



Developer Guide

Amazon Quantum Ledger Database (Amazon QLDB)



Amazon Quantum Ledger Database (Amazon QLDB): Developer Guide

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon QLDB?	1
Amazon QLDB video	1
Amazon QLDB pricing	2
Getting started with QLDB	2
Overview	2
Journal first	3
Immutable	4
Cryptographically verifiable	5
SQL-like and document flexible	6
Open-source developer tools	7
Serverless and highly available	7
Enterprise grade	7
From relational to ledger	7
Core concepts	10
QLDB data object model	11
Journal-first transactions	12
Querying your data	14
Data storage	14
QLDB API model	14
Next steps	15
Journal contents	16
Block example	16
Block contents	19
Redacted revisions	21
Sample application	22
See also	23
QLDB glossary	23
Accessing Amazon QLDB	27
Prerequisites	27
Sign up for an AWS account	27
Create a user with administrative access	28
Manage QLDB permissions in IAM	29
Grant programmatic access	29
How to access Amazon QLDB	31

Using the console	32
PartiQL editor quick reference	32
Using the AWS CLI (management API only)	37
Installing and configuring the AWS CLI	38
Using the AWS CLI with QLDB	38
Using the Amazon QLDB shell (data API only)	39
Prerequisites	40
Installing the shell	40
Invoking the shell	41
Shell parameters	41
Command reference	43
Running individual statements	44
Managing transactions	45
Exiting the shell	47
Example	47
Using the API	48
Getting started with the console	49
Prerequisites and considerations	50
Setting up permissions	51
Step 1: Create a new ledger	52
Step 2: Create tables, indexes, and sample data	54
Step 3: Query the tables	63
Step 4: Modify documents	65
Step 5: View the revision history	68
Step 6: Verify a document	71
To request a digest	71
To verify a document revision	73
Step 7: Clean up	74
Next steps	75
Getting started with the driver	76
Java driver	77
Driver resources	78
Prerequisites	78
Setting your default AWS credentials and Region	79
Installation	79
Quick start tutorial	82

Cookbook reference	91
.NET driver	108
Driver resources	109
Prerequisites	109
Installation	110
Quick start tutorial	111
Cookbook reference	135
Go driver	169
Driver resources	169
Prerequisites	169
Installation	170
Quick start tutorial	171
Cookbook reference	183
Node.js driver	197
Driver resources	198
Prerequisites	198
Installation	199
Setup recommendations	204
Quick start tutorial	207
Cookbook reference	226
Python driver	247
Driver resources	248
Prerequisites	248
Installation	249
Quick start tutorial	250
Cookbook reference	257
Session management with the driver	270
Session lifecycle	271
Session expiration	271
Session handling in the QLDB driver	271
Driver recommendations	274
Configuring the QldbDriver object	274
Retrying on exceptions	277
Optimizing performance	278
Running multiple statements per transaction	279
Driver retry policy	284

Types of retryable errors	284
Default retry policy	285
Common errors	285
Sample application tutorial	288
Java tutorial	289
Node.js tutorial	458
Python tutorial	520
Working with Amazon Ion	608
Prerequisites	609
Bool	609
Int	612
Float	616
Decimal	620
Timestamp	624
String	628
Blob	631
List	634
Struct	639
Null values and dynamic types	645
Down-converting to JSON	651
Working with data and history	652
Creating tables with indexes and inserting documents	653
Creating tables and indexes	654
Inserting documents	655
Querying your data	657
Basic queries	658
Projections and filters	660
Joins	660
Nested data	661
Querying document metadata	663
Committed view	664
Joining the committed and user views	666
Using the BY clause to query document ID	666
Joining on document ID	667
Updating and deleting documents	668
Making document revisions	668

Querying revision history	670
History function	670
History query example	671
Redacting document revisions	674
Redaction stored procedure	675
Checking whether a redaction is complete	676
Redaction example	676
Deleting and redacting an active revision	679
Redacting a particular field within a revision	680
Optimizing query performance	680
Transaction timeout limit	681
Concurrency conflicts	681
Optimal query patterns	681
Query patterns to avoid	683
Monitoring performance	684
Getting PartiQL statement statistics	684
I/O usage	685
Timing information	691
Querying the system catalog	698
Managing tables	699
Tagging tables on creation	700
Dropping tables	700
Querying the history of inactive tables	701
Reactivating tables	701
Managing indexes	702
Creating indexes	702
Describing indexes	704
Dropping indexes	705
Common errors	706
Unique IDs	707
Properties	708
Usage	708
Examples	708
Concurrency model	709
Optimistic concurrency control	709
Using indexes to avoid full table scans	710

Insertion OCC conflicts	711
Making transactions idempotent	713
Redaction OCC conflicts	713
Managing concurrent sessions	714
Verification	715
What kind of data can you verify in QLDB?	716
What does data integrity mean?	717
How does verification work?	717
Hashing	717
Digest	719
Merkle tree	719
Proof	719
Verification example	720
How does data redaction affect verification?	721
Recalculating a revision hash	721
Getting started with verification	721
Step 1: Requesting a digest	722
AWS Management Console	723
QLDB API	724
Step 2: Verifying your data	725
AWS Management Console	725
QLDB API	727
Verification results	728
Using a proof to recalculate your digest	730
Tutorial: Verifying data using an AWS SDK	730
Prerequisites	731
Step 1: Request a digest	732
Step 2: Query the document revision	733
Step 3: Request a proof for the revision	735
Step 4: Recalculate the digest from the revision	740
Step 5: Request a proof for the journal block	742
Step 6: Recalculate the digest from the block	747
Run the full code example	755
Common errors	779
Exporting journal data	782
Requesting an export	783

AWS Management Console	783
QLDB API	786
Export job expiration	787
Export output	788
Manifest files	788
Data objects	790
Down-converting to JSON	794
Export processor library (Java)	795
Export permissions	795
Create a permissions policy	796
Create an IAM role	798
Common errors	800
Streams	803
Common use cases	804
Consuming your stream	804
Delivery guarantee	805
Delivery latency considerations	805
Getting started with streams	806
Creating and managing streams	806
Stream parameters	807
Stream ARN	808
AWS Management Console	809
Stream states	811
Handling impaired streams	812
Developing with streams	813
QLDB journal stream APIs	813
Sample applications	814
Stream records	817
Control records	818
Block summary records	819
Revision details records	821
Handling duplicate and out-of-order records	822
Stream permissions	822
Create a permissions policy	823
Create an IAM role	826
Common errors	828

Ledger management	831
Basic operations for ledgers	831
Creating a ledger	832
Describing a ledger	836
Updating a ledger	838
Updating a ledger permissions mode	841
Deleting a ledger	843
Listing ledgers	844
AWS CloudFormation resources	846
QLDB and AWS CloudFormation templates	846
Learn more about AWS CloudFormation	846
Tagging resources	847
Supported resources in Amazon QLDB	848
Tag naming and usage conventions	848
Managing tags	849
Tagging resources on creation	849
Security	851
Data protection	852
Encryption at rest	853
Encryption in transit	870
Identity and Access Management	871
Audience	872
Authenticating with identities	872
Managing access using policies	875
How Amazon QLDB works with IAM	878
Getting started with the standard permissions mode	887
Identity-based policy examples	899
Cross-service confused deputy prevention	917
AWS managed policies	920
Troubleshooting	924
Logging and monitoring	926
Monitoring tools	927
Monitoring with Amazon CloudWatch	929
Automating with CloudWatch Events	934
Logging Amazon QLDB API calls with AWS CloudTrail	935
Compliance validation	954

Resilience	956
Storage durability	956
Data durability features	957
Infrastructure security	958
AWS PrivateLink	958
Troubleshooting	963
Running transactions using the QLDB driver	963
Exporting journal data	966
Streaming journal data	968
Verifying journal data	970
PartiQL reference	973
What is PartiQL?	974
PartiQL in Amazon QLDB	974
PartiQL quick tips in QLDB	974
PartiQL reference conventions	975
Data types	976
QLDB documents	977
Ion document structure	978
PartiQL-Ion type mapping	979
Document ID	979
Querying Ion with PartiQL	980
Syntax and semantics	981
Backtick notation	983
Path navigation	984
Aliasing	984
PartiQL specification	985
PartiQL commands	985
DDL statements	986
DML statements	986
CREATE INDEX	987
CREATE TABLE	989
DELETE	992
DROP INDEX	994
DROP TABLE	995
FROM (INSERT, REMOVE, or SET)	997
INSERT	1002

SELECT	1005
UPDATE	1011
UNDROP TABLE	1016
PartiQL functions	1017
Aggregate functions	1018
Conditional functions	1018
Date and time functions	1018
Scalar functions	1019
String functions	1019
Data type formatting functions	1019
AVG	1019
CAST	1021
CHAR_LENGTH	1025
CHARACTER_LENGTH	1026
COALESCE	1026
COUNT	1027
DATE_ADD	1029
DATE_DIFF	1031
EXISTS	1032
EXTRACT	1034
LOWER	1036
MAX	1037
MIN	1038
NULLIF	1039
SIZE	1041
SUBSTRING	1042
SUM	1044
TO_STRING	1045
TO_TIMESTAMP	1047
TRIM	1049
TXID	1050
UPPER	1051
UTCNOW	1052
Timestamp format strings	1053
PartiQL stored procedures	1055
REDACT_REVISION	1056

PartiQL operators	1060
Arithmetic operators	1060
Comparison operators	1061
Logical operators	1061
String operators	1062
Reserved keywords	1062
Amazon Ion reference	1069
What is Amazon Ion?	1069
Ion specification	1070
JSON compatible	1070
Extensions from JSON	1071
Ion text example	1072
API references	1072
Amazon Ion code examples	1073
API reference	1088
Actions	1089
Amazon QLDB	1090
Amazon QLDB Session	1165
Data Types	1173
Amazon QLDB	1174
Amazon QLDB Session	1191
Common Errors	1214
Common Parameters	1215
Quotas and limits	1219
Default quotas	1219
Fixed quotas	1220
Ledger quota	1221
Document size	1222
Transaction size	1222
Naming constraints	1223
Related information	1224
Technical documentation	1224
GitHub repositories	1225
AWS blog posts and articles	1226
Media	1228
General AWS resources	1229

Release history 1231

What is Amazon QLDB?

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon Quantum Ledger Database (Amazon QLDB) is a fully managed ledger database that provides a transparent, immutable, and cryptographically verifiable transaction log owned by a central trusted authority. You can use Amazon QLDB to track all application data changes, and maintain a complete and verifiable history of changes over time. To learn more about the variety of database options available on Amazon Web Services, see [Choosing the right database for your organization on AWS](#).

Ledgers are typically used to record a history of economic and financial activity in an organization. Many organizations build applications with ledger-like functionality because they want to maintain an accurate history of their applications' data. For example, they might want to track the history of credits and debits in banking transactions, verify the data lineage of an insurance claim, or trace the movement of an item in a supply chain network. Ledger applications are often implemented using custom audit tables or audit trails created in relational databases.

Amazon QLDB is a new class of database that helps eliminate the need to engage in the complex development effort of building your own ledger-like applications. With QLDB, the history of changes to your data is immutable—it can't be overwritten or altered in place. And using cryptography, you can verify that there have been no unintended changes to your application's data. QLDB uses an immutable transactional log, known as a *journal*. The journal is append-only and is composed of a sequenced and hash-chained set of *blocks* that contain your committed data.

Amazon QLDB video

For an overview of Amazon QLDB and how it can benefit you, watch this [QLDB overview video](#) on YouTube.

Amazon QLDB pricing

With Amazon QLDB, you pay only for what you use with no minimum fees or mandatory service usage. You pay only for the resources your ledger database consumes, and you do not need to provision in advance.

For more information, see [Amazon QLDB pricing](#).

Getting started with QLDB

We recommend that you begin by reading the following topics:

- [Overview of Amazon QLDB](#) – To get a high-level overview of QLDB.
- [Core concepts and terminology in Amazon QLDB](#) – To learn fundamental QLDB concepts and terminology.
- [Accessing Amazon QLDB](#) – To learn how to access QLDB using the AWS Management Console, API, or AWS Command Line Interface (AWS CLI).
- [How Amazon QLDB works with IAM](#) – To learn how to control access to QLDB using AWS Identity and Access Management (IAM).

To get started quickly with the QLDB console, see [Getting started with the Amazon QLDB console](#).

To learn about developing with QLDB using an AWS provided driver, see [Getting started with the Amazon QLDB driver](#).

Overview of Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following sections provide a high-level overview of Amazon QLDB service components and how they interact.

Topics

- [Journal first](#)
- [Immutable](#)
- [Cryptographically verifiable](#)
- [SQL-like and document flexible](#)
- [Open-source developer tools](#)
- [Serverless and highly available](#)
- [Enterprise grade](#)

Journal first

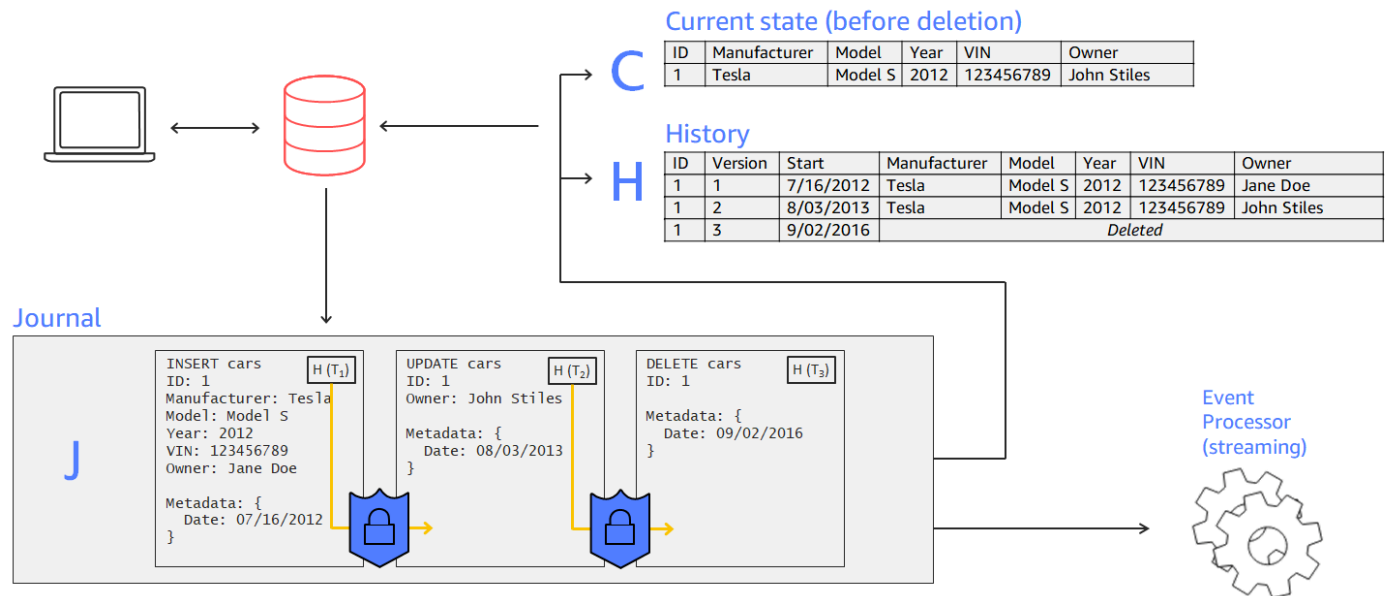
In traditional database architecture, you generally write data in tables as part of a transaction. A transaction log—typically an internal implementation—records all of the transactions and the database modifications that they make. The transaction log is a critical component of the database. You need the log to replay transactions in the event of a system failure, disaster recovery, or data replication. However, database transaction logs aren't immutable and aren't designed to provide direct and easy access to users.

In Amazon QLDB, the journal is the core of the database. Structurally similar to a transaction log, the journal is an immutable, append-only data structure that stores your application data along with the associated metadata. All write transactions, including updates and deletes, are committed to the journal first.

QLDB uses the journal to determine the current state of your ledger data by materializing it into queryable, user-defined tables. These tables also provide an accessible history of all transaction data, including document revisions and metadata. In addition, the journal handles concurrency, sequencing, cryptographic verification, and availability of the ledger data.

The following diagram illustrates the QLDB journal architecture.

Amazon QLDB: the journal is the database



- In this example, an application connects to a ledger and runs transactions that insert, update, and delete a document into a table named cars.
- The data is first written to the journal in sequenced order.
- Then the data is materialized into the table with built-in views. These views let you query both the current state and the complete history of the car, with each revision assigned a version number.
- You can also export or stream data directly from the journal.

Immutable

Because the QLDB journal is append-only, it keeps a full record of all changes to your data that can't be modified or overwritten. There are no APIs or other methods to alter any committed data in place. This journal structure lets you access and query the full history of your ledger.

Note

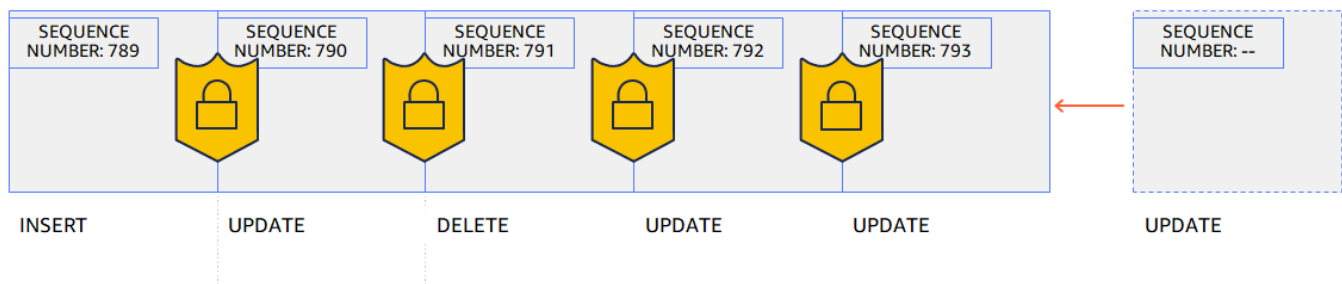
The only exception to immutability that QLDB supports is *data redaction*. With this feature, you can comply with regulatory statutes such as the General Data Protection Regulation (GDPR) in the European Union and the California Consumer Privacy Act (CCPA).

QLDB provides a redaction operation that lets you permanently delete inactive document revisions in the history of a table. This operation deletes only the user data in the specified revision, and leaves the journal sequence and the document metadata unchanged. This maintains the overall data integrity of your ledger. For more information, see [Redacting document revisions](#).

QLDB writes one block to the journal in a transaction. Each block contains entry objects that represent the documents that you insert, update, and delete, along with the statements that you ran to commit them. These blocks are sequenced and hash-chained to guarantee data integrity.

The following diagram illustrates this journal structure.

Records cannot be altered



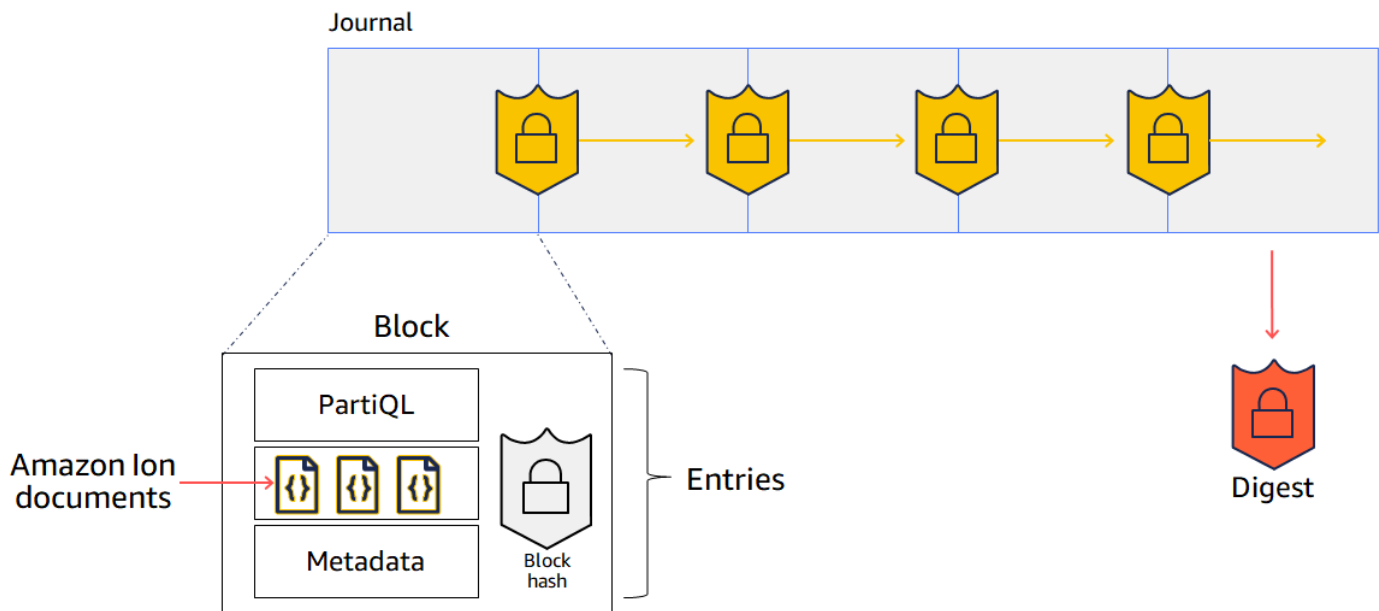
The diagram shows that transactions are committed to the journal as blocks that are hash-chained for verification. Each block has a sequence number to specify its address.

Cryptographically verifiable

Journal blocks are sequenced and chained together with cryptographic hashing techniques, similar to blockchains. QLDB uses the journal's hash chain to provide transactional data integrity using a cryptographic verification method. Using a *digest* (a hash value that represents a journal's full hash chain as of a point in time) and a *Merkle audit proof* (a mechanism that proves the validity of any node within a binary hash tree), you can verify that there have been no unintended changes to your data at any time.

The following diagram shows a digest that covers a journal's full hash chain at a point in time.

Hash chaining using SHA-256



In this diagram, the journal blocks are hashed using the SHA-256 cryptographic hash function and are sequentially chained to subsequent blocks. Each block contains entries that include your data documents, metadata, and the PartiQL statements that ran in the transaction.

For more information, see [Data verification in Amazon QLDB](#).

SQL-like and document flexible

QLDB uses PartiQL as its query language and Amazon Ion as its document-oriented data model. PartiQL is an open-source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. When you're querying flat documents, the syntax is the same as using SQL to query relational tables. To learn more about the QLDB implementation of PartiQL, see the [Amazon QLDB PartiQL reference](#).

Amazon Ion is a superset of JSON. Ion is an open-source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data. To learn more about Ion in QLDB, see the [Amazon Ion data format reference in Amazon QLDB](#).

For a high-level comparison of the core components and features in traditional relational databases versus QLDB, see [From relational to ledger](#).

Open-source developer tools

To simplify application development, QLDB provides open-source drivers in various programming languages. You can use these drivers to interact with the transactional data API by running PartiQL statements on a ledger and processing the results of those statements. For information and tutorials about the driver languages that are currently supported, see [Getting started with the Amazon QLDB driver](#).

Amazon Ion also provides client libraries that process Ion data for you. For developer guides and code examples of processing Ion data, see the [Amazon Ion documentation](#) on GitHub.

Serverless and highly available

QLDB is fully managed, serverless, and highly available. The service automatically scales to support the demands of your application, and you don't need to provision instances or capacity. Multiple copies of your data are replicated within an Availability Zone and across Availability Zones in an AWS Region.

Enterprise grade

QLDB transactions are fully compliant with atomicity, consistency, isolation, and durability (ACID) properties. QLDB uses optimistic concurrency control (OCC), and transactions operate with full serializability—the highest level of isolation. This means that there's no risk of seeing phantom reads, dirty reads, write skew, or other similar concurrency issues. For more information, see [Amazon QLDB concurrency model](#).

From relational to ledger

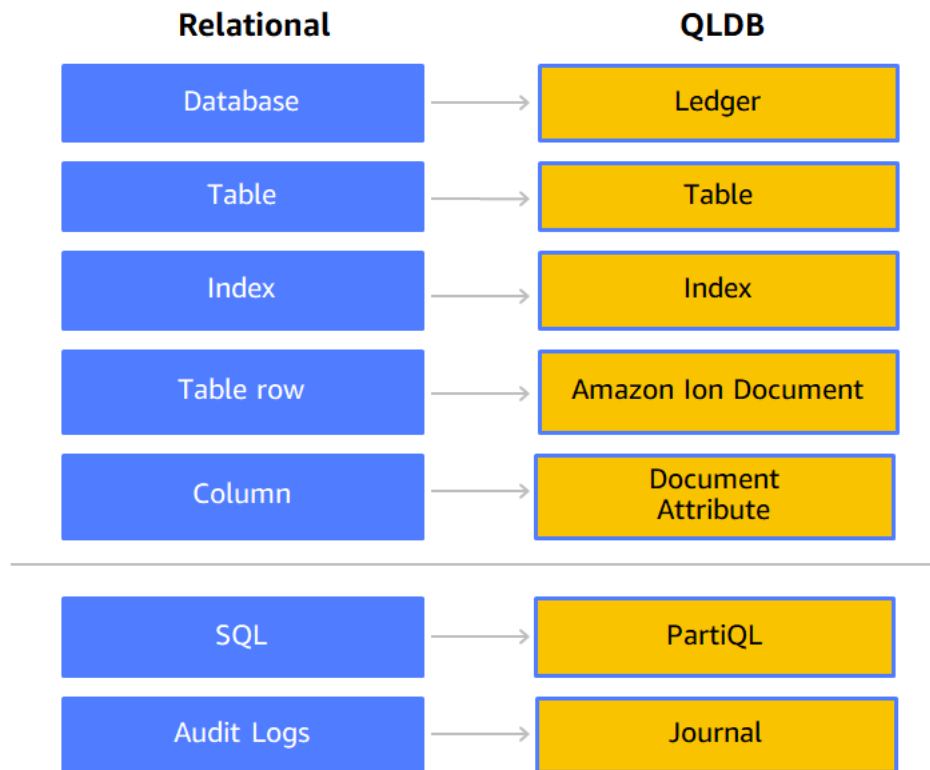
Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

If you're an application developer, you might have some experience using a relational database management system (RDBMS) and Structured Query Language (SQL). As you begin working with Amazon QLDB, you will encounter many similarities. As you progress to more advanced topics, you

will also encounter powerful new features that QLDB has built on the RDBMS foundation. This section describes common database components and operations, comparing and contrasting them with their equivalents in QLDB.

The following diagram shows the mapping constructs of the core components between a traditional RDBMS and Amazon QLDB.



The following table shows the primary high-level similarities and differences of built-in operational features between a traditional RDBMS and QLDB.

Operation	RDBMS	QLDB
Creating tables	CREATE TABLE statement that defines all column names and data types	CREATE TABLE statement that doesn't define any table attributes or data types to allow schemaless and open content
Creating indexes	CREATE INDEX statement	CREATE INDEX statement for any top-level fields on a table

Operation	RDBMS	QLDB
Inserting data	INSERT statement that specifies values within a new row or tuple that adheres to the schema as defined by the table	INSERT statement that specifies values within a new document in any valid Amazon Ion format regardless of the existing documents in the table
Querying data	SELECT-FROM-WHERE statement	SELECT-FROM-WHERE statement in the same syntax as SQL when querying flat documents
Updating data	UPDATE-SET-WHERE statement	UPDATE-SET-WHERE statement in the same syntax as SQL when updating flat documents
Deleting data	DELETE-FROM-WHERE statement	DELETE-FROM-WHERE statement in the same syntax as SQL when deleting flat documents
Nested and semistructured data	Flat rows or tuples only	Documents that can have any structured, semistructured, or nested data as supported by the Amazon Ion data format and the PartiQL query language
Querying metadata	No built-in metadata	SELECT statement that queries from the built-in committed view of a table
Querying revision history	No built-in data history	SELECT statement that queries from the built-in history function

Operation	RDBMS	QLDB
Cryptographic verification	No built-in cryptography or immutability	APIs that return a digest of a journal and a proof that verifies the integrity of any document revision relative to that digest

For an overview of the core concepts and terminology in QLDB, see [Core concepts](#).

For detailed information about the process of creating, querying, and managing your data in a ledger, see [Working with data and history](#).

Core concepts and terminology in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section provides an overview of the core concepts and terminology in Amazon QLDB, including ledger structure and how a ledger manages data. As a ledger database, QLDB differs from other document-oriented databases when it comes to the following key concepts.

Topics

- [QLDB data object model](#)
- [Journal-first transactions](#)
- [Querying your data](#)
- [Data storage](#)
- [QLDB API model](#)
- [Next steps](#)

QLDB data object model

The fundamental data object model in Amazon QLDB is described as follows:

1. Ledger

Your first step is to create a *ledger*, which is the primary AWS resource type in QLDB. To learn how to create a ledger, see [Step 1: Create a new ledger](#) in *Getting started with the console*, or [Basic operations for Amazon QLDB ledgers](#).

For both the ALLOW_ALL and STANDARD permissions modes of a ledger, you create AWS Identity and Access Management (IAM) policies that grant permissions to run API operations on this ledger resource.

Ledger ARN format:

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}
```

2. Journal and tables

To start writing data in a QLDB ledger, you first create a *table* with a basic [CREATE TABLE](#) statement. Ledger data consists of revisions of documents that are committed to the ledger's *journal*. You commit document revisions to the ledger in the context of user-defined tables. In QLDB, a table represents a materialized view of a collection of document revisions from the journal.

In the STANDARD permissions mode of a ledger, you must create IAM policies that grant permissions to run PartiQL statements on this table resource. With permissions on a table resource, you can run statements that access the current state of the table. You can also query the revision history of the table by using the built-in `history()` function.

Table ARN format:

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/table/${table-id}
```

For more information about granting permissions on a ledger and its associated resources, see [How Amazon QLDB works with IAM](#).

3. Documents

Tables consist of revisions of [QLDB documents](#), which are datasets in [Amazon Ion](#) struct format. A *document revision* represents a single version of a sequence of documents that are identified by a unique document ID.

QLDB stores the complete change history of your committed documents. A table lets you query the current state of its documents, while the `history()` function lets you query the entire revision history of a table's documents. For details on querying and writing revisions, see [Working with data and history](#).

4. System catalog

Each ledger also provides a system-defined *catalog* resource that you can query to list all of the tables and indexes in a ledger. In the STANDARD permissions mode of a ledger, you need the `qldb:PartiQLSelect` permission on this catalog resource to do the following:

- Run SELECT statements on the system catalog table [information_schema.user_tables](#).
- View table and index information on the ledger details page on the [QLDB console](#).
- View the list of tables and indexes in the *PartiQL editor* on the QLDB console.

Catalog ARN format:

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/information_schema/
user_tables
```

Journal-first transactions

When an application reads or writes data in a QLDB ledger, it does so in a database transaction. All transactions are subject to limits as defined in [Quotas and limits in Amazon QLDB](#). Within a transaction, QLDB does the following steps:

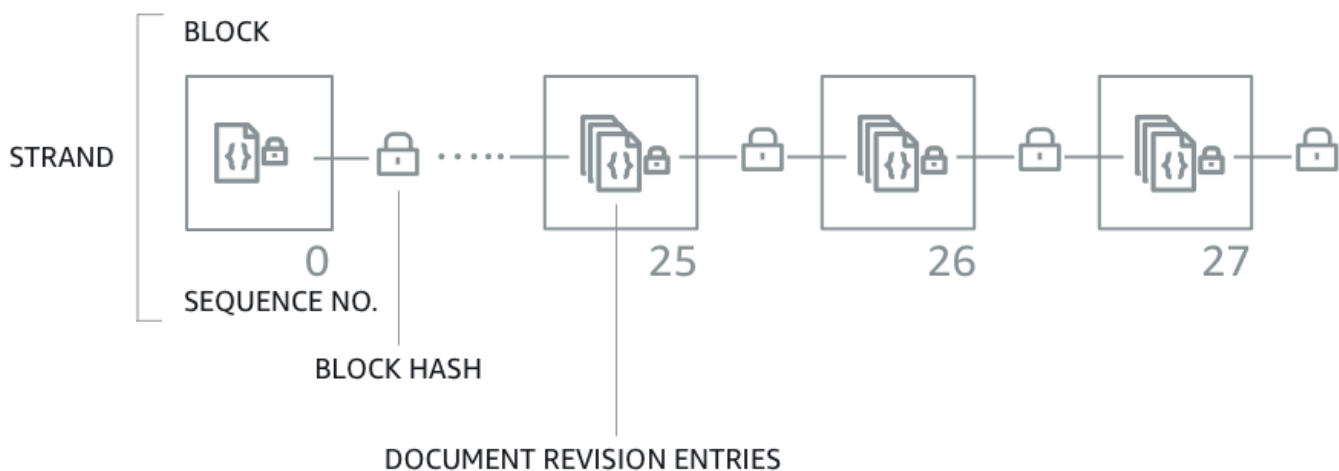
1. Read the current state of the data from the ledger.
2. Perform the statements provided in the transaction, and then check for any conflicts using [optimistic concurrency control \(OCC\)](#) to ensure fully serializable isolation.
3. If no OCC conflicts are found, return the transaction results as follows:
 - For reads, return the result set and commit the SELECT statements to the journal in an append-only manner.

- For writes, commit any updates, deletes, or newly inserted data to the journal in an append-only manner.

The *journal* represents a complete and immutable history of all the changes to your data. QLDB writes one chained *block* to the journal in a transaction. Each block contains *entry* objects that represent the document revisions that you insert, update, and delete, along with the [PartiQL](#) statements that committed them.

The following diagram illustrates this journal structure.

QLDB JOURNAL



The diagram shows that transactions are committed to the journal as blocks that contain document revision entries. Each block is hashed and chained to subsequent blocks for [verification](#). Each block has a sequence number to specify its address within the strand.

Note

In Amazon QLDB, a strand is a partition of your ledger's journal. QLDB currently supports journals with a single strand only.

For information about the data contents in a block, see [Journal contents in Amazon QLDB](#).

Querying your data

QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. A ledger provides queryable table views of your data based on the transaction information that is committed to the journal. A table *view* in QLDB is a subset of the data in a table. Views are maintained in real time, so that they're always available for applications to query.

You can query the following system-defined views using PartiQL SELECT statements:

- *User* – The latest active revision of only the data that you wrote in the table (that is, the current state of your user data). This is the default view in QLDB.
- *Committed* – The latest active revision of both your user data and the system-generated metadata. This is the full system-defined table that corresponds directly to your user table.

In addition to these queryable views, you can query the revision history of your data by using the built-in [History function](#). The history function returns both your user data and the associated metadata in the same schema as the *committed view*.

Data storage

There are two types of data storage in QLDB:

- *Journal storage* – The disk space that is used by a ledger's journal. The journal is append-only and contains the complete, immutable, and verifiable history of all the changes to your data.
- *Indexed storage* – The disk space that is used by a ledger's tables, indexes, and indexed history. Indexed storage consists of ledger data that is optimized for high-performance queries.

After your data is committed to the journal, it's materialized into the tables that you defined. These tables are optimized for faster and more efficient queries. When an application uses the transactional data API to read data, it accesses the tables and indexes that are stored in your indexed storage.

QLDB API model

QLDB provides two types of APIs that your application code can interact with:

- *Amazon QLDB* – The QLDB resource management API (also known as the *control plane*). This API is used only for managing ledger resources and for non-transactional data operations. You can

use these operations to create, delete, describe, list, and update ledgers. You can also verify data cryptographically, and export or stream journal blocks.

- *Amazon QLDB Session* – The QLDB transactional data API. You can use this API to run data transactions on a ledger with [PartiQL](#) statements.

Important

Instead of interacting directly with the *QLDB Session* API, we recommend using the QLDB driver or the QLDB shell to run data transactions on a ledger.

- If you're working with an AWS SDK, use the QLDB driver. The driver provides a high-level abstraction layer above the *QLDB Session* data API and manages the `SendCommand` operation for you. For information and a list of supported programming languages, see [Getting started with the driver](#).
- If you're working with the AWS CLI, use the QLDB shell. The shell is a command line interface that uses the QLDB driver to interact with a ledger. For information, see [Using the Amazon QLDB shell \(data API only\)](#).

For more information about these API operations, see the [Amazon QLDB API reference](#).

Next steps

To learn how to use a ledger with your data, see [Working with data and history in Amazon QLDB](#) and follow the examples that describe the process of creating tables, inserting data, and running basic queries. This guide explains how these concepts work in depth, using sample data and query examples for context.

To get started quickly with a sample application tutorial using the QLDB console, see [Getting started with the Amazon QLDB console](#).

For a list of the key terms and definitions described in this section, see the [Amazon QLDB glossary](#).

Journal contents in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, the *journal* is the immutable transactional log that stores the complete and verifiable history of all the changes to your data. The journal is append-only and is composed of a sequenced and hash-chained set of *blocks* that contain your committed data and other system metadata. QLDB writes one chained block to the journal in a transaction.

This section provides an example of a journal block with sample data and describes the contents of a block.

Topics

- [Block example](#)
- [Block contents](#)
- [Redacted revisions](#)
- [Sample application](#)
- [See also](#)

Block example

A journal block contains transaction metadata along with entries that represent the document revisions that were committed in the transaction and the [PartiQL](#) statements that committed them.

The following is an example of a block with sample data.

Note

This block example is provided for informational purposes only. The hashes shown aren't real calculated hash values.

```

{
  blockAddress:{
    strandId:"4o5UuzWSW5PIo0Gm5jPA6J",
    sequenceNo:25
  },
  transactionId:"3gtB8Q8dfIMA8lQ5pzHAMo",
  blockTimestamp:2022-06-08T18:46:46.512Z,
  blockHash:{{QS5lJt8vRxT30L90GL5oU1pxFte+U1EwakYBCrvGQ4A=}},
  entriesHash:{{buYYc5kV4rrRtJAsrIQnfnhgkzfQ8BKjI0C2vFnYQEw=}},
  previousBlockHash:{{I1UKRIWUgkM1X6042kcoZ/eN1rn0uxhDTc08zw9kZ5I=}},
  entriesHashList:[
    {{BUCXP6oYgmug2AfPZcAZup2lKolJNTbTuV5RA1VaFpo=}},
    {{cTIRkjuULzp/4KaUEsb/S7+TG8FvpFiZHT4tEJGcANc=}},
    {{3aktJSMYJ3C5StZv4WIJLu/w3D8mGtduZvP0ldKUaUM=}},
    {{GPKIJ1+o8mMZmPj/35ZQXoca2z64MVYMCwqs/g080IM=}}
  ],
  transactionInfo:{
    statements:[
      {
        statement:"INSERT INTO VehicleRegistration VALUE ?",
        startTime:2022-06-08T18:46:46.063Z,
        statementDigest:{{KY2nL6UGUPs5lXCLVXcUaBxcEIop0Jvk4MEjcFVBfwI=}}
      },
      {
        statement:"SELECT p_id FROM Person p BY p_id WHERE p.FirstName = ? and
p.LastName = ?",
        startTime:2022-06-08T18:46:46.173Z,
        statementDigest:{{QS2nfB8XBf2ozlDx0nvtsli0YDSmNHMYC3IRH4Uh690=}}
      },
      {
        statement:"UPDATE VehicleRegistration r SET r.Owners.PrimaryOwner.PersonId = ?
WHERE r.VIN = ?",
        startTime:2022-06-08T18:46:46.278Z,
        statementDigest:{{nGtIA9Qh0/dwIpl0R8J5CTeqyUVtNUQgXfltdUo2Aq4=}}
      },
      {
        statement:"DELETE FROM DriversLicense l WHERE l.LicenseNumber = ?",
        startTime:2022-06-08T18:46:46.385Z,
        statementDigest:{{ka783dcEP58Q9AVQ1m9N0Jd3JAmEvXLjz100jN1BojQ=}}
      }
    ],
    documents:{
      HwVFkn8IMRa0xjze5xcgga:{

```

```

        tableName:"VehicleRegistration",
        tableId:"HQZ6cgIMUi204Lq1tT4oaJ",
        statements:[0,2]
    },
    IiPTRxLGJZa342zHFCFT15:{
        tableName:"DriversLicense",
        tableId:"BvtXEB1JxZg0lJlBAAtbtSV",
        statements:[3]
    }
}
},
revisions:[
    {
        hash:{{FR1IWcWew0yw1TnRklo2YMF/qtwb7ohsu5FD8A4DSVg=}}
    },
    {
        blockAddress:{
            strandId:"4o5UuzWSW5PIo0Gm5jPA6J",
            sequenceNo:25
        },
        hash:{{6TTHbcfIVdWoFC/j90B0Zi0JdHzhjSXo1tW+uHd6Dj4=}},
        data:{
            VIN:"1N4AL11D75C109151",
            LicensePlateNumber:"LEWISR261LL",
            State:"WA",
            City:"Seattle",
            PendingPenaltyTicketAmount:90.25,
            ValidFromDate:2017-08-21,
            ValidToDate:2020-05-11,
            Owners:{
                PrimaryOwner:{
                    PersonId:"3Ax20JIix5J2ulu2rCMvo2"
                },
                SecondaryOwners:[]
            }
        },
        metadata:{
            id:"HwVFkn8IMRa0xjze5xcgga",
            version:0,
            txTime:2022-06-08T18:46:46.492Z,
            txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
        }
    },
    {

```



```
    blockAddress:{
      strandId:"4o5UuzWSW5PIo0Gm5jPA6J",
      sequenceNo:25
    },
    hash:{{ZVF/f1uSqd5DIMqzI04CCHaCGFK/J0Jf5AFzSEk0190=}},
    metadata:{
      id:"IiPTRxLGJZa342zHFCFT15",
      version:1,
      txTime:2022-06-08T18:46:46.492Z,
      txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
    }
  }
]
```

In the `revisions` field, some revision objects might only contain a hash value and no other attributes. These are internal-only system revisions that don't contain user data. The hashes of these revisions are part of the journal's full hash chain, which is required for cryptographic verification.

Block contents

A journal block has the following fields:

blockAddress

The location of the block in the journal. An address is an [Amazon Ion](#) structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"B1FTjlSXze9BIh1K0szcE3",sequenceNo:14}`

transactionId

The unique ID of the transaction that committed the block.

blockTimestamp

The timestamp when the block was committed to the journal.

blockHash

The 256-bit hash value that uniquely represents the block. This is the hash of the concatenation of `entriesHash` and `previousBlockHash`.

entriesHash

The hash that represents all of the entries within the block, including internal-only system entries. This is the root hash of the [Merkle tree](#) in which the leaf nodes consist of all the hashes in `entriesHashList`.

previousBlockHash

The hash of the previous chained block in the journal.

entriesHashList

The list of hashes that represent each entry within the block. This list can include the following entry hashes:

- The `lon` hash that represents `transactionInfo`. This value is calculated by taking the `lon` hash of the entire `transactionInfo` structure.
- The root hash of the Merkle tree in which the leaf nodes consist of all the hashes in `revisions`.
- The `lon` hash that represents `redactionInfo`. This hash only exists in blocks that were committed by a redaction transaction. Its value is calculated by taking the `lon` hash of the entire `redactionInfo` structure.
- Hashes that represent internal-only system metadata. These hashes might not exist in all blocks.

transactionInfo

An Amazon `lon` structure that contains information about the statements in the transaction that committed the block. This structure has the following fields:

- `statements` – The list of PartiQL statements and the `startTime` when they started running. Each statement has a `statementDigest` hash, which is required to calculate the hash of the `transactionInfo` structure.
- `documents` – The document IDs that were updated by the statements. Each document includes the `tableName` and `tableId` that it belongs to, and the index of each statement that updated it.

revisions

The list of document revisions that were committed in the block. Each revision structure contains all of the fields from the [committed view](#) of the revision.

This can also include hashes that represent internal-only system revisions that are part of the full hash chain of a journal.

Redacted revisions

In Amazon QLDB, a DELETE statement only logically deletes a document by creating a new revision that marks it as deleted. QLDB also supports a *data redaction* operation that lets you permanently delete inactive document revisions in the history of a table.

The redaction operation deletes only the user data in the specified revision, and leaves the journal sequence and the document metadata unchanged. This maintains the overall data integrity of your ledger. For more information and an example of a redaction operation, see [Redacting document revisions](#).

Redacted revision example

Consider the previous [block example](#). In this block, suppose that you redact the revision that has a document ID of HwVFkn8IMRa0xjze5xcgga and a version number of 0.

After the redaction is complete, the user data in the revision (represented by the data structure) is replaced by a new dataHash field. The value of this field is the Ion hash of the removed data structure. As a result, the ledger maintains its overall data integrity and remains cryptographically verifiable through the existing verification API operations.

The following revision example shows the results of this redaction, with the new dataHash field highlighted in *red italics*.

Note

This revision example is provided for informational purposes only. The hashes shown aren't real calculated hash values.

```
...
{
  blockAddress:{
    strandId:"4o5UuzWSW5PIo0Gm5jPA6J",
    sequenceNo:25
  },
  hash:{{6TTHbcfIVdWoFC/j90B0Zi0JdHzhjSXo1tW+uHd6Dj4=}},
}
```

```
dataHash:{{s83jd7sfhsdfhksj7hskjdfjfpIPP/DP2hvionas2d4=}},
metadata:{
  id:"HwVFkn8IMRa0xjze5xcgga",
  version:0,
  txTime:2022-06-08T18:46:46.492Z,
  txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
}
}
...
```

QLDB also appends a new block to the journal for the completed redaction request. This block includes an additional `redactionInfo` entry that contains a list of revisions that were redacted in the transaction, as shown in the following example.

```
...
redactionInfo:{
  revisions:[
    {
      blockAddress:{
        strandId:"4o5UuzWSW5PIo0Gm5jPA6J",
        sequenceNo:25
      },
      tableId:"HQZ6cgIMUi204Lq1tT4oaJ",
      documentId:"HwVFkn8IMRa0xjze5xcgga",
      version:0
    }
  ]
}
...
```

Sample application

For a Java code example that validates a journal's hash chain using exported data, see the GitHub repository [aws-samples/amazon-qldb-dmv-sample-java](https://github.com/aws-samples/amazon-qldb-dmv-sample-java). This sample application includes the following class files:

- [ValidateQldbHashChain.java](#) – Contains tutorial code that exports journal blocks from a ledger and uses the exported data to validate the hash chain between blocks.
- [JournalBlock.java](#) – Contains a method named `verifyBlockHash()` that demonstrates how to calculate each individual hash component within a block. This method is called by the tutorial code in `ValidateQldbHashChain.java`.

For instructions on how to download and install this complete sample application, see [Installing the Amazon QLDB Java sample application](#). Before you run the tutorial code, make sure that you follow Steps 1–3 in the [Java tutorial](#) to set up a sample ledger and load it with sample data.

See also

For more information about journals in QLDB, see the following topics:

- [Exporting journal data from Amazon QLDB](#) – To learn how to export journal data to Amazon Simple Storage Service (Amazon S3).
- [Streaming journal data from Amazon QLDB](#) – To learn how to stream journal data to Amazon Kinesis Data Streams.
- [Data verification in Amazon QLDB](#) – To learn about cryptographic verification of journal data.

Amazon QLDB glossary

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following are definitions for key terms that you might encounter as you work with Amazon QLDB.

[block](#) | [digest](#) | [document](#) | [document ID](#) | [document revision](#) | [entry](#) | [field](#) | [index](#) | [indexed storage](#) | [journal](#) | [journal block](#) | [journal storage](#) | [journal strand](#) | [journal tip](#) | [ledger](#) | [proof](#) | [revision](#) | [session](#) | [strand](#) | [table](#) | [table view](#) | [view](#)

block

An object that is committed to the journal in a transaction. A single transaction writes one block in the journal, so a block can only be associated with one transaction. A block contains entries that represent the document revisions that were committed in the transaction, along with the [PartiQL](#) statements that committed them.

Each block also has a hash value for verification. A block hash is calculated from the entry hashes within that block combined with the hash of the previous chained block.

digest

A 256-bit hash value that uniquely represents your ledger's entire history of document revisions as of a point in time. A digest hash is calculated from your journal's full hash chain as of the latest committed block in the journal at that time.

QLDB lets you generate a digest as a secure output file. Then, you can use that output file to verify the integrity of your document revisions relative to that hash.

document

A set of data in [Amazon Ion](#) struct format that can be inserted, updated, and deleted in a table. A QLDB document can have structured, semistructured, nested, and schema-less data.

document ID

The universally unique identifier (UUID) that QLDB assigns to each document that you insert into a table. This ID is a 128-bit number that is represented in a Base62-encoded alphanumeric string with a fixed length of 22 characters.

document revision

An Ion structure that represents a single version of a sequence of documents that are identified by a unique document ID. A revision includes both your user data (that is, the data that you wrote in the table) and system-generated metadata. Each revision is associated with a table, and is uniquely identified by a combination of the document ID and a zero-based version number.

entry

An object that is contained in a block. Entries represent document revisions that are inserted, updated, and deleted in a transaction, along with the PartiQL statements that committed them.

Each entry also has a hash value for verification. An entry hash is calculated from the revision hashes or the statement hashes within that entry.

field

A name-value pair that makes up each attribute of a QLDB document. The name is a symbol token, and the value is unrestricted.

index

A data structure that you can create on a table to optimize the performance of data retrieval operations. For information about indexes in QLDB, see [CREATE INDEX](#) in the *Amazon QLDB PartiQL reference*.

indexed storage

The disk space that is used by a ledger's tables, indexes, and indexed history. Indexed storage consists of ledger data that is optimized for high-performance queries.

journal

The hash-chained set of all blocks that are committed in your ledger. The journal is append-only and represents a complete and immutable history of all the changes to your ledger data.

journal block

See [block](#).

journal storage

The disk space that is used by a ledger's journal.

journal strand

See [strand](#).

journal tip

The latest committed block in a journal at a point in time.

ledger

An instance of an Amazon QLDB ledger database resource. This is the primary AWS resource type in QLDB. A ledger consists of both *journal storage* and *indexed storage*. After ledger data is committed to the journal, it's available to query in tables of Amazon Ion document revisions.

proof

The ordered list of 256-bit hash values that QLDB returns for a given digest and document revision. It consists of the hashes that are required by a Merkle tree model to chain the given revision hash to the digest hash. You use a proof to verify the integrity of your revisions relative to the digest. For more information, see [Data verification in Amazon QLDB](#).

revision

See [document revision](#).

session

An object that manages information about your data transaction requests and responses to and from a ledger. An *active session* (one that is actively running a transaction) represents a single connection to a ledger. QLDB supports one actively running transaction per session.

strand

A partition of a journal. QLDB currently supports journals with a single strand only.

table

A materialized view of an unordered collection of document revisions that are committed in the ledger's journal.

table view

A queryable subset of the data in a table, based on transactions committed to the journal. In a PartiQL statement, a view is denoted by a prefix qualifier (starting with `_ql_`) for a table name.

You can query the following system-defined views using SELECT statements:

- *User* – The latest active revision of only the data that you wrote in the table (that is, the current state of your user data). This is the default view in QLDB.
- *Committed* – The latest active revision of both your user data and the system-generated metadata. This is the full system-defined table that corresponds directly to your user table. For example: `_ql_committed_TableName`.

view

See [table view](#).

Accessing Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can access Amazon QLDB using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the QLDB API. The following sections describe how to use these options and the prerequisites for using them.

Prerequisites

Before you can access QLDB, you must set up an AWS account if you haven't already done so.

Topics

- [Sign up for an AWS account](#)
- [Create a user with administrative access](#)
- [Manage QLDB permissions in IAM](#)
- [Grant programmatic access \(optional\)](#)

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign

administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Assign access to additional users

- In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.
- Assign users to a group, and then assign single sign-on access to the group.

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Manage QLDB permissions in IAM

For information about using AWS Identity and Access Management (IAM) to manage QLDB permissions for users, see [How Amazon QLDB works with IAM](#).

Grant programmatic access (optional)

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use.

Which user needs programmatic access?	To	By
		- For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i>. - For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i> .

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. - For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS Command Line Interface User Guide</i>. - For AWS SDKs and tools, see Authenticate using long-term credentials in the <i>AWS SDKs and Tools Reference Guide</i>. - For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

How to access Amazon QLDB

After completing the prerequisites to set up an AWS account, see the following topics to learn more about how to access QLDB:

- [Using the console](#)
- [Using the AWS CLI \(management API only\)](#)
- [Using the Amazon QLDB shell \(data API only\)](#)
- [Using the API](#)

Accessing Amazon QLDB using the console

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can access the AWS Management Console for Amazon QLDB at <https://console.aws.amazon.com/qldb>.

You can use the console to do the following in QLDB:

- Create, delete, describe, and list ledgers.
- Run [PartiQL](#) statements by using the *PartiQL editor*.
- Manage tags for QLDB resources.
- Verify journal data cryptographically.
- Export or stream journal blocks.

To learn how to create an Amazon QLDB ledger and set it up with sample application data, see [Getting started with the Amazon QLDB console](#).

PartiQL editor quick reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB supports a subset of [PartiQL](#) as its query language and [Amazon Ion](#) as its document-oriented data format. For a complete guide and more detailed information about the QLDB implementation of PartiQL, see the [Amazon QLDB PartiQL reference](#).

The following topics provide a quick reference overview of how to use PartiQL in QLDB.

Topics

- [PartiQL quick tips in QLDB](#)
- [Commands](#)
- [System-defined views](#)
- [Basic syntax rules](#)
- [PartiQL editor keyboard shortcuts](#)

PartiQL quick tips in QLDB

The following is a short summary of tips and best practices for working with PartiQL in QLDB:

- **Understand concurrency and transaction limits** – All statements, including SELECT queries, are subject to [optimistic concurrency control \(OCC\)](#) conflicts and [transaction limits](#), including a 30-second transaction timeout.
- **Use indexes** – Use high-cardinality indexes and run targeted queries to optimize your statements and avoid full table scans. To learn more, see [Optimizing query performance](#).
- **Use equality predicates** – Indexed lookups require an *equality* operator (= or IN). Inequality operators (<, >, LIKE, BETWEEN) don't qualify for indexed lookups and result in full table scans.
- **Use inner joins only** – QLDB supports inner joins only. As a best practice, join on fields that are indexed for each table that you're joining. Choose high-cardinality indexes for both the join criteria and the equality predicates.

Commands

QLDB supports the following PartiQL commands.

Data definition language (DDL)

Command	Description
CREATE INDEX	Creates an index for a top-level document field on a table.
CREATE TABLE	Creates a table.
DROP INDEX	Deletes an index from a table.

Command	Description
DROP TABLE	Deactivates an existing table.
UNDROP TABLE	Reactivates an inactive table.

Data manipulation language (DML)

Command	Description
DELETE	Marks an active document as deleted by creating a new, final revision of the document.
FROM (INSERT, REMOVE, or SET)	Semantically the same as UPDATE.
INSERT	Adds one or more documents to a table.
SELECT	Retrieves data from one or more tables.
UPDATE	Updates, inserts, or removes specific elements within a document.

DML statement examples

INSERT

```
INSERT INTO VehicleRegistration VALUE
{
  'VIN' : 'KM8SRDHF6EU074761', --string
  'RegNum' : 1722, --integer
  'PendingPenaltyTicketAmount' : 130.75, --decimal
  'Owners' : { --nested struct
    'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSab0s' },
    'SecondaryOwners' : [ --list of structs
      { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
      { 'PersonId': 'IN7MvYtUjKp1GMZu0F6CG9' }
    ]
  },
  'ValidToDate' : `2020-06-25T` --Ion timestamp literal with day precision
}
```


UPDATE-INSERT

```
UPDATE Vehicle AS v
INSERT INTO v VALUE 26500 AT 'Mileage'
WHERE v.VIN = '1N4AL11D75C109151'
```

UPDATE-REMOVE

```
UPDATE Person AS p
REMOVE p.Address
WHERE p.GovId = '111-22-3333'
```

SELECT – Correlated subquery

```
SELECT r.VIN, o.SecondaryOwners
FROM VehicleRegistration AS r, @r.Owners AS o
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

SELECT – Inner join

```
SELECT v.Make, v.Model, r.Owners
FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

SELECT – Get document ID using BY clause

```
SELECT r_id FROM VehicleRegistration AS r BY r_id
WHERE r.VIN = '1HVBBAANXWH544237'
```

System-defined views

QLDB supports the following system-defined views of a table.

View	Description
<i>table_name</i>	The default user view of a table that includes the current state of your user data only.

View	Description
<code>_ql_committed_<i>table_name</i></code>	The full system-defined committed view of a table that includes the current state of both your user data and system-generated metadata, such as a document ID.
<code>history(<i>table_name</i>)</code>	The built-in history function that returns the complete revision history of a table.

Basic syntax rules

QLDB supports the following basic syntax rules for PartiQL.

Character	Description
'	Single quotes denote string values, or field names in Amazon Ion structures.
"	Double quotes denote quoted identifiers, such as a reserved word that is used as a table name.
`	Backticks denote Ion literal values.
.	Dot notation accesses field names of a parent structure.
[]	Square brackets define an Ion list, or denote a zero-based ordinal number for an existing list.
{ }	Curly braces define an Ion struct.
<< >>	Double angle brackets define a PartiQL bag, which is an unordered collection. You use a bag to insert multiple documents into a table.
Case sensitivity	All QLDB system object names—including field names and table names—are case sensitive.

PartiQL editor keyboard shortcuts

The *PartiQL editor* on the QLDB console supports the following keyboard shortcuts.

Action	macOS	Windows
Run	Cmd+Return	Ctrl+Enter
Comment	Cmd+/ 	Ctrl+/
Clear	Cmd+Shift+Delete	Ctrl+Shift+Delete

Accessing Amazon QLDB using the AWS CLI (management API only)

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can use the AWS Command Line Interface (AWS CLI) to control multiple AWS services from the command line and automate them through scripts. You can use the AWS CLI for one-time operations as needed. You can also use it to embed Amazon QLDB operations within utility scripts.

For CLI access, you need an access key ID and a secret access key. Use temporary credentials instead of long-term access keys when possible. Temporary credentials include an access key ID, a secret access key, and a security token that indicates when the credentials expire. For more information, see [Using temporary credentials with AWS resources](#) in the *IAM User Guide*.

For a complete list and usage examples of all the commands available for QLDB in the AWS CLI, see the [AWS CLI Command Reference](#).

Note

The AWS CLI only supports the `qldb` management API operations that are listed in the [Amazon QLDB API reference](#). This API is used only for managing ledger resources and for non-transactional data operations.

To run data transactions with the `qldb-session` API using a command line interface, see [Accessing Amazon QLDB using the QLDB shell \(data API only\)](#).

Topics

- [Installing and configuring the AWS CLI](#)
- [Using the AWS CLI with QLDB](#)

Installing and configuring the AWS CLI

The AWS CLI runs on Linux, macOS, or Windows. To install and configure it, see the following instructions in the *AWS Command Line Interface User Guide*:

1. [Installing or updating the latest version of the AWS CLI](#)
2. [Quick setup](#)

Using the AWS CLI with QLDB

The command line format consists of an Amazon QLDB operation name, followed by the parameters for that operation. The AWS CLI supports a shorthand syntax for the parameter values, in addition to JSON.

Use `help` to list all available commands in QLDB:

```
aws qldb help
```

You can also use `help` to describe a specific command and learn more about its usage:

```
aws qldb create-ledger help
```

For example, to create a ledger:

```
aws qldb create-ledger --name my-example-ledger --permissions-mode STANDARD
```

Accessing Amazon QLDB using the QLDB shell (data API only)

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB provides a command line shell for interaction with the transactional data API. With the QLDB shell, you can run [PartiQL](#) statements on ledger data.

The latest version of this shell is written in Rust and is open source in the GitHub repository [awslabs/amazon-qldb-shell](#) on the default main branch. The Python version (v1) is also still available for use in the same repository on the master branch.

Note

The Amazon QLDB shell only supports the `qldb-session` transactional data API. This API is used only for running PartiQL statements on a QLDB ledger. To interact with the `qldb` management API operations using a command line interface, see [Accessing Amazon QLDB using the AWS CLI \(management API only\)](#).

This tool isn't intended to be incorporated into an application or adopted for production purposes. The objective of this tool is to let you rapidly experiment with QLDB and PartiQL.

The following sections describe how to get started with the QLDB shell.

Topics

- [Prerequisites](#)
- [Installing the shell](#)
- [Invoking the shell](#)
- [Shell parameters](#)
- [Command reference](#)
- [Running individual statements](#)

- [Managing transactions](#)
- [Exiting the shell](#)
- [Example](#)

Prerequisites

Before you get started with the QLDB shell, you must do the following:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#). This includes the following:
 1. Sign up for AWS.
 2. Create a user with the appropriate QLDB permissions.
 3. Grant programmatic access for development.
2. Set up your AWS credentials and your default AWS Region. For instructions, see [Configuration basics](#) in the *AWS Command Line Interface User Guide*.

For a complete list of available Regions, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

3. For any ledgers in the STANDARD permissions mode, create IAM policies that grant you permissions to run PartiQL statements on the appropriate tables. To learn how to create these policies, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Installing the shell

To install the latest version of the QLDB shell, see the [README.md](#) file on GitHub. QLDB provides pre-built binary files for Linux, macOS, and Windows in the [Releases](#) section of the GitHub repository.

For macOS, the shell integrates with the aws/tap [Homebrew](#) tap. To install the shell on macOS using Homebrew, run the following commands.

```
$ xcode-select --install # Required to use Homebrew
$ brew tap aws/tap # Add AWS as a Homebrew tap
$ brew install qlldbshell
```

Configuration

After installation, the shell loads the default configuration file that is located at `$XDG_CONFIG_HOME/qldbshell/config.ion` during initialization. On Linux and macOS, this file is typically at `~/.config/qldbshell/config.ion`. If such a file doesn't exist, the shell runs with default settings.

You can create a `config.ion` file manually after installation. This configuration file uses the [Amazon Ion](#) data format. The following is an example of a minimal `config.ion` file.

```
{
  default_ledger: "my-example-ledger"
}
```

If `default_ledger` isn't set in your configuration file, the `--ledger` parameter is required when you invoke the shell. For a full list of configuration options, see the [README.md](#) file on GitHub.

Invoking the shell

To invoke the QLDB shell on your command line terminal for a specific ledger, run the following command. Replace *my-example-ledger* with your ledger name.

```
$ qldb --ledger my-example-ledger
```

This command connects to your default AWS Region. To explicitly specify the Region, you can run the command with the `--region` or `--qldb-session-endpoint` parameter, as described in the following section.

After invoking a `qldb` shell session, you can enter the following types of input:

- [Shell commands](#)
- [Single PartiQL statements in separate transactions](#)
- [Multiple PartiQL statements within a transaction](#)

Shell parameters

For a full list of available flags and options for invoking a shell, run the `qldb` command with the `--help` flag, as follows.

```
$ qlldb --help
```

The following are some key flags and options for the `qlldb` command. You can add these optional parameters to override the AWS Region, credentials profile, endpoint, results format, and other configuration options.

Usage

```
$ qlldb [FLAGS] [OPTIONS]
```

FLAGS

-h, --help

Prints help information.

-v, --verbose

Configures the logging verbosity. By default, the shell logs errors only. To increase the verbosity level, repeat this argument (for example, `-vv`). The highest level is `-vvv` which corresponds to trace verbosity.

-V, --version

Prints version information.

OPTIONS

-l, --ledger *LEDGER_NAME*

The name of the ledger to connect to. This is a required shell parameter if `default_ledger` isn't set in your `config.ion` file. In this file, you can set additional options, such as the Region.

-c, --config *CONFIG_FILE*

The file where you can define any shell configuration options. For formatting details and a full list of configuration options, see the [README.md](#) file on GitHub.

-f, --format *ion|table*

The output format of your query results. The default is `ion`.

-p, --profile *PROFILE*

The location of your AWS credentials profile to use for authentication.

If not provided, the shell uses your default AWS profile, which is located at `~/.aws/credentials`.

-r, --region *REGION_CODE*

The AWS Region code of the QLDB ledger to connect to. For example: `us-east-1`.

If not provided, the shell connects to your default AWS Region as specified in your AWS profile.

-s, --qldb-session-endpoint *QLDB_SESSION_ENDPOINT*

The `qldb-session` API endpoint to connect to.

For a complete list of available QLDB Regions and endpoints, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

Command reference

After you invoke a `qldb` session, the shell supports the following keys and database commands:

Shell keys

Key	Function description
Enter	Runs the statement.
Escape+Enter (macOS, Linux) Shift+Enter (Windows)	Starts a new line to enter a statement that spans multiple lines. You can also copy input text with multiple lines and paste it into the shell. For instructions on setting up Option instead of Escape as a Meta key in macOS, see the OS X Daily site.
Ctrl+C	Cancels the current command.
Ctrl+D	Signals end of file (EOF) and exits the current level of the shell. If not in an active transacti

Key	Function description
	on, exits the shell. In an active transaction, aborts the transaction.

Shell database commands

Command	Function description
help	Displays the help information.
begin	Begins a transaction.
start transaction	
commit	Commits your transaction to the ledger's journal.
abort	Stops your transaction and rejects any changes that you made.
exit	Exits the shell.
quit	

Note

All QLDB shell commands are case insensitive.

Running individual statements

Except for the database commands and shell meta commands listed in [README.md](#), the shell interprets each command that you enter as a separate PartiQL statement. By default, the shell enables auto-commit mode. This mode is configurable.

In the auto-commit mode, the shell implicitly runs each statement in its own transaction and automatically commits the transaction if no errors are found. This means that you don't have to run `start transaction` (or `begin`) and `commit` manually each time you run a statement.

Managing transactions

Alternatively, the QLDB shell lets you manually control transactions. You can run multiple statements within a transaction interactively, or non-interactively by batching commands and statements sequentially.

Interactive transactions

To run an interactive transaction, do the following steps.

1. To begin a transaction, enter the `begin` command.

```
qldb> begin
```

After you begin a transaction, the shell displays the following command prompt.

```
qldb *>
```

2. Then, each statement that you enter runs in the same transaction.

- For example, you can run a single statement as follows.

```
qldb *> SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'
```

After you press **Enter**, the shell displays the results of the statement.

- You can also enter multiple statements or commands separated by a semicolon (;) delimiter as follows.

```
qldb *> SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'; commit
```

3. To end the transaction, enter one of the following commands.

- Enter the `commit` command to commit your transaction to the ledger's journal.

```
qldb *> commit
```

- Enter the `abort` command to stop your transaction and reject any changes that you made.

```
qldb *> abort
```

```
transaction was aborted
```

Transaction timeout limit

An interactive transaction adheres to QLDB's [transaction timeout limit](#). If you don't commit a transaction within **30 seconds** of starting it, QLDB automatically expires the transaction and rejects any changes made during the transaction.

Then, instead of displaying the statement results, the shell displays an expiration error message and returns to the normal command prompt. To retry, you must enter the begin command again to begin a new transaction.

```
transaction failed after 1 attempts, last error: communication failure:  
Transaction 2UMpiJ5hh7WLjVgEiML0o0 has expired
```

Non-interactive transactions

You can run a complete transaction with multiple statements by batching commands and statements sequentially as follows.

```
qldb> begin; SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'; SELECT * FROM  
Person p, DriversLicense l WHERE p.GovId = l.LicenseNumber; commit
```

You must separate each command and statement with a semicolon (;) delimiter. If any statement in the transaction isn't valid, the shell automatically rejects the transaction. The shell doesn't proceed with any subsequent statements that you entered.

You can also set up multiple transactions.

```
qldb> begin; statement1; commit; begin; statement2; statement3; commit
```

Similar to the previous example, if a transaction fails, the shell doesn't proceed with any subsequent transactions or statements that you entered.

If you don't end a transaction, the shell switches to interactive mode and prompts you for the next command or statement.

```
qldb> begin; statement1; commit; begin
```

```
qldb *>
```

Exiting the shell

To exit the current qldb shell session, enter the `exit` or `quit` command, or use the keyboard shortcut **Ctrl+D** when the shell isn't in a transaction.

```
qldb> exit
$
```

```
qldb> quit
$
```

Example

For information about writing PartiQL statements in QLDB, see the [Amazon QLDB PartiQL reference](#).

Example

The following example shows a common sequence of basic commands.

Note

The QLDB shell runs each PartiQL statement in this example in its own transaction. This example assumes that the ledger `test-ledger` already exists and is active.

```
$ qldb --ledger test-ledger --region us-east-1

qldb> CREATE TABLE TestTable
qldb> INSERT INTO TestTable `{"Name": "John Doe"}`
qldb> SELECT * FROM TestTable
qldb> DROP TABLE TestTable
qldb> exit
```

Accessing Amazon QLDB using the API

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can use the AWS Management Console and the AWS Command Line Interface (AWS CLI) to work interactively with Amazon QLDB. However, to get the most out of QLDB, you can write application code with a QLDB driver or an AWS SDK to interact with your ledger using the APIs.

The driver lets your application interact with QLDB using the *transactional data* API. The AWS SDK supports interaction with the QLDB *resource management* API. For more information about these APIs, see the [Amazon QLDB API reference](#).

The driver provides support for QLDB in [Java](#), [.NET](#), [Go](#), [Node.js](#), and [Python](#). To get started quickly with these languages, see [Getting started with the Amazon QLDB driver](#).

Before you can use a QLDB driver or an AWS SDK in your application, you must grant programmatic access. For more information, see [Grant programmatic access](#).

Getting started with the Amazon QLDB console

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This tutorial guides you through steps to create your first Amazon QLDB ledger and populate it with tables and sample data. The sample ledger you create in this scenario is a database for a department of motor vehicles (DMV) application that tracks the complete historical information about vehicle registrations.

The history of an asset is a common use case for QLDB because it involves a variety of scenarios and operations that highlight the usefulness of a ledger database. With QLDB you can directly access, query, and verify the full history of changes to your data in a document-oriented database that supports SQL-like query capabilities.

As you work through this tutorial, the following topics explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to verify a registration document cryptographically, and concludes by cleaning up resources and deleting the sample ledger.

Each step in the tutorial has instructions for using the AWS Management Console.

Topics

- [Tutorial prerequisites and considerations](#)
- [Step 1: Create a new ledger](#)
- [Step 2: Create tables, indexes, and sample data in a ledger](#)
- [Step 3: Query the tables in a ledger](#)
- [Step 4: Modify documents in a ledger](#)
- [Step 5: View the revision history for a document](#)
- [Step 6: Verify a document in a ledger](#)
- [Step 7 \(optional\): Clean up resources](#)
- [Getting started with Amazon QLDB: Next steps](#)

Tutorial prerequisites and considerations

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Before you start this Amazon QLDB tutorial, make sure that you complete the following prerequisites:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#), if you haven't already done so. These steps include signing up for AWS and creating an administrative user.
2. Follow the instructions in [Setting up permissions](#) to set up IAM permissions for your QLDB resources. To complete all of the steps in this tutorial, you need full administrative access to your ledger resources through the AWS Management Console.

Note

If you're already signed in as a user with full AWS administrative permissions, you can skip this step.

3. (Optional) QLDB encrypts data at rest using a key in AWS Key Management Service (AWS KMS). You can choose one of the following types of AWS KMS keys:
 - **AWS owned KMS key** – Use a KMS key that is owned and managed by AWS on your behalf. This is the default option and requires no additional setup.
 - **Customer managed KMS key** – Use a symmetric encryption KMS key in your account that you create, own, and manage. QLDB doesn't support [asymmetric keys](#).

This option requires you to create a KMS key or use an existing key in your account. For instructions on creating a customer managed key, see [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*.

You can specify a customer managed KMS key by using an ID, alias, or Amazon Resource Name (ARN). To learn more, see [Key identifiers \(KeyId\)](#) in the *AWS Key Management Service Developer Guide*.

Note

Cross-Region keys are not supported. The specified KMS key must be in the same AWS Region as your ledger.

Setting up permissions

In this step, you set up full access permissions through the console for all QLDB resources in your AWS account. To grant these permissions quickly, use the AWS managed policy [AmazonQLDBConsoleFullAccess](#).

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:
 - Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
 - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

Important

For the purposes of this tutorial, you grant yourself full administrative access to all QLDB resources. For production use cases, however, follow the security best practice of [granting least privilege](#), or granting only the permissions required to perform a task. For examples, see [Identity-based policy examples for Amazon QLDB](#).

To create a ledger named `vehicle-registration`, proceed to [Step 1: Create a new ledger](#).

Step 1: Create a new ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`. Then, you confirm that the status of the ledger is **Active**. You can also verify any tags that you added to the ledger.

When you create a ledger, *deletion protection* is enabled by default. Deletion protection is a feature in QLDB that prevents ledgers from being deleted by any user. You can disable deletion protection when you create a ledger using the QLDB API or the AWS Command Line Interface (AWS CLI).

To create a new ledger

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Getting started**.
3. On the **Create your first ledger** card, choose **Create Ledger**.
4. On the **Create Ledger** page, do the following:
 - **Ledger information** – The **Ledger name** should be pre-populated with **vehicle-registration**.
 - **Permissions mode** – The permissions mode to assign to the ledger. Choose one of the following options:
 - **Allow all** – A legacy permissions mode that enables access control with API-level granularity for ledgers.

This mode allows users who have the `SendCommand` API permission for this ledger to run all PartiQL commands (hence, `ALLOW_ALL`) on any tables in the specified ledger. This

mode disregards any table-level or command-level IAM permissions policies that you create for the ledger.

- **Standard** – (*Recommended*) A permissions mode that enables access control with finer granularity for ledgers, tables, and PartiQL commands. We strongly recommend using this permissions mode to maximize the security of your ledger data.

By default, this mode denies all requests to run any PartiQL commands on any tables in this ledger. To allow PartiQL commands, you must create IAM permissions policies for specific table resources and PartiQL actions, in addition to the `SendCommand` API permission for the ledger. For information, see [Getting started with the standard permissions mode in Amazon QLDB](#).

- **Encrypt data at rest** – The key in AWS Key Management Service (AWS KMS) to use for encryption of data at rest. Choose one of the following options:
 - **Use AWS owned KMS key** – Use a KMS key that is owned and managed by AWS on your behalf. This is the default option and requires no additional setup.
 - **Choose a different AWS KMS key** – Use a symmetric encryption KMS key in your account that you create, own, and manage.

To create a new key by using the AWS KMS console, choose **Create an AWS KMS key**. For more information, see [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*.

To use an existing KMS key, choose one from the dropdown list or specify a KMS key ARN.

- **Tags** – (Optional) Add metadata to the ledger by attaching tags as key-value pairs. You can add tags to your ledger to help organize and identify them. For more information, see [Tagging Amazon QLDB resources](#).

Choose **Add tag**, and then enter any key-value pairs as appropriate.

5. When the settings are as you want them, choose **Create ledger**.

 Note

You can access your QLDB ledger when its status becomes **Active**. This can take several minutes.

6. In the list of **Ledgers**, locate `vehicle-registration` and confirm that the ledger's status is **Active**.

7. (Optional) Choose the `vehicle-registration` ledger name. On the **vehicle-registration** ledger details page, confirm that any tags that you added to the ledger appear on the **Tags** card. You can also edit your ledger tags using this console page.

To create tables in the `vehicle-registration` ledger, proceed to [Step 2: Create tables, indexes, and sample data in a ledger](#).

Step 2: Create tables, indexes, and sample data in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

When your Amazon QLDB ledger is active, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the `vehicle-registration` ledger:

- `VehicleRegistration`
- `Vehicle`
- `Person`
- `DriversLicense`

You also create the following indexes.

Table name	Field
<code>VehicleRegistration</code>	<code>VIN</code>
<code>VehicleRegistration</code>	<code>LicensePlateNumber</code>
<code>Vehicle</code>	<code>VIN</code>

Table name	Field
Person	GovId
DriversLicense	LicensePlateNumber
DriversLicense	PersonId

You can use the QLDB console to automatically create these tables with indexes and load them with sample data. Or, you can use the **PartiQL editor** on the console to manually run each [PartiQL](#) statement step-by-step.

Automatic option

To create tables, indexes, and sample data

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Getting started**.
3. Under **Automatic option** on the **Sample application data** card, choose vehicle-registration in the list of ledgers.
4. Choose **Load sample data**.

If the operation finishes successfully, the console displays the message **Sample data loaded**.

This script runs all statements in a single transaction. If any part of the transaction fails, every statement is rolled back, and an appropriate error message is displayed. You can retry the operation after addressing any issues.

Note

- One possible cause for a transaction failure is attempting to create duplicate tables. Your request to load sample data will fail if any of the following table names already exist in your ledger: `VehicleRegistration`, `Vehicle`, `Person`, and `DriversLicense`.

Instead, try loading this sample data in an empty ledger.

- This script runs parameterized INSERT statements. So, these PartiQL statements are recorded in your journal blocks with bind parameters instead of the literal data. For example, you might see the following statement in a journal block, where the question mark (?) is a variable placeholder for the document contents.

```
INSERT INTO Vehicle ?
```

Manual option

You insert documents into `VehicleRegistration` with an empty `PrimaryOwner` field, and into `DriversLicense` with an empty `PersonId` field. Later, you populate these fields with the system-assigned document id from the `Person` table.

Tip

As a best practice, use this document id metadata field as a foreign key. For more information, see [Querying document metadata](#).

To create tables, indexes, and sample data

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **PartiQL editor**.
3. Choose the `vehicle-registration` ledger.
4. Start by creating four tables. QLDB supports open content and doesn't enforce schema, so you don't specify attributes or data types.

In the query editor window, enter the following statement, and then choose **Run**. To run the statement, you can also use the keyboard shortcut **Ctrl+Enter** for Windows, or **Cmd+Return** for macOS. For more keyboard shortcuts, see [PartiQL editor keyboard shortcuts](#).

```
CREATE TABLE VehicleRegistration
```

Repeat this step for each of the following.

```
CREATE TABLE Vehicle
```

```
CREATE TABLE Person
```

```
CREATE TABLE DriversLicense
```

- Next, create indexes that optimize query performance for each table.

 Important

QLDB requires an index to efficiently look up a document. Without an index, QLDB needs to do a full table scan when reading documents. This can cause performance problems on large tables, including concurrency conflicts and transaction timeouts. To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator (= or IN) on an indexed field or a document ID. For more information, see [Optimizing query performance](#).

In the query editor window, enter the following statement, and then choose **Run**.

```
CREATE INDEX ON VehicleRegistration (VIN)
```

Repeat this step for the following.

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

```
CREATE INDEX ON Person (GovId)
```

```
CREATE INDEX ON DriversLicense (LicensePlateNumber)
```

```
CREATE INDEX ON DriversLicense (PersonId)
```

- After creating your indexes, you can start loading data into your tables. In this step, insert documents into the Person table with personal information about owners of the vehicles that the ledger is tracking.

In the query editor window, enter the following statement, and then choose **Run**.

```
INSERT INTO Person
<< {
  'FirstName' : 'Raul',
  'LastName' : 'Lewis',
  'DOB' : `1963-08-19T`,
  'GovId' : 'LEWISR261LL',
  'GovIdType' : 'Driver License',
  'Address' : '1719 University Street, Seattle, WA, 98109'
},
{
  'FirstName' : 'Brent',
  'LastName' : 'Logan',
  'DOB' : `1967-07-03T`,
  'GovId' : 'LOGANB486CG',
  'GovIdType' : 'Driver License',
  'Address' : '43 Stockert Hollow Road, Everett, WA, 98203'
},
{
  'FirstName' : 'Alexis',
  'LastName' : 'Pena',
  'DOB' : `1974-02-10T`,
  'GovId' : '744 849 301',
  'GovIdType' : 'SSN',
  'Address' : '4058 Melrose Street, Spokane Valley, WA, 99206'
},
{
  'FirstName' : 'Melvin',
  'LastName' : 'Parker',
  'DOB' : `1976-05-22T`,
  'GovId' : 'P626-168-229-765',
  'GovIdType' : 'Passport',
  'Address' : '4362 Ryder Avenue, Seattle, WA, 98101'
},
{
  'FirstName' : 'Salvatore',
  'LastName' : 'Spencer',
  'DOB' : `1997-11-15T`,
  'GovId' : 'S152-780-97-415-0',
  'GovIdType' : 'Passport',
  'Address' : '4450 Honeysuckle Lane, Seattle, WA, 98101'
```



```
} >>
```

7. Then, populate the `DriversLicense` table with documents that include driver's license information for each vehicle owner.

In the query editor window, enter the following statement, and then choose **Run**.

```
INSERT INTO DriversLicense
<< {
  'LicensePlateNumber' : 'LEWISR261LL',
  'LicenseType' : 'Learner',
  'ValidFromDate' : `2016-12-20T`,
  'ValidToDate' : `2020-11-15T`,
  'PersonId' : ''
},
{
  'LicensePlateNumber' : 'LOGANB486CG',
  'LicenseType' : 'Probationary',
  'ValidFromDate' : `2016-04-06T`,
  'ValidToDate' : `2020-11-15T`,
  'PersonId' : ''
},
{
  'LicensePlateNumber' : '744 849 301',
  'LicenseType' : 'Full',
  'ValidFromDate' : `2017-12-06T`,
  'ValidToDate' : `2022-10-15T`,
  'PersonId' : ''
},
{
  'LicensePlateNumber' : 'P626-168-229-765',
  'LicenseType' : 'Learner',
  'ValidFromDate' : `2017-08-16T`,
  'ValidToDate' : `2021-11-15T`,
  'PersonId' : ''
},
{
  'LicensePlateNumber' : 'S152-780-97-415-0',
  'LicenseType' : 'Probationary',
  'ValidFromDate' : `2015-08-15T`,
  'ValidToDate' : `2021-08-21T`,
  'PersonId' : ''
} >>
```

- Now, populate the `VehicleRegistration` table with vehicle registration documents. These documents include a nested `Owners` structure that stores the primary and secondary owners.

In the query editor window, enter the following statement, and then choose **Run**.

```
INSERT INTO VehicleRegistration
<< {
  'VIN' : '1N4AL11D75C109151',
  'LicensePlateNumber' : 'LEWISR261LL',
  'State' : 'WA',
  'City' : 'Seattle',
  'PendingPenaltyTicketAmount' : 90.25,
  'ValidFromDate' : `2017-08-21T`,
  'ValidToDate' : `2020-05-11T`,
  'Owners' : {
    'PrimaryOwner' : { 'PersonId': '' },
    'SecondaryOwners' : []
  }
},
{
  'VIN' : 'KM8SRDHF6EU074761',
  'LicensePlateNumber' : 'CA762X',
  'State' : 'WA',
  'City' : 'Kent',
  'PendingPenaltyTicketAmount' : 130.75,
  'ValidFromDate' : `2017-09-14T`,
  'ValidToDate' : `2020-06-25T`,
  'Owners' : {
    'PrimaryOwner' : { 'PersonId': '' },
    'SecondaryOwners' : []
  }
},
{
  'VIN' : '3HGGK5G53FM761765',
  'LicensePlateNumber' : 'CD820Z',
  'State' : 'WA',
  'City' : 'Everett',
  'PendingPenaltyTicketAmount' : 442.30,
  'ValidFromDate' : `2011-03-17T`,
  'ValidToDate' : `2021-03-24T`,
  'Owners' : {
    'PrimaryOwner' : { 'PersonId': '' },
    'SecondaryOwners' : []
  }
}
```

```

    }
  },
  {
    'VIN' : '1HVBBAANXWH544237',
    'LicensePlateNumber' : 'LS477D',
    'State' : 'WA',
    'City' : 'Tacoma',
    'PendingPenaltyTicketAmount' : 42.20,
    'ValidFromDate' : `2011-10-26T`,
    'ValidToDate' : `2023-09-25T`,
    'Owners' : {
      'PrimaryOwner' : { 'PersonId': '' },
      'SecondaryOwners' : []
    }
  },
  {
    'VIN' : '1C4RJFAG0FC625797',
    'LicensePlateNumber' : 'TH393F',
    'State' : 'WA',
    'City' : 'Olympia',
    'PendingPenaltyTicketAmount' : 30.45,
    'ValidFromDate' : `2013-09-02T`,
    'ValidToDate' : `2024-03-19T`,
    'Owners' : {
      'PrimaryOwner' : { 'PersonId': '' },
      'SecondaryOwners' : []
    }
  }
} >>

```

9. Lastly, populate the `Vehicle` table with documents describing the vehicles that are registered in your ledger.

In the query editor window, enter the following statement, and then choose **Run**.

```

INSERT INTO Vehicle
<< {
  'VIN' : '1N4AL11D75C109151',
  'Type' : 'Sedan',
  'Year' : 2011,
  'Make' : 'Audi',
  'Model' : 'A5',
  'Color' : 'Silver'
},

```

```
{
  'VIN' : 'KM8SRDHF6EU074761',
  'Type' : 'Sedan',
  'Year' : 2015,
  'Make' : 'Tesla',
  'Model' : 'Model S',
  'Color' : 'Blue'
},
{
  'VIN' : '3HGGK5G53FM761765',
  'Type' : 'Motorcycle',
  'Year' : 2011,
  'Make' : 'Ducati',
  'Model' : 'Monster 1200',
  'Color' : 'Yellow'
},
{
  'VIN' : '1HVBBAANXWH544237',
  'Type' : 'Semi',
  'Year' : 2009,
  'Make' : 'Ford',
  'Model' : 'F 150',
  'Color' : 'Black'
},
{
  'VIN' : '1C4RJFAG0FC625797',
  'Type' : 'Sedan',
  'Year' : 2019,
  'Make' : 'Mercedes',
  'Model' : 'CLK 350',
  'Color' : 'White'
} >>
```

Next, you can use SELECT statements to read data from the tables in the vehicle-registration ledger. Proceed to [Step 3: Query the tables in a ledger](#).

Step 3: Query the tables in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses PartiQL as its query language and Amazon Ion as its document-oriented data model.

PartiQL is an open-source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open-source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use SELECT statements to read data from the tables in the vehicle-registration ledger.

Warning

When you run a query in QLDB without an indexed lookup, it invokes a full table scan. PartiQL supports such queries because it's SQL compatible. However, *don't* run table scans for production use cases in QLDB. Table scans can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator on an indexed field or a document ID; for example, WHERE indexedField = 123 or WHERE indexedField IN (456, 789). For more information, see [Optimizing query performance](#).

To query the tables

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **PartiQL editor**.

3. Choose the `vehicle-registration` ledger.
4. In the query editor window, enter the following statement to query the `Vehicle` table for a particular vehicle identification number (VIN) that you added to the ledger, and then choose **Run**.

To run the statement, you can also use the keyboard shortcut **Ctrl+Enter** for Windows, or **Cmd+Return** for macOS. For more keyboard shortcuts, see [PartiQL editor keyboard shortcuts](#).

```
SELECT * FROM Vehicle AS v
WHERE v.VIN = '1N4AL11D75C109151'
```

5. You can write inner join queries. This query example joins `Vehicle` with `VehicleRegistration` and returns registration information along with attributes of the registered vehicle for a specified VIN.

Enter the following statement, and then choose **Run**.

```
SELECT v.VIN, r.LicensePlateNumber, r.State, r.City, r.Owners
FROM Vehicle AS v, VehicleRegistration AS r
WHERE v.VIN = '1N4AL11D75C109151'
AND v.VIN = r.VIN
```

You can also join the `Person` and `DriversLicense` tables to see attributes related to the drivers who were added to the ledger.

Repeat this step for the following.

```
SELECT * FROM Person AS p, DriversLicense AS l
WHERE p.GovId = l.LicensePlateNumber
```

To learn about modifying documents in the tables in the `vehicle-registration` ledger, see [Step 4: Modify documents in a ledger](#).

Step 4: Modify documents in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Now that you have data to work with, you can start making changes to documents in the vehicle-registration ledger in Amazon QLDB. For example, consider the Audi A5 with VIN 1N4AL11D75C109151. This car is initially owned by a driver named Raul Lewis in Seattle, WA.

Suppose that Raul sells the car to a resident in Everett, WA named Brent Logan. Then, Brent and Alexis Pena decide to get married. Brent wants to add Alexis as a secondary owner on the registration. In this step, the following data manipulation language (DML) statements demonstrate how to make the appropriate changes in your ledger to reflect these events.

Tip

As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table). For more information about views in QLDB, see [Core concepts](#). To learn more about metadata, see [Querying document metadata](#).

To modify documents

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **PartiQL editor**.
3. Choose the vehicle-registration ledger.

Note

If you set up your ledger using the console's automatic **Load sample data** feature, skip ahead to step 6.

4. If you manually ran INSERT statements to load the sample data, continue with these steps.

To initially register Raul as this vehicle's owner, start by finding his system-assigned document `id` in the `Person` table. This field is included in the document's metadata, which you can query in the system-defined view of the table, called the *committed view*.

In the query editor window, enter the following statement, and then choose **Run**.

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Raul' and p.data.LastName = 'Lewis'
```

The prefix `_ql_committed_` is a reserved prefix signifying that you want to query the committed view of the `Person` table. In this view, your data is nested in the `data` field, and `metadata` is nested in the `metadata` field.

5. Now, use this `id` in an UPDATE statement to modify the appropriate document in the `VehicleRegistration` table. Enter the following statement, and then choose **Run**.

```
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = '294jJ3YUoH1IEEm8GSab0s' --replace with your
id
WHERE r.VIN = '1N4AL11D75C109151'
```

Confirm that you modified the `Owners` field by issuing this statement.

```
SELECT r.Owners FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

6. To transfer the vehicle's ownership to Brent in the city of Everett, first find his `id` from the `Person` table with the following statement.

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Brent' and p.data.LastName = 'Logan'
```


Next, use this id to update the PrimaryOwner and the City in the VehicleRegistration table.

```
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = '7NmE8YLPbXc0IqesJy1rpR', --replace with your
id
    r.City = 'Everett'
WHERE r.VIN = '1N4AL11D75C109151'
```

Confirm that you modified the PrimaryOwner and City fields by issuing this statement.

```
SELECT r.Owners.PrimaryOwner, r.City
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

7. To add Alexis as a secondary owner of the car, find her Person id.

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Alexis' and p.data.LastName = 'Pena'
```

Then, insert this id into the SecondaryOwners list with the following [FROM-INSERT](#) DML statement.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners
    VALUE { 'PersonId' : '5Ufgdlnj06gF5CWc0Iu64s' } --replace with your id
```

Confirm that you modified SecondaryOwners by issuing this statement.

```
SELECT r.Owners.SecondaryOwners FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

To review these changes in the vehicle-registration ledger, see [Step 5: View the revision history for a document](#).

Step 5: View the revision history for a document

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After modifying registration data for the car with VIN 1N4AL11D75C109151, you can query the history of all its registered owners and any other updated fields. You can see all revisions of a document that you inserted, updated, and deleted by querying the built-in [History function](#).

The history function returns revisions from the *committed view* of your table, which includes both your application data and the associated metadata. The metadata shows exactly when each revision was made, in what order, and which transaction committed them.

In this step, you query the revision history of a document in the VehicleRegistration table in the vehicle-registration ledger.

To view the revision history

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **PartiQL editor**.
3. Choose the vehicle-registration ledger.
4. To query the history of a document, start by finding its unique id. In addition to querying the committed view, another way of getting a document id is to use the BY keyword in the table's default user view. To learn more, see [Using the BY clause to query document ID](#).

In the query editor window, enter the following statement, and then choose **Run**.

```
SELECT r_id FROM VehicleRegistration AS r BY r_id
WHERE r.VIN = '1N4AL11D75C109151'
```

5. Next, you can use this id value to query the history function. Enter the following statement, and then choose **Run**. Be sure to replace the id value with your own document ID as appropriate.

```
SELECT h.data.VIN, h.data.City, h.data.Owners
```

```
FROM history(VehicleRegistration) AS h
WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
```

Note

For the purposes of this tutorial, this history query returns all revisions of document ID ADR2LQq48kB9neZDupQrMm. As a best practice, however, qualify a history query with both a document ID and a date range (start time and end time).

In QLDB, every SELECT query is processed in a transaction and is subject to a [transaction timeout limit](#). History queries that include a start time and end time gain the benefit of date range qualification. For more information, see [History function](#).

The history function returns documents in the same schema as the committed view. This example projects your modified vehicle registration data. The output should look similar to the following.

VIN	City	Owners
"1N4AL11D 75C109151"	"Seattle"	{PrimaryOwner:{PersonId:""}, SecondaryOwners:[]}
"1N4AL11D 75C109151"	"Seattle"	{PrimaryOwner:{PersonId:"29 4jJ3YUoH1IEEm8GSab0s"}, SecondaryOwners:[]}
"1N4AL11D 75C109151"	"Everett"	{PrimaryOwner:{PersonId:"7N mE8YLPbXc0IqesJy1rpR"}, SecondaryOwners:[]}
"1N4AL11D 75C109151"	"Everett"	{PrimaryOwner:{PersonId:"7N mE8YLPbXc0IqesJy1rpR"}, SecondaryOwners:[{PersonId: "5Ufgdlnj06gF5CWc0Iu64s"}]}

Note

The history query might not always return document revisions in sequential order.

Review the output and confirm that the changes reflect what you did in [Step 4: Modify documents in a ledger](#).

- Then, you can inspect the document metadata of each revision. Enter the following statement, and then choose **Run**. Again, be sure to replace the `id` value with your own document ID as appropriate.

```
SELECT VALUE h.metadata
FROM history(VehicleRegistration) AS h
WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
```

The output should look similar to the following.

version	id	txTime	txId
0	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T19:20:360d-3Z	"FMoVdWuPxJg3k466Iz4i75"
1	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:40:199d-3Z	"KWByxe842Xw8DNHcvARP0t"
2	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:44:432d-3Z	"EKwD0JRwbHpFvmAyJ2Kdh9"
3	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:49:254d-3Z	"96EiZd7vCmJ6RAv0vTZ4YA"

These metadata fields provide details on when each item was modified, and by which transaction. From this data, you can infer the following:

- The document is uniquely identified by its system-assigned `id`: `ADR2LQq48kB9neZDupQrMm`. This is a universally unique identifier (UUID) that is represented in a Base62-encoded string.
- The `txTime` shows that the initial revision of the document (version 0) was created at `2019-05-23T19:20:360d-3Z`.
- Each subsequent transaction creates a new revision with the same document `id`, an incremented version number, and an updated `txId` and `txTime`.

To verify a document revision cryptographically in the vehicle-registration ledger, proceed to [Step 6: Verify a document in a ledger](#).

Step 6: Verify a document in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. In this example, Alexis and Brent decide to upgrade to a new model by trading in the vehicle with VIN 1N4AL11D75C109151 at a car dealership. The dealership starts the process by verifying the vehicle's ownership with the registration office.

To learn more about how verification and cryptographic hashing work in QLDB, see [Data verification in Amazon QLDB](#).

In this step, you verify a document revision in the vehicle-registration ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To request a digest

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.

2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select `vehicle-registration`.
4. Choose **Get digest**. The **Get digest** dialog box displays the following digest details:
 - **Digest** – The SHA-256 hash value of the digest that you requested.
 - **Digest tip address** – The latest [block](#) location in the journal covered by the digest that you requested. An address has the following two fields:
 - `strandId` – The unique ID of the journal strand that contains the block.
 - `sequenceNo` – The index number that specifies the location of the block within the strand.
 - **Ledger** – The ledger name for which you requested a digest.
 - **Date** – The timestamp when you requested the digest.
5. Review the digest information. Then choose **Save**. You can keep the default file name, or enter a new name.

This step saves a plaintext file with contents in [Amazon Ion](#) format. The file has a file name extension of `.ion.txt` and contains all the digest information that was listed on the preceding dialog box. The following is an example of a digest file's contents. The order of the fields can vary depending on your browser.

```
{
  "digest": "42zaJ0fV8iGutVGNaIuzQWhD5Xb/5B9lScHnvxPXm9E=",
  "digestTipAddress": "{strandId:\"B1FTjlSXze9BIh1K0szcE3\",sequenceNo:73}",
  "ledger": "vehicle-registration",
  "date": "2019-04-17T16:57:26.749Z"
}
```

6. Save this file where you can access it later. In the following steps, you use this file to verify a document revision against.

After you have a ledger digest saved, you can start the process of verifying a document revision against that digest.

Note

In a production use case for verification, you use a digest that was previously saved rather than doing the two tasks consecutively. As a best practice, request and save the digest as soon as a revision that you want to verify later is written to the journal.

To verify a document revision

1. First, query your ledger for the `id` and `blockAddress` of the document revision that you want to verify. These fields are included in the document's metadata, which you can query in the committed view.

The document `id` is a system-assigned unique ID string. The `blockAddress` is an Ion structure that specifies the block location where the revision was committed.

In the navigation pane of the QLDB console, choose **PartiQL editor**.

2. Choose the `vehicle-registration` ledger.
3. In the query editor window, enter the following statement, and then choose **Run**.

```
SELECT r.metadata.id, r.blockAddress
FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN = '1N4AL11D75C109151'
```

4. Copy and save the `id` and `blockAddress` values that your query returns. Be sure to omit the double quotes for the `id` field. In Amazon Ion, string data types are delimited with double quotes.
5. Now that you have a document revision selected, you can start the process of verifying it.

In the navigation pane, choose **Verification**.

6. On the **Verify document** form, under **Specify the document that you want to verify**, enter the following input parameters:
 - **Ledger** – Choose `vehicle-registration`.
 - **Block address** – The `blockAddress` value returned by your query in step 3.
 - **Document ID** – The `id` value returned by your query in step 3.

7. Under **Specify the digest to use for verification**, select the digest that you previously saved by choosing **Choose digest**. If the file is valid, this auto-populates all the digest fields on your console. Or, you can manually copy and paste the following values directly from your digest file:
 - **Digest** – The digest value from your digest file.
 - **Digest tip address** – The `digestTipAddress` value from your digest file.
8. Review your document and digest input parameters, and then choose **Verify**.

The console automates two steps for you:

- a. Request a proof from QLDB for your specified document.
- b. Use the proof returned by QLDB to call a client-side API, which verifies your document revision against the provided digest.

The console displays the results of your request in the **Verification results** card. For more information, see [Verification results](#).

9. To test the verification logic, repeat steps 6–8 under **To verify a document revision**, but change a single character in the **Digest** input string. This should cause your **Verify** request to fail with an appropriate error message.

If you no longer need to use the vehicle-registration ledger, proceed to [Step 7 \(optional\): Clean up resources](#).

Step 7 (optional): Clean up resources

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can continue using the vehicle-registration ledger. However, if you no longer need it, you should delete it.

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API, AWS Command Line Interface (AWS CLI), or QLDB console.

To delete the ledger

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. If deletion protection is enabled for this ledger, you must first turn it off.

In the list of ledgers, select `vehicle-registration`, and then choose **Edit ledger**.

4. On the **Edit ledger** page, turn off **Deletion protection**, and then choose **Confirm changes**.
5. In the list of ledgers, select `vehicle-registration` again, and then choose **Delete**.
6. Confirm this by entering **delete vehicle-registration** in the field provided.

To learn more about working with ledgers in QLDB, see [Getting started with Amazon QLDB: Next steps](#).

Getting started with Amazon QLDB: Next steps

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

For more information about using Amazon QLDB, see the following topics:

- [Working with data and history in Amazon QLDB](#)
- [Getting started with the Amazon QLDB driver](#)
- [Amazon QLDB concurrency model](#)
- [Exporting journal data from Amazon QLDB](#)
- [Streaming journal data from Amazon QLDB](#)
- [Data verification in Amazon QLDB](#)
- [Amazon QLDB PartiQL reference](#)

Getting started with the Amazon QLDB driver

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This chapter contains hands-on tutorials to help you learn about developing with Amazon QLDB by using the QLDB driver. The driver is built on top of the AWS SDK, which supports interaction with the [QLDB API](#).

QLDB session abstraction

The driver provides a high-level abstraction layer above the transactional data API (*QLDB Session*). It streamlines the process of running [PartiQL](#) statements on ledger data by managing [SendCommand](#) API calls. These API calls require several parameters that the driver handles for you, including the management of sessions, transactions, and retry policy in the case of errors. The driver also has performance optimizations and applies best practices for interacting with QLDB.

Note

To interact with the resource management API operations that are listed in the [Amazon QLDB API reference](#), you use the AWS SDK directly instead of the driver. You use the management API only for managing ledger resources and for non-transactional data operations, such as exporting, streaming, and data verification.

Amazon Ion support

In addition, the driver uses [Amazon Ion](#) libraries to provide support for handling Ion data when running transactions. These libraries also take care of calculating the hash of Ion values. QLDB requires these Ion hashes to check the integrity of data transaction requests.

Driver terminology

This tool is called a *driver* because it's comparable to other database drivers that provide developer-friendly interfaces. These drivers similarly encapsulate logic that converts a standard set of commands and functions into specific calls that are required by the service's low-level API.

The driver is open source on GitHub and is available for the following programming languages:

- [Java driver](#)
- [.NET driver](#)
- [Go driver](#)
- [Node.js driver](#)
- [Python driver](#)

For general driver information for all supported programming languages, and additional tutorials, see the following topics:

- [Session management with the driver](#)
- [Driver recommendations](#)
- [Driver retry policy](#)
- [Common errors](#)
- [Sample application tutorial](#)
- [Working with Amazon Ion](#)
- [Getting PartiQL statement statistics](#)

Amazon QLDB driver for Java

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

To work with data in your ledger, you can connect to Amazon QLDB from your Java application by using an AWS provided driver. The following topics describe how to get started with the QLDB driver for Java.

Topics

- [Driver resources](#)
- [Prerequisites](#)
- [Setting your default AWS credentials and Region](#)
- [Installation](#)
- [Amazon QLDB driver for Java – Quick start tutorial](#)
- [Amazon QLDB driver for Java – Cookbook reference](#)

Driver resources

For more information about the functionality supported by the Java driver, see the following resources:

- API reference: [2.x](#), [1.x](#)
- [Driver source code \(GitHub\)](#)
- [Sample application source code \(GitHub\)](#)
- [Ledger loader framework \(GitHub\)](#)
- [Amazon Ion code examples](#)

Prerequisites

Before you get started with the QLDB driver for Java, you must do the following:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#). This includes the following:
 1. Sign up for AWS.
 2. Create a user with the appropriate QLDB permissions.
 3. Grant programmatic access for development.
2. Set up a Java development environment by downloading and installing the following:
 1. Java SE Development Kit 8, such as [Amazon Corretto 8](#).
 2. (Optional) Java integrated development environment (IDE) of your choice, such as [Eclipse](#) or [IntelliJ](#).

3. Configure your development environment for the AWS SDK for Java by [Setting your default AWS credentials and Region](#).

Next, you can download the complete tutorial sample application—or you can install only the driver in a Java project and run short code examples.

- To install the QLDB driver and the AWS SDK for Java in an existing project, proceed to [Installation](#).
- To set up a project and run short code examples that demonstrate basic data transactions on a ledger, see the [Quick start tutorial](#).
- To run more in-depth examples of both data and management API operations in the complete tutorial sample application, see the [Java tutorial](#).

Setting your default AWS credentials and Region

The QLDB driver and the underlying [AWS SDK for Java](#) require that you provide AWS credentials to your application at runtime. The code examples in this guide assume that you're using an AWS credentials file, as described in [Set default credentials and Region](#) in the *AWS SDK for Java 2.x Developer Guide*.

As part of these steps, you should also set your default AWS Region to determine your default QLDB endpoint. The code examples connect to QLDB in your default AWS Region. For a complete list of Regions where QLDB is available, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

The following is an example of an AWS credentials file named `~/.aws/credentials`, where the tilde character (`~`) represents your home directory.

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

Substitute your own AWS credentials values for the values *your_access_key_id* and *your_secret_access_key*.

Installation

QLDB supports the following Java driver versions and their AWS SDK dependencies.

Driver version	AWS SDK	Status	Latest release date
1.x	AWS SDK for Java 1.x	Production release	March 20, 2020
2.x	AWS SDK for Java 2.x	Production release	June 4, 2021

To install the QLDB driver, we recommend using a dependency management system, such as Gradle or Maven. For example, add the following artifact as a dependency in your Java project.

2.x

Gradle

Add this dependency in your `build.gradle` configuration file.

```
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
    '2.3.1'
}
```

Maven

Add this dependency in your `pom.xml` configuration file.

```
<dependencies>
  <dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>2.3.1</version>
  </dependency>
</dependencies>
```

This artifact automatically includes the AWS SDK for Java 2.x core module, [Amazon Ion](#) libraries, and other required dependencies.

1.x

Gradle

Add this dependency in your `build.gradle` configuration file.

```
dependencies {
```

```
compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
'1.1.0'
}
```

Maven

Add this dependency in your `pom.xml` configuration file.

```
<dependencies>
  <dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>1.1.0</version>
  </dependency>
</dependencies>
```

This artifact automatically includes the AWS SDK for Java core module, [Amazon Ion](#) libraries, and other required dependencies.

Important

Amazon Ion namespace – When importing Amazon Ion classes in your application, you must use the package that is under the namespace `com.amazon.ion`. The AWS SDK for Java depends on another Ion package under the namespace `software.amazon.ion`, but this is a legacy package that is not compatible with the QLDB driver.

For short code examples of how to run basic data transactions on a ledger, see the [Cookbook reference](#).

Other optional libraries

Optionally, you can also add the following useful libraries in your project. These artifacts are required dependencies in the [Java tutorial](#) sample application.

1. [aws-java-sdk-qldb](#) – The QLDB module of the AWS SDK for Java. The minimum QLDB supported version is `1.11.785`.

Use this module in your application to interact directly with the management API operations listed in the [Amazon QLDB API reference](#).

2. [jackson-dataformat-ion](#) – FasterXML's Jackson data format module for Ion. The sample application requires version 2.10.0 or later.

Gradle

Add these dependencies in your `build.gradle` configuration file.

```
dependencies {  
    compile group: 'com.amazonaws', name: 'aws-java-sdk-qldb', version: '1.11.785'  
    compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-ion', version: '2.10.0'  
}
```

Maven

Add these dependencies in your `pom.xml` configuration file.

```
<dependencies>  
  <dependency>  
    <groupId>com.amazonaws</groupId>  
    <artifactId>aws-java-sdk-qldb</artifactId>  
    <version>1.11.785</version>  
  </dependency>  
  <dependency>  
    <groupId>com.fasterxml.jackson.dataformat</groupId>  
    <artifactId>jackson-dataformat-ion</artifactId>  
    <version>2.10.0</version>  
  </dependency>  
</dependencies>
```

Amazon QLDB driver for Java – Quick start tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this tutorial, you learn how to set up a simple application using the latest version of the Amazon QLDB driver for Java. This guide includes steps for installing the driver and short code examples of basic *create*, *read*, *update*, and *delete* (CRUD) operations. For more in-depth examples that demonstrate these operations in a full sample application, see the [Java tutorial](#).

Topics

- [Prerequisites](#)
- [Step 1: Set up your project](#)
- [Step 2: Initialize the driver](#)
- [Step 3: Create a table and an index](#)
- [Step 4: Insert a document](#)
- [Step 5: Query the document](#)
- [Step 6: Update the document](#)
- [Running the complete application](#)

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the [Prerequisites](#) for the Java driver, if you haven't already done so. This includes signing up for AWS, granting programmatic access for development, and installing a Java integrated development environment (IDE).
2. Create a ledger named `quick-start`.

To learn how to create a ledger, see [Basic operations for Amazon QLDB ledgers](#) or [Step 1: Create a new ledger](#) in *Getting started with the console*.

Step 1: Set up your project

First, set up your Java project. We recommend using the [Maven](#) dependency management system for this tutorial.

Note

If you use an IDE that has features to automate these setup steps, you can skip ahead to [Step 2: Initialize the driver](#).

1. Create a folder for your application.

```
$ mkdir myproject
$ cd myproject
```

2. Enter the following command to initialize your project from a Maven template. Replace *project-package*, *project-name*, and *maven-template* with your own values as appropriate.

```
$ mvn archetype:generate
  -DgroupId=project-package \
  -DartifactId=project-name \
  -DarchetypeArtifactId=maven-template \
  -DinteractiveMode=false
```

For *maven-template*, you can use the basic Maven template: `maven-archetype-quickstart`

3. To add the [QLDB driver for Java](#) as a project dependency, navigate to your newly created `pom.xml` file and add the following artifact.

```
<dependency>
  <groupId>software.amazon.qlldb</groupId>
  <artifactId>amazon-qlldb-driver-java</artifactId>
  <version>2.3.1</version>
</dependency>
```

This artifact automatically includes the [AWS SDK for Java 2.x](#) core module, [Amazon Ion](#) libraries, and other required dependencies. Your `pom.xml` file should now look similar to the following.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>software.amazon.qldb</groupId>
  <artifactId>qldb-quickstart</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>qldb-quickstart</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>software.amazon.qldb</groupId>
      <artifactId>amazon-qldb-driver-java</artifactId>
      <version>2.3.1</version>
    </dependency>
  </dependencies>
</project>
```

4. Open the App.java file.

Then, incrementally add the code examples in the following steps to try some basic CRUD operations. Or, you can skip the step-by-step tutorial and instead run the [complete application](#).

Step 2: Initialize the driver

Initialize an instance of the driver that connects to the ledger named quick-start. Add the following code to your App.java file.

```
import java.util.*;
import com.amazon.ion.*;
import com.amazon.ion.system.*;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.*;

public final class App {
```

```
public static IonSystem ionSys = IonSystemBuilder.standard().build();
public static QldbDriver qldbDriver;

public static void main(final String... args) {
    System.out.println("Initializing the driver");
    qldbDriver = QldbDriver.builder()
        .ledger("quick-start")
        .transactionRetryPolicy(RetryPolicy
            .builder()
            .maxRetries(3)
            .build())
        .sessionClientBuilder(QldbSessionClient.builder())
        .build();
}
}
```

Step 3: Create a table and an index

The following code example shows how to run `CREATE TABLE` and `CREATE INDEX` statements.

In the main method, add the following code that creates a table named `People` and an index for the `lastName` field on that table. [Indexes](#) are required to optimize query performance and help to limit [optimistic concurrency control \(OCC\)](#) conflict exceptions.

```
// Create a table and an index in the same transaction
qldbDriver.execute(txn -> {
    System.out.println("Creating a table and an index");
    txn.execute("CREATE TABLE People");
    txn.execute("CREATE INDEX ON People(lastName)");
});
```

Step 4: Insert a document

The following code example shows how to run an `INSERT` statement. QLDB supports the [PartiQL](#) query language (SQL compatible) and the [Amazon Ion](#) data format (superset of JSON).

Add the following code that inserts a document into the `People` table.

```
// Insert a document
qldbDriver.execute(txn -> {
    System.out.println("Inserting a document");
```

```
IonStruct person = ionSys.newEmptyStruct();
person.put("firstName").newString("John");
person.put("lastName").newString("Doe");
person.put("age").newInt(32);
txn.execute("INSERT INTO People ?", person);
});
```

This example uses a question mark (?) as a variable placeholder to pass the document information to the statement. When you use placeholders, you must pass a value of type `IonValue`.

Tip

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [IonList](#) (explicitly cast as an `IonValue`) to the statement as follows.

```
// people is an IonList explicitly cast as an IonValue
txn.execute("INSERT INTO People ?", (IonValue) people);
```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . >>) when passing an `IonList`. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Step 5: Query the document

The following code example shows how to run a `SELECT` statement.

Add the following code that queries a document from the `People` table.

```
// Query the document
qlldbDriver.execute(txn -> {
    System.out.println("Querying the table");
    Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
        ionSys.newString("Doe"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("firstName")); // prints John
    System.out.println(person.get("lastName")); // prints Doe
    System.out.println(person.get("age")); // prints 32
});
```

Step 6: Update the document

The following code example shows how to run an UPDATE statement.

1. Add the following code that updates a document in the `People` table by updating age to 42.

```
// Update the document
qlldbDriver.execute(txn -> {
    System.out.println("Updating the document");
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(ionSys.newInt(42));
    parameters.add(ionSys.newString("Doe"));
    txn.execute("UPDATE People SET age = ? WHERE lastName = ?", parameters);
});
```

2. Query the document again to see the updated value.

```
// Query the updated document
qlldbDriver.execute(txn -> {
    System.out.println("Querying the table for the updated document");
    Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
        ionSys.newString("Doe"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("firstName")); // prints John
    System.out.println(person.get("lastName")); // prints Doe
    System.out.println(person.get("age")); // prints 32
});
```

3. Use Maven or your IDE to compile and run the `App.java` file.

Running the complete application

The following code example is the complete version of the `App.java` application. Instead of doing the previous steps individually, you can also copy and run this code example from start to end. This application demonstrates some basic CRUD operations on the ledger named `quick-start`.

Note

Before you run this code, make sure that you don't already have an active table named `People` in the `quick-start` ledger.

On the first line, replace *project-package* with the groupId value that you used for the Maven command in [Step 1: Set up your project](#).

```
package project-package;

import java.util.*;
import com.amazon.ion.*;
import com.amazon.ion.system.*;
import software.amazon.awssdk.services.qldb.session.QldbSessionClient;
import software.amazon.qldb.*;

public class App {
    public static IonSystem ionSys = IonSystemBuilder.standard().build();
    public static QldbDriver qldbDriver;

    public static void main(final String... args) {
        System.out.println("Initializing the driver");
        qldbDriver = QldbDriver.builder()
            .ledger("quick-start")
            .transactionRetryPolicy(RetryPolicy
                .builder()
                .maxRetries(3)
                .build())
            .sessionClientBuilder(QldbSessionClient.builder())
            .build();

        // Create a table and an index in the same transaction
        qldbDriver.execute(txn -> {
            System.out.println("Creating a table and an index");
            txn.execute("CREATE TABLE People");
            txn.execute("CREATE INDEX ON People(lastName)");
        });

        // Insert a document
        qldbDriver.execute(txn -> {
            System.out.println("Inserting a document");
            IonStruct person = ionSys.newEmptyStruct();
            person.put("firstName").newString("John");
            person.put("lastName").newString("Doe");
            person.put("age").newInt(32);
            txn.execute("INSERT INTO People ?", person);
        });
    }
}
```

```
// Query the document
qlldbDriver.execute(txn -> {
    System.out.println("Querying the table");
    Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
        ionSys.newString("Doe"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("firstName")); // prints John
    System.out.println(person.get("lastName")); // prints Doe
    System.out.println(person.get("age")); // prints 32
});

// Update the document
qlldbDriver.execute(txn -> {
    System.out.println("Updating the document");
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(ionSys.newInt(42));
    parameters.add(ionSys.newString("Doe"));
    txn.execute("UPDATE People SET age = ? WHERE lastName = ?", parameters);
});

// Query the updated document
qlldbDriver.execute(txn -> {
    System.out.println("Querying the table for the updated document");
    Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
        ionSys.newString("Doe"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("firstName")); // prints John
    System.out.println(person.get("lastName")); // prints Doe
    System.out.println(person.get("age")); // prints 42
});
}
```

Use Maven or your IDE to compile and run the `App.java` file.

Amazon QLDB driver for Java – Cookbook reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This reference guide shows common use cases of the Amazon QLDB driver for Java. It provides Java code examples that demonstrate how to use the driver to run basic *create, read, update, and delete* (CRUD) operations. It also includes code examples for processing Amazon Ion data. In addition, this guide highlights best practices for making transactions idempotent and implementing uniqueness constraints.

Note

Where applicable, some use cases have different code examples for each supported major version of the QLDB driver for Java.

Contents

- [Importing the driver](#)
- [Instantiating the driver](#)
- [CRUD operations](#)
 - [Creating tables](#)
 - [Creating indexes](#)
 - [Reading documents](#)
 - [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
 - [Updating documents](#)
 - [Deleting documents](#)
 - [Running multiple statements in a transaction](#)
- [Retry logic](#)

- [Implementing uniqueness constraints](#)
- [Working with Amazon Ion](#)
 - [Importing the Ion packages](#)
 - [Initializing Ion](#)
 - [Creating Ion objects](#)
 - [Reading Ion objects](#)

Importing the driver

The following code example imports the driver, the QLDB session client, Amazon Ion packages, and other related dependencies.

2.x

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;

import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;
```

1.x

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;

import com.amazonaws.services.qldbsession.AmazonQLDBSessionClientBuilder;
import software.amazon.qldb.PooledQldbDriver;
import software.amazon.qldb.Result;
```

Instantiating the driver

The following code example creates a driver instance that connects to a specified ledger name, and uses specified [retry logic](#) with a custom retry limit.

Note

This example also instantiates an Amazon Ion system object (IonSystem). You need this object to process Ion data when running some data operations in this reference. To learn more, see [Working with Amazon Ion](#).

2.x

```
QldbDriver qldbDriver = QldbDriver.builder()
    .ledger("vehicle-registration")
    .transactionRetryPolicy(RetryPolicy
        .builder()
        .maxRetries(3)
        .build())
    .sessionClientBuilder(QldbSessionClient.builder())
    .build();

IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

1.x

```
PooledQldbDriver qldbDriver = PooledQldbDriver.builder()
    .withLedger("vehicle-registration")
    .withRetryLimit(3)
    .withSessionClientBuilder(AmazonQLDBSessionClientBuilder.standard())
    .build();

IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

CRUD operations

QLDB runs *create*, *read*, *update*, and *delete* (CRUD) operations as part of a transaction.

Warning

As a best practice, make your write transactions strictly idempotent.

Making transactions idempotent

We recommend that you make write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can run multiple times and produce identical results each time.

For example, consider a transaction that inserts a document into a table named `Person`. The transaction should first check whether or not the document already exists in the table. Without this check, the table might end up with duplicate documents.

Suppose that QLDB successfully commits the transaction on the server side, but the client times out while waiting for a response. If the transaction isn't idempotent, the same document could be inserted more than once in the case of a retry.

Using indexes to avoid full table scans

We also recommend that you run statements with a `WHERE` predicate clause using an *equality* operator on an indexed field or a document ID; for example, `WHERE indexedField = 123` or `WHERE indexedField IN (456, 789)`. Without this indexed lookup, QLDB needs to do a table scan, which can lead to transaction timeouts or *optimistic concurrency control* (OCC) conflicts.

For more information about OCC, see [Amazon QLDB concurrency model](#).

Implicitly created transactions

The [QldbDriver.execute](#) method accepts a lambda function that receives an instance of [Executor](#), which you can use to run statements. The `Executor` instance wraps an implicitly created transaction.

You can run statements within the lambda function by using the `Executor.execute` method. The driver implicitly commits the transaction when the lambda function returns.

The following sections show how to run basic CRUD operations, specify custom retry logic, and implement uniqueness constraints.

Note

Where applicable, these sections provide code examples of processing Amazon Ion data using both the built-in Ion library and the Jackson Ion mapper library. To learn more, see [Working with Amazon Ion](#).

Contents

- [Creating tables](#)
- [Creating indexes](#)
- [Reading documents](#)
- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)
- [Running multiple statements in a transaction](#)
- [Retry logic](#)
- [Implementing uniqueness constraints](#)

Creating tables

```
qldbDriver.execute(txn -> {  
    txn.execute("CREATE TABLE Person");  
});
```

Creating indexes

```
qldbDriver.execute(txn -> {  
    txn.execute("CREATE INDEX ON Person(GovId)");  
});
```

Reading documents

```
// Assumes that Person table has documents as follows:  
// { GovId: "TOYENC486FH", FirstName: "Brent" }  
  
qldbDriver.execute(txn -> {  
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");  
    IonStruct person = (IonStruct) result.iterator().next();  
    System.out.println(person.get("GovId")); // prints TOYENC486FH  
    System.out.println(person.get("FirstName")); // prints Brent  
});
```

Using query parameters

The following code example uses an Ion type query parameter.

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
        SYSTEM.newString("TOYENC486FH"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

The following code example uses multiple query parameters.

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?",
        SYSTEM.newString("TOYENC486FH"),
        SYSTEM.newString("Brent"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

The following code example uses a list of query parameters.

```
qldbDriver.execute(txn -> {
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(SYSTEM.newString("TOYENC486FH"));
    parameters.add(SYSTEM.newString("ROEE1"));
    parameters.add(SYSTEM.newString("YH844"));
    Result result = txn.execute("SELECT * FROM Person WHERE GovId IN (?, ?, ?)",
        parameters);
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

Using the Jackson mapper

```
// Assumes that Person table has documents as follows:
// {GovId: "TOYENC486FH", FirstName: "Brent" }

qldbDriver.execute(txn -> {
```

```
try {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId =
'TOYENC486FH'");
    Person person = MAPPER.readValue(result.iterator().next(), Person.class);
    System.out.println(person.getFirstName()); // prints Brent
    System.out.println(person.getGovId()); // prints TOYENC486FH
} catch (IOException e) {
    e.printStackTrace();
}
});
```

Using query parameters

The following code example uses an Ion type query parameter.

```
qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            MAPPER.writeValueAsIonValue("TOYENC486FH"));
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

The following code example uses multiple query parameters.

```
qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ? AND FirstName
= ?",
            MAPPER.writeValueAsIonValue("TOYENC486FH"),
            MAPPER.writeValueAsIonValue("Brent"));
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

The following code example uses a list of query parameters.

```
qldbDriver.execute(txn -> {
    try {
        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(MAPPER.writeValueAsIonValue("TOYENC486FH"));
        parameters.add(MAPPER.writeValueAsIonValue("ROEE1"));
        parameters.add(MAPPER.writeValueAsIonValue("YH844"));
        Result result = txn.execute("SELECT * FROM Person WHERE GovId IN (?, ?, ?)",
parameters);
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

Note

When you run a query without an indexed lookup, it invokes a full table scan. In this example, we recommend having an [index](#) on the GovId field to optimize performance. Without an index on GovId, queries can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting documents

The following code example inserts Ion data types.

```
qldbDriver.execute(txn -> {
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
        SYSTEM.newString("TOYENC486FH"));
    // Check if there is a result
    if (!result.iterator().hasNext()) {
        IonStruct person = SYSTEM.newEmptyStruct();
        person.put("GovId").newString("TOYENC486FH");
        person.put("FirstName").newString("Brent");
        // Insert the document
    }
});
```



```
        txn.execute("INSERT INTO Person ?", person);
    }
});
```

Using the Jackson mapper

The following code example inserts `Ion` data types.

```
qldbDriver.execute(txn -> {
    try {
        // Check if a document with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            MAPPER.writeValueAsIonValue("TOYENC486FH"));
        // Check if there is a result
        if (!result.iterator().hasNext()) {
            // Insert the document
            txn.execute("INSERT INTO Person ?",
                MAPPER.writeValueAsIonValue(new Person("Brent", "TOYENC486FH")));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

This transaction inserts a document into the `Person` table. Before inserting, it first checks if the document already exists in the table. **This check makes the transaction idempotent in nature.** Even if you run this transaction multiple times, it won't cause any unintended side effects.

Note

In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting multiple documents in one statement

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [IonList](#) (explicitly cast as an `IonValue`) to the statement as follows.

```
// people is an IonList explicitly cast as an IonValue
txn.execute("INSERT INTO People ?", (IonValue) people);
```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . . >>) when passing an IonList. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Why is the explicit cast required?

The [TransactionExecutor.execute](#) method is overloaded. It accepts a variable number of IonValue arguments (*varargs*), or a single List<IonValue> argument. In [ion-java](#), IonList is implemented as a List<IonValue>.

Java defaults to the most specific method implementation when you call an overloaded method. In this case, when you pass an IonList parameter, it defaults to the method that takes a List<IonValue>. When invoked, this method implementation passes the IonValue elements of the list as distinct values. So, to invoke the varargs method instead, you must explicitly cast an IonList parameter as an IonValue.

Updating documents

```
qldbDriver.execute(txn -> {
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(SYSTEM.newString("John"));
    parameters.add(SYSTEM.newString("TOYENC486FH"));
    txn.execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", parameters);
});
```

Using the Jackson mapper

```
qldbDriver.execute(txn -> {
    try {
        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(MAPPER.writeValueAsIonValue("John"));
        parameters.add(MAPPER.writeValueAsIonValue("TOYENC486FH"));
        txn.execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", parameters);
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Deleting documents

```
qlldbDriver.execute(txn -> {  
    txn.execute("DELETE FROM Person WHERE GovId = ?",  
        SYSTEM.newString("TOYENC486FH"));  
});
```

Using the Jackson mapper

```
qlldbDriver.execute(txn -> {  
    try {  
        txn.execute("DELETE FROM Person WHERE GovId = ?",  
            MAPPER.writeValueAsIonValue("TOYENC486FH"));  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
});
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Running multiple statements in a transaction

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because  
// you'd  
// set your UPDATE to filter on vin and insured, and check if you updated something or  
// not.  
public static boolean InsureCar(qlldbDriver qlldbDriver, final String vin) {  
    final IonSystem ionSystem = IonSystemBuilder.standard().build();
```

```

final IonString ionVin = ionSystem.newString(vin);

return qlldbDriver.execute(txn -> {
    Result result = txn.execute(
        "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
        ionVin);
    if (!result.isEmpty()) {
        txn.execute("UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
        return true;
    }
    return false;
});
}

```

Retry logic

The driver's `execute` method has a built-in retry mechanism that retries the transaction if a retryable exception occurs (such as timeouts or OCC conflicts).

2.x

The maximum number of retry attempts and the backoff strategy are configurable.

The default retry limit is 4, and the default backoff strategy is [DefaultQldbTransactionBackoffStrategy](#). You can set the retry configuration per driver instance and also per transaction by using an instance of [RetryPolicy](#).

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for an instance of the driver.

```

public void retry() {
    QldbDriver qlldbDriver = QldbDriver.builder()
        .ledger("vehicle-registration")
        .transactionRetryPolicy(RetryPolicy.builder()
            .maxRetries(2)
            .backoffStrategy(new CustomBackOffStrategy()).build())
        .sessionClientBuilder(QldbSessionClient.builder())
        .build();
}

private class CustomBackOffStrategy implements BackoffStrategy {

    @Override

```

```

    public Duration calculateDelay(RetryPolicyContext retryPolicyContext) {
        return Duration.ofMillis(1000 * retryPolicyContext.retriesAttempted());
    }
}

```

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for a particular transaction. This configuration for `execute` overrides the retry logic that is set for the driver instance.

```

public void retry() {
    Result result = qlldbDriver.execute(txn -> { txn.execute("SELECT * FROM Person
    WHERE GovId = ?",
        SYSTEM.newString("TOYENC486FH")); },
        RetryPolicy.builder()
            .maxRetries(2)
            .backoffStrategy(new CustomBackOffStrategy())
            .build());
}

private class CustomBackOffStrategy implements BackoffStrategy {

    // Configuring a custom backoff which increases delay by 1s for each attempt.
    @Override
    public Duration calculateDelay(RetryPolicyContext retryPolicyContext) {
        return Duration.ofMillis(1000 * retryPolicyContext.retriesAttempted());
    }
}

```

1.x

The maximum number of retry attempts is configurable. You can configure the retry limit by setting the `retryLimit` property when initializing `PooledQldbDriver`.

The default retry limit is 4.

Implementing uniqueness constraints

QLDB doesn't support unique indexes, but you can implement this behavior in your application.

Suppose that you want to implement a uniqueness constraint on the `GovId` field in the `Person` table. To do this, you can write a transaction that does the following:

1. Assert that the table has no existing documents with a specified GovId.
2. Insert the document if the assertion passes.

If a competing transaction concurrently passes the assertion, only one of the transactions will commit successfully. The other transaction will fail with an OCC conflict exception.

The following code example shows how to implement this uniqueness constraint logic.

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
        SYSTEM.newString("TOYENC486FH"));
    // Check if there is a result
    if (!result.iterator().hasNext()) {
        IonStruct person = SYSTEM.newEmptyStruct();
        person.put("GovId").newString("TOYENC486FH");
        person.put("FirstName").newString("Brent");
        // Insert the document
        txn.execute("INSERT INTO Person ?", person);
    }
});
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Working with Amazon Ion

There are multiple ways to process Amazon Ion data in QLDB. You can use built-in methods from the [Ion library](#) to create and modify documents flexibly as needed. Or, you can use FasterXML's [Jackson data format module for Ion](#) to map Ion documents to *plain old Java object* (POJO) models.

The following sections provide code examples of processing Ion data using both techniques.

Contents

- [Importing the Ion packages](#)
- [Initializing Ion](#)

- [Creating Ion objects](#)
- [Reading Ion objects](#)

Importing the Ion packages

Add the artifact [ion-java](#) as a dependency in your Java project.

Gradle

```
dependencies {  
    compile group: 'com.amazon.ion', name: 'ion-java', version: '1.6.1'  
}
```

Maven

```
<dependencies>  
  <dependency>  
    <groupId>com.amazon.ion</groupId>  
    <artifactId>ion-java</artifactId>  
    <version>1.6.1</version>  
  </dependency>  
</dependencies>
```

Import the following Ion packages.

```
import com.amazon.ion.IonStruct;  
import com.amazon.ion.IonSystem;  
import com.amazon.ion.system.IonSystemBuilder;
```

Using the Jackson mapper

Add the artifact [jackson-dataformat-ion](#) as a dependency in your Java project. QLDB requires version 2.10.0 or later.

Gradle

```
dependencies {  
    compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-ion', version: '2.10.0'
```

```
}
```

Maven

```
<dependencies>
  <dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-ion</artifactId>
    <version>2.10.0</version>
  </dependency>
</dependencies>
```

Import the following Ion packages.

```
import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonSystemBuilder;

import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.value.IonValueMapper;
```

Initializing Ion

```
IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

Using the Jackson mapper

```
IonObjectMapper MAPPER = new IonValueMapper(IonSystemBuilder.standard().build());
```

Creating Ion objects

The following code example creates an Ion object by using the `IonStruct` interface and its built-in methods.

```
IonStruct ionStruct = SYSTEM.newEmptyStruct();

ionStruct.put("GovId").newString("T0YENC486FH");
ionStruct.put("FirstName").newString("Brent");
```



```
System.out.println(ionStruct.toPrettyString()); // prints a nicely formatted copy of
ionStruct
```

Using the Jackson mapper

Suppose that you have a JSON-mapped model class named `Person`, as follows.

```
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public static class Person {
    private final String firstName;
    private final String govId;

    @JsonCreator
    public Person(@JsonProperty("FirstName") final String firstName,
                 @JsonProperty("GovId") final String govId) {
        this.firstName = firstName;
        this.govId = govId;
    }

    @JsonProperty("FirstName")
    public String getFirstName() {
        return firstName;
    }

    @JsonProperty("GovId")
    public String getGovId() {
        return govId;
    }
}
```

The following code example creates an `IonStruct` object from an instance of `Person`.

```
IonStruct ionStruct = (IonStruct) MAPPER.writeValueAsIonValue(new Person("Brent",
"TOYENC486FH"));
```

Reading Ion objects

The following code example prints each field of the `ionStruct` instance.

```
// ionStruct is an instance of IonStruct
```

```
System.out.println(ionStruct.get("GovId")); // prints TOYENC486FH
System.out.println(ionStruct.get("FirstName")); // prints Brent
```

Using the Jackson mapper

The following code example reads an `IonStruct` object and maps it to an instance of `Person`.

```
// ionStruct is an instance of IonStruct
IonReader reader = IonReaderBuilder.standard().build(ionStruct);
Person person = MAPPER.readValue(reader, Person.class);
System.out.println(person.getFirstName()); // prints Brent
System.out.println(person.getGovId()); // prints TOYENC486FH
```

For more information about working with Ion, see the [Amazon Ion documentation](#) on GitHub. For more code examples of working with Ion in QLDB, see [Working with Amazon Ion data types in Amazon QLDB](#).

Amazon QLDB driver for .NET

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

To work with data in your ledger, you can connect to Amazon QLDB from your Microsoft .NET application by using an AWS provided driver. The driver targets **.NET Standard 2.0**. More specifically, it supports **.NET Core (LTS) 2.1+** and **.NET Framework 4.5.2+**. For information on compatibility, see [.NET Standard](#) on the *Microsoft Docs* site.

We highly recommend using the *Ion object mapper* to completely bypass the need to manually convert between Amazon Ion types and native C# types.

The following topics describe how to get started with the QLDB driver for .NET.

Topics

- [Driver resources](#)
- [Prerequisites](#)

- [Installation](#)
- [Amazon QLDB driver for .NET – Quick start tutorial](#)
- [Amazon QLDB driver for .NET – Cookbook reference](#)

Driver resources

For more information about the functionality supported by the .NET driver, see the following resources:

- [API reference](#)
- [Driver source code \(GitHub\)](#)
- [Sample application source code \(GitHub\)](#)
- [Amazon Ion Cookbook](#)
- [Ion object mapper \(GitHub\)](#)

Prerequisites

Before you get started with the QLDB driver for .NET, you must do the following:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#). This includes the following:
 1. Sign up for AWS.
 2. Create a user with the appropriate QLDB permissions.
 3. Grant programmatic access for development.
2. Download and install the .NET Core SDK version 2.1 or later from the [Microsoft .NET downloads](#) site.
3. (Optional) Install an integrated development environment (IDE) of your choice, such as Visual Studio, Visual Studio for Mac, or Visual Studio Code. You can download these from the [Microsoft Visual Studio](#) site.
4. Configure your development environment for the [AWS SDK for .NET](#):
 1. Set up your AWS credentials. We recommend creating a shared credentials file.

For instructions, see [Configuring AWS credentials using a credentials file](#) in the *AWS SDK for .NET Developer Guide*.

2. Set your default AWS Region. To learn how, see [AWS Region selection](#).

For a complete list of available Regions, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

Next, you can set up a basic sample application and run short code examples—or you can install the driver in an existing .NET project.

- To install the QLDB driver and the AWS SDK for .NET in an existing project, proceed to [Installation](#).
- To set up a project and run short code examples that demonstrate basic data transactions on a ledger, see the [Quick start tutorial](#).

Installation

Use the NuGet package manager to install the QLDB driver for .NET. We recommend using Visual Studio or an IDE of your choice to add project dependencies. The driver package name is [Amazon.QLDB.Driver](#).

For example in Visual Studio, open the **NuGet Package Manager Console** on the **Tools** menu. Then, enter the following command at the PM> prompt.

```
PM> Install-Package Amazon.QLDB.Driver
```

Installing the driver also installs its dependencies, including the AWS SDK for .NET and [Amazon Ion](#) packages.

Install the Ion object mapper

Version 1.3.0 of the QLDB driver for .NET introduces support for accepting and returning native C# data types without the need to work with Amazon Ion. To use this feature, add the following package to your project.

- [Amazon.QLDB.Driver.Serialization](#) – A library that can map Ion values to C# *plain old CLR objects* (POCO), and the other way around. This Ion object mapper lets your application interact directly with native C# data types without the need to work with Ion. For a short guide on how to use this library, see the [SERIALIZATION.md](#) file in the GitHub repository `aws-labs/amazon-qlldb-driver-dotnet`.

To install this package, enter the following command.

```
PM> Install-Package Amazon.QLDB.Driver.Serialization
```

For short code examples of how to run basic data transactions on a ledger, see the [Cookbook reference](#).

Amazon QLDB driver for .NET – Quick start tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this tutorial, you learn how to set up a simple application using the Amazon QLDB driver for .NET. This guide includes steps for installing the driver and short code examples of basic *create*, *read*, *update*, and *delete* (CRUD) operations.

Topics

- [Prerequisites](#)
- [Step 1: Set up your project](#)
- [Step 2: Initialize the driver](#)
- [Step 3: Create a table and an index](#)
- [Step 4: Insert a document](#)
- [Step 5: Query the document](#)
- [Step 6: Update the document](#)
- [Running the complete application](#)

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the [Prerequisites](#) for the .NET driver, if you haven't already done so. This includes signing up for AWS, granting programmatic access for development, and installing the .NET Core SDK.

2. Create a ledger named quick-start.

To learn how to create a ledger, see [Basic operations for Amazon QLDB ledgers](#) or [Step 1: Create a new ledger](#) in *Getting started with the console*.

Step 1: Set up your project

First, set up your .NET project.

1. To create and run a template application, enter the following dotnet commands on a terminal such as *bash*, *PowerShell*, or *Command Prompt*.

```
$ dotnet new console --output Amazon.QLDB.QuickStartGuide
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

This template creates a folder named `Amazon.QLDB.QuickStartGuide`. In that folder, it creates a project with the same name and a file named `Program.cs`. The program contains code that displays the output `Hello World!`.

2. Use the NuGet package manager to install the QLDB driver for .NET. We recommend using Visual Studio or an IDE of your choice to add dependencies to your project. The driver package name is [Amazon.QLDB.Driver](#).
 - For example in Visual Studio, open the **NuGet Package Manager Console** on the **Tools** menu. Then, enter the following command at the `PM>` prompt.

```
PM> Install-Package Amazon.QLDB.Driver
```

- Or, you can enter the following commands on your terminal.

```
$ cd Amazon.QLDB.QuickStartGuide
$ dotnet add package Amazon.QLDB.Driver
```

Installing the driver also installs its dependencies, including the [AWS SDK for .NET](#) and [Amazon Ion](#) libraries.

3. Install the driver's serialization library.

```
PM> Install-Package Amazon.QLDB.Driver.Serialization
```

4. Open the `Program.cs` file.

Then, incrementally add the code examples in the following steps to try some basic CRUD operations. Or, you can skip the step-by-step tutorial and instead run the [complete application](#).

Note

- **Choosing between synchronous and asynchronous APIs** – The driver provides synchronous and asynchronous APIs. For high-demand applications that handle multiple requests without blocking, we recommend using the asynchronous APIs for improved performance. The driver offers synchronous APIs as an added convenience for existing code bases that are written synchronously.

This tutorial includes both synchronous and asynchronous code examples. For more information about the APIs, see the [IQldbDriver](#) and [IAsyncQldbDriver](#) interfaces in the API documentation.

- **Processing Amazon Ion data** – This tutorial provides code examples of processing Amazon Ion data using the [Ion object mapper](#) by default. QLDB introduced the Ion object mapper in version 1.3.0 of the .NET driver. Where applicable, this tutorial also provides code examples using the standard [Ion library](#) as an alternative. To learn more, see [Working with Amazon Ion](#).

Step 2: Initialize the driver

Initialize an instance of the driver that connects to the ledger named `quick-start`. Add the following code to your `Program.cs` file.

Async

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
```

```

    public class Person
    {
        public string FirstName { get; set; }

        public string LastName { get; set; }

        public int Age { get; set; }

        public override string ToString()
        {
            return FirstName + ", " + LastName + ", " + Age.ToString();
        }
    }

    static async Task Main(string[] args)
    {
        Console.WriteLine("Create the async QLDB driver");
        IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
            .WithLedger("quick-start")
            .WithSerializer(new ObjectSerializer())
            .Build();
    }
}

```

Sync

```

using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }
        }
    }
}

```



```
        public override string ToString()
        {
            return FirstName + ", " + LastName + ", " + Age.ToString();
        }
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Create the sync QLDB driver");
        IQldbDriver driver = QldbDriver.Builder()
            .WithLedger("quick-start")
            .WithSerializer(new ObjectSerializer())
            .Build();
    }
}
```

Using the Ion library

Async

```
using System;
using System.Threading.Tasks;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();
        }
    }
}
```

```
}
```

Sync

```
using System;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();
        }
    }
}
```

Step 3: Create a table and an index

For the rest of this tutorial through *Step 6*, you need to append the following code examples to the previous code example.

In this step, the following code shows how to run `CREATE TABLE` and `CREATE INDEX` statements. It creates a table named `Person` and an index for the `firstName` field on that table. [Indexes](#) are required to optimize query performance and help to limit [optimistic concurrency control \(OCC\)](#) conflict exceptions.

Async

```
Console.WriteLine("Creating the table and index");

// Creates the table and the index in the same transaction.
```

```
// Note: Any code within the lambda can potentially execute multiple times due to
retries.
// For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
await driver.Execute(async txn =>
{
    await txn.Execute("CREATE TABLE Person");
    await txn.Execute("CREATE INDEX ON Person(firstName)");
});
```

Sync

```
Console.WriteLine("Creating the tables and index");

// Creates the table and the index in the same transaction.
// Note: Any code within the lambda can potentially execute multiple times due to
retries.
// For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
driver.Execute(txn =>
{
    txn.Execute("CREATE TABLE Person");
    txn.Execute("CREATE INDEX ON Person(firstName)");
});
```

Step 4: Insert a document

The following code example shows how to run an INSERT statement. QLDB supports the [PartiQL](#) query language (SQL compatible) and the [Amazon Ion](#) data format (superset of JSON).

Add the following code that inserts a document into the Person table.

Async

```
Console.WriteLine("Inserting a document");

Person myPerson = new Person {
    FirstName = "John",
    LastName = "Doe",
    Age = 32
};
```

```
await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?", myPerson);
    await txn.Execute(myQuery);
});
```

Sync

```
Console.WriteLine("Inserting a document");

Person myPerson = new Person {
    FirstName = "John",
    LastName = "Doe",
    Age = 32
};

driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?", myPerson);
    txn.Execute(myQuery);
});
```

Using the Ion library

Async

```
Console.WriteLine("Inserting a document");

// This is one way of creating Ion values. We can also use an IonLoader.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
ionPerson.SetField("age", valueFactory.NewInt(32));

await driver.Execute(async txn =>
{
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Sync

```
Console.WriteLine("Inserting a document");

// This is one way of creating Ion values, we can also use an IonLoader.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/
driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
ionPerson.SetField("age", valueFactory.NewInt(32));

driver.Execute(txn =>
{
    txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Tip

To insert multiple documents by using a single [INSERT](#) statement, you can pass an [Ion list](#) type parameter to the statement as follows.

```
// people is an Ion list
txn.Execute("INSERT INTO Person ?", people);
```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . . >>) when passing an Ion list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Step 5: Query the document

The following code example shows how to run a SELECT statement.

Add the following code that queries a document from the Person table.

Async

```
Console.WriteLine("Querying the table");
```

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IAsyncResult<Person> selectResult = await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE FirstName
= ?", "John");
    return await txn.Execute(myQuery);
});

await foreach (Person person in selectResult)
{
    Console.WriteLine(person);
    // John, Doe, 32
}
```

Sync

```
Console.WriteLine("Querying the table");

// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult<Person> selectResult = driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE FirstName
= ?", "John");
    return txn.Execute(myQuery);
});

foreach (Person person in selectResult)
{
    Console.WriteLine(person);
    // John, Doe, 32
}
```

Using the Ion library

Async

```
Console.WriteLine("Querying the table");

IIonValue ionFirstName = valueFactory.NewString("John");
```

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IAsyncResult selectResult = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName);
});

await foreach (IIonValue row in selectResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

Sync

```
Console.WriteLine("Querying the table");

IIonValue ionFirstName = valueFactory.NewString("John");

// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult selectResult = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE firstName = ?", ionFirstName);
});

foreach (IIonValue row in selectResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

This example uses a question mark (?) as a variable placeholder to pass the document information to the statement. When you use placeholders, you must pass a value of type `IonValue`.

Step 6: Update the document

The following code example shows how to run an UPDATE statement.

1. Add the following code that updates a document in the Person table by updating age to 42.

Async

```
Console.WriteLine("Updating the document");

await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age = ? WHERE
    FirstName = ?", 42, "John");
    await txn.Execute(myQuery);
});
```

Sync

```
Console.WriteLine("Updating the document");

driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age = ? WHERE
    FirstName = ?", 42, "John");
    txn.Execute(myQuery);
});
```

2. Query the document again to see the updated value.

Async

```
Console.WriteLine("Querying the table for the updated document");

IAsyncResult<Person> updateResult = await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE
    FirstName = ?", "John");
    return await txn.Execute(myQuery);
});

await foreach (Person person in updateResult)
{
    Console.WriteLine(person);
    // John, Doe, 42
}
```


Sync

```
Console.WriteLine("Querying the table for the updated document");

IResult<Person> updateResult = driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE
    FirstName = ?", "John");
    return txn.Execute(myQuery);
});

foreach (Person person in updateResult)
{
    Console.WriteLine(person);
    // John, Doe, 42
}
```

3. To run the application, enter the following command from the parent directory of your `Amazon.QLDB.QuickStartGuide` project directory.

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

Using the Ion library

1. Add the following code that updates a document in the `Person` table by updating age to 42.

Async

```
Console.WriteLine("Updating the document");

IIonValue ionIntAge = valueFactory.NewInt(42);
IIonValue ionFirstName2 = valueFactory.NewString("John");

await driver.Execute(async txn =>
{
    await txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
        ionIntAge, ionFirstName2);
});
```

Sync

```
Console.WriteLine("Updating a document");

IIonValue ionIntAge = valueFactory.NewInt(42);
IIonValue ionFirstName2 = valueFactory.NewString("John");

driver.Execute(txn =>
{
    txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?", ionIntAge,
        ionFirstName2);
});
```

2. Query the document again to see the updated value.

Async

```
Console.WriteLine("Querying the table for the updated document");

IIonValue ionFirstName3 = valueFactory.NewString("John");

IAsyncResult updateResult = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
        ionFirstName3 );
});

await foreach (IIonValue row in updateResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

Sync

```
Console.WriteLine("Querying the table for the updated document");

IIonValue ionFirstName3 = valueFactory.NewString("John");

IResult updateResult = driver.Execute(txn =>
{
```

```
        return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
            ionFirstName3);
    });

    foreach (IIonValue row in updateResult)
    {
        Console.WriteLine(row.GetField("firstName").StringValue);
        Console.WriteLine(row.GetField("lastName").StringValue);
        Console.WriteLine(row.GetField("age").IntValue);
    }
}
```

3. To run the application, enter the following command from the parent directory of your `Amazon.QLDB.QuickStartGuide` project directory.

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

Running the complete application

The following code example is the complete version of the `Program.cs` application. Instead of doing the previous steps individually, you can also copy and run this code example from start to end. This application demonstrates some basic CRUD operations on the ledger named `quick-start`.

Note

Before you run this code, make sure that you don't already have an active table named `Person` in the `quick-start` ledger.

Async

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
```

```
{
    public string FirstName { get; set; }

    public string LastName { get; set; }

    public int Age { get; set; }

    public override string ToString()
    {
        return FirstName + ", " + LastName + ", " + Age.ToString();
    }
}

static async Task Main(string[] args)
{
    Console.WriteLine("Create the async QLDB driver");
    IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
        .WithLedger("quick-start")
        .WithSerializer(new ObjectSerializer())
        .Build();

    Console.WriteLine("Creating the table and index");

    // Creates the table and the index in the same transaction.
    // Note: Any code within the lambda can potentially execute multiple
times due to retries.
    // For more information, see: https://docs.aws.amazon.com/qldb/latest/developerguide/driver-retry-policy
    await driver.Execute(async txn =>
    {
        await txn.Execute("CREATE TABLE Person");
        await txn.Execute("CREATE INDEX ON Person(firstName)");
    });

    Console.WriteLine("Inserting a document");

    Person myPerson = new Person {
        FirstName = "John",
        LastName = "Doe",
        Age = 32
    };

    await driver.Execute(async txn =>
    {
```

```

        IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?",
myPerson);
        await txn.Execute(myQuery);
    });

    Console.WriteLine("Querying the table");

    // The result from driver.Execute() is buffered into memory because once
the
    // transaction is committed, streaming the result is no longer possible.
    IAsyncResult<Person> selectResult = await driver.Execute(async txn =>
    {
        IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
WHERE FirstName = ?", "John");
        return await txn.Execute(myQuery);
    });

    await foreach (Person person in selectResult)
    {
        Console.WriteLine(person);
        // John, Doe, 32
    }

    Console.WriteLine("Updating the document");

    await driver.Execute(async txn =>
    {
        IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age
= ? WHERE FirstName = ?", 42, "John");
        await txn.Execute(myQuery);
    });

    Console.WriteLine("Querying the table for the updated document");

    IAsyncResult<Person> updateResult = await driver.Execute(async txn =>
    {
        IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
WHERE FirstName = ?", "John");
        return await txn.Execute(myQuery);
    });

    await foreach (Person person in updateResult)
    {
        Console.WriteLine(person);
    }

```

```

        // John, Doe, 42
    }
}
}
}

```

Sync

```

using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB driver");
            IQLdbDriver driver = QLdbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();

            Console.WriteLine("Creating the table and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
            times due to retries.

```

```

        // For more information, see: https://docs.aws.amazon.com/qldb/latest/
        developerguide/driver-retry-policy
        driver.Execute(txn =>
        {
            txn.Execute("CREATE TABLE Person");
            txn.Execute("CREATE INDEX ON Person(firstName)");
        });

        Console.WriteLine("Inserting a document");

        Person myPerson = new Person {
            FirstName = "John",
            LastName = "Doe",
            Age = 32
        };

        driver.Execute(txn =>
        {
            IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?",
myPerson);
            txn.Execute(myQuery);
        });

        Console.WriteLine("Querying the table");

        // The result from driver.Execute() is buffered into memory because once
the
        // transaction is committed, streaming the result is no longer possible.
        IResult<Person> selectResult = driver.Execute(txn =>
        {
            IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
WHERE FirstName = ?", "John");
            return txn.Execute(myQuery);
        });

        foreach (Person person in selectResult)
        {
            Console.WriteLine(person);
            // John, Doe, 32
        }

        Console.WriteLine("Updating the document");

        driver.Execute(txn =>

```

```

        {
            IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age
= ? WHERE FirstName = ?", 42, "John");
            txn.Execute(myQuery);
        });

        Console.WriteLine("Querying the table for the updated document");

        IResult<Person> updateResult = driver.Execute(txn =>
        {
            IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
WHERE FirstName = ?", "John");
            return txn.Execute(myQuery);
        });

        foreach (Person person in updateResult)
        {
            Console.WriteLine(person);
            // John, Doe, 42
        }
    }
}
}

```

Using the Ion library

Async

```

using System;
using System.Threading.Tasks;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();
    }
}

```



```

static async Task Main(string[] args)
{
    Console.WriteLine("Create the async QLDB driver");
    IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
        .WithLedger("quick-start")
        .Build();

    Console.WriteLine("Creating the table and index");

    // Creates the table and the index in the same transaction.
    // Note: Any code within the lambda can potentially execute multiple
times due to retries.
    // For more information, see: https://docs.aws.amazon.com/qldb/latest/developerguide/driver-retry-policy
    await driver.Execute(async txn =>
    {
        await txn.Execute("CREATE TABLE Person");
        await txn.Execute("CREATE INDEX ON Person(firstName)");
    });

    Console.WriteLine("Inserting a document");

    // This is one way of creating Ion values. We can also use an IonLoader.
    // For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
    IIonValue ionPerson = valueFactory.NewEmptyStruct();
    ionPerson.SetField("firstName", valueFactory.NewString("John"));
    ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
    ionPerson.SetField("age", valueFactory.NewInt(32));

    await driver.Execute(async txn =>
    {
        await txn.Execute("INSERT INTO Person ?", ionPerson);
    });

    Console.WriteLine("Querying the table");

    IIonValue ionFirstName = valueFactory.NewString("John");

    // The result from driver.Execute() is buffered into memory because once
the
    // transaction is committed, streaming the result is no longer possible.
    IAsyncResult selectResult = await driver.Execute(async txn =>
    {

```

```

        return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName);
    });

    await foreach (IIonValue row in selectResult)
    {
        Console.WriteLine(row.GetField("firstName").StringValue);
        Console.WriteLine(row.GetField("lastName").StringValue);
        Console.WriteLine(row.GetField("age").IntValue);
    }

    Console.WriteLine("Updating the document");

    IIonValue ionIntAge = valueFactory.NewInt(42);
    IIonValue ionFirstName2 = valueFactory.NewString("John");

    await driver.Execute(async txn =>
    {
        await txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
ionIntAge, ionFirstName2);
    });

    Console.WriteLine("Querying the table for the updated document");

    IIonValue ionFirstName3 = valueFactory.NewString("John");

    IAsyncResult updateResult = await driver.Execute(async txn =>
    {
        return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName3);
    });

    await foreach (IIonValue row in updateResult)
    {
        Console.WriteLine(row.GetField("firstName").StringValue);
        Console.WriteLine(row.GetField("lastName").StringValue);
        Console.WriteLine(row.GetField("age").IntValue);
    }
}
}
}

```

Sync

```
using System;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();

            Console.WriteLine("Creating the tables and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
            // times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/developerguide/driver-retry-policy
            driver.Execute(txn =>
            {
                txn.Execute("CREATE TABLE Person");
                txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            // This is one way of creating Ion values. We can also use an IonLoader.
            // For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
            IIonValue ionPerson = valueFactory.NewEmptyStruct();
            ionPerson.SetField("firstName", valueFactory.NewString("John"));
            ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
            ionPerson.SetField("age", valueFactory.NewInt(32));

            driver.Execute(txn =>
```

```

        {
            txn.Execute("INSERT INTO Person ?", ionPerson);
        });

        Console.WriteLine("Querying the table");

        IIonValue ionFirstName = valueFactory.NewString("John");

        // The result from driver.Execute() is buffered into memory because once
the
        // transaction is committed, streaming the result is no longer possible.
        IResult selectResult = driver.Execute(txn =>
        {
            return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName);
        });

        foreach (IIonValue row in selectResult)
        {
            Console.WriteLine(row.GetField("firstName").StringValue);
            Console.WriteLine(row.GetField("lastName").StringValue);
            Console.WriteLine(row.GetField("age").IntValue);
        }

        Console.WriteLine("Updating a document");

        IIonValue ionIntAge = valueFactory.NewInt(42);
        IIonValue ionFirstName2 = valueFactory.NewString("John");

        driver.Execute(txn =>
        {
            txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
ionIntAge, ionFirstName2);
        });

        Console.WriteLine("Querying the table for the updated document");

        IIonValue ionFirstName3 = valueFactory.NewString("John");

        IResult updateResult = driver.Execute(txn =>
        {
            return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName3);
        });

```

```
        foreach (IIonValue row in updateResult)
        {
            Console.WriteLine(row.GetField("firstName").StringValue);
            Console.WriteLine(row.GetField("lastName").StringValue);
            Console.WriteLine(row.GetField("age").IntValue);
        }
    }
}
```

To run the complete application, enter the following command from the parent directory of your `Amazon.QLDB.QuickStartGuide` project directory.

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

Amazon QLDB driver for .NET – Cookbook reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This reference guide shows common use cases of the Amazon QLDB driver for .NET. It provides C# code examples that demonstrate how to use the driver to run basic *create, read, update, and delete* (CRUD) operations. It also includes code examples for processing Amazon Ion data. In addition, this guide highlights best practices for making transactions idempotent and implementing uniqueness constraints.

Note

This topic provides code examples of processing Amazon Ion data using the [Ion object mapper](#) by default. QLDB introduced the Ion object mapper in version 1.3.0 of the .NET driver. Where applicable, this topic also provides code examples using the standard [Ion library](#) as an alternative. To learn more, see [Working with Amazon Ion](#).

Contents

- [Importing the driver](#)
- [Instantiating the driver](#)
- [CRUD operations](#)
 - [Creating tables](#)
 - [Creating indexes](#)
 - [Reading documents](#)
 - [Using query parameters](#)
 - [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
 - [Updating documents](#)
 - [Deleting documents](#)
 - [Running multiple statements in a transaction](#)
 - [Retry logic](#)
 - [Implementing uniqueness constraints](#)
- [Working with Amazon Ion](#)
 - [Importing the Ion module](#)
 - [Creating Ion types](#)
 - [Getting an Ion binary dump](#)
 - [Getting an Ion text dump](#)

Importing the driver

The following code example imports the driver.

```
using Amazon.QLDB.Driver;  
using Amazon.QLDB.Driver.Generic;  
using Amazon.QLDB.Driver.Serialization;
```

Using the Ion library

```
using Amazon.QLDB.Driver;  
using Amazon.IonDotnet.Builders;
```

Instantiating the driver

The following code example creates an instance of the driver that connects to a specified ledger name using default settings.

Async

```
IAsyncQldbDriver driver = AsyncQldbDriver.Builder()  
    .WithLedger("vehicle-registration")  
    // Add Serialization library  
    .WithSerializer(new ObjectSerializer())  
    .Build();
```

Sync

```
IQldbDriver driver = QldbDriver.Builder()  
    .WithLedger("vehicle-registration")  
    // Add Serialization library  
    .WithSerializer(new ObjectSerializer())  
    .Build();
```

Using the Ion library

Async

```
IAsyncQldbDriver driver = AsyncQldbDriver.Builder().WithLedger("vehicle-  
registration").Build();
```

Sync

```
IQldbDriver driver = QldbDriver.Builder().WithLedger("vehicle-  
registration").Build();
```

CRUD operations

QLDB runs *create*, *read*, *update*, and *delete* (CRUD) operations as part of a transaction.

⚠ Warning

As a best practice, make your write transactions strictly idempotent.

Making transactions idempotent

We recommend that you make write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can run multiple times and produce identical results each time.

For example, consider a transaction that inserts a document into a table named `Person`. The transaction should first check whether or not the document already exists in the table. Without this check, the table might end up with duplicate documents.

Suppose that QLDB successfully commits the transaction on the server side, but the client times out while waiting for a response. If the transaction isn't idempotent, the same document could be inserted more than once in the case of a retry.

Using indexes to avoid full table scans

We also recommend that you run statements with a `WHERE` predicate clause using an *equality* operator on an indexed field or a document ID; for example, `WHERE indexedField = 123` or `WHERE indexedField IN (456, 789)`. Without this indexed lookup, QLDB needs to do a table scan, which can lead to transaction timeouts or *optimistic concurrency control* (OCC) conflicts.

For more information about OCC, see [Amazon QLDB concurrency model](#).

Implicitly created transactions

The [Amazon.QLDB.Driver.IQldbDriver.Execute](#) method accepts a lambda function that receives an instance of [Amazon.QLDB.Driver.TransactionExecutor](#), which you can use to run statements. The instance of `TransactionExecutor` wraps an implicitly created transaction.

You can run statements within the lambda function by using the `Execute` method of the transaction executor. The driver implicitly commits the transaction when the lambda function returns.

The following sections show how to run basic CRUD operations, specify custom retry logic, and implement uniqueness constraints.

Contents

- [Creating tables](#)
- [Creating indexes](#)
- [Reading documents](#)
 - [Using query parameters](#)
- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)
- [Running multiple statements in a transaction](#)
- [Retry logic](#)
- [Implementing uniqueness constraints](#)

Creating tables

Async

```
IAsyncResult<Table> createResult = await driver.Execute(async txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE TABLE Person");
    return await txn.Execute(query);
});

await foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the created table ID:
    // { tableId: 4o5Uk090cjC6PpJpLahceE }
}
```

Sync

```
IResult<Table> createResult = driver.Execute( txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE TABLE Person");
    return txn.Execute(query);
});

foreach (var result in createResult)
```

```
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the created table ID:
    // { tableId: 4o5Uk090cjC6PpJpLahceE }
}
```

Using the Ion library

Async

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("CREATE TABLE Person");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created table ID:
    // {
    //     tableId: "4o5Uk090cjC6PpJpLahceE"
    // }
}
```

Sync

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult result = driver.Execute(txn =>
{
    return txn.Execute("CREATE TABLE Person");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created table ID:
    // {
    //     tableId: "4o5Uk090cjC6PpJpLahceE"
    // }
}
```

```
}
```

Creating indexes

Async

```
IAsyncResult<Table> createResult = await driver.Execute(async txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE INDEX ON Person(firstName)");
    return await txn.Execute(query);
});

await foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the updated table ID:
    // { tableId: 4o5Uk090cjC6PpJpLahceE }
}
```

Sync

```
IResult<Table> createResult = driver.Execute(txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE INDEX ON Person(firstName)");
    return txn.Execute(query);
});

foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the updated table ID:
    // { tableId: 4o5Uk090cjC6PpJpLahceE }
}
```

Using the Ion library

Async

```
IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("CREATE INDEX ON Person(GovId)");
});
```

```
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated table ID:
    // {
    //     tableId: "4o5Uk090cjC6PpJpLahceE"
    // }
}
```

Sync

```
IResult result = driver.Execute(txn =>
{
    return txn.Execute("CREATE INDEX ON Person(GovId)");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated table ID:
    // {
    //     tableId: "4o5Uk090cjC6PpJpLahceE"
    // }
}
```

Reading documents

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// Person class is defined as follows:
// public class Person
// {
//     public string GovId { get; set; }
//     public string FirstName { get; set; }
// }

IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId =
'TOYENC486FH'"));
});
```

```
});

await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

Note

When you run a query without an indexed lookup, it invokes a full table scan. In this example, we recommend having an [index](#) on the GovId field to optimize performance. Without an index on GovId, queries can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Using query parameters

The following code example uses a C# type query parameter.

```
IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE FirstName
= ?", "Brent"));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

The following code example uses multiple C# type query parameters.

```
IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId = ?
AND FirstName = ?", "TOYENC486FH", "Brent"));
});

await foreach (Person person in result)
{
```

```
Console.WriteLine(person.GovId); // Prints TOYENC486FH.  
Console.WriteLine(person.FirstName); // Prints Brent.  
}
```

The following code example uses an array of C# type query parameters.

```
// Assumes that Person table has documents as follows:  
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }  
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }  
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }  
  
string[] ids = {  
    "TOYENC486FH",  
    "ROEE1C1AABH",  
    "YH844DA7LDB"  
};  
  
IAsyncResult<Person> result = await driver.Execute(async txn =>  
{  
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId IN  
    (?, ?, ?)", ids));  
});  
  
await foreach (Person person in result)  
{  
    Console.WriteLine(person.FirstName); // Prints Brent on first iteration.  
    // Prints Jim on second iteration.  
    // Prints Mary on third iteration.  
}
```

The following code example uses a C# list as a value.

```
// Assumes that Person table has document as follows:  
// { "GovId": "TOYENC486FH",  
//   "FirstName" : "Brent",  
//   "Vehicles": [  
//       { "Make": "Volkswagen",  
//         "Model": "Golf"},  
//       { "Make": "Honda",  
//         "Model": "Civic"}  
//   ]  
// }  
// Person class is defined as follows:
```

```
// public class Person
// {
//     public string GovId { get; set; }
//     public string FirstName { get; set; }
//     public List<Vehicle> Vehicles { get; set; }
// }
// Vehicle class is defined as follows:
// public class Vehicle
// {
//     public string Make { get; set; }
//     public string Model { get; set; }
// }

List<Vehicle> vehicles = new List<Vehicle>
{
    new Vehicle
    {
        Make = "Volkswagen",
        Model = "Golf"
    },
    new Vehicle
    {
        Make = "Honda",
        Model = "Civic"
    }
};

IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE Vehicles
= ?", vehicles));
});

await foreach (Person person in result)
{
    Console.WriteLine("{}");
    Console.WriteLine($" GovId: {person.GovId},");
    Console.WriteLine($" FirstName: {person.FirstName},");
    Console.WriteLine($" Vehicles: [");
    foreach (Vehicle vehicle in person.Vehicles)
    {
        Console.WriteLine("  {}");
        Console.WriteLine($"    Make: {vehicle.Make},");
        Console.WriteLine($"    Model: {vehicle.Model},");
    }
}
```

```

        Console.WriteLine("  },");
    }
    Console.WriteLine("  ]");
    Console.WriteLine("}");
    // Prints:
    // {
    //     GovId: TOYENC486FH,
    //     FirstName: Brent,
    //     Vehicles: [
    //     {
    //         Make: Volkswagen,
    //         Model: Golf
    //     },
    //     {
    //         Make: Honda,
    //         Model: Civic
    //     },
    //     ]
    // }
}

```

Using the Ion library

Async

```

// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}

```

Sync

```

// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }

```



```
IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Note

When you run a query without an indexed lookup, it invokes a full table scan. In this example, we recommend having an [index](#) on the GovId field to optimize performance. Without an index on GovId, queries can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

The following code example uses an Ion type query parameter.

Async

```
IValueFactory valueFactory = new ValueFactory();
IIonValue ionFirstName = valueFactory.NewString("Brent");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE FirstName = ?",
        ionFirstName);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
IValueFactory valueFactory = new ValueFactory();
IIonValue ionFirstName = valueFactory.NewString("Brent");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE FirstName = ?", ionFirstName);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

The following code example uses multiple query parameters.

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("Brent");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?", ionGovId, ionFirstName);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("Brent");

IResult result = driver.Execute(txn =>
{
```

```

        return txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?",
            ionGovId, ionFirstName);
    });

    foreach (IIonValue row in result)
    {
        Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
        Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
    }

```

The following code example uses a list of query parameters.

Async

```

// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }

IIonValue[] ionIds = {
    valueFactory.NewString("TOYENC486FH"),
    valueFactory.NewString("ROEE1C1AABH"),
    valueFactory.NewString("YH844DA7LDB")
};

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId IN (?, ?, ?)", ionIds);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent on
    first iteration.
                                                                // Prints Jim on
    second iteration.
                                                                // Prints Mary on
    third iteration.
}

```

Sync

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "R0EE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }

IIonValue[] ionIds = {
    valueFactory.NewString("TOYENC486FH"),
    valueFactory.NewString("R0EE1C1AABH"),
    valueFactory.NewString("YH844DA7LDB")
};

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId IN (?, ?, ?)", ionIds);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent on
    first iteration.
                                                                // Prints Jim on
    second iteration.
                                                                // Prints Mary on
    third iteration.
}
```

The following code example uses an Ion list as a value. To learn more about using different Ion types, see [Working with Amazon Ion data types in Amazon QLDB](#).

Async

```
// Assumes that Person table has document as follows:
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//     { "Make": "Volkswagen",
//       "Model": "Golf"},
//     { "Make": "Honda",
//       "Model": "Civic"}
//   ]
// }
```

```
// }

IIonValue ionVehicle1 = valueFactory.NewEmptyStruct();
ionVehicle1.SetField("Make", valueFactory.NewString("Volkswagen"));
ionVehicle1.SetField("Model", valueFactory.NewString("Golf"));

IIonValue ionVehicle2 = valueFactory.NewEmptyStruct();
ionVehicle2.SetField("Make", valueFactory.NewString("Honda"));
ionVehicle2.SetField("Model", valueFactory.NewString("Civic"));

IIonValue ionVehicles = valueFactory.NewEmptyList();
ionVehicles.Add(ionVehicle1);
ionVehicles.Add(ionVehicle2);

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE Vehicles = ?",
        ionVehicles);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // Prints:
    // {
    //     GovId: "TOYENC486FN",
    //     FirstName: "Brent",
    //     Vehicles: [
    //         {
    //             Make: "Volkswagen",
    //             Model: "Golf"
    //         },
    //         {
    //             Make: "Honda",
    //             Model: "Civic"
    //         }
    //     ]
    // }
}
```

Sync

```
// Assumes that Person table has document as follows:
```

```
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//     { "Make": "Volkswagen",
//       "Model": "Golf"},
//     { "Make": "Honda",
//       "Model": "Civic"}
//   ]
// }

IIonValue ionVehicle1 = valueFactory.NewEmptyStruct();
ionVehicle1.SetField("Make", valueFactory.NewString("Volkswagen"));
ionVehicle1.SetField("Model", valueFactory.NewString("Golf"));

IIonValue ionVehicle2 = valueFactory.NewEmptyStruct();
ionVehicle2.SetField("Make", valueFactory.NewString("Honda"));
ionVehicle2.SetField("Model", valueFactory.NewString("Civic"));

IIonValue ionVehicles = valueFactory.NewEmptyList();
ionVehicles.Add(ionVehicle1);
ionVehicles.Add(ionVehicle2);

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE Vehicles = ?", ionVehicles);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // Prints:
    // {
    //   GovId: "TOYENC486FN",
    //   FirstName: "Brent",
    //   Vehicles: [
    //     {
    //       Make: "Volkswagen",
    //       Model: "Golf"
    //     },
    //     {
    //       Make: "Honda",
    //       Model: "Civic"
    //     }
    //   ]
    // }
```

```
// }  
}
```

Inserting documents

The following code example inserts Ion data types.

```
string govId = "TOYENC486FH";  
  
Person person = new Person  
{  
    GovId = "TOYENC486FH",  
    FirstName = "Brent"  
};  
  
await driver.Execute(async txn =>  
{  
    // Check if a document with GovId:TOYENC486FH exists  
    // This is critical to make this transaction idempotent  
    IAsyncResult<Person> result = await txn.Execute(txn.Query<Person>("SELECT * FROM  
Person WHERE GovId = ?", govId));  
  
    // Check if there is a record in the cursor.  
    int count = await result.CountAsync();  
    if (count > 0)  
    {  
        // Document already exists, no need to insert  
        return;  
    }  
  
    // Insert the document.  
    await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", person));  
});
```

Using the Ion library

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");  
  
IIonValue ionPerson = valueFactory.NewEmptyStruct();  
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
```

```

ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult result = await txn.Execute("SELECT * FROM Person WHERE GovId = ?",
ionGovId);

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});

```

Sync

```

IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

driver.Execute(txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IResult result = txn.Execute("SELECT * FROM Person WHERE GovId = ?", ionGovId);

    // Check if there is a record in the cursor.
    int count = result.Count();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }
}

```



```
// Insert the document.  
txn.Execute("INSERT INTO Person ?", ionPerson);  
});
```

This transaction inserts a document into the `Person` table. Before inserting, it first checks if the document already exists in the table. **This check makes the transaction idempotent in nature.** Even if you run this transaction multiple times, it won't cause any unintended side effects.

Note

In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting multiple documents in one statement

To insert multiple documents by using a single [INSERT](#) statement, you can pass a C# `List` parameter to the statement as follows.

```
Person person1 = new Person  
{  
    FirstName = "Brent",  
    GovId = "TOYENC486FH"  
};  
  
Person person2 = new Person  
{  
    FirstName = "Jim",  
    GovId = "R0EE1C1AABH"  
};  
  
List<Person> people = new List<Person>();  
people.Add(person1);  
people.Add(person2);  
  
IAsyncResult<Document> result = await driver.Execute(async txn =>  
{  
    return await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", people));  
});
```

```
await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the created documents' ID:
    // { documentId: 6BFt5eJQDFLBW2aR8LPw42 }
    // { documentId: K5Zrcb6N3gmIEHgGhwoyKF }
}
```

Using the Ion library

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [Ion list](#) to the statement as follows.

Async

```
IIonValue ionPerson1 = valueFactory.NewEmptyStruct();
ionPerson1.SetField("FirstName", valueFactory.NewString("Brent"));
ionPerson1.SetField("GovId", valueFactory.NewString("TOYENC486FH"));

IIonValue ionPerson2 = valueFactory.NewEmptyStruct();
ionPerson2.SetField("FirstName", valueFactory.NewString("Jim"));
ionPerson2.SetField("GovId", valueFactory.NewString("ROEE1C1AABH"));

IIonValue ionPeople = valueFactory.NewEmptyList();
ionPeople.Add(ionPerson1);
ionPeople.Add(ionPerson2);

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("INSERT INTO Person ?", ionPeople);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created documents' ID:
    // {
    //     documentId: "6BFt5eJQDFLBW2aR8LPw42"
    // }
    //
    // {
    //     documentId: "K5Zrcb6N3gmIEHgGhwoyKF"
    // }
```

```
}

```

Sync

```

IIonValue ionPerson1 = valueFactory.NewEmptyStruct();
ionPerson1.SetField("FirstName", valueFactory.NewString("Brent"));
ionPerson1.SetField("GovId", valueFactory.NewString("TOYENC486FH"));

IIonValue ionPerson2 = valueFactory.NewEmptyStruct();
ionPerson2.SetField("FirstName", valueFactory.NewString("Jim"));
ionPerson2.SetField("GovId", valueFactory.NewString("ROEE1C1AABH"));

IIonValue ionPeople = valueFactory.NewEmptyList();
ionPeople.Add(ionPerson1);
ionPeople.Add(ionPerson2);

IResult result = driver.Execute(txn =>
{
    return txn.Execute("INSERT INTO Person ?", ionPeople);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created documents' ID:
    // {
    //     documentId: "6BFt5eJQDFLBW2aR8LPw42"
    // }
    //
    // {
    //     documentId: "K5Zrcb6N3gmIEHgGhwoyKF"
    // }
}

```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . . >>) when passing an Ion list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Updating documents

```

string govId = "TOYENC486FH";
string firstName = "John";

```

```

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("UPDATE Person SET FirstName = ? WHERE
GovId = ?", firstName , govId));
});

await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the updated document ID:
    // { documentId: Djg30Zoltqy5M4BFsA2jSJ }
}

```

Using the Ion library

Async

```

IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("John");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?",
ionFirstName , ionGovId);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}

```

Sync

```

IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("John");

IResult result = driver.Execute(txn =>
{

```

```

        return txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?",
            ionFirstName , ionGovId);
    });

    foreach (IIonValue row in result)
    {
        Console.WriteLine(row.ToPrettyString());
        // The statement returns the updated document ID:
        // {
        //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
        // }
    }
}

```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Deleting documents

```

string govId = "TOYENC486FH";

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("DELETE FROM Person WHERE GovId = ?",
        govId));
});

await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the updated document ID:
    // { documentId: Djg30Zoltqy5M4BFsA2jSJ }
}

```

Using the Ion library

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("DELETE FROM Person WHERE GovId = ?", ionGovId);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the deleted document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("DELETE FROM Person WHERE GovId = ?", ionGovId);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the deleted document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Running multiple statements in a transaction

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
// you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
// not.
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.Generic.IAsyncResult<Vehicle> result = await txn.Execute(
            txn.Query<Vehicle>("SELECT insured FROM Vehicles WHERE vin = ? AND insured
= FALSE", vin));

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                txn.Query<Document>("UPDATE Vehicles SET insured = TRUE WHERE vin = ?",
vin));
            return true;
        }
        return false;
    });
}
```

Using the Ion library

Async

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
// because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
// or not.
```

```
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    ValueFactory valueFactory = new ValueFactory();
    IIonValue ionVin = valueFactory.NewString(vin);

    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.IAsyncResult result = await txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
            ionVin);

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

Retry logic

For information about the driver's built-in retry logic, see [Understanding retry policy with the driver in Amazon QLDB](#).

Implementing uniqueness constraints

QLDB doesn't support unique indexes, but you can implement this behavior in your application.

Suppose that you want to implement a uniqueness constraint on the GovId field in the Person table. To do this, you can write a transaction that does the following:

1. Assert that the table has no existing documents with a specified GovId.
2. Insert the document if the assertion passes.

If a competing transaction concurrently passes the assertion, only one of the transactions will commit successfully. The other transaction will fail with an OCC conflict exception.

The following code example shows how to implement this uniqueness constraint logic.

```
string govId = "TOYENC486FH";

Person person = new Person
{
    GovId = "TOYENC486FH",
    FirstName = "Brent"
};

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult<Person> result = await txn.Execute(txn.Query<Person>("SELECT * FROM
Person WHERE GovId = ?", govId));

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", person));
});
```

Using the Ion library

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
```

```

    IAsyncResult result = await txn.Execute("SELECT * FROM Person WHERE GovId = ?",
ionGovId);

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});

```

Sync

```

IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

driver.Execute(txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IResult result = txn.Execute("SELECT * FROM Person WHERE GovId = ?", ionGovId);

    // Check if there is a record in the cursor.
    int count = result.Count();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    txn.Execute("INSERT INTO Person ?", ionPerson);
});

```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Working with Amazon Ion

There are multiple ways to process Amazon Ion data in QLDB. You can use the [Ion library](#) to create and modify Ion values. Or, you can use the [Ion object mapper](#) to map C# *plain old CLR objects* (POCO) to and from Ion values. Version 1.3.0 of the QLDB driver for .NET introduces support for the Ion object mapper.

The following sections provide code examples of processing Ion data using both techniques.

Contents

- [Importing the Ion module](#)
- [Creating Ion types](#)
- [Getting an Ion binary dump](#)
- [Getting an Ion text dump](#)

Importing the Ion module

```
using Amazon.IonObjectMapper;
```

Using the Ion library

```
using Amazon.IonDotnet.Builders;
```

Creating Ion types

The following code example shows how to create Ion values from C# objects using the Ion object mapper.

```
// Assumes that Person class is defined as follows:
```

```
// public class Person
// {
//     public string FirstName { get; set; }
//     public int Age { get; set; }
// }

// Initialize the Ion Object Mapper
IonSerializer ionSerializer = new IonSerializer();

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
    Age = 13
};

// Serialize the C# object into stream using the Ion Object Mapper
Stream stream = ionSerializer.Serialize(person);

// Load will take in stream and return a datagram; a top level container of Ion values.
IIonValue ionDatagram = IonLoader.Default.Load(stream);

// To get the Ion value within the datagram, we call GetElementAt(0).
IIonValue ionPerson = ionDatagram.GetElementAt(0);

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

Using the Ion library

The following code examples show the two ways to create Ion values using the Ion library.

Using ValueFactory

```
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;

IValueFactory valueFactory = new ValueFactory();

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("age", valueFactory.NewInt(13));
```

```
Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

Using IonLoader

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;

// Load will take in Ion text and return a datagram; a top level container of Ion
// values.
IIonValue ionDatagram = IonLoader.Default.Load("{firstName: \"John\", age: 13}");

// To get the Ion value within the datagram, we call GetElementAt(0).
IIonValue ionPerson = ionDatagram.GetElementAt(0);

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

Getting an Ion binary dump

```
// Initialize the Ion Object Mapper with Ion binary serialization format
IonSerializer ionSerializer = new IonSerializer(new IonSerializationOptions
{
    Format = IonSerializationFormat.BINARY
});

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
    Age = 13
};

MemoryStream stream = (MemoryStream) ionSerializer.Serialize(person);
Console.WriteLine(BitConverter.ToString(stream.ToArray()));
```

Using the Ion library

```
// ionObject is an Ion struct
MemoryStream stream = new MemoryStream();
using (var writer = IonBinaryWriterBuilder.Build(stream))
{
```

```
ionObject.WriteTo(writer);
writer.Finish();
}

Console.WriteLine(BitConverter.ToString(stream.ToArray()));
```

Getting an Ion text dump

```
// Initialize the Ion Object Mapper
IonSerializer ionSerializer = new IonSerializer(new IonSerializationOptions
{
    Format = IonSerializationFormat.TEXT
});

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
    Age = 13
};

MemoryStream stream = (MemoryStream) ionSerializer.Serialize(person);
Console.WriteLine(System.Text.Encoding.UTF8.GetString(stream.ToArray()));
```

Using the Ion library

```
// ionObject is an Ion struct
StringWriter sw = new StringWriter();
using (var writer = IonTextWriterBuilder.Build(sw))
{
    ionObject.WriteTo(writer);
    writer.Finish();
}

Console.WriteLine(sw.ToString());
```

For more information about working with Ion, see the [Amazon Ion documentation](#) on GitHub. For more code examples of working with Ion in QLDB, see [Working with Amazon Ion data types in Amazon QLDB](#).

Amazon QLDB driver for Go

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

To work with data in your ledger, you can connect to Amazon QLDB from your Go application by using an AWS provided driver. The following topics describe how to get started with the QLDB driver for Go.

Topics

- [Driver resources](#)
- [Prerequisites](#)
- [Installation](#)
- [Amazon QLDB driver for Go – Quick start tutorial](#)
- [Amazon QLDB driver for Go – Cookbook reference](#)

Driver resources

For more information about the functionality supported by the Go driver, see the following resources:

- API reference: [3.x](#), [2.x](#), [1.x](#)
- [Driver source code \(GitHub\)](#)
- [Amazon Ion Cookbook](#)

Prerequisites

Before you get started with the QLDB driver for Go, you must do the following:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#). This includes the following:
 1. Sign up for AWS.

2. Create a user with the appropriate QLDB permissions.
3. Grant programmatic access for development.
2. (Optional) Install an integrated development environment (IDE) of your choice. For a list of commonly used IDEs for Go, see [Editor plugins and IDEs](#) on the Go website.
3. Download and install one of the following versions of Go from the [Go downloads](#) site:
 - **1.15 or later** – QLDB driver for Go v3
 - **1.14** – QLDB driver for Go v1 or v2
4. Configure your development environment for the [AWS SDK for Go](#):

1. Set up your AWS credentials. We recommend creating a shared credentials file.

For instructions, see [Specifying Credentials](#) in the *AWS SDK for Go Developer Guide*.

2. Set your default AWS Region. To learn how, see [Specifying the AWS Region](#).

For a complete list of available Regions, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

Next, you can set up a basic sample application and run short code examples—or you can install the driver in an existing Go project.

- To install the QLDB driver and the AWS SDK for Go in an existing project, proceed to [Installation](#).
- To set up a project and run short code examples that demonstrate basic data transactions on a ledger, see the [Quick start tutorial](#).

Installation

The QLDB driver for Go is open source in the GitHub repository [awslabs/amazon-qldb-driver-go](#). QLDB supports the following driver versions and their Go dependencies.

Driver version	Go version	Status	Latest release date
1.x	1.14 or later	Production release	June 16, 2021
2.x	1.14 or later	Production release	July 21, 2021
3.x	1.15 or later	Production release	November 10, 2022

To install the driver

1. Ensure that your project is using [Go modules](#) to install project dependencies.
2. In your project directory, enter the following `go get` command.

3.x

```
$ go get -u github.com/awslabs/amazon-qlldb-driver-go/v3/qlldbdriver
```

2.x

```
$ go get -u github.com/awslabs/amazon-qlldb-driver-go/v2/qlldbdriver
```

Installing the driver also installs its dependencies, including the [AWS SDK for Go](#) or [AWS SDK for Go v2](#), and [Amazon Ion](#) packages.

For short code examples of how to run basic data transactions on a ledger, see the [Cookbook reference](#).

Amazon QLDB driver for Go – Quick start tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this tutorial, you learn how to set up a simple application using the latest version of the Amazon QLDB driver for Go. This guide includes steps for installing the driver and short code examples of basic *create*, *read*, *update*, and *delete* (CRUD) operations.

Topics

- [Prerequisites](#)
- [Step 1: Install the driver](#)
- [Step 2: Import the packages](#)
- [Step 3: Initialize the driver](#)

- [Step 4: Create a table and index](#)
- [Step 5: Insert a document](#)
- [Step 6: Query the document](#)
- [Step 7: Update the document](#)
- [Step 8: Query the updated document](#)
- [Step 9: Drop the table](#)
- [Running the complete application](#)

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the [Prerequisites](#) for the Go driver, if you haven't already done so. This includes signing up for AWS, granting programmatic access for development, and installing Go.
2. Create a ledger named `quick-start`.

To learn how to create a ledger, see [Basic operations for Amazon QLDB ledgers](#) or [Step 1: Create a new ledger](#) in *Getting started with the console*.

Step 1: Install the driver

Ensure that your project is using [Go modules](#) to install project dependencies.

In your project directory, enter the following `go get` command.

```
$ go get -u github.com/awslabs/amazon-qldb-driver-go/v3/qlbdbdriver
```

Installing the driver also installs its dependencies, including the [AWS SDK for Go v2](#) and [Amazon Ion](#) packages.

Step 2: Import the packages

Import the following AWS packages.

```
import (  
    "context"  
    "fmt"
```

```

"github.com/amzn/ion-go/ion"
"github.com/aws/aws-sdk-go/aws"
"github.com/aws/aws-sdk-go-v2/config"
"github.com/aws/aws-sdk-go-v2/service/qlldbSession"
"github.com/aws-labs/amazon-qlldb-driver-go/v3/qlldbdriver"
)

```

Step 3: Initialize the driver

Initialize an instance of the driver that connects to the ledger named `quick-start`.

```

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}

qlldbSession := qlldbSession.NewFromConfig(cfg, func(options *qlldbSession.Options) {
    options.Region = "us-east-1"
})
driver, err := qlldbdriver.New(
    "quick-start",
    qlldbSession,
    func(options *qlldbdriver.DriverOptions) {
        options.LoggerVerbosity = qlldbdriver.LogInfo
    })
if err != nil {
    panic(err)
}

defer driver.Shutdown(context.Background())

```

Note

In this code example, replace `us-east-1` with the AWS Region where you created your ledger.

Step 4: Create a table and index

The following code example shows how to run `CREATE TABLE` and `CREATE INDEX` statements.

```
_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}), error) {
    _, err := txn.Execute("CREATE TABLE People")
    if err != nil {
        return nil, err
    }

    // When working with QLDB, it's recommended to create an index on fields we're
    // filtering on.
    // This reduces the chance of OCC conflict exceptions with large datasets.
    _, err = txn.Execute("CREATE INDEX ON People (firstName)")
    if err != nil {
        return nil, err
    }

    _, err = txn.Execute("CREATE INDEX ON People (age)")
    if err != nil {
        return nil, err
    }

    return nil, nil
})
if err != nil {
    panic(err)
}
```

This code creates a table named `People`, and indexes for the `firstName` and `age` fields on that table. [Indexes](#) are required to optimize query performance and help to limit [optimistic concurrency control \(OCC\)](#) conflict exceptions.

Step 5: Insert a document

The following code examples show how to run an INSERT statement. QLDB supports the [PartiQL](#) query language (SQL compatible) and the [Amazon Ion](#) data format (superset of JSON).

Using literal PartiQL

The following code inserts a document into the `People` table using a string literal PartiQL statement.

```
_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}), error) {
```

```
    return txn.Execute("INSERT INTO People {'firstName': 'Jane', 'lastName': 'Doe',
'age': 77}")
  })
  if err != nil {
    panic(err)
  }
```

Using Ion data types

Similar to Go's built-in [JSON package](#), you can marshal and unmarshal Go data types to and from Ion.

1. Suppose that you have the following Go structure named Person.

```
type Person struct {
    FirstName string `ion:"firstName"`
    LastName  string `ion:"lastName"`
    Age       int    `ion:"age"`
}
```

2. Create an instance of Person.

```
person := Person{"John", "Doe", 54}
```

The driver marshals an Ion-encoded text representation of person for you.

Important

For marshal and unmarshal to work properly, the field names of the Go data structure must be exported (first letter capitalized).

3. Pass the person instance to the transaction's Execute method.

```
_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("INSERT INTO People ?", person)
})
if err != nil {
    panic(err)
}
```

This example uses a question mark (?) as a variable placeholder to pass the document information to the statement. When you use placeholders, you must pass an Ion-encoded text value.

Tip

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [list](#) to the statement as follows.

```
// people is a list
txn.Execute("INSERT INTO People ?", people)
```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . . >>) when passing a list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Step 6: Query the document

The following code example shows how to run a SELECT statement.

```
p, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE age =
54")
    if err != nil {
        return nil, err
    }

    // Assume the result is not empty
    hasNext := result.Next(txn)
    if !hasNext && result.Err() != nil {
        return nil, result.Err()
    }

    ionBinary := result.GetCurrentData()

    temp := new(Person)
    err = ion.Unmarshal(ionBinary, temp)
    if err != nil {
```

```

        return nil, err
    }

    return *temp, nil
})
if err != nil {
    panic(err)
}

var returnedPerson Person
returnedPerson = p.(Person)

if returnedPerson != person {
    fmt.Print("Queried result does not match inserted struct")
}

```

This example queries your document from the `People` table, assumes that the result set isn't empty, and returns your document from the result.

Step 7: Update the document

The following code example shows how to run an `UPDATE` statement.

```

person.Age += 10

_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("UPDATE People SET age = ? WHERE firstName = ?", person.Age,
person.FirstName)
})
if err != nil {
    panic(err)
}

```

Step 8: Query the updated document

The following code example queries the `People` table by `firstName` and returns all of the documents in the result set.

```

p, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
firstName = ?", person.FirstName)

```

```

    if err != nil {
        return nil, err
    }

    var people []Person
    for result.Next(txn) {
        ionBinary := result.GetCurrentData()

        temp := new(Person)
        err = ion.Unmarshal(ionBinary, temp)
        if err != nil {
            return nil, err
        }

        people = append(people, *temp)
    }
    if result.Err() != nil {
        return nil, result.Err()
    }

    return people, nil
}))
if err != nil {
    panic(err)
}

var people []Person
people = p.([]Person)

updatedPerson := Person{"John", "Doe", 64}
if people[0] != updatedPerson {
    fmt.Print("Queried result does not match updated struct")
}

```

Step 9: Drop the table

The following code example shows how to run a DROP TABLE statement.

```

_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("DROP TABLE People")
}))
if err != nil {

```



```
    panic(err)
}
```

Running the complete application

The following code example is the complete version of the application. Instead of doing the previous steps individually, you can also copy and run this code example from start to end. This application demonstrates some basic CRUD operations on the ledger named quick-start.

Note

Before you run this code, make sure that you don't already have an active table named People in the quick-start ledger.

```
package main

import (
    "context"
    "fmt"

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qlldb-driver-go/v2/qlddbdriver"
)

func main() {
    awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
    qlldbSession := qlldbsession.New(awsSession)

    driver, err := qlddbdriver.New(
        "quick-start",
        qlldbSession,
        func(options *qlddbdriver.DriverOptions) {
            options.LoggerVerbosity = qlddbdriver.LogInfo
        })
    if err != nil {
        panic(err)
    }
}
```

```

defer driver.Shutdown(context.Background())

_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}), error) {
    _, err := txn.Execute("CREATE TABLE People")
    if err != nil {
        return nil, err
    }

    // When working with QLDB, it's recommended to create an index on fields we're
    filtering on.
    // This reduces the chance of OCC conflict exceptions with large datasets.
    _, err = txn.Execute("CREATE INDEX ON People (firstName)")
    if err != nil {
        return nil, err
    }

    _, err = txn.Execute("CREATE INDEX ON People (age)")
    if err != nil {
        return nil, err
    }

    return nil, nil
})
if err != nil {
    panic(err)
}

_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}), error) {
    return txn.Execute("INSERT INTO People {'firstName': 'Jane', 'lastName': 'Doe',
'age': 77}")
})
if err != nil {
    panic(err)
}

type Person struct {
    FirstName string `ion:"firstName"`
    LastName  string `ion:"lastName"`
    Age       int    `ion:"age"`
}

person := Person{"John", "Doe", 54}

```

```

_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("INSERT INTO People ?", person)
})
if err != nil {
    panic(err)
}

p, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
age = 54")
    if err != nil {
        return nil, err
    }

    // Assume the result is not empty
    hasNext := result.Next(txn)
    if !hasNext && result.Err() != nil {
        return nil, result.Err()
    }

    ionBinary := result.GetCurrentData()

    temp := new(Person)
    err = ion.Unmarshal(ionBinary, temp)
    if err != nil {
        return nil, err
    }

    return *temp, nil
})
if err != nil {
    panic(err)
}

var returnedPerson Person
returnedPerson = p.(Person)

if returnedPerson != person {
    fmt.Print("Queried result does not match inserted struct")
}

```

```

    person.Age += 10

    _, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
        return txn.Execute("UPDATE People SET age = ? WHERE firstName = ?", person.Age,
person.FirstName)
    })
    if err != nil {
        panic(err)
    }

    p, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
        result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
firstName = ?", person.FirstName)
        if err != nil {
            return nil, err
        }

        var people []Person
        for result.Next(txn) {
            ionBinary := result.GetCurrentData()

            temp := new(Person)
            err = ion.Unmarshal(ionBinary, temp)
            if err != nil {
                return nil, err
            }

            people = append(people, *temp)
        }
        if result.Err() != nil {
            return nil, result.Err()
        }

        return people, nil
    })
    if err != nil {
        panic(err)
    }

    var people []Person
    people = p.([]Person)

```

```
updatedPerson := Person{"John", "Doe", 64}
if people[0] != updatedPerson {
    fmt.Print("Queried result does not match updated struct")
}

_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("DROP TABLE People")
})
if err != nil {
    panic(err)
}
}
```

Amazon QLDB driver for Go – Cookbook reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This reference guide shows common use cases of the Amazon QLDB driver for Go. It provides Go code examples that demonstrate how to use the driver to run basic *create, read, update, and delete* (CRUD) operations. It also includes code examples for processing Amazon Ion data. In addition, this guide highlights best practices for making transactions idempotent and implementing uniqueness constraints.

Note

Where applicable, some use cases have different code examples for each supported major version of the QLDB driver for Go.

Contents

- [Importing the driver](#)
- [Instantiating the driver](#)
- [CRUD operations](#)

- [Creating tables](#)
- [Creating indexes](#)
- [Reading documents](#)
 - [Using query parameters](#)
- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)
- [Running multiple statements in a transaction](#)
- [Retry logic](#)
- [Implementing uniqueness constraints](#)
- [Working with Amazon Ion](#)
 - [Importing the Ion module](#)
 - [Creating Ion types](#)
 - [Getting Ion binary](#)
 - [Getting Ion text](#)

Importing the driver

The following code example imports the driver and other required AWS packages.

3.x

```
import (  
  
    "github.com/amzn/ion-go/ion"  
    "github.com/aws/aws-sdk-go/aws"  
    "github.com/aws/aws-sdk-go-v2/config"  
    "github.com/aws/aws-sdk-go-v2/service/qldbSession"  
    "github.com/aws-labs/amazon-qlldb-driver-go/v3/qlbdbdriver"  
)
```

2.x

```
import (  
  

```

```
"github.com/amzn/ion-go/ion"  
"github.com/aws/aws-sdk-go/aws"  
"github.com/aws/aws-sdk-go/aws/session"  
"github.com/aws/aws-sdk-go/service/qldbsession"  
"github.com/awslabs/amazon-qlldb-driver-go/v2/qlbdbdriver"  
)
```

Note

This example also imports the Amazon Ion package (`amzn/ion-go/ion`). You need this package to process Ion data when running some data operations in this reference. To learn more, see [Working with Amazon Ion](#).

Instantiating the driver

The following code example creates an instance of the driver that connects to a specified ledger name in a specified AWS Region.

3.x

```
cfg, err := config.LoadDefaultConfig(context.TODO())  
if err != nil {  
    panic(err)  
}  
  
qlldbSession := qlldbSession.NewFromConfig(cfg, func(options *qlldbSession.Options) {  
    options.Region = "us-east-1"  
})  
driver, err := qlbdbDriver.New(  
    "vehicle-registration",  
    qlldbSession,  
    func(options *qlbdbDriver.DriverOptions) {  
        options.LoggerVerbosity = qlbdbDriver.LogInfo  
    })  
if err != nil {  
    panic(err)  
}  
  
defer driver.Shutdown(context.Background())
```

2.x

```
awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-  
east-1")))
qldbSession := qlldbSession.New(awsSession)

driver, err := qlldbdriver.New(  
    "vehicle-registration",  
    qldbSession,  
    func(options *qlldbdriver.DriverOptions) {  
        options.LoggerVerbosity = qlldbdriver.LogInfo  
    })  
if err != nil {  
    panic(err)  
}
```

CRUD operations

QLDB runs *create*, *read*, *update*, and *delete* (CRUD) operations as part of a transaction.

Warning

As a best practice, make your write transactions strictly idempotent.

Making transactions idempotent

We recommend that you make write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can run multiple times and produce identical results each time.

For example, consider a transaction that inserts a document into a table named `Person`. The transaction should first check whether or not the document already exists in the table. Without this check, the table might end up with duplicate documents.

Suppose that QLDB successfully commits the transaction on the server side, but the client times out while waiting for a response. If the transaction isn't idempotent, the same document could be inserted more than once in the case of a retry.

Using indexes to avoid full table scans

We also recommend that you run statements with a `WHERE` predicate clause using an *equality* operator on an indexed field or a document ID; for example, `WHERE indexedField = 123` or `WHERE indexedField IN (456, 789)`. Without this indexed lookup, QLDB needs to do a table scan, which can lead to transaction timeouts or *optimistic concurrency control* (OCC) conflicts.

For more information about OCC, see [Amazon QLDB concurrency model](#).

Implicitly created transactions

The [QLDBDriver.Execute](#) function accepts a lambda function that receives an instance of [Transaction](#), which you can use to run statements. The instance of `Transaction` wraps an implicitly created transaction.

You can run statements within the lambda function by using the `Transaction.Execute` function. The driver implicitly commits the transaction when the lambda function returns.

The following sections show how to run basic CRUD operations, specify custom retry logic, and implement uniqueness constraints.

Contents

- [Creating tables](#)
- [Creating indexes](#)
- [Reading documents](#)
 - [Using query parameters](#)
- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)
- [Running multiple statements in a transaction](#)
- [Retry logic](#)
- [Implementing uniqueness constraints](#)

Creating tables

```
result, err := driver.Execute(context.Background(), func(txn qlldriver.Transaction)
(interface{}, error) {
```

```
    return txn.Execute("CREATE TABLE Person")
  })
```

Creating indexes

```
result, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("CREATE INDEX ON Person(GovId)")
})
```

Reading documents

```
var decodedResult map[string]interface{}

// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName": "Brent" }
_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    result, err := txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'")
    if err != nil {
        return nil, err
    }
    for result.Next(txn) {
        ionBinary := result.GetCurrentData()
        err = ion.Unmarshal(ionBinary, &decodedResult)
        if err != nil {
            return nil, err
        }
        fmt.Println(decodedResult) // prints map[GovId: TOYENC486FH FirstName:Brent]
    }
    if result.Err() != nil {
        return nil, result.Err()
    }
    return nil, nil
})
if err != nil {
    panic(err)
}
```

Using query parameters

The following code example uses a native type query parameter.

```
result, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("SELECT * FROM Person WHERE GovId = ?", "TOYENC486FH")
})
if err != nil {
    panic(err)
}
```

The following code example uses multiple query parameters.

```
result, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?",
        "TOYENC486FH", "Brent")
})
if err != nil {
    panic(err)
}
```

The following code example uses a list of query parameters.

```
govIDs := []string{"TOYENC486FH", "R0EE1", "YH844"}

result, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("SELECT * FROM Person WHERE GovId IN (?, ?, ?)", govIDs...)
})
if err != nil {
    panic(err)
}
```

Note

When you run a query without an indexed lookup, it invokes a full table scan. In this example, we recommend having an [index](#) on the GovId field to optimize performance. Without an index on GovId, queries can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting documents

The following code example inserts native data types.

```
_, err = driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    // Check if a document with a GovId of TOYENC486FH exists
    // This is critical to make this transaction idempotent
    result, err := txn.Execute("SELECT * FROM Person WHERE GovId = ?", "TOYENC486FH")
    if err != nil {
        return nil, err
    }
    // Check if there are any results
    if result.Next(txn) {
        // Document already exists, no need to insert
    } else {
        person := map[string]interface{}{
            "GovId": "TOYENC486FH",
            "FirstName": "Brent",
        }
        _, err = txn.Execute("INSERT INTO Person ?", person)
        if err != nil {
            return nil, err
        }
    }
    return nil, nil
})
```

This transaction inserts a document into the Person table. Before inserting, it first checks if the document already exists in the table. **This check makes the transaction idempotent in nature.** Even if you run this transaction multiple times, it won't cause any unintended side effects.

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting multiple documents in one statement

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [list](#) to the statement as follows.

```
// people is a list
txn.Execute("INSERT INTO People ?", people)
```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . >>) when passing a list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Updating documents

The following code example uses native data types.

```
result, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", "John",
        "TOYENC486FH")
})
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Deleting documents

The following code example uses native data types.

```
result, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    return txn.Execute("DELETE FROM Person WHERE GovId = ?", "TOYENC486FH")
})
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Running multiple statements in a transaction

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
// you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
// not.
func InsureCar(driver *qldbdriver.QLDBDriver, vin string) (bool, error) {
    insured, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {

        result, err := txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
        if err != nil {
            return false, err
        }

        hasNext := result.Next(txn)
        if !hasNext && result.Err() != nil {
            return false, result.Err()
        }

        if hasNext {
            _, err = txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
            if err != nil {
                return false, err
            }
            return true, nil
        }
        return false, nil
    })
    if err != nil {
        panic(err)
    }
}
```

```
    return insured.(bool), err
}
```

Retry logic

The driver's `Execute` function has a built-in retry mechanism that retries the transaction if a retryable exception occurs (such as timeouts or OCC conflicts). The maximum number of retry attempts and the backoff strategy are configurable.

The default retry limit is 4, and the default backoff strategy is [ExponentialBackoffStrategy](#) with a base of 10 milliseconds. You can set the retry policy per driver instance and also per transaction by using an instance of [RetryPolicy](#).

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for an instance of the driver.

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qlldbsession"
    "github.com/awslabs/amazon-qlldb-driver-go/v2/qlddbdriver"
)

func main() {
    awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
    qlldbSession := qlldbsession.New(awsSession)

    // Configuring retry limit to 2
    retryPolicy := qlddbdriver.RetryPolicy{MaxRetryLimit: 2}

    driver, err := qlddbdriver.New("test-ledger", qlldbSession, func(options
*qlddbdriver.DriverOptions) {
        options.RetryPolicy = retryPolicy
    })
    if err != nil {
        panic(err)
    }

    // Configuring an exponential backoff strategy with base of 20 milliseconds
    retryPolicy = qlddbdriver.RetryPolicy{
        MaxRetryLimit: 2,
```

```

    Backoff: qlbdbdriver.ExponentialBackoffStrategy{SleepBase: 20, SleepCap: 4000,
    }}

    driver, err = qlbdbdriver.New("test-ledger", qlldbSession, func(options
    *qlbdbdriver.DriverOptions) {
        options.RetryPolicy = retryPolicy
    })
    if err != nil {
        panic(err)
    }
}

```

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for a particular anonymous function. The `SetRetryPolicy` function overrides the retry policy that is set for the driver instance.

```

import (
    "context"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qlldb-driver-go/v2/qlbdbdriver"
)

func main() {
    awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
    east-1")))
    qlldbSession := qlldbsession.New(awsSession)

    // Configuring retry limit to 2
    retryPolicy1 := qlbdbdriver.RetryPolicy{MaxRetryLimit: 2}

    driver, err := qlbdbdriver.New("test-ledger", qlldbSession, func(options
    *qlbdbdriver.DriverOptions) {
        options.RetryPolicy = retryPolicy1
    })
    if err != nil {
        panic(err)
    }

    // Configuring an exponential backoff strategy with base of 20 milliseconds
    retryPolicy2 := qlbdbdriver.RetryPolicy{
        MaxRetryLimit: 2,
    }
}

```



```

    Backoff: qlldbdriver.ExponentialBackoffStrategy{SleepBase: 20, SleepCap: 4000,
    }}

// Overrides the retry policy set by the driver instance
driver.SetRetryPolicy(retryPolicy2)

driver.Execute(context.Background(), func(txn qlldbdriver.Transaction) (interface{},
error) {
    return txn.Execute("CREATE TABLE Person")
})
}

```

Implementing uniqueness constraints

QLDB doesn't support unique indexes, but you can implement this behavior in your application.

Suppose that you want to implement a uniqueness constraint on the GovId field in the Person table. To do this, you can write a transaction that does the following:

1. Assert that the table has no existing documents with a specified GovId.
2. Insert the document if the assertion passes.

If a competing transaction concurrently passes the assertion, only one of the transactions will commit successfully. The other transaction will fail with an OCC conflict exception.

The following code example shows how to implement this uniqueness constraint logic.

```

govID := "TOYENC486FH"

document := map[string]interface{}{
    "GovId":      "TOYENC486FH",
    "FirstName": "Brent",
}

result, err := driver.Execute(context.Background(), func(txn qlldbdriver.Transaction)
(interface{}, error) {
    // Check if doc with GovId = govID exists
    result, err := txn.Execute("SELECT * FROM Person WHERE GovId = ?", govID)
    if err != nil {
        return nil, err
    }
}

```

```
// Check if there are any results
if result.Next(txn) {
    // Document already exists, no need to insert
    return nil, nil
}
return txn.Execute("INSERT INTO Person ?", document)
})
if err != nil {
    panic(err)
}
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Working with Amazon Ion

The following sections show how to use the Amazon Ion module to process Ion data.

Contents

- [Importing the Ion module](#)
- [Creating Ion types](#)
- [Getting Ion binary](#)
- [Getting Ion text](#)

Importing the Ion module

```
import "github.com/amzn/ion-go/ion"
```

Creating Ion types

The Ion library for Go currently doesn't support the Document Object Model (DOM), so you can't create Ion data types. But you can marshal and unmarshal between Go native types and Ion binary when working with QLDB.

Getting Ion binary

```
aDict := map[string]interface{}{
    "GovId": "TOYENC486FH",
    "FirstName": "Brent",
}

ionBytes, err := ion.MarshalBinary(aDict)
if err != nil {
    panic(err)
}

fmt.Println(ionBytes) // prints [224 1 0 234 238 151 129 131 222 147 135 190 144 133 71
111 118 73 100 137 70 105 114 115 116 78 97 109 101 222 148 138 139 84 79 89 69 78 67
52 56 54 70 72 139 133 66 114 101 110 116]
```

Getting Ion text

```
aDict := map[string]interface{}{
    "GovId": "TOYENC486FH",
    "FirstName": "Brent",
}

ionBytes, err := ion.MarshalText(aDict)
if err != nil {
    panic(err)
}

fmt.Println(string(ionBytes)) // prints {FirstName:"Brent",GovId:"TOYENC486FH"}
```

For more information about Ion, see the [Amazon Ion documentation](#) on GitHub. For more code examples of working with Ion in QLDB, see [Working with Amazon Ion data types in Amazon QLDB](#).

Amazon QLDB driver for Node.js

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

To work with data in your ledger, you can connect to Amazon QLDB from your Node.js application by using an AWS provided driver. The following topics describe how to get started with the QLDB driver for Node.js.

Topics

- [Driver resources](#)
- [Prerequisites](#)
- [Installation](#)
- [Setup recommendations](#)
- [Amazon QLDB driver for Node.js – Quick start tutorial](#)
- [Amazon QLDB driver for Node.js – Cookbook reference](#)

Driver resources

For more information about the functionality supported by the Node.js driver, see the following resources:

- API reference: [3.x](#), [2.x](#), [1.x](#)
- [Driver source code \(GitHub\)](#)
- [Sample application source code \(GitHub\)](#)
- [Amazon Ion code examples](#)
- [Build a simple CRUD operation and data stream on QLDB using AWS Lambda \(AWS Blog\)](#)

Prerequisites

Before you get started with the QLDB driver for Node.js, you must do the following:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#). This includes the following:
 1. Sign up for AWS.
 2. Create a user with the appropriate QLDB permissions.
 3. Grant programmatic access for development.
2. Install Node.js version 14.x or later from the [Node.js downloads](#) site. (Previous versions of the driver support Node.js version 10.x or later.)

3. Configure your development environment for the [AWS SDK for JavaScript in Node.js](#):

1. Set up your AWS credentials. We recommend creating a shared credentials file.

For instructions, see [Loading credentials in Node.js from the shared credentials file](#) in the *AWS SDK for JavaScript Developer Guide*.

2. Set your default AWS Region. To learn how, see [Setting the AWS Region](#).

For a complete list of available Regions, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

Next, you can download the complete tutorial sample application—or you can install only the driver in a Node.js project and run short code examples.

- To install the QLDB driver and the AWS SDK for JavaScript in Node.js in an existing project, proceed to [Installation](#).
- To set up a project and run short code examples that demonstrate basic data transactions on a ledger, see the [Quick start tutorial](#).
- To run more in-depth examples of both data and management API operations in the complete tutorial sample application, see the [Node.js tutorial](#).

Installation

QLDB supports the following driver versions and their Node.js dependencies.

Driver version	Node.js version	Status	Latest release date
1.x	10.x or later	Production release	June 5, 2020
2.x	10.x or later	Production release	May 6, 2021
3.x	14.x or later	Production release	November 10, 2023

To install the QLDB driver using [npm \(the Node.js package manager\)](#), enter the following command from your project root directory.

3.x

```
npm install amazon-qlldb-driver-nodejs
```

2.x

```
npm install amazon-qlldb-driver-nodejs@2.2.0
```

1.x

```
npm install amazon-qlldb-driver-nodejs@1.0.0
```

The driver has peer dependencies on the following packages. You must also install these packages as dependencies in your project.

3.x

Modular aggregated QLDB client (management API)

```
npm install @aws-sdk/client-qlldb
```

Modular aggregated *QLDB Session* client (data API)

```
npm install @aws-sdk/client-qlldb-session
```

Amazon Ion data format

```
npm install ion-js
```

Pure JavaScript implementation of BigInt

```
npm install jsbi
```

2.x

AWS SDK for JavaScript

```
npm install aws-sdk
```

Amazon Ion data format

```
npm install ion-js@4.0.0
```

Pure JavaScript implementation of BigInt

```
npm install jsbi@3.1.1
```

1.x

AWS SDK for JavaScript

```
npm install aws-sdk
```

Amazon Ion data format

```
npm install ion-js@4.0.0
```

Pure JavaScript implementation of BigInt

```
npm install jsbi@3.1.1
```

Using the driver to connect to a ledger

Then you can import the driver and use it to connect to a ledger. The following TypeScript code example shows how to create a driver instance for a specified ledger name and AWS Region.

3.x

```
import { Agent } from 'https';
import { QLDBSessionClientConfig } from "@aws-sdk/client-qlldb-session";
import { QldbDriver, RetryConfig } from 'amazon-qlldb-driver-nodejs';
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";

const maxConcurrentTransactions: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
```

```

    httpAgent: new Agent({
      maxSockets: maxConcurrentTransactions
    })
  };

  const serviceConfigurationOptions: QLDBSessionClientConfig = {
    region: "us-east-1"
  };

  //Use driver's default backoff function for this example (no second parameter
  //provided to RetryConfig)
  const retryConfig: RetryConfig = new RetryConfig(retryLimit);
  const qlldbDriver: QldbDriver = new QldbDriver("testLedger",
    serviceConfigurationOptions, lowLevelClientHttpOptions, maxConcurrentTransactions,
    retryConfig);

  qlldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
  });

```

2.x

```

import { Agent } from 'https';
import { QldbDriver, RetryConfig } from 'amazon-qlldb-driver-nodejs';

const maxConcurrentTransactions: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const agentForQldb: Agent = new Agent({
  keepAlive: true,
  maxSockets: maxConcurrentTransactions
});

const serviceConfigurationOptions = {
  region: "us-east-1",
  httpOptions: {
    agent: agentForQldb
  }
};

//Use driver's default backoff function for this example (no second parameter
//provided to RetryConfig)

```



```
const retryConfig: RetryConfig = new RetryConfig(retryLimit);
const qlldbDriver: QldbDriver = new QldbDriver("testLedger",
  serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);

qlldbDriver.getTableNames().then(function(tableNames: string[]) {
  console.log(tableNames);
});
```

1.x

```
import { Agent } from 'https';
import { QldbDriver } from 'amazon-qlldb-driver-nodejs';

const poolLimit: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const agentForQldb: Agent = new Agent({
  keepAlive: true,
  maxSockets: poolLimit
});

const serviceConfigurationOptions = {
  region: "us-east-1",
  httpOptions: {
    agent: agentForQldb
  }
};

const qlldbDriver: QldbDriver = new QldbDriver("testLedger",
  serviceConfigurationOptions, retryLimit, poolLimit);
qlldbDriver.getTableNames().then(function(tableNames: string[]) {
  console.log(tableNames);
});
```

For short code examples of how to run basic data transactions on a ledger, see the [Cookbook reference](#).

Setup recommendations

Reusing connections with keep-alive

QLDB Node.js driver v3

The default Node.js HTTP/HTTPS agent creates a new TCP connection for every new request. To avoid the cost of establishing a new connection, the AWS SDK for JavaScript v3 reuses TCP connections by default. For more information and to learn how to disable reusing connections, see [Reusing connections with keep-alive in Node.js](#) in the *AWS SDK for JavaScript Developer Guide*.

We recommend using the default setting to reuse connections in the QLDB driver for Node.js. During driver initialization, set the low-level client HTTP option `maxSockets` to the same value that you set for `maxConcurrentTransactions`.

For example, see the following JavaScript or TypeScript code.

JavaScript

```
const qlldb = require('amazon-qlldb-driver-nodejs');
const https = require('https');

//Replace this value as appropriate for your application
const maxConcurrentTransactions = 50;

const agentForQldb = new https.Agent({
  //Set this to the same value as `maxConcurrentTransactions` (previously called
  `poolLimit`)
  //Do not rely on the default value of `Infinity`
  "maxSockets": maxConcurrentTransactions
});

const lowLevelClientHttpOptions = {
  httpAgent: agentForQldb
}

let driver = new qlldb.QldbDriver("testLedger", undefined, lowLevelClientHttpOptions,
maxConcurrentTransactions);
```

TypeScript

```
import { Agent } from 'https';
```

```
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";
import { QldbDriver } from 'amazon-qlldb-driver-nodejs';

//Replace this value as appropriate for your application
const maxConcurrentTransactions: number = 50;

const agentForQldb: Agent = new Agent({
  //Set this to the same value as `maxConcurrentTransactions` (previously called
  `poolLimit`)
  //Do not rely on the default value of `Infinity`
  maxSockets: maxConcurrentTransactions
});

const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
  httpAgent: agentForQldb
};

let driver = new QldbDriver("testLedger", undefined, lowLevelClientHttpOptions,
  maxConcurrentTransactions);
```

QLDB Node.js driver v2

The default Node.js HTTP/HTTPS agent creates a new TCP connection for every new request. To avoid the cost of establishing a new connection, we recommend reusing an existing connection.

To reuse connections in the QLDB driver for Node.js, use one of the following options:

- During driver initialization, set the following low-level client HTTP options:
 - `keepAlive` – `true`
 - `maxSockets` – The same value that you set for `maxConcurrentTransactions`

For example, see the following JavaScript or TypeScript code.

JavaScript

```
const qldb = require('amazon-qlldb-driver-nodejs');
const https = require('https');

//Replace this value as appropriate for your application
const maxConcurrentTransactions = 50;

const agentForQldb = new https.Agent({
```

```
"keepAlive": true,
//Set this to the same value as `maxConcurrentTransactions`(previously called
`poolLimit`)
//Do not rely on the default value of `Infinity`
"maxSockets": maxConcurrentTransactions
});

const serviceConfiguration = { "httpOptions": {
  "agent": agentForQldb
}};

let driver = new qldb.QldbDriver("testLedger", serviceConfiguration,
  maxConcurrentTransactions);
```

TypeScript

```
import { Agent } from 'https';
import { ClientConfiguration } from 'aws-sdk/clients/acm';
import { QldbDriver } from 'amazon-qldb-driver-nodejs';

//Replace this value as appropriate for your application
const maxConcurrentTransactions: number = 50;

const agentForQldb: Agent = new Agent({
  keepAlive: true,
  //Set this to the same value as `maxConcurrentTransactions`(previously called
  `poolLimit`)
  //Do not rely on the default value of `Infinity`
  maxSockets: maxConcurrentTransactions
});

const serviceConfiguration: ClientConfiguration = { httpOptions: {
  agent: agentForQldb
}};

let driver = new QldbDriver("testLedger", serviceConfiguration,
  maxConcurrentTransactions);
```

- Alternatively, you can set the `AWS_NODEJS_CONNECTION_REUSE_ENABLED` environment variable to 1. For more information, see [Reusing Connections with Keep-Alive in Node.js](#) in the *AWS SDK for JavaScript Developer Guide*.

Note

If you set this environment variable, it affects all of the AWS services that use the AWS SDK for JavaScript.

Amazon QLDB driver for Node.js – Quick start tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this tutorial, you learn how to set up a simple application using the Amazon QLDB driver for Node.js. This guide includes steps for installing the driver, and short JavaScript and TypeScript code examples of basic *create*, *read*, *update*, and *delete* (CRUD) operations. For more in-depth examples that demonstrate these operations in a full sample application, see the [Node.js tutorial](#).

Note

Where applicable, some steps have different code examples for each supported major version of the QLDB driver for Node.js.

Topics

- [Prerequisites](#)
- [Step 1: Set up your project](#)
- [Step 2: Initialize the driver](#)
- [Step 3: Create a table and an index](#)
- [Step 4: Insert a document](#)
- [Step 5: Query the document](#)
- [Step 6: Update the document](#)

- [Running the complete application](#)

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the [Prerequisites](#) for the Node.js driver, if you haven't already done so. This includes signing up for AWS, granting programmatic access for development, and installing Node.js.
2. Create a ledger named `quick-start`.

To learn how to create a ledger, see [Basic operations for Amazon QLDB ledgers](#) or [Step 1: Create a new ledger](#) in *Getting started with the console*.

If you're using TypeScript, you must also do the following setup steps.

Using TypeScript

To install TypeScript

1. Install the TypeScript package. The QLDB driver runs on TypeScript 3.8.x.

```
$ npm install --global typescript@3.8.0
```

2. After the package is installed, run the following command to make sure that the TypeScript compiler is installed.

```
$ tsc --version
```

To run the code in the following steps, note that you must first transpile your TypeScript file to executable JavaScript code, as follows.

```
$ tsc app.ts; node app.js
```

Step 1: Set up your project

First, set up your Node.js project.

1. Create a folder for your application.

```
$ mkdir myproject
$ cd myproject
```

2. To initialize your project, enter the following npm command and answer the questions that are asked during the setup. You can use defaults for most of the questions.

```
$ npm init
```

3. Install the Amazon QLDB driver for Node.js.

- Using version 3.x

```
$ npm install amazon-qlldb-driver-nodejs --save
```

- Using version 2.x

```
$ npm install amazon-qlldb-driver-nodejs@2.2.0 --save
```

- Using version 1.x

```
$ npm install amazon-qlldb-driver-nodejs@1.0.0 --save
```

4. Install the peer dependencies of the driver.

- Using version 3.x

```
$ npm install @aws-sdk/client-qlldb-session --save
$ npm install ion-js --save
$ npm install jsbi --save
```

- Using version 2.x or 1.x

```
$ npm install aws-sdk --save
$ npm install ion-js@4.0.0 --save
$ npm install jsbi@3.1.1 --save
```

5. Create a new file named `app.js` for JavaScript, or `app.ts` for TypeScript.

Then, incrementally add the code examples in the following steps to try some basic CRUD operations. Or, you can skip the step-by-step tutorial and instead run the [complete application](#).

Step 2: Initialize the driver

Initialize an instance of the driver that connects to the ledger named `quick-start`. Add the following code to your `app.js` or `app.ts` file.

Using version 3.x

JavaScript

```
var qlldb = require('amazon-qlldb-driver-nodejs');
var https = require('https');

function main() {
    const maxConcurrentTransactions = 10;
    const retryLimit = 4;

    const agentForQldb = new https.Agent({
        maxSockets: maxConcurrentTransactions
    });

    const lowLevelClientHttpOptions = {
        httpAgent: agentForQldb
    }

    const serviceConfigurationOptions = {
        region: "us-east-1"
    };

    // Use driver's default backoff function for this example (no second parameter
    // provided to RetryConfig)
    var retryConfig = new qlldb.RetryConfig(retryLimit);
    var driver = new qlldb.QldbDriver("quick-start", serviceConfigurationOptions,
        lowLevelClientHttpOptions, maxConcurrentTransactions, retryConfig);
}

main();
```

TypeScript

```
import { Agent } from "https";
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";
import { QLDBSessionClientConfig } from "@aws-sdk/client-qlldb-session";
import { QldbDriver, RetryConfig } from "amazon-qlldb-driver-nodejs";
```



```
function main(): void {
    const maxConcurrentTransactions: number = 10;
    const agentForQldb: Agent = new Agent({
        maxSockets: maxConcurrentTransactions
    });

    const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
        httpAgent: agentForQldb
    };

    const serviceConfigurationOptions: QLDBSessionClientConfig = {
        region: "us-east-1"
    };

    const retryLimit: number = 4;
    // Use driver's default backoff function for this example (no second parameter
    // provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
    const driver: QldbDriver = new QldbDriver("quick-start",
        serviceConfigurationOptions, lowLevelClientHttpOptions, maxConcurrentTransactions,
        retryConfig);
}

if (require.main === module) {
    main();
}
```

Note

- In this code example, replace *us-east-1* with the AWS Region where you created your ledger.
- For simplicity, the remaining code examples in this guide use a driver with default settings, as specified in the following example for version 1.x. You can also use your own driver instance with a custom `RetryConfig` instead.

Using version 2.x

JavaScript

```
var qlldb = require('amazon-qlldb-driver-nodejs');
var https = require('https');

function main() {
    var maxConcurrentTransactions = 10;
    var retryLimit = 4;

    var agentForQldb = new https.Agent({
        keepAlive: true,
        maxSockets: maxConcurrentTransactions
    });

    var serviceConfigurationOptions = {
        region: "us-east-1",
        httpOptions: {
            agent: agentForQldb
        }
    };

    // Use driver's default backoff function for this example (no second parameter
    // provided to RetryConfig)
    var retryConfig = new qlldb.RetryConfig(retryLimit);
    var driver = new qlldb.QldbDriver("quick-start", serviceConfigurationOptions,
    maxConcurrentTransactions, retryConfig);
}

main();
```

TypeScript

```
import { QldbDriver, RetryConfig } from "amazon-qlldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/acm";
import { Agent } from "https";

function main(): void {
    const maxConcurrentTransactions: number = 10;
    const agentForQldb: Agent = new Agent({
        keepAlive: true,
        maxSockets: maxConcurrentTransactions
```

```
});  
const serviceConfigurationOptions: ClientConfiguration = {  
  region: "us-east-1",  
  httpOptions: {  
    agent: agentForQldb  
  }  
};  
const retryLimit: number = 4;  
// Use driver's default backoff function for this example (no second parameter  
provided to RetryConfig)  
const retryConfig: RetryConfig = new RetryConfig(retryLimit);  
const driver: QldbDriver = new QldbDriver("quick-start",  
serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);  
}  
  
if (require.main === module) {  
  main();  
}
```

Note

- In this code example, replace *us-east-1* with the AWS Region where you created your ledger.
- Version 2.x introduces the new optional parameter `RetryConfig` for initializing `QldbDriver`.
- For simplicity, the remaining code examples in this guide use a driver with default settings, as specified in the following example for version 1.x. You can also use your own driver instance with a custom `RetryConfig` instead.
- This code example initializes a driver that reuses existing connections by setting keep-alive options. To learn more, see [Setup recommendations](#) for the Node.js driver.

Using version 1.x

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');  
  
function main() {
```

```
// Use default settings
const driver = new qlldb.QldbDriver("quick-start");
}

main();
```

TypeScript

```
import { QldbDriver } from "amazon-qlldb-driver-nodejs";

function main(): void {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");
}

if (require.main === module) {
    main();
}
```

Note

You can set the `AWS_REGION` environment variable to specify the Region. For more information, see [Setting the AWS Region](#) in the *AWS SDK for JavaScript Developer Guide*.

Step 3: Create a table and an index

The following code examples show how to run `CREATE TABLE` and `CREATE INDEX` statements.

1. Add the following function that creates a table named `People`.

JavaScript

```
async function createTable(txn) {
    await txn.execute("CREATE TABLE People");
}
```

TypeScript

```
async function createTable(txn: TransactionExecutor): Promise<void> {
```

```
    await txn.execute("CREATE TABLE People");
}
```

2. Add the following function that creates an index for the `firstName` field on the `People` table. [Indexes](#) are required to optimize query performance and help to limit [optimistic concurrency control \(OCC\)](#) conflict exceptions.

JavaScript

```
async function createIndex(txn) {
    await txn.execute("CREATE INDEX ON People (firstName)");
}
```

TypeScript

```
async function createIndex(txn: TransactionExecutor): Promise<void> {
    await txn.execute("CREATE INDEX ON People (firstName)");
}
```

3. In the main function, you first call `createTable`, and then call `createIndex`.

JavaScript

```
const qlldb = require('amazon-qlldb-driver-nodejs');

async function main() {
    // Use default settings
    const driver = new qlldb.QLldbDriver("quick-start");

    await driver.executeLambda(async (txn) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
    });

    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qlldb-driver-nodejs";

async function main(): Promise<void> {
  // Use default settings
  const driver: QldbDriver = new QldbDriver("quick-start");

  await driver.executeLambda(async (txn: TransactionExecutor) => {
    console.log("Create table People");
    await createTable(txn);
    console.log("Create index on firstName");
    await createIndex(txn);
  });

  driver.close();
}

if (require.main === module) {
  main();
}
```

4. Run the code to create the table and index.

JavaScript

```
$ node app.js
```

TypeScript

```
$ tsc app.ts; node app.js
```

Step 4: Insert a document

The following code example shows how to run an INSERT statement. QLDB supports the [PartiQL](#) query language (SQL compatible) and the [Amazon Ion](#) data format (superset of JSON).

1. Add the following function that inserts a document into the `People` table.

JavaScript

```
async function insertDocument(txn) {
  const person = {
    firstName: "John",
    lastName: "Doe",
    age: 42
  };
  await txn.execute("INSERT INTO People ?", person);
}
```

TypeScript

```
async function insertDocument(txn: TransactionExecutor): Promise<void> {
  const person: Record<string, any> = {
    firstName: "John",
    lastName: "Doe",
    age: 42
  };
  await txn.execute("INSERT INTO People ?", person);
}
```

This example uses a question mark (?) as a variable placeholder to pass the document information to the statement. The execute method supports values in both Amazon Ion types and Node.js native types.

Tip

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [list](#) to the statement as follows.

```
// people is a list
txn.execute("INSERT INTO People ?", people);
```

You don't enclose the variable placeholder (?) in double angle brackets (<<...>>) when passing a list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

2. In the main function, remove the `createTable` and `createIndex` calls, and add a call to `insertDocument`.

JavaScript

```
const qlldb = require('amazon-qlldb-driver-nodejs');

async function main() {
  // Use default settings
  const driver = new qlldb.QldbDriver("quick-start");

  await driver.executeLambda(async (txn) => {
    console.log("Insert document");
    await insertDocument(txn);
  });

  driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qlldb-driver-nodejs";

async function main(): Promise<void> {
  // Use default settings
  const driver: QldbDriver = new QldbDriver("quick-start");

  await driver.executeLambda(async (txn: TransactionExecutor) => {
    console.log("Insert document");
    await insertDocument(txn);
  });

  driver.close();
}

if (require.main === module) {
  main();
}
```


Step 5: Query the document

The following code example shows how to run a SELECT statement.

1. Add the following function that queries a document from the `People` table.

JavaScript

```
async function fetchDocuments(txn) {
    return await txn.execute("SELECT firstName, age, lastName FROM People WHERE
    firstName = ?", "John");
}
```

TypeScript

```
async function fetchDocuments(txn: TransactionExecutor): Promise<dom.Value[]> {
    return (await txn.execute("SELECT firstName, age, lastName FROM People WHERE
    firstName = ?", "John")).getResultList();
}
```

2. In the main function, add the following call to `fetchDocuments` after the call to `insertDocument`.

JavaScript

```
const qlldb = require('amazon-qlldb-driver-nodejs');

async function main() {
    // Use default settings
    const driver = new qlldb.QLldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}
```

```
}  
  
main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";  
import { dom } from "ion-js";  
  
async function main(): Promise<void> {  
    // Use default settings  
    const driver: QldbDriver = new QldbDriver("quick-start");  
  
    const resultList: dom.Value[] = await driver.executeLambda(async (txn:  
TransactionExecutor) => {  
        console.log("Insert document");  
        await insertDocument(txn);  
        console.log("Fetch document");  
        return await fetchDocuments(txn);  
    });  
  
    // Pretty print the result list  
    console.log("The result List is ", JSON.stringify(resultList, null, 2));  
    driver.close();  
}  
  
if (require.main === module) {  
    main();  
}
```

Step 6: Update the document

The following code example shows how to run an UPDATE statement.

1. Add the following function that updates a document in the People table by changing lastName to "Stiles".

JavaScript

```
async function updateDocuments(txn) {
```

```
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
        "Stiles", "John");
}
```

TypeScript

```
async function updateDocuments(txn: TransactionExecutor): Promise<void> {
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
        "Stiles", "John");
}
```

2. In the main function, add the following call to `updateDocuments` after the call to `fetchDocuments`. Then, call `fetchDocuments` again to see the updated results.

JavaScript

```
const qlldb = require('amazon-qlldb-driver-nodejs');

async function main() {
    // Use default settings
    const driver = new qlldb.QLldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qlldb-driver-nodejs";
import { dom } from "ion-js";

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
TransactionExecutor) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        return await fetchDocuments(txn);
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

if (require.main === module) {
    main();
}
```

3. Run the code to insert, query, and update a document.

JavaScript

```
$ node app.js
```

TypeScript

```
$ tsc app.ts; node app.js
```

Running the complete application

The following code examples are the complete versions of `app.js` and `app.ts`. Instead of doing the previous steps individually, you can also run this code from start to end. This application demonstrates some basic CRUD operations on the ledger named `quick-start`.

Note

Before you run this code, make sure that you don't already have an active table named `People` in the `quick-start` ledger.

JavaScript

```
const qlldb = require('amazon-qlldb-driver-nodejs');

async function createTable(txn) {
    await txn.execute("CREATE TABLE People");
}

async function createIndex(txn) {
    await txn.execute("CREATE INDEX ON People (firstName)");
}

async function insertDocument(txn) {
    const person = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}

async function fetchDocuments(txn) {
    return await txn.execute("SELECT firstName, age, lastName FROM People WHERE
    firstName = ?", "John");
}

async function updateDocuments(txn) {
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
    "Stiles", "John");
}
```

```

async function main() {
    // Use default settings
    const driver = new qlldb.QldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

main();

```

TypeScript

```

import { QldbDriver, TransactionExecutor } from "amazon-qlldb-driver-nodejs";
import { dom } from "ion-js";

async function createTable(txn: TransactionExecutor): Promise<void> {
    await txn.execute("CREATE TABLE People");
}

async function createIndex(txn: TransactionExecutor): Promise<void> {
    await txn.execute("CREATE INDEX ON People (firstName)");
}

async function insertDocument(txn: TransactionExecutor): Promise<void> {
    const person: Record<string, any> = {

```

```
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}

async function fetchDocuments(txn: TransactionExecutor): Promise<dom.Value[]> {
    return (await txn.execute("SELECT firstName, age, lastName FROM People WHERE
    firstName = ?", "John")).getResultList();
}

async function updateDocuments(txn: TransactionExecutor): Promise<void> {
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
    "Stiles", "John");
};

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
TransactionExecutor) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        return await fetchDocuments(txn);
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

if (require.main === module) {
    main();
}
```

```
}
```

To run the complete application, enter the following command.

JavaScript

```
$ node app.js
```

TypeScript

```
$ tsc app.ts; node app.js
```

Amazon QLDB driver for Node.js – Cookbook reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This reference guide shows common use cases of the Amazon QLDB driver for Node.js. It provides JavaScript and TypeScript code examples that demonstrate how to use the driver to run basic *create, read, update, and delete* (CRUD) operations. It also includes code examples for processing Amazon Ion data. In addition, this guide highlights best practices for making transactions idempotent and implementing uniqueness constraints.

Contents

- [Importing the driver](#)
- [Instantiating the driver](#)
- [CRUD operations](#)
 - [Creating tables](#)
 - [Creating indexes](#)
 - [Reading documents](#)
 - [Using query parameters](#)

- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)
- [Running multiple statements in a transaction](#)
- [Retry logic](#)
- [Implementing uniqueness constraints](#)
- [Working with Amazon Ion](#)
 - [Importing the Ion module](#)
 - [Creating Ion types](#)
 - [Getting an Ion binary dump](#)
 - [Getting an Ion text dump](#)

Importing the driver

The following code example imports the driver.

JavaScript

```
var qlldb = require('amazon-qlldb-driver-nodejs');  
var ionjs = require('ion-js');
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qlldb-driver-nodejs";  
import { dom, dumpBinary, load } from "ion-js";
```

Note

This example also imports the Amazon Ion package (`ion-js`). You need this package to process Ion data when running some data operations in this reference. To learn more, see [Working with Amazon Ion](#).

Instantiating the driver

The following code example creates an instance of the driver that connects to a specified ledger name using default settings.

JavaScript

```
const qlldbDriver = new qlldb.QldbDriver("vehicle-registration");
```

TypeScript

```
const qlldbDriver: QldbDriver = new QldbDriver("vehicle-registration");
```

CRUD operations

QLDB runs *create*, *read*, *update*, and *delete* (CRUD) operations as part of a transaction.

Warning

As a best practice, make your write transactions strictly idempotent.

Making transactions idempotent

We recommend that you make write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can run multiple times and produce identical results each time.

For example, consider a transaction that inserts a document into a table named `Person`. The transaction should first check whether or not the document already exists in the table. Without this check, the table might end up with duplicate documents.

Suppose that QLDB successfully commits the transaction on the server side, but the client times out while waiting for a response. If the transaction isn't idempotent, the same document could be inserted more than once in the case of a retry.

Using indexes to avoid full table scans

We also recommend that you run statements with a `WHERE` predicate clause using an *equality* operator on an indexed field or a document ID; for example, `WHERE indexedField = 123` or

WHERE indexedField IN (456, 789). Without this indexed lookup, QLDB needs to do a table scan, which can lead to transaction timeouts or *optimistic concurrency control* (OCC) conflicts.

For more information about OCC, see [Amazon QLDB concurrency model](#).

Implicitly created transactions

The [QldbDriver.executeLambda](#) method accepts a lambda function that receives an instance of [TransactionExecutor](#), which you can use to run statements. The instance of `TransactionExecutor` wraps an implicitly created transaction.

You can run statements within the lambda function by using the [execute](#) method of the transaction executor. The driver implicitly commits the transaction when the lambda function returns.

Note

The `execute` method supports both Amazon Ion types and Node.js native types. If you pass a Node.js native type as an argument to `execute`, the driver converts it to an Ion type using the `ion-js` package (provided that conversion for the given Node.js data type is supported). For supported data types and conversion rules, see the [Ion JavaScript DOM README](#).

The following sections show how to run basic CRUD operations, specify custom retry logic, and implement uniqueness constraints.

Contents

- [Creating tables](#)
- [Creating indexes](#)
- [Reading documents](#)
 - [Using query parameters](#)
- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)

- [Running multiple statements in a transaction](#)
- [Retry logic](#)
- [Implementing uniqueness constraints](#)

Creating tables

JavaScript

```
(async function() {  
    await qlldbDriver.executeLambda(async (txn) => {  
        await txn.execute("CREATE TABLE Person");  
    });  
})();
```

TypeScript

```
(async function(): Promise<void> {  
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {  
        await txn.execute('CREATE TABLE Person');  
    });  
})();
```

Creating indexes

JavaScript

```
(async function() {  
    await qlldbDriver.executeLambda(async (txn) => {  
        await txn.execute("CREATE INDEX ON Person (GovId)");  
    });  
})();
```

TypeScript

```
(async function(): Promise<void> {  
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {  
        await txn.execute('CREATE INDEX ON Person (GovId)');  
    });  
})();
```

Reading documents

JavaScript

```
(async function() {
  // Assumes that Person table has documents as follows:
  // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
  await qlldbDriver.executeLambda(async (txn) => {
    const results = (await txn.execute("SELECT * FROM Person WHERE GovId =
'TOYENC486FH'")).getResultList();
    for (let result of results) {
      console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
      console.log(result.get('FirstName')); // prints [String: 'Brent']
    }
  });
})();
```

TypeScript

```
(async function(): Promise<void> {
  // Assumes that Person table has documents as follows:
  // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
  await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
    const results: dom.Value[] = (await txn.execute("SELECT * FROM Person WHERE
GovId = 'TOYENC486FH'")).getResultList();
    for (let result of results) {
      console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
      console.log(result.get('FirstName')); // prints [String: 'Brent']
    }
  });
})();
```

Using query parameters

The following code example uses a native type query parameter.

JavaScript

```
(async function() {
  // Assumes that Person table has documents as follows:
  // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
  await qlldbDriver.executeLambda(async (txn) => {
```

```

    const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
    'TOYENC486FH')).getResultList();
    for (let result of results) {
        console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
        console.log(result.get('FirstName')); // prints [String: 'Brent']
    }
  });
}());

```

TypeScript

```

(async function(): Promise<void> {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
GovId = ?', 'TOYENC486FH')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());

```

The following code example uses an Ion type query parameter.

JavaScript

```

(async function() {
    await qlldbDriver.executeLambda(async (txn) => {
        const govId = ionjs.load("TOYENC486FH");

        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
govId)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());

```

TypeScript

```
(async function(): Promise<void> {
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govId: dom.Value = load("TOYENC486FH");

        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
GovId = ?', govId)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
})();
```

The following code example uses multiple query parameters.

JavaScript

```
(async function() {
    await qlldbDriver.executeLambda(async (txn) => {
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ? AND
FirstName = ?', 'TOYENC486FH', 'Brent')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
})();
```

TypeScript

```
(async function(): Promise<void> {
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
GovId = ? AND FirstName = ?', 'TOYENC486FH', 'Brent')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
})();
```

```
}());
```

The following code example uses a list of query parameters.

JavaScript

```
(async function() {
    await qlldbDriver.executeLambda(async (txn) => {
        const govIds = ['TOYENC486FH', 'LOGANB486CG', 'LEWISR261LL'];
        /*
        Assumes that Person table has documents as follows:
        { "GovId": "TOYENC486FH", "FirstName": "Brent" }
        { "GovId": "LOGANB486CG", "FirstName": "Brent" }
        { "GovId": "LEWISR261LL", "FirstName": "Raul" }
        */
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId IN
        (?, ?, ?)', ...govIds)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId'));
            console.log(result.get('FirstName'));
        }
        /*
        prints:
        [String: 'TOYENC486FH']
        [String: 'Brent']
        [String: 'LOGANB486CG']
        [String: 'Brent']
        [String: 'LEWISR261LL']
        [String: 'Raul']
        */
    });
})();
```

TypeScript

```
(async function(): Promise<void> {
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govIds: string[] = ['TOYENC486FH', 'LOGANB486CG', 'LEWISR261LL'];
        /*
        Assumes that Person table has documents as follows:
        { "GovId": "TOYENC486FH", "FirstName": "Brent" }
        { "GovId": "LOGANB486CG", "FirstName": "Brent" }
        */
    });
})();
```



```

    { "GovId": "LEWISR261LL", "FirstName": "Raul" }
    */
    const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
GovId IN (?, ?, ?)', ...govIds)).getResultList();
    for (let result of results) {
        console.log(result.get('GovId'));
        console.log(result.get('FirstName'));
    }
    /*
prints:
[String: 'TOYENC486FH']
[String: 'Brent']
[String: 'LOGANB486CG']
[String: 'Brent']
[String: 'LEWISR261LL']
[String: 'Raul']
    */
}
});
}());

```

Note

When you run a query without an indexed lookup, it invokes a full table scan. In this example, we recommend having an [index](#) on the GovId field to optimize performance. Without an index on GovId, queries can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting documents

The following code example inserts native data types.

JavaScript

```

(async function() {
    await qlldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
    });
});

```

```

        if (results.length == 0) {
            const doc = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            await txn.execute('INSERT INTO Person ?', doc);
        }
    });
}());

```

TypeScript

```

(async function(): Promise<void> {
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
GovId = ?', 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc: Record<string, string> = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            await txn.execute('INSERT INTO Person ?', doc);
        }
    });
}());

```

The following code example inserts Ion data types.

JavaScript

```

(async function() {
    await qlldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc = {

```

```
        'FirstName': 'Brent',
        'GovId': 'TOYENC486FH',
    };
    // Create a sample Ion doc
    const ionDoc = ionjs.load(ionjs.dumpBinary(doc));

    await txn.execute('INSERT INTO Person ?', ionDoc);
  }
});
}());
```

TypeScript

```
(async function(): Promise<void> {
  await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
    // Check if doc with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
GovId = ?', 'TOYENC486FH')).getResultList();
    // Insert the document after ensuring it doesn't already exist
    if (results.length == 0) {
      const doc: Record<string, string> = {
        'FirstName': 'Brent',
        'GovId': 'TOYENC486FH',
      };
      // Create a sample Ion doc
      const ionDoc: dom.Value = load(dumpBinary(doc));

      await txn.execute('INSERT INTO Person ?', ionDoc);
    }
  });
}());
```

This transaction inserts a document into the Person table. Before inserting, it first checks if the document already exists in the table. **This check makes the transaction idempotent in nature.** Even if you run this transaction multiple times, it won't cause any unintended side effects.

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting multiple documents in one statement

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [list](#) to the statement as follows.

```
// people is a list
txn.execute("INSERT INTO People ?", people);
```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . . >>) when passing a list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Updating documents

The following code example uses native data types.

JavaScript

```
(async function() {
  await qlldbDriver.executeLambda(async (txn) => {
    await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?', 'John',
      'TOYENC486FH');
  });
})();
```

TypeScript

```
(async function(): Promise<void> {
  await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
    await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?', 'John',
      'TOYENC486FH');
  });
})();
```

The following code example uses Ion data types.

JavaScript

```
(async function() {
  await qlldbDriver.executeLambda(async (txn) => {
    const firstName = ionjs.load("John");
    const govId = ionjs.load("TOYENC486FH");

    await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?',
      firstName, govId);
  });
})();
```

TypeScript

```
(async function(): Promise<void> {
  await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
    const firstName: dom.Value = load("John");
    const govId: dom.Value = load("TOYENC486FH");

    await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?',
      firstName, govId);
  });
})();
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Deleting documents

The following code example uses native data types.

JavaScript

```
(async function() {
  await qlldbDriver.executeLambda(async (txn) => {
```

```
        await txn.execute('DELETE FROM Person WHERE GovId = ?', 'TOYENC486FH');
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('DELETE FROM Person WHERE GovId = ?', 'TOYENC486FH');
    });
})();
```

The following code example uses Ion data types.

JavaScript

```
(async function() {
    await qlldbDriver.executeLambda(async (txn) => {
        const govId = ionjs.load("TOYENC486FH");

        await txn.execute('DELETE FROM Person WHERE GovId = ?', govId);
    });
})();
```

TypeScript

```
(async function(): Promise<void> {
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govId: dom.Value = load("TOYENC486FH");

        await txn.execute('DELETE FROM Person WHERE GovId = ?', govId);
    });
})();
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Running multiple statements in a transaction

TypeScript

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
// because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
// or not.
async function insureCar(driver: QldbDriver, vin: string): Promise<boolean> {

    return await driver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
            vin)).getResultList();

        if (results.length > 0) {
            await txn.execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin);
            return true;
        }
        return false;
    });
};
```

Retry logic

The driver's `executeLambda` method has a built-in retry mechanism that retries the transaction if a retryable exception occurs (such as timeouts or OCC conflicts). The maximum number of retry attempts and the backoff strategy are configurable.

The default retry limit is 4, and the default backoff strategy is [defaultBackoffFunction](#) with a base of 10 milliseconds. You can set the retry configuration per driver instance and also per transaction by using an instance of [RetryConfig](#).

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for an instance of the driver.

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');
```

```
// Configuring retry limit to 2
const retryConfig = new qlldb.RetryConfig(2);
const qlldbDriver = new qlldb.QldbDriver("test-ledger", undefined, undefined,
  retryConfig);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff = (retryAttempt, error, transactionId) => {
  return 1000 * retryAttempt;
};

const retryConfigCustomBackoff = new qlldb.RetryConfig(2, customBackoff);
const qlldbDriverCustomBackoff = new qlldb.QldbDriver("test-ledger", undefined,
  undefined, retryConfigCustomBackoff);
```

TypeScript

```
import { BackoffFunction, QldbDriver, RetryConfig } from "amazon-qlldb-driver-nodejs"

// Configuring retry limit to 2
const retryConfig: RetryConfig = new RetryConfig(2);
const qlldbDriver: QldbDriver = new QldbDriver("test-ledger", undefined, undefined,
  retryConfig);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff: BackoffFunction = (retryAttempt: number, error: Error,
  transactionId: string) => {
  return 1000 * retryAttempt;
};

const retryConfigCustomBackoff: RetryConfig = new RetryConfig(2, customBackoff);
const qlldbDriverCustomBackoff: QldbDriver = new QldbDriver("test-ledger", undefined,
  undefined, retryConfigCustomBackoff);
```

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for a particular lambda execution. This configuration for `executeLambda` overrides the retry logic that is set for the driver instance.

JavaScript

```
var qlldb = require('amazon-qlldb-driver-nodejs');

// Configuring retry limit to 2
```



```

const retryConfig1 = new qlldb.RetryConfig(2);
const qlldbDriver = new qlldb.QldbDriver("test-ledger", undefined, undefined,
    retryConfig1);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff = (retryAttempt, error, transactionId) => {
    return 1000 * retryAttempt;
};

const retryConfig2 = new qlldb.RetryConfig(2, customBackoff);

// The config `retryConfig1` will be overridden by `retryConfig2`
(async function() {
    await qlldbDriver.executeLambda(async (txn) => {
        await txn.execute('CREATE TABLE Person');
    }, retryConfig2);
})();

```

TypeScript

```

import { BackoffFunction, QldbDriver, RetryConfig, TransactionExecutor } from
    "amazon-qlldb-driver-nodejs"

// Configuring retry limit to 2
const retryConfig1: RetryConfig = new RetryConfig(2);
const qlldbDriver: QldbDriver = new QldbDriver("test-ledger", undefined, undefined,
    retryConfig1);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff: BackoffFunction = (retryAttempt: number, error: Error,
    transactionId: string) => {
    return 1000 * retryAttempt;
};

const retryConfig2: RetryConfig = new RetryConfig(2, customBackoff);

// The config `retryConfig1` will be overridden by `retryConfig2`
(async function(): Promise<void> {
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('CREATE TABLE Person');
    }, retryConfig2);
})();

```

Implementing uniqueness constraints

QLDB doesn't support unique indexes, but you can implement this behavior in your application.

Suppose that you want to implement a uniqueness constraint on the GovId field in the Person table. To do this, you can write a transaction that does the following:

1. Assert that the table has no existing documents with a specified GovId.
2. Insert the document if the assertion passes.

If a competing transaction concurrently passes the assertion, only one of the transactions will commit successfully. The other transaction will fail with an OCC conflict exception.

The following code example shows how to implement this uniqueness constraint logic.

JavaScript

```
const govId = 'TOYENC486FH';
const document = {
  'FirstName': 'Brent',
  'GovId': 'TOYENC486FH',
};
(async function() {
  await qlldbDriver.executeLambda(async (txn) => {
    // Check if doc with GovId = govId exists
    const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
govId)).getResultList();
    // Insert the document after ensuring it doesn't already exist
    if (results.length == 0) {
      await txn.execute('INSERT INTO Person ?', document);
    }
  });
})();
```

TypeScript

```
const govId: string = 'TOYENC486FH';
const document: Record<string, string> = {
  'FirstName': 'Brent',
  'GovId': 'TOYENC486FH',
};
(async function(): Promise<void> {
```

```
await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
    // Check if doc with GovId = govId exists
    const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
GovId = ?', govId)).getResultList();
    // Insert the document after ensuring it doesn't already exist
    if (results.length == 0) {
        await txn.execute('INSERT INTO Person ?', document);
    }
});
})();
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Working with Amazon Ion

The following sections show how to use the Amazon Ion module to process Ion data.

Contents

- [Importing the Ion module](#)
- [Creating Ion types](#)
- [Getting an Ion binary dump](#)
- [Getting an Ion text dump](#)

Importing the Ion module

JavaScript

```
var ionjs = require('ion-js');
```

TypeScript

```
import { dom, dumpBinary, dumpText, load } from "ion-js";
```

Creating Ion types

The following code example creates an Ion object from Ion text.

JavaScript

```
const ionText = '{GovId: "TOYENC486FH", FirstName: "Brent"}';
const ionObj = ionjs.load(ionText);

console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

TypeScript

```
const ionText: string = '{GovId: "TOYENC486FH", FirstName: "Brent"}';
const ionObj: dom.Value = load(ionText);

console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

The following code example creates an Ion object from a Node.js dictionary.

JavaScript

```
const aDict = {
  'GovId': 'TOYENC486FH',
  'FirstName': 'Brent'
};
const ionObj = ionjs.load(ionjs.dumpBinary(aDict));
console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

TypeScript

```
const aDict: Record<string, string> = {
  'GovId': 'TOYENC486FH',
  'FirstName': 'Brent'
};
const ionObj: dom.Value = load(dumpBinary(aDict));
console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

Getting an Ion binary dump

JavaScript

```
// ionObj is an Ion struct
console.log(ionjs.dumpBinary(ionObj).toString()); // prints
224,1,0,234,238,151,129,131,222,147,135,190,144,133,71,111,118,73,100,137,70,105,114,115,11
```

TypeScript

```
// ionObj is an Ion struct
console.log(dumpBinary(ionObj).toString()); // prints
224,1,0,234,238,151,129,131,222,147,135,190,144,133,71,111,118,73,100,137,70,105,114,115,11
```

Getting an Ion text dump

JavaScript

```
// ionObj is an Ion struct
console.log(ionjs.dumpText(ionObj)); // prints
{GovId:"TOYENC486FH",FirstName:"Brent"}
```

TypeScript

```
// ionObj is an Ion struct
console.log(dumpText(ionObj)); // prints {GovId:"TOYENC486FH",FirstName:"Brent"}
```

For more information about Ion, see the [Amazon Ion documentation](#) on GitHub. For more code examples of working with Ion in QLDB, see [Working with Amazon Ion data types in Amazon QLDB](#).

Amazon QLDB driver for Python

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

To work with data in your ledger, you can connect to Amazon QLDB from your Python application by using an AWS provided driver. The following topics describe how to get started with the QLDB driver for Python.

Topics

- [Driver resources](#)
- [Prerequisites](#)
- [Installation](#)
- [Amazon QLDB driver for Python – Quick start tutorial](#)
- [Amazon QLDB driver for Python – Cookbook reference](#)

Driver resources

For more information about the functionality supported by the Python driver, see the following resources:

- API reference: [3.x](#), [2.x](#)
- [Driver source code \(GitHub\)](#)
- [Sample application source code \(GitHub\)](#)
- [Amazon Ion code examples](#)

Prerequisites

Before you get started with the QLDB driver for Python, you must do the following:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#). This includes the following:
 1. Sign up for AWS.
 2. Create a user with the appropriate QLDB permissions.
 3. Grant programmatic access for development.
2. Install one of the following versions of Python from the [Python downloads](#) site:
 - **3.6 or later** – QLDB driver for Python v3
 - **3.4 or later** – QLDB driver for Python v2

3. Set up your AWS credentials and your default AWS Region. For instructions, see [Quickstart](#) in the AWS SDK for Python (Boto3) documentation.

For a complete list of available Regions, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

Next, you can download the complete tutorial sample application—or you can install only the driver in a Python project and run short code examples.

- To install the QLDB driver and the AWS SDK for Python (Boto3) in an existing project, proceed to [Installation](#).
- To set up a project and run short code examples that demonstrate basic data transactions on a ledger, see the [Quick start tutorial](#).
- To run more in-depth examples of both data and management API operations in the complete tutorial sample application, see the [Python tutorial](#).

Installation

QLDB supports the following driver versions and their Python dependencies.

Driver version	Python version	Status	Latest release date
2.x	3.4 or later	Production release	May 7, 2020
3.x	3.6 or later	Production release	October 28, 2021

To install the QLDB driver from PyPI using `pip` (a package manager for Python), enter the following at the command line.

3.x

```
pip install pyqldb
```

2.x

```
pip install pyqldb==2.0.2
```

Installing the driver also installs its dependencies, including the [AWS SDK for Python \(Boto3\)](#) and [Amazon Ion](#) packages.

Using the driver to connect to a ledger

Then you can import the driver and use it to connect to a ledger. The following Python code example shows how to create a session for a specified ledger name.

3.x

```
from pyqldb.driver.qldb_driver import QldbDriver
qldb_driver = QldbDriver(ledger_name='testLedger')

for table in qldb_driver.list_tables():
    print(table)
```

2.x

```
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver

qldb_driver = PooledQldbDriver(ledger_name='testLedger')
qldb_session = qldb_driver.get_session()

for table in qldb_session.list_tables():
    print(table)
```

For short code examples of how to run basic data transactions on a ledger, see the [Cookbook reference](#).

Amazon QLDB driver for Python – Quick start tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this tutorial, you learn how to set up a simple application using the latest version of the Amazon QLDB driver for Python. This guide includes steps for installing the driver and short code examples

of basic *create*, *read*, *update*, and *delete* (CRUD) operations. For more in-depth examples that demonstrate these operations in a full sample application, see the [Python tutorial](#).

Topics

- [Prerequisites](#)
- [Step 1: Set up your project](#)
- [Step 2: Initialize the driver](#)
- [Step 3: Create a table and an index](#)
- [Step 4: Insert a document](#)
- [Step 5: Query the document](#)
- [Step 6: Update the document](#)
- [Running the complete application](#)

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the [Prerequisites](#) for the Python driver, if you haven't already done so. This includes signing up for AWS, granting programmatic access for development, and installing Python version 3.6 or later.
2. Create a ledger named `quick-start`.

To learn how to create a ledger, see [Basic operations for Amazon QLDB ledgers](#) or [Step 1: Create a new ledger](#) in *Getting started with the console*.

Step 1: Set up your project

First, set up your Python project.

Note

If you use an IDE that has features to automate these setup steps, you can skip ahead to [Step 2: Initialize the driver](#).

1. Create a folder for your application.

```
$ mkdir myproject
$ cd myproject
```

2. To install the QLDB driver for Python from PyPI, enter the following pip command.

```
$ pip install pyqldb
```

Installing the driver also installs its dependencies, including the [AWS SDK for Python \(Boto3\)](#) and [Amazon Ion](#) packages.

3. Create a new file named `app.py`.

Then, incrementally add the code examples in the following steps to try some basic CRUD operations. Or, you can skip the step-by-step tutorial and instead run the [complete application](#).

Step 2: Initialize the driver

Initialize an instance of the driver that connects to the ledger named `quick-start`. Add the following code to your `app.py` file.

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

# Configure retry limit to 3
retry_config = RetryConfig(retry_limit=3)

# Initialize the driver
print("Initializing the driver")
qldb_driver = QldbDriver("quick-start", retry_config=retry_config)
```

Step 3: Create a table and an index

The following code example shows how to run `CREATE TABLE` and `CREATE INDEX` statements.

Add the following code that creates a table named `People` and an index for the `lastName` field on that table. [Indexes](#) are required to optimize query performance and help to limit [optimistic concurrency control \(OCC\)](#) conflict exceptions.

```
def create_table(transaction_executor):
```

```
print("Creating a table")
transaction_executor.execute_statement("Create TABLE People")

def create_index(transaction_executor):
    print("Creating an index")
    transaction_executor.execute_statement("CREATE INDEX ON People(lastName)")

# Create a table
qldb_driver.execute_lambda(lambda executor: create_table(executor))

# Create an index on the table
qldb_driver.execute_lambda(lambda executor: create_index(executor))
```

Step 4: Insert a document

The following code example shows how to run an INSERT statement. QLDB supports the [PartiQL](#) query language (SQL compatible) and the [Amazon Ion](#) data format (superset of JSON).

Add the following code that inserts a document into the People table.

```
def insert_documents(transaction_executor, arg_1):
    print("Inserting a document")
    transaction_executor.execute_statement("INSERT INTO People ?", arg_1)

# Insert a document
doc_1 = { 'firstName': "John",
          'lastName': "Doe",
          'age': 32,
        }

qldb_driver.execute_lambda(lambda x: insert_documents(x, doc_1))
```

This example uses a question mark (?) as a variable placeholder to pass the document information to the statement. The `execute_statement` method supports values in both Amazon Ion types and Python native types.

Tip

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [list](#) to the statement as follows.

```
# people is a list
```

```
transaction_executor.execute_statement("INSERT INTO Person ?", people)
```

You don't enclose the variable placeholder (?) in double angle brackets (<<...>>) when passing a list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Step 5: Query the document

The following code example shows how to run a SELECT statement.

Add the following code that queries a document from the `People` table.

```
def read_documents(transaction_executor):
    print("Querying the table")
    cursor = transaction_executor.execute_statement("SELECT * FROM People WHERE
lastName = ?", 'Doe')

    for doc in cursor:
        print(doc["firstName"])
        print(doc["lastName"])
        print(doc["age"])

# Query the table
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

Step 6: Update the document

The following code example shows how to run an UPDATE statement.

1. Add the following code that updates a document in the `People` table by updating age to 42.

```
def update_documents(transaction_executor, age, lastName):
    print("Updating the document")
    transaction_executor.execute_statement("UPDATE People SET age = ? WHERE
lastName = ?", age, lastName)

# Update the document
age = 42
lastName = 'Doe'
```

```
qldb_driver.execute_lambda(lambda x: update_documents(x, age, lastName))
```

2. Query the table again to see the updated value.

```
# Query the updated document
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

3. To run the application, enter the following command from your project directory.

```
$ python app.py
```

Running the complete application

The following code example is the complete version of the `app.py` application. Instead of doing the previous steps individually, you can also copy and run this code example from start to end. This application demonstrates some basic CRUD operations on the ledger named `quick-start`.

Note

Before you run this code, make sure that you don't already have an active table named `People` in the `quick-start` ledger.

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

def create_table(transaction_executor):
    print("Creating a table")
    transaction_executor.execute_statement("CREATE TABLE People")

def create_index(transaction_executor):
    print("Creating an index")
    transaction_executor.execute_statement("CREATE INDEX ON People(lastName)")

def insert_documents(transaction_executor, arg_1):
    print("Inserting a document")
    transaction_executor.execute_statement("INSERT INTO People ?", arg_1)

def read_documents(transaction_executor):
    print("Querying the table")
```

```
    cursor = transaction_executor.execute_statement("SELECT * FROM People WHERE
lastName = ?", 'Doe')

    for doc in cursor:
        print(doc["firstName"])
        print(doc["lastName"])
        print(doc["age"])

def update_documents(transaction_executor, age, lastName):
    print("Updating the document")
    transaction_executor.execute_statement("UPDATE People SET age = ? WHERE lastName
= ?", age, lastName)

# Configure retry limit to 3
retry_config = RetryConfig(retry_limit=3)

# Initialize the driver
print("Initializing the driver")
qldb_driver = QldbDriver("quick-start", retry_config=retry_config)

# Create a table
qldb_driver.execute_lambda(lambda executor: create_table(executor))

# Create an index on the table
qldb_driver.execute_lambda(lambda executor: create_index(executor))

# Insert a document
doc_1 = { 'firstName': "John",
          'lastName': "Doe",
          'age': 32,
        }

qldb_driver.execute_lambda(lambda x: insert_documents(x, doc_1))

# Query the table
qldb_driver.execute_lambda(lambda executor: read_documents(executor))

# Update the document
age = 42
lastName = 'Doe'

qldb_driver.execute_lambda(lambda x: update_documents(x, age, lastName))
```

```
# Query the table for the updated document
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

To run the complete application, enter the following command from your project directory.

```
$ python app.py
```

Amazon QLDB driver for Python – Cookbook reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This reference guide shows common use cases of the Amazon QLDB driver for Python. It provides Python code examples that demonstrate how to use the driver to run basic *create*, *read*, *update*, and *delete* (CRUD) operations. It also includes code examples for processing Amazon Ion data. In addition, this guide highlights best practices for making transactions idempotent and implementing uniqueness constraints.

Note

Where applicable, some use cases have different code examples for each supported major version of the QLDB driver for Python.

Contents

- [Importing the driver](#)
- [Instantiating the driver](#)
- [CRUD operations](#)
 - [Creating tables](#)
 - [Creating indexes](#)
 - [Reading documents](#)
 - [Using query parameters](#)

- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)
- [Running multiple statements in a transaction](#)
- [Retry logic](#)
- [Implementing uniqueness constraints](#)
- [Working with Amazon Ion](#)
 - [Importing the Ion module](#)
 - [Creating Ion types](#)
 - [Getting an Ion binary dump](#)
 - [Getting an Ion text dump](#)

Importing the driver

The following code example imports the driver.

3.x

```
from pyqldb.driver.qldb_driver import QldbDriver
import amazon.ion.simpleion as simpleion
```

2.x

```
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
import amazon.ion.simpleion as simpleion
```

Note

This example also imports the Amazon Ion package (`amazon.ion.simpleion`). You need this package to process Ion data when running some data operations in this reference. To learn more, see [Working with Amazon Ion](#).

Instantiating the driver

The following code example creates an instance of the driver that connects to a specified ledger name using default settings.

3.x

```
qldb_driver = QldbDriver(ledger_name='vehicle-registration')
```

2.x

```
qldb_driver = PooledQldbDriver(ledger_name='vehicle-registration')
```

CRUD operations

QLDB runs *create*, *read*, *update*, and *delete* (CRUD) operations as part of a transaction.

Warning

As a best practice, make your write transactions strictly idempotent.

Making transactions idempotent

We recommend that you make write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can run multiple times and produce identical results each time.

For example, consider a transaction that inserts a document into a table named `Person`. The transaction should first check whether or not the document already exists in the table. Without this check, the table might end up with duplicate documents.

Suppose that QLDB successfully commits the transaction on the server side, but the client times out while waiting for a response. If the transaction isn't idempotent, the same document could be inserted more than once in the case of a retry.

Using indexes to avoid full table scans

We also recommend that you run statements with a `WHERE` predicate clause using an *equality* operator on an indexed field or a document ID; for example, `WHERE indexedField = 123` or

WHERE indexedField IN (456, 789). Without this indexed lookup, QLDB needs to do a table scan, which can lead to transaction timeouts or *optimistic concurrency control* (OCC) conflicts.

For more information about OCC, see [Amazon QLDB concurrency model](#).

Implicitly created transactions

The [pyqldb.driver.qldb_driver.execute_lambda](#) method accepts a lambda function that receives an instance of [pyqldb.execution.executor.Executor](#), which you can use to run statements. The instance of `Executor` wraps an implicitly created transaction.

You can run statements within the lambda function by using the [execute_statement](#) method of the transaction executor. The driver implicitly commits the transaction when the lambda function returns.

Note

The `execute_statement` method supports both Amazon Ion types and Python native types. If you pass a Python native type as an argument to `execute_statement`, the driver converts it to an Ion type using the `amazon.ion.simpleion` module (provided that conversion for the given Python data type is supported). For supported data types and conversion rules, see the [simpleion source code](#).

The following sections show how to run basic CRUD operations, specify custom retry logic, and implement uniqueness constraints.

Contents

- [Creating tables](#)
- [Creating indexes](#)
- [Reading documents](#)
 - [Using query parameters](#)
- [Inserting documents](#)
 - [Inserting multiple documents in one statement](#)
- [Updating documents](#)
- [Deleting documents](#)
- [Running multiple statements in a transaction](#)

- [Retry logic](#)
- [Implementing uniqueness constraints](#)

Creating tables

```
def create_table(transaction_executor):
    transaction_executor.execute_statement("CREATE TABLE Person")

qldb_driver.execute_lambda(lambda executor: create_table(executor))
```

Creating indexes

```
def create_index(transaction_executor):
    transaction_executor.execute_statement("CREATE INDEX ON Person(GovId)")

qldb_driver.execute_lambda(lambda executor: create_index(executor))
```

Reading documents

```
# Assumes that Person table has documents as follows:
# { "GovId": "TOYENC486FH", "FirstName": "Brent" }

def read_documents(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'")

    for doc in cursor:
        print(doc["GovId"]) # prints TOYENC486FH
        print(doc["FirstName"]) # prints Brent

qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

Using query parameters

The following code example uses a native type query parameter.

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?",
    'TOYENC486FH')
```

The following code example uses an lon type query parameter.

```
name = ion.loads('Brent')
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE FirstName
= ?", name)
```

The following code example uses multiple query parameters.

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?
AND FirstName = ?", 'TOYENC486FH', "Brent")
```

The following code example uses a list of query parameters.

```
gov_ids = ['TOYENC486FH', 'R0EE1', 'YH844']
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId IN
(?,?,?)", *gov_ids)
```

Note

When you run a query without an indexed lookup, it invokes a full table scan. In this example, we recommend having an [index](#) on the GovId field to optimize performance. Without an index on GovId, queries can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting documents

The following code example inserts native data types.

```
def insert_documents(transaction_executor, arg_1):
    # Check if doc with GovId:TOYENC486FH exists
    # This is critical to make this transaction idempotent
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
= ?", 'TOYENC486FH')
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", arg_1)
```

```
doc_1 = { 'FirstName': "Brent",
          'GovId': 'TOYENC486FH',
          }

qlldb_driver.execute_lambda(lambda executor: insert_documents(executor, doc_1))
```

The following code example inserts Ion data types.

```
def insert_documents(transaction_executor, arg_1):
    # Check if doc with GovId:TOYENC486FH exists
    # This is critical to make this transaction idempotent
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
    = ?", 'TOYENC486FH')
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", arg_1)

doc_1 = { 'FirstName': 'Brent',
          'GovId': 'TOYENC486FH',
          }

# create a sample Ion doc
ion_doc_1 = simpleion.loads(simpleion.dumps(doc_1))

qlldb_driver.execute_lambda(lambda executor: insert_documents(executor, ion_doc_1))
```

This transaction inserts a document into the Person table. Before inserting, it first checks if the document already exists in the table. **This check makes the transaction idempotent in nature.** Even if you run this transaction multiple times, it won't cause any unintended side effects.

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Inserting multiple documents in one statement

To insert multiple documents by using a single [INSERT](#) statement, you can pass a parameter of type [list](#) to the statement as follows.

```
# people is a list
transaction_executor.execute_statement("INSERT INTO Person ?", people)
```

You don't enclose the variable placeholder (?) in double angle brackets (<< . . . >>) when passing a list. In manual PartiQL statements, double angle brackets denote an unordered collection known as a *bag*.

Updating documents

The following code example uses native data types.

```
def update_documents(transaction_executor, gov_id, name):
    transaction_executor.execute_statement("UPDATE Person SET FirstName = ? WHERE
    GovId = ?", name, gov_id)

gov_id = 'TOYENC486FH'
name = 'John'

qldb_driver.execute_lambda(lambda executor: update_documents(executor, gov_id, name))
```

The following code example uses Ion data types.

```
def update_documents(transaction_executor, gov_id, name):
    transaction_executor.execute_statement("UPDATE Person SET FirstName = ? WHERE GovId
    = ?", name, gov_id)

# Ion datatypes
gov_id = simpleion.loads('TOYENC486FH')
name = simpleion.loads('John')

qldb_driver.execute_lambda(lambda executor: update_documents(executor, gov_id, name))
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Deleting documents

The following code example uses native data types.

```
def delete_documents(transaction_executor, gov_id):
    cursor = transaction_executor.execute_statement("DELETE FROM Person WHERE GovId
    = ?", gov_id)

gov_id = 'TOYENC486FH'

qldb_driver.execute_lambda(lambda executor: delete_documents(executor, gov_id))
```

The following code example uses Ion data types.

```
def delete_documents(transaction_executor, gov_id):
    cursor = transaction_executor.execute_statement("DELETE FROM Person WHERE GovId
    = ?", gov_id)

# Ion datatypes
gov_id = simpleion.loads('TOYENC486FH')

qldb_driver.execute_lambda(lambda executor: delete_documents(executor, gov_id))
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Running multiple statements in a transaction

```
# This code snippet is intentionally trivial. In reality you wouldn't do this because
# you'd
# set your UPDATE to filter on vin and insured, and check if you updated something or
# not.

def do_insure_car(transaction_executor, vin):
    cursor = transaction_executor.execute_statement(
        "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
    first_record = next(cursor, None)
    if first_record:
        transaction_executor.execute_statement(
            "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
        return True
    else:
        return False

def insure_car(qlldb_driver, vin_to_insure):
    return qlldb_driver.execute_lambda(
        lambda executor: do_insure_car(executor, vin_to_insure))
```

Retry logic

The driver's `execute_lambda` method has a built-in retry mechanism that retries the transaction if a retryable exception occurs (such as timeouts or OCC conflicts).

3.x

The maximum number of retry attempts and the backoff strategy are configurable.

The default retry limit is 4, and the default backoff strategy is [Exponential Backoff and Jitter](#) with a base of 10 milliseconds. You can set the retry configuration per driver instance and also per transaction by using an instance of [pyqlldb.config.retry_config.RetryConfig](#).

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for an instance of the driver.

```
from pyqlldb.config.retry_config import RetryConfig
from pyqlldb.driver.qlldb_driver import QldbDriver

# Configuring retry limit to 2
```



```

retry_config = RetryConfig(retry_limit=2)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config)

# Configuring a custom backoff which increases delay by 1s for each attempt.
def custom_backoff(retry_attempt, error, transaction_id):
    return 1000 * retry_attempt

retry_config_custom_backoff = RetryConfig(retry_limit=2,
    custom_backoff=custom_backoff)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config_custom_backoff)

```

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for a particular lambda execution. This configuration for `execute_lambda` overrides the retry logic that is set for the driver instance.

```

from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

# Configuring retry limit to 2
retry_config_1 = RetryConfig(retry_limit=4)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config_1)

# Configuring a custom backoff which increases delay by 1s for each attempt.
def custom_backoff(retry_attempt, error, transaction_id):
    return 1000 * retry_attempt

retry_config_2 = RetryConfig(retry_limit=2, custom_backoff=custom_backoff)

# The config `retry_config_1` will be overridden by `retry_config_2`
qldb_driver.execute_lambda(lambda txn: txn.execute_statement("CREATE TABLE Person"),
    retry_config_2)

```

2.x

The maximum number of retry attempts is configurable. You can configure the retry limit by setting the `retry_limit` property when initializing `PooledQldbDriver`.

The default retry limit is 4.

Implementing uniqueness constraints

QLDB doesn't support unique indexes, but you can implement this behavior in your application.

Suppose that you want to implement a uniqueness constraint on the GovId field in the Person table. To do this, you can write a transaction that does the following:

1. Assert that the table has no existing documents with a specified GovId.
2. Insert the document if the assertion passes.

If a competing transaction concurrently passes the assertion, only one of the transactions will commit successfully. The other transaction will fail with an OCC conflict exception.

The following code example shows how to implement this uniqueness constraint logic.

```
def insert_documents(transaction_executor, gov_id, document):
    # Check if doc with GovId = gov_id exists
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?", gov_id)
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", document)

qlldb_driver.execute_lambda(lambda executor: insert_documents(executor, gov_id, document))
```

Note

In this example, we recommend having an index on the GovId field to optimize performance. Without an index on GovId, statements can have more latency and can also lead to OCC conflict exceptions or transaction timeouts.

Working with Amazon Ion

The following sections show how to use the Amazon Ion module to process Ion data.

Contents

- [Importing the Ion module](#)

- [Creating Ion types](#)
- [Getting an Ion binary dump](#)
- [Getting an Ion text dump](#)

Importing the Ion module

```
import amazon.ion.simpleion as simpleion
```

Creating Ion types

The following code example creates an Ion object from Ion text.

```
ion_text = '{GovId: "TOYENC486FH", FirstName: "Brent"}'  
ion_obj = simpleion.loads(ion_text)  
  
print(ion_obj['GovId']) # prints TOYENC486FH  
print(ion_obj['Name']) # prints Brent
```

The following code example creates an Ion object from a Python dict.

```
a_dict = { 'GovId': 'TOYENC486FH',  
           'FirstName': "Brent"  
         }  
ion_obj = simpleion.loads(simpleion.dumps(a_dict))  
  
print(ion_obj['GovId']) # prints TOYENC486FH  
print(ion_obj['FirstName']) # prints Brent
```

Getting an Ion binary dump

```
# ion_obj is an Ion struct  
print(simpleion.dumps(ion_obj)) # b'\xe0\x01\x00\xea\xee\x97\x81\x83\xde\x93\x87\xbe  
\x90\x85GovId\x89FirstName\xde\x94\x8a\x8bTOYENC486FH\x8b\x85Brent'
```

Getting an Ion text dump

```
# ion_obj is an Ion struct  
print(simpleion.dumps(ion_obj, binary=False)) # prints $ion_1_0  
{GovId:'TOYENC486FH',FirstName:"Brent"}
```

For more information about working with Ion, see the [Amazon Ion documentation](#) on GitHub. For more code examples of working with Ion in QLDB, see [Working with Amazon Ion data types in Amazon QLDB](#).

Understanding session management with the driver in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

If you have experience using a relational database management system (RDBMS), you might be familiar with concurrent connections. QLDB doesn't have the same concept of a traditional RDBMS connection because transactions are run with HTTP request and response messages.

In QLDB, the analogous concept is an *active session*. A session is conceptually similar to a user login—it manages information about your data transaction requests to a ledger. An active session is one that is actively running a transaction. It can also be a session that recently finished a transaction where the service anticipates it will start another transaction immediately. QLDB supports one actively running transaction per session.

The limit of concurrent active sessions per ledger is defined in [Quotas and limits in Amazon QLDB](#). After this limit is reached, any session that tries to start a transaction will result in an error (`LimitExceededException`).

For best practices for configuring a session pool in your application using the QLDB driver, see [Configuring the QldbDriver object](#) in *Amazon QLDB driver recommendations*.

Contents

- [Session lifecycle](#)
- [Session expiration](#)
- [Session handling in the QLDB driver](#)
 - [Session pooling overview](#)

- [Session pooling and transaction logic](#)
- [Returning sessions to the pool](#)

Session lifecycle

The following sequence of [QLDB Session API](#) operations represents the typical lifecycle of a QLDB session:

1. `StartSession`
2. `StartTransaction`
3. `ExecuteStatement`
4. `CommitTransaction`
5. Repeat steps 2–4 to start more transactions (one transaction at a time).
6. `EndSession`

Session expiration

QLDB expires and discards a session after a total lifetime of **13–17 minutes**, regardless of whether it's actively running a transaction. Sessions can be lost or impaired for a number of reasons, such as hardware failure, network failure, or application restarts. QLDB enforces a maximum lifetime on sessions to ensure that the client application is resilient to session failure.

Session handling in the QLDB driver

Although you can use an AWS SDK to interact directly with the *QLDB Session API*, this adds complexity and requires you to compute a commit digest. QLDB uses this commit digest to ensure transaction integrity. Instead of interacting directly with this API, we recommend using the QLDB driver.

The driver provides a high-level abstraction layer above the transactional data API. It streamlines the process of running [PartiQL](#) statements on ledger data by managing [SendCommand](#) API calls. These API calls require several parameters that the driver handles for you, including the management of sessions, transactions, and retry policy in case of errors.

Topics

- [Session pooling overview](#)
- [Session pooling and transaction logic](#)
- [Returning sessions to the pool](#)

Session pooling overview

In older versions of the QLDB driver (for example, [Java v1.1.0](#)), we provided two implementations of the driver object: a standard, nonpooled `QldbDriver` and a `PooledQldbDriver`. As the name suggests, the `PooledQldbDriver` maintains a pool of sessions that are reused across transactions.

Based on user feedback, developers prefer to get the pooling feature and its benefits by default, instead of using the standard driver. So, we removed the `PooledQldbDriver` and moved the session-pooling functionality to `QldbDriver`. This change is included in the latest release of each driver (for example, [Java v2.0.0](#)).

The driver provides three levels of abstractions:

- **Driver** (pooled driver implementation) – The top-level abstraction. The driver maintains and manages a pool of sessions. When you ask the driver to run a transaction, the driver chooses a session from the pool and uses it to run the transaction. If the transaction fails because of a session error (`InvalidSessionException`), the driver uses another session to retry the transaction. Essentially, the driver offers a fully managed session experience.
- **Session** – One level below the driver abstraction. A session is contained in a pool, and the driver manages the lifecycle of the session. If a transaction fails, the driver retries it up to a specified number of attempts using the same session. But if the session encounters an error (`InvalidSessionException`), QLDB discards it internally. The driver is then responsible for getting another session from the pool to retry the transaction.
- **Transaction** – The lowest level of abstraction. A transaction is contained in a session, and the session manages the lifecycle of the transaction. The session is responsible for retrying the transaction in the case of an error. The session also ensures that it doesn't leak an open transaction that has not been committed or canceled.

In the latest version of each driver, you can perform operations at the driver level of abstraction only. You have no direct control over individual sessions and transactions (that is, there are no API operations to manually start a new session or transaction).

Session pooling and transaction logic

The latest release of each driver no longer provides a nonpooled implementation of the driver object. By default, the `QldbDriver` object manages the session pool. When you make a call to run a transaction, the driver does the following steps:

1. The driver checks if it has reached the session-pool limit. If so, the driver immediately throws a `NoSessionAvailable` exception. Otherwise, it proceeds to the next step.
2. The driver checks if the pool has an available session.
 - If a session is available in the pool, the driver uses it to run a transaction.
 - If no session is available in the pool, the driver creates a new session and uses it to run a transaction.
3. After the driver gets a session in step 2, the driver calls the `execute` operation on the session instance.
4. In the `execute` operation of the session, the driver tries to start a transaction by calling `startTransaction`.
 - If the session isn't valid, the `startTransaction` call fails, and the driver returns to step 1.
 - If the `startTransaction` call succeeds, the driver proceeds to the next step.
5. The driver runs your lambda expression. This lambda expression can contain one or more calls to run PartiQL statements. After the lambda expression finishes running without any errors, the driver proceeds to commit the transaction.
6. The commit of the transaction can have one of two results:
 - The commit succeeds, and the driver returns control to your application code.
 - The commit fails due to an optimistic concurrency control (OCC) conflict. In this case, the driver retries steps 4–6 using the same session. The maximum number of retry attempts is configurable in your application code. The default limit is 4.

Note

If an `InvalidSessionException` is thrown during steps 4–6, the driver marks the session as closed and returns to step 1 to retry the transaction.

If any other exception is thrown during steps 4–6, the driver checks if the exception can be retried. If so, it retries the transaction up to the specified number of retry attempts. If not, it propagates (bubbles up and throws) the exception to your application code.

Returning sessions to the pool

If an `InvalidSessionException` occurs at any time during the course of an active transaction, the driver doesn't return the session to the pool. QLDB discards the session instead, and the driver gets another session from the pool. In all other cases, the driver returns the session to the pool.

Amazon QLDB driver recommendations

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes best practices for configuring and using the Amazon QLDB driver for any supported language. The code examples provided are specifically for Java.

These recommendations apply for most typical use cases, but one size doesn't fit all. Use the following recommendations as you see fit for your application.

Topics

- [Configuring the `QldbDriver` object](#)
- [Retrying on exceptions](#)
- [Optimizing performance](#)
- [Running multiple statements per transaction](#)

Configuring the `QldbDriver` object

The `QldbDriver` object manages connections to your ledger by maintaining a pool of *sessions* that are reused across transactions. A [session](#) represents a single connection to the ledger. QLDB supports one actively running transaction per session.

⚠ Important

For older driver versions, the session pooling functionality is still in the `PooledQldbDriver` object instead of `QldbDriver`. If you're using one of the following versions, replace any mentions of `QldbDriver` with `PooledQldbDriver` for the rest of this topic.

Driver	Version
Java	1.1.0 or earlier
.NET	0.1.0-beta
Node.js	1.0.0-rc.1 or earlier
Python	2.0.2 or earlier

The `PooledQldbDriver` object is deprecated in the latest version of the drivers. We recommend that you upgrade to the latest version and convert any instances of `PooledQldbDriver` to `QldbDriver`.

Configure QldbDriver as a global object

To optimize the use of drivers and sessions, ensure that only one global instance of the driver exists in your application instance. For example in Java, you can use *dependency injection* frameworks such as [Spring](#), [Google Guice](#), or [Dagger](#). The following code example shows how to configure `QldbDriver` as a singleton.

```
@Singleton
public QldbDriver qldbDriver (AWSCredentialsProvider credentialsProvider,
                              @Named(LEDGER_NAME_CONFIG_PARAM) String ledgerName)
{
    QldbSessionClientBuilder builder = QldbSessionClient.builder();
    if (null != credentialsProvider) {
        builder.credentialsProvider(credentialsProvider);
    }
    return QldbDriver.builder()
        .ledger(ledgerName)
```

```
        .transactionRetryPolicy(RetryPolicy
            .builder()
            .maxRetries(3)
            .build())
        .sessionClientBuilder(builder)
        .build();
    }
```

Configure the retry attempts

The driver automatically retries transactions when common transient exceptions (such as `SocketTimeoutException` or `NoHttpResponseException`) occur. To set the maximum number of retry attempts, you can use the `maxRetries` parameter of the `transactionRetryPolicy` configuration object when creating an instance of `QldbDriver`. (For older driver versions as listed in the previous section, use the `retryLimit` parameter of `PooledQldbDriver`.)

The default value of `maxRetries` is 4.

Client-side errors such as `InvalidParameterException` can't be retried. When they occur, the transaction is aborted, the session is returned to the pool, and the exception is thrown to the driver's client.

Configure the maximum number of concurrent sessions and transactions

The maximum number of ledger sessions that are used by an instance of `QldbDriver` to run transactions is defined by its `maxConcurrentTransactions` parameter. (For older driver versions as listed in the previous section, this is defined by the `poolLimit` parameter of `PooledQldbDriver`.)

This limit must be greater than zero and less than or equal to the maximum number of open HTTP connections that the session client allows, as defined by the specific AWS SDK. For example in Java, the maximum number of connections is set in the [ClientConfiguration](#) object.

The default value of `maxConcurrentTransactions` is the maximum connection setting of your AWS SDK.

When you configure the `QldbDriver` in your application, take the following scaling considerations:

- Your pool should always have at least as many sessions as the number of concurrently running transactions that you plan to have.

- In a multi-threaded model where a supervisor thread delegates to worker threads, the driver should have at least as many sessions as the number of worker threads. Otherwise, at peak load, threads will be waiting in line for an available session.
- The service limit of concurrent active sessions per ledger is defined in [Quotas and limits in Amazon QLDB](#). Ensure that you don't configure more than this limit of concurrent sessions to be used for a single ledger across all clients.

Retrying on exceptions

When retrying on exceptions that occur in QLDB, consider the following recommendations.

Retrying on `OccConflictException`

Optimistic concurrency control (OCC) conflict exceptions occur when the data that the transaction is accessing has changed since the start of the transaction. QLDB throws this exception while trying to commit the transaction. The driver retries the transaction up to as many times as `maxRetries` is configured.

For more information about OCC and best practices for using indexes to limit OCC conflicts, see [Amazon QLDB concurrency model](#).

Retrying on other exceptions outside of `QldbDriver`

To retry a transaction outside of the driver when custom, application-defined exceptions are thrown during runtime, you must wrap the transaction. For example in Java, the following code shows how to use the [Resilience4J](#) library to retry a transaction in QLDB.

```
private final RetryConfig retryConfig = RetryConfig.custom()
    .maxAttempts(MAX_RETRIES)
    .intervalFunction(IntervalFunction.ofExponentialRandomBackoff())
    // Retry this exception
    .retryExceptions(InvalidSessionException.class, MyRetryableException.class)
    // But fail for any other type of exception extended from RuntimeException
    .ignoreExceptions(RuntimeException.class)
    .build();

// Method callable by a client
public void myTransactionWithRetries(Params params) {
    Retry retry = Retry.of("registerDriver", retryConfig);

    Function<Params, Void> transactionFunction = Retry.decorateFunction(
```

```
        retry,
        parameters -> transactionNoReturn(params));
transactionFunction.apply(params);
}

private Void transactionNoReturn(Params params) {
    try (driver.execute(txn -> {
        // Transaction code
    }));
}
return null;
}
```

Note

Retrying a transaction outside of the QLDB driver has a multiplier effect. For example, if `QldbDriver` is configured to retry three times, and the custom retry logic also retries three times, the same transaction can be retried up to nine times.

Making transactions idempotent

As a best practice, make your write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can run multiple times and produce identical results each time.

To learn more, see [Amazon QLDB concurrency model](#).

Optimizing performance

To optimize performance when you run transactions using the driver, take the following considerations:

- The execute operation always makes a minimum of three `SendCommand` API calls to QLDB, including the following commands:
 1. `StartTransaction`
 2. `ExecuteStatement`

This command is invoked for each PartiQL statement that you run in the execute block.
 3. `CommitTransaction`

Consider the total number of API calls that are made when you calculate the overall workload of your application.

- In general, we recommend starting with a single-threaded writer and optimizing transactions by batching multiple statements within a single transaction. Maximize the quotas on transaction size, document size, and number of documents per transaction, as defined in [Quotas and limits in Amazon QLDB](#).
- If batching isn't sufficient for large transaction loads, you can try multi-threading by adding additional writers. However, you should carefully consider your application requirements for document and transaction sequencing and the additional complexity that this introduces.

Running multiple statements per transaction

As described in the [previous section](#), you can run multiple statements per transaction to optimize performance of your application. In the following code example, you query a table and then update a document in that table within a transaction. You do this by passing a lambda expression to the `execute` operation.

Java

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
// because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
// or not.
public static boolean InsureCar(qlldbDriver qlldbDriver, final String vin) {
    final IonSystem ionSystem = IonSystemBuilder.standard().build();
    final IonString ionVin = ionSystem.newString(vin);

    return qlldbDriver.execute(txn -> {
        Result result = txn.execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
            ionVin);
        if (!result.isEmpty()) {
            txn.execute("UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

.NET

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
// because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
// or not.
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    ValueFactory valueFactory = new ValueFactory();
    IIonValue ionVin = valueFactory.NewString(vin);

    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.IAsyncResult result = await txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
            ionVin);

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

Go

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
// because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
// or not.
func InsureCar(driver *qldbdriver.QLDBDriver, vin string) (bool, error) {
    insured, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {

        result, err := txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
        if err != nil {
            return false, err
        }
    })
}
```

```

    }

    hasNext := result.Next(txn)
    if !hasNext && result.Err() != nil {
        return false, result.Err()
    }

    if hasNext {
        _, err = txn.Execute(
            "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
        if err != nil {
            return false, err
        }
        return true, nil
    }
    return false, nil
}))
if err != nil {
    panic(err)
}

return insured.(bool), err
}

```

Node.js

```

// This code snippet is intentionally trivial. In reality you wouldn't do this
// because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
// or not.
async function insureCar(driver: QldbDriver, vin: string): Promise<boolean> {

    return await driver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
            vin)).getResultList();

        if (results.length > 0) {
            await txn.execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin);
            return true;
        }
        return false;
    })
}

```

```
});  
};
```

Python

```
# This code snippet is intentionally trivial. In reality you wouldn't do this  
# because you'd  
# set your UPDATE to filter on vin and insured, and check if you updated something  
# or not.  
  
def do_insure_car(transaction_executor, vin):  
    cursor = transaction_executor.execute_statement(  
        "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)  
    first_record = next(cursor, None)  
    if first_record:  
        transaction_executor.execute_statement(  
            "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)  
        return True  
    else:  
        return False  
  
def insure_car(qlldb_driver, vin_to_insure):  
    return qlldb_driver.execute_lambda(  
        lambda executor: do_insure_car(executor, vin_to_insure))
```

The driver's `execute` operation implicitly starts a session and a transaction in that session. Each statement that you run in the lambda expression is wrapped in the transaction. After all of the statements run, the driver auto-commits the transaction. If any statement fails after the automatic retry limit is exhausted, the transaction is aborted.

Propagate exceptions in a transaction

When running multiple statements per transaction, we generally don't recommend that you catch and swallow exceptions within the transaction.

For example in Java, the following program catches any instance of `RuntimeException`, logs the error, and continues. This code example is considered bad practice because the transaction succeeds even when the `UPDATE` statement fails. So, the client might assume that the update succeeded when it didn't.

⚠ Warning

Don't use this code example. It's provided to show an anti-pattern example that is considered bad practice.

```
// DO NOT USE this code example because it is considered bad practice
public static void main(final String... args) {
    ConnectToLedger.getDriver().execute(txn -> {
        final Result selectTableResult = txn.execute("SELECT * FROM Vehicle WHERE VIN
='123456789'");
        // Catching an error inside the transaction is an anti-pattern because the
operation might
        // not succeed.
        // In this example, the transaction succeeds even when the update statement
fails.
        // So, the client might assume that the update succeeded when it didn't.
        try {
            processResults(selectTableResult);
            String model = // some code that extracts the model
            final Result updateResult = txn.execute("UPDATE Vehicle SET model = ? WHERE
VIN = '123456789'",
                Constants.MAPPER.writeValueAsIonValue(model));
        } catch (RuntimeException e) {
            log.error("Exception when updating the Vehicle table {}", e.getMessage());
        }
    });
    log.info("Vehicle table updated successfully.");
}
```

Propagate (bubble up) the exception instead. If any part of the transaction fails, let the execute operation abort the transaction so that the client can handle the exception accordingly.

Understanding retry policy with the driver in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The Amazon QLDB driver uses a retry policy to handle transient exceptions by transparently retrying a failed transaction. These exceptions, such as `CapacityExceededException` and `RateExceededException`, typically correct themselves after a period of time. If the transaction that failed with the exception is retried after a suitable delay, it's likely to succeed. This helps to improve the stability of the application that uses QLDB.

Topics

- [Types of retryable errors](#)
- [Default retry policy](#)

Types of retryable errors

The driver automatically retries a transaction if and only if any of the following exceptions occur during an operation within that transaction:

- [CapacityExceededException](#) – Returned when the request exceeds the processing capacity of the ledger.
- [InvalidSessionException](#) – Returned when a session is no longer valid or if the session doesn't exist.
- [LimitExceededException](#) – Returned if a resource limit such as number of active sessions is exceeded.
- [OccConflictException](#) – Returned when a transaction can't be written to the journal due to a failure in the verification phase of *optimistic concurrency control* (OCC).
- [RateExceededException](#) – Returned when the rate of requests exceeds the allowed throughput.

Default retry policy

The retry policy consists of a retry condition and a backoff strategy. The retry condition defines when a transaction should be retried, while the backoff strategy defines how long to wait before retrying the transaction.

When creating an instance of the driver, the default retry policy specifies to retry up to four times, and to use an exponential backoff strategy. The exponential backoff strategy uses a minimum delay of 10 milliseconds and a maximum delay of 5000 milliseconds, with equal jitter. If the transaction can't be committed successfully within the retry policy, we recommend trying the transaction at another time.

The concept of exponential backoff is to use progressively longer wait times between retries for consecutive error responses. For more information, see the AWS blog post [Exponential Backoff and Jitter](#).

Common errors from the Amazon QLDB driver

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes runtime errors that can be thrown by the Amazon QLDB driver when interacting with the [QLDB Session API](#).

The following is a list of common exceptions that are returned by the driver. Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

CapacityExceededException

Message: Capacity exceeded

Amazon QLDB rejected the request because it exceeded the processing capacity of the ledger. QLDB enforces an internal scaling limit per ledger to maintain the health and performance of the service. This limit varies depending on the workload size of each individual request. For

example, a request can have an increased workload if it performs inefficient data transactions, such as table scans that result from a non-index qualified query.

We recommend that you wait before retrying the request. If your application consistently encounters this exception, optimize your statements and decrease the rate and volume of the requests that you send to the ledger. Examples of statement optimization include running fewer statements per transaction and tuning your table indexes. To learn how to optimize statements and avoid table scans, see [Optimizing query performance](#).

We also recommend using the latest version of the QLDB driver. The driver has a default retry policy that uses [Exponential Backoff and Jitter](#) to automatically retry on exceptions such as this. The concept of exponential backoff is to use progressively longer wait times between retries for consecutive error responses.

InvalidSessionException

Message: Transaction *transactionId* has expired

A transaction exceeded its maximum lifetime. A transaction can run for up to 30 seconds before being committed. After this timeout limit, any work done on the transaction is rejected, and QLDB discards the session. This limit protects the client from leaking sessions by starting transactions and not committing or canceling them.

If this is a common exception in your application, it's likely that transactions are simply taking too long to run. If transaction runtime is taking longer than 30 seconds, optimize your statements to speed up the transactions. Examples of statement optimization include running fewer statements per transaction and tuning your table indexes. For more information, see [Optimizing query performance](#).

InvalidSessionException

Message: Session *sessionId* has expired

QLDB discarded the session because it exceeded its maximum total lifetime. QLDB discards sessions after 13–17 minutes, regardless of an active transaction. Sessions can be lost or impaired for a number of reasons, such as hardware failure, network failure, or application restarts. So, QLDB enforces a maximum lifetime on sessions to ensure that client software is resilient to session failure.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the latest version of the QLDB driver, which manages the session pool and its health on the application's behalf.

InvalidSessionException

Message: No such session

The client tried to transact with QLDB using a session that doesn't exist. Assuming that the client is using a session that previously existed, the session might no longer exist because of one of the following:

- If a session is involved in an internal server failure (that is, an error with HTTP response code 500), QLDB might choose to discard the session completely, rather than allow the customer to transact with a session of uncertain state. Then, any retry attempts on that session fail with this error.
- Expired sessions are eventually forgotten by QLDB. Then, any attempts to continue using the session result in this error, rather than the initial `InvalidSessionException`.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the latest version of the QLDB driver, which manages the session pool and its health on the application's behalf.

RateExceededException

Message: The rate was exceeded

QLDB throttled a client based on the caller's identity. QLDB enforces throttling on a per-Region, per-account basis using a [token bucket](#) throttling algorithm. QLDB does this to help the performance of the service, and to ensure fair usage for all QLDB customers. For example, trying to acquire a large number of concurrent sessions using the `StartSessionRequest` operation might lead to throttling.

To maintain your application health and mitigate further throttling, you can retry on this exception using [Exponential Backoff and Jitter](#). The concept of exponential backoff is to use progressively longer wait times between retries for consecutive error responses. We recommend using the latest version of the QLDB driver. The driver has a default retry policy that uses exponential backoff and jitter to automatically retry on exceptions such as this.

The latest version of the QLDB driver can also help if your application is consistently getting throttled by QLDB for `StartSessionRequest` calls. The driver maintains a pool of sessions that are reused across transactions, which can help to reduce the number of `StartSessionRequest` calls that your application makes. To request an increase in API throttling limits, contact the [AWS Support Center](#).

LimitExceededException

Message: Exceeded the session limit

A ledger exceeded its quota (also known as a *limit*) on the number of active sessions. This quota is defined in [Quotas and limits in Amazon QLDB](#). A ledger's active session count is eventually consistent, and ledgers consistently running near the quota might periodically see this exception.

To maintain your application's health, we recommend retrying on this exception. To avoid this exception, ensure that you have not configured more than 1,500 concurrent sessions to be used for a single ledger across all clients. For example, you can use the [maxConcurrentTransactions](#) method of the [Amazon QLDB driver for Java](#) to configure the maximum number of available sessions in a driver instance.

QldbClientException

Message: A streamed result is only valid when the parent transaction is open

The transaction is closed, and it can't be used to retrieve the results from QLDB. A transaction closes when it's either committed or canceled.

This exception occurs when the client is working directly with the `Transaction` object, and it's trying to retrieve results from QLDB after committing or canceling a transaction. To mitigate this issue, the client must read the data before closing the transaction.

Getting started with Amazon QLDB using a sample application tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this tutorial, you use the Amazon QLDB driver with an AWS SDK to create a QLDB ledger and populate it with sample data. The driver lets your application interact with QLDB using the

transactional data API. The AWS SDK supports interaction with the QLDB resource management API.

The sample ledger that you create in this scenario is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations. The following topics explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to verify a registration document cryptographically, and it concludes by cleaning up resources and deleting the sample ledger.

This sample application tutorial is available for the following programming languages.

Topics

- [Amazon QLDB Java tutorial](#)
- [Amazon QLDB Node.js tutorial](#)
- [Amazon QLDB Python tutorial](#)

Amazon QLDB Java tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this implementation of the tutorial sample application, you use the Amazon QLDB driver with the AWS SDK for Java to create a QLDB ledger and populate it with sample data.

As you work through this tutorial, you can refer to the [AWS SDK for Java API Reference](#) for management API operations. For transactional data operations, you can refer to the [QLDB Driver for Java API Reference](#).

Note

Where applicable, some tutorial steps have different commands or code examples for each supported major version of the QLDB driver for Java.

Topics

- [Installing the Amazon QLDB Java sample application](#)
- [Step 1: Create a new ledger](#)
- [Step 2: Test connectivity to the ledger](#)
- [Step 3: Create tables, indexes, and sample data](#)
- [Step 4: Query the tables in a ledger](#)
- [Step 5: Modify documents in a ledger](#)
- [Step 6: View the revision history for a document](#)
- [Step 7: Verify a document in a ledger](#)
- [Step 8: Export and validate journal data in a ledger](#)
- [Step 9 \(optional\): Clean up resources](#)

Installing the Amazon QLDB Java sample application

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes how to install and run the provided Amazon QLDB sample application for the step-by-step Java tutorial. The use case for this sample application is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations.

The DMV sample application for Java is open source in the GitHub repository [aws-samples/amazon-qldb-dmv-sample-java](#).

Prerequisites

Before you get started, make sure that you complete the QLDB driver for Java [Prerequisites](#). This includes the following:

1. Sign up for AWS.
2. Create a user with the appropriate QLDB permissions. To complete all of the steps in this tutorial, you need full administrative access to your ledger resource through the QLDB API.

3. If you're using an IDE other than AWS Cloud9, install Java and grant programmatic access for development.

Installation

The following steps describe how to download and set up the sample application with a local development environment. Or, you can automate setup of the sample application by using AWS Cloud9 as your IDE, and an AWS CloudFormation template to provision your development resources.

Local development environment

These instructions describe how to download and install the QLDB Java sample application using your own resources and development environment.

To download and run the sample application

1. Enter the following command to clone the sample application from GitHub.

2.x

```
git clone https://github.com/aws-samples/amazon-qldb-dmv-sample-java.git
```

1.x

```
git clone -b v1.2.0 https://github.com/aws-samples/amazon-qldb-dmv-sample-java.git
```

This package includes the Gradle configuration and complete code from the [Java tutorial](#).

2. Load and run the provided application.
 - If you're using Eclipse:
 - a. Start Eclipse, and on the **Eclipse** menu, choose **File, Import**, and then **Existing Gradle Project**.
 - b. In the project root directory, browse and select the application directory that contains the `build.gradle` file. Then, choose **Finish** to use the default Gradle settings for the import.

- c. You can try running the `ListLedgers` program as an example. Open the context (right-click) menu for the `ListLedgers.java` file, and choose **Run as Java Application**.
- If you're using IntelliJ:
 - a. Start IntelliJ, and on the **IntelliJ** menu, choose **File** and then **Open**.
 - b. In the project root directory, browse and select the application directory that contains the `build.gradle` file. Then, choose **OK**. Keep the default settings, and choose **OK** again.
 - c. You can try running the `ListLedgers` program as an example. Open the context (right-click) menu for the `ListLedgers.java` file, and choose **Run 'ListLedgers'**.
3. Proceed to [Step 1: Create a new ledger](#) to start the tutorial and create a ledger.

AWS Cloud9

These instructions describe how to automate setup of the Amazon QLDB vehicle registration sample application for Java, using [AWS Cloud9](#) as your IDE. In this guide, you use an [AWS CloudFormation](#) template to provision your development resources.

For more information about AWS Cloud9, see the [AWS Cloud9 User Guide](#). To learn more about AWS CloudFormation, see the [AWS CloudFormation User Guide](#).

Topics

- [Part 1: Provision your resources](#)
- [Part 2: Set up your IDE](#)
- [Part 3: Run the QLDB DMV sample application](#)

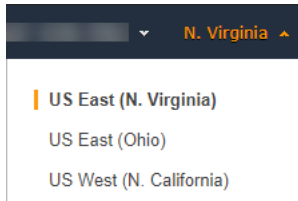
Part 1: Provision your resources

In this first step, you use AWS CloudFormation to provision the resources required to set up your development environment with the Amazon QLDB sample application.

To open the AWS CloudFormation console and load the QLDB sample application template

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.

Switch to a Region that supports QLDB. For a complete list, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*. The following screenshot of the AWS Management Console shows US East (N. Virginia) as the selected AWS Region.



2. On the AWS CloudFormation console, choose **Create stack**, and then choose **With new resources (standard)**.
3. On the **Create stack** page under **Specify template**, choose **Amazon S3 URL**.
4. Enter the following URL, and choose **Next**.

```
https://amazon-qldb-assets.s3.amazonaws.com/templates/QLDB-DMV-SampleApp.yml
```

5. Enter a **Stack name** (such as **qlldb-sample-app**), and choose **Next**.
6. You can add any tags as appropriate and keep the default options. Then choose **Next**.
7. Review your stack settings, and choose **Create stack**. The AWS CloudFormation script might take a few minutes to finish.

This script provisions your AWS Cloud9 environment with an associated Amazon Elastic Compute Cloud (Amazon EC2) instance that you use to run the QLDB sample application in this tutorial. It also clones the [aws-samples/amazon-qlldb-dmv-sample-java](#) repository from GitHub into your AWS Cloud9 development environment.

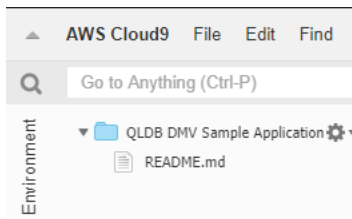
Part 2: Set up your IDE

In this step, you finish the setup of your cloud development environment. You download and run a provided shell script to set up your AWS Cloud9 IDE with sample application's dependencies.

To set up your AWS Cloud9 environment

1. Open the AWS Cloud9 console at <https://console.aws.amazon.com/cloud9/>.
2. Under **Your environments**, locate the card for the environment named **QLDB DMV Sample Application**, and choose **Open IDE**. Your environment might take a minute to load as the underlying EC2 instance launches.

Your AWS Cloud9 environment is preconfigured with the system dependencies that you need to run the tutorial. In the **Environment** navigation pane of your console, confirm that you see a folder named `QLDB DMV Sample Application`. The following screenshot of the AWS Cloud9 console shows the `QLDB DMV Sample Application` environment folder pane.



If you don't see a navigation pane, toggle the **Environment** tab on the left side of your console. If you don't see any folders in the pane, enable **Show Environment Root** using the settings icon



).

3. On the bottom pane of your console, you should see an open bash terminal window. If you don't see this, choose **New Terminal** from the **Window** menu at the top of your console.
4. Next, download and run a setup script to install OpenJDK 8 and—if applicable—check out the appropriate branch from the Git repository. In the AWS Cloud9 terminal that you created in the previous step, run the following two commands in order:

2.x

```
aws s3 cp s3://amazon-qlldb-assets/setup-scripts/dmv-setup-v2.sh .
```

```
sh dmv-setup-v2.sh
```

1.x

```
aws s3 cp s3://amazon-qlldb-assets/setup-scripts/dmv-setup.sh .
```

```
sh dmv-setup.sh
```

Upon completion, you should see the following message printed in the terminal:

```
** DMV Sample App setup completed , enjoy!! **
```

5. Take a moment to browse the sample application code in AWS Cloud9, particularly in the following directory path: `src/main/java/software/amazon/qldb/tutorial`.

Part 3: Run the QLDB DMV sample application

In this step, you learn how to run the Amazon QLDB DMV sample application tasks using AWS Cloud9. To run the sample code, go back to your AWS Cloud9 terminal or create a new terminal window as you did in *Part 2: Set Up Your IDE*.

To run the sample application

1. Run the following command in your terminal to switch to the project root directory:

```
cd ~/environment/amazon-qldb-dmv-sample-java
```

Ensure that you're running the examples in the following directory path.

```
/home/ec2-user/environment/amazon-qldb-dmv-sample-java/
```

2. The following command shows the Gradle syntax to run each task.

```
./gradlew run -Dtutorial=Task
```

For example, run the following command to list all of the ledgers in your AWS account and current Region.

```
./gradlew run -Dtutorial=ListLedgers
```

3. Proceed to [Step 1: Create a new ledger](#) to start the tutorial and create a ledger.
4. (Optional) After you complete the tutorial, clean up your AWS CloudFormation resources if you no longer need them.
 - a. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>, and delete the stack that you created in *Part 1: Provision Your Resources*.
 - b. Also delete the AWS Cloud9 stack that the AWS CloudFormation template created for you.

Step 1: Create a new ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`.

To create a new ledger

1. Review the following file (`Constants.java`), which contains constant values that are used by all of the other programs in this tutorial.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
*/

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;

/**
 * Constant values used throughout this tutorial.
 */
public final class Constants {
    public static final int RETRY_LIMIT = 4;
    public static final String LEDGER_NAME = "vehicle-registration";
    public static final String STREAM_NAME = "vehicle-registration-stream";
    public static final String VEHICLE_REGISTRATION_TABLE_NAME =
"VehicleRegistration";
    public static final String VEHICLE_TABLE_NAME = "Vehicle";
    public static final String PERSON_TABLE_NAME = "Person";
    public static final String DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
    public static final String VIN_INDEX_NAME = "VIN";
    public static final String PERSON_GOV_ID_INDEX_NAME = "GovId";
    public static final String
VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
    public static final String DRIVER_LICENSE_NUMBER_INDEX_NAME =
"LicenseNumber";
    public static final String DRIVER_LICENSE_PERSONID_INDEX_NAME = "PersonId";
    public static final String JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-
tutorial-journal-export";
    public static final String USER_TABLES = "information_schema.user_tables";
    public static final String LEDGER_NAME_WITH_TAGS = "tags";
    public static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    public static final IonObjectMapper MAPPER = new IonValueMapper(SYSTEM);

    private Constants() { }

    static {
        MAPPER.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    }
}
```

1.x

⚠ Important

For the Amazon Ion package, you must use the namespace `com.amazon.ion` in your application. The AWS SDK for Java depends on another Ion package under the namespace `software.amazon.ion`, but this is a legacy package that is not compatible with the QLDB driver.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qlldb.tutorial;

import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.fasterxml.jackson.databind.SerializationFeature;
```



```

import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;

/**
 * Constant values used throughout this tutorial.
 */
public final class Constants {
    public static final int RETRY_LIMIT = 4;
    public static final String LEDGER_NAME = "vehicle-registration";
    public static final String STREAM_NAME = "vehicle-registration-stream";
    public static final String VEHICLE_REGISTRATION_TABLE_NAME =
"VehicleRegistration";
    public static final String VEHICLE_TABLE_NAME = "Vehicle";
    public static final String PERSON_TABLE_NAME = "Person";
    public static final String DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
    public static final String VIN_INDEX_NAME = "VIN";
    public static final String PERSON_GOV_ID_INDEX_NAME = "GovId";
    public static final String
VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
    public static final String DRIVER_LICENSE_NUMBER_INDEX_NAME =
"LicenseNumber";
    public static final String DRIVER_LICENSE_PERSONID_INDEX_NAME = "PersonId";
    public static final String JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-
tutorial-journal-export";
    public static final String USER_TABLES = "information_schema.user_tables";
    public static final String LEDGER_NAME_WITH_TAGS = "tags";
    public static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    public static final IonObjectMapper MAPPER = new IonValueMapper(SYSTEM);

    private Constants() { }

    static {
        MAPPER.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    }
}

```

Note

This Constants class includes an instance of the open-source Jackson `IonValueMapper` class. You can use this mapper to process your [Amazon Ion](#) data when doing read and write transactions.

The `CreateLedger.java` file also has a dependency on the following program (`DescribeLedger.java`), which describes the current status of your ledger.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.DescribeLedgerRequest;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Describe a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:

```

```

    * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
    */
public final class DescribeLedger {
    public static AmazonQLDB client = CreateLedger.getClient();
    public static final Logger log = LoggerFactory.getLogger(DescribeLedger.class);

    private DescribeLedger() { }

    public static void main(final String... args) {
        try {

            describe(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to describe a ledger!", e);
        }
    }

    /**
     * Describe a ledger.
     *
     * @param name
     *           Name of the ledger to describe.
     * @return {@link DescribeLedgerResult} from QLDB.
     */
    public static DescribeLedgerResult describe(final String name) {
        log.info("Let's describe ledger with name: {}", name);
        DescribeLedgerRequest request = new DescribeLedgerRequest().withName(name);
        DescribeLedgerResult result = client.describeLedger(request);
        log.info("Success. Ledger description: {}", result);
        return result;
    }
}

```

2. Compile and run the `CreateLedger.java` program to create a ledger named `vehicle-registration`.

2.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0

```

```

*
* Permission is hereby granted, free of charge, to any person obtaining a copy
of this
* software and associated documentation files (the "Software"), to deal in the
Software
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

```

```
package software.amazon.qldb.tutorial;
```

```

import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.CreateLedgerRequest;
import com.amazonaws.services.qldb.model.CreateLedgerResult;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;
import com.amazonaws.services.qldb.model.LedgerState;
import com.amazonaws.services.qldb.model.PermissionsMode;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Create a ledger and wait for it to be active.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 * </p>
 */
public final class CreateLedger {

```

```
    public static final Logger log =
        LoggerFactory.getLogger(CreateLedger.class);
    public static final Long LEDGER_CREATION_POLL_PERIOD_MS = 10_000L;
    public static String endpoint = null;
    public static String region = null;
    public static AmazonQLDB client = getClient();

    private CreateLedger() {
    }

    /**
     * Build a low-level QLDB client.
     *
     * @return {@link AmazonQLDB} control plane client.
     */
    public static AmazonQLDB getClient() {
        AmazonQLDBClientBuilder builder = AmazonQLDBClientBuilder.standard();
        if (null != endpoint && null != region) {
            builder.setEndpointConfiguration(new
                AwsClientBuilder.EndpointConfiguration(endpoint, region));
        }
        return builder.build();
    }

    public static void main(final String... args) throws Exception {
        try {
            client = getClient();

            create(Constants.LEDGER_NAME);

            waitForActive(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to create the ledger!", e);
            throw e;
        }
    }

    /**
     * Create a new ledger with the specified ledger name.
     *
     * @param ledgerName Name of the ledger to be created.
     * @return {@link CreateLedgerResult} from QLDB.
     */
}
```

```

    public static CreateLedgerResult create(final String ledgerName) {
        log.info("Let's create the ledger with name: {}...", ledgerName);
        CreateLedgerRequest request = new CreateLedgerRequest()
            .WithName(ledgerName)
            .withPermissionsMode(PermissionsMode.ALLOW_ALL);
        CreateLedgerResult result = client.createLedger(request);
        log.info("Success. Ledger state: {}.", result.getState());
        return result;
    }

    /**
     * Wait for a newly created ledger to become active.
     *
     * @param ledgerName Name of the ledger to wait on.
     * @return {@link DescribeLedgerResult} from QLDB.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static DescribeLedgerResult waitForActive(final String ledgerName)
    throws InterruptedException {
        log.info("Waiting for ledger to become active...");
        while (true) {
            DescribeLedgerResult result = DescribeLedger.describe(ledgerName);
            if (result.getState().equals(LedgerState.ACTIVE.name())) {
                log.info("Success. Ledger is active and ready to use.");
                return result;
            }
            log.info("The ledger is still creating. Please wait...");
            Thread.sleep(LEDGER_CREATION_POLL_PERIOD_MS);
        }
    }
}

```

1.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software

```

```
* without restriction, including without limitation the rights to use, copy,
* modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
* and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
* COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
* ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
* THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.CreateLedgerRequest;
import com.amazonaws.services.qldb.model.CreateLedgerResult;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;
import com.amazonaws.services.qldb.model.LedgerState;
import com.amazonaws.services.qldb.model.PermissionsMode;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Create a ledger and wait for it to be active.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class CreateLedger {
    public static final Logger log =
        LoggerFactory.getLogger(CreateLedger.class);
    public static final Long LEDGER_CREATION_POLL_PERIOD_MS = 10_000L;
    public static AmazonQLDB client = getClient();
```

```
private CreateLedger() { }

/**
 * Build a low-level QLDB client.
 *
 * @return {@link AmazonQLDB} control plane client.
 */
public static AmazonQLDB getClient() {
    return AmazonQLDBClientBuilder.standard().build();
}

public static void main(final String... args) throws Exception {
    try {

        create(Constants.LEDGER_NAME);

        waitForActive(Constants.LEDGER_NAME);

    } catch (Exception e) {
        log.error("Unable to create the ledger!", e);
        throw e;
    }
}

/**
 * Create a new ledger with the specified ledger name.
 *
 * @param ledgerName
 *        Name of the ledger to be created.
 * @return {@link CreateLedgerResult} from QLDB.
 */
public static CreateLedgerResult create(final String ledgerName) {
    log.info("Let's create the ledger with name: {}....", ledgerName);
    CreateLedgerRequest request = new CreateLedgerRequest()
        .withName(ledgerName)
        .withPermissionsMode(PermissionsMode.ALLOW_ALL);
    CreateLedgerResult result = client.createLedger(request);
    log.info("Success. Ledger state: {}.", result.getState());
    return result;
}

/**
 * Wait for a newly created ledger to become active.
 *
```



```
    * @param ledgerName
    *         Name of the ledger to wait on.
    * @return {@link DescribeLedgerResult} from QLDB.
    * @throws InterruptedException if thread is being interrupted.
    */
    public static DescribeLedgerResult waitForActive(final String ledgerName)
    throws InterruptedException {
        log.info("Waiting for ledger to become active...");
        while (true) {
            DescribeLedgerResult result = DescribeLedger.describe(ledgerName);
            if (result.getState().equals(LedgerState.ACTIVE.name())) {
                log.info("Success. Ledger is active and ready to use.");
                return result;
            }
            log.info("The ledger is still creating. Please wait...");
            Thread.sleep(LEDGER_CREATION_POLL_PERIOD_MS);
        }
    }
}
```

Note

- In the `createLedger` call, you must specify a ledger name and a permissions mode. We recommend using the STANDARD permissions mode to maximize the security of your ledger data.
- When you create a ledger, *deletion protection* is enabled by default. This is a feature in QLDB that prevents ledgers from being deleted by any user. You have the option of disabling deletion protection on ledger creation using the QLDB API or the AWS Command Line Interface (AWS CLI).
- Optionally, you can also specify tags to attach to your ledger.

To verify your connection to the new ledger, proceed to [Step 2: Test connectivity to the ledger](#).

Step 2: Test connectivity to the ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this step, you verify that you can connect to the vehicle-registration ledger in Amazon QLDB using the transactional data API endpoint.

To test connectivity to the ledger

1. Review the following program (`ConnectToLedger.java`), which creates a data session connection to the vehicle-registration ledger.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
```

```
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import java.net.URI;
import java.net.URISyntaxException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.services.qldb.session.QldbSessionClient;
import software.amazon.awssdk.services.qldb.session.QldbSessionClientBuilder;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.RetryPolicy;

/**
 * Connect to a session for a given ledger using default settings.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class ConnectToLedger {
    public static final Logger log =
        LoggerFactory.getLogger(ConnectToLedger.class);
    public static AwsCredentialsProvider credentialsProvider;
    public static String endpoint = null;
    public static String ledgerName = Constants.LEDGER_NAME;
    public static String region = null;
    public static QldbDriver driver;

    private ConnectToLedger() {
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @param retryAttempts How many times the transaction will be retried in
     *     case of a retryable issue happens like Optimistic Concurrency Control
     *     exception,
     *     server side failures or network issues.
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver(int retryAttempts) {
```

```

        QldbSessionClientBuilder builder = getAmazonQldbSessionClientBuilder();
        return QldbDriver.builder()
            .ledger(ledgerName)
            .transactionRetryPolicy(RetryPolicy
                .builder()
                .maxRetries(retryAttempts)
                .build())
            .sessionClientBuilder(builder)
            .build();
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver() {
        QldbSessionClientBuilder builder = getAmazonQldbSessionClientBuilder();
        return QldbDriver.builder()
            .ledger(ledgerName)
            .transactionRetryPolicy(RetryPolicy.builder()

.maxRetries(Constants.RETRY_LIMIT).build())
            .sessionClientBuilder(builder)
            .build();
    }

    /**
     * Creates a QldbSession builder that is passed to the QldbDriver to connect
     * to the Ledger.
     *
     * @return An instance of the AmazonQLDBSessionClientBuilder
     */
    public static QldbSessionClientBuilder getAmazonQldbSessionClientBuilder() {
        QldbSessionClientBuilder builder = QldbSessionClient.builder();
        if (null != endpoint && null != region) {
            try {
                builder.endpointOverride(new URI(endpoint));
            } catch (URISyntaxException e) {
                throw new IllegalArgumentException(e);
            }
        }
        if (null != credentialsProvider) {
            builder.credentialsProvider(credentialsProvider);
        }
    }

```

```
    }
    return builder;
}

/**
 * Create a pooled driver for creating sessions.
 *
 * @return The pooled driver for creating sessions.
 */
public static QldbDriver getDriver() {
    if (driver == null) {
        driver = createQldbDriver();
    }
    return driver;
}

public static void main(final String... args) {
    Iterable<String> tables = ConnectToLedger.getDriver().getTableNames();
    log.info("Existing tables in the ledger:");
    for (String table : tables) {
        log.info("- {} ", table);
    }
}
}
```

Note

- To run data operations on your ledger, you must create an instance of the `QldbDriver` class to connect to a specific ledger. This is a different client object than the `AmazonQLDB` client that you used in the previous step to create the ledger. That previous client is only used to run the management API operations listed in the [Amazon QLDB API reference](#).
- First, create a `QldbDriver` object. You must specify a ledger name when you create this driver.

Then, you can use this driver's `execute` method to run PartiQL statements.

- Optionally, you can specify a maximum number of retry attempts for transaction exceptions. The `execute` method automatically retries optimistic concurrency

control (OCC) conflicts and other common transient exceptions up to this configurable limit. The default value is 4.

If the transaction still fails after the limit is reached, then the driver throws the exception. To learn more, see [Understanding retry policy with the driver in Amazon QLDB](#).

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.qldb.session.AmazonQLDBSessionClientBuilder;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
import software.amazon.qldb.PooledQldbDriver;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.exceptions.QldbClientException;

/**
 * Connect to a session for a given ledger using default settings.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class ConnectToLedger {
    public static final Logger log =
        LoggerFactory.getLogger(ConnectToLedger.class);
    public static AWSCredentialsProvider credentialsProvider;
    public static String endpoint = null;
    public static String ledgerName = Constants.LEDGER_NAME;
    public static String region = null;
    private static PooledQldbDriver driver;

    private ConnectToLedger() {
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static PooledQldbDriver createQldbDriver() {
        AmazonQLDBSessionClientBuilder builder =
            AmazonQLDBSessionClientBuilder.standard();
        if (null != endpoint && null != region) {
            builder.setEndpointConfiguration(new
                AwsClientBuilder.EndpointConfiguration(endpoint, region));
        }
        if (null != credentialsProvider) {
            builder.setCredentials(credentialsProvider);
        }
        return PooledQldbDriver.builder()
            .withLedger(ledgerName)
            .withRetryLimit(Constants.RETRY_LIMIT)
            .withSessionClientBuilder(builder)
```

```
        .build();
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static PooledQldbDriver getDriver() {
        if (driver == null) {
            driver = createQldbDriver();
        }
        return driver;
    }

    /**
     * Connect to a ledger through a {@link QldbDriver}.
     *
     * @return {@link QldbSession}.
     */
    public static QldbSession createQldbSession() {
        return getDriver().getSession();
    }

    public static void main(final String... args) {
        try (QldbSession qldbSession = createQldbSession()) {
            log.info("Listing table names ");
            for (String tableName : qldbSession.getTableNames()) {
                log.info(tableName);
            }
        } catch (QldbClientException e) {
            log.error("Unable to create session.", e);
        }
    }
}
```

Note

- To run data operations on your ledger, you must create an instance of the `PooledQldbDriver` or `QldbDriver` class to connect to a specific ledger. This is a different client object than the AmazonQLDB client that you used in the

previous step to create the ledger. That previous client is only used to run the management API operations listed in the [Amazon QLDB API reference](#).

We recommend using `PooledQldbDriver` unless you need to implement a custom session pool with `QldbDriver`. The default pool size for `PooledQldbDriver` is the [maximum number of open HTTP connections](#) that the session client allows.

- First, create a `PooledQldbDriver` object. You must specify a ledger name when you create this driver.

Then, you can use this driver's `execute` method to run PartiQL statements. Or, you can manually create a session from this pooled driver object and use the session's `execute` method. A session represents a single connection with the ledger.

- Optionally, you can specify a maximum number of retry attempts for transaction exceptions. The `execute` method automatically retries optimistic concurrency control (OCC) conflicts and other common transient exceptions up to this configurable limit. The default value is 4.

If the transaction still fails after the limit is reached, then the driver throws the exception. To learn more, see [Understanding retry policy with the driver in Amazon QLDB](#).

2. Compile and run the `ConnectToLedger.java` program to test your data session connectivity to the `vehicle-registration` ledger.

Overriding the AWS Region

The sample application connects to QLDB in your default AWS Region, which you can set as described in the prerequisite step [Setting your default AWS credentials and Region](#). You can also change the Region by modifying the QLDB session client builder properties.

2.x

The following code example instantiates a new `QldbSessionClientBuilder` object.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.qldb.session.QldbSessionClientBuilder;
```

```
// This client builder will default to US East (Ohio)
QldbSessionClientBuilder builder = QldbSessionClient.builder()
    .region(Region.US_EAST_2);
```

You can use the `region` method to run your code against QLDB in any Region where it's available. For a complete list, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

1.x

The following code example instantiates a new `AmazonQLDBSessionClientBuilder` object.

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.qldb.session.AmazonQLDBSessionClientBuilder;

// This client builder will default to US East (Ohio)
AmazonQLDBSessionClientBuilder builder = AmazonQLDBSessionClientBuilder.standard()
    .withRegion(Regions.US_EAST_2);
```

You can use the `withRegion` method to run your code against QLDB in any Region where it's available. For a complete list, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

To create tables in the vehicle-registration ledger, proceed to [Step 3: Create tables, indexes, and sample data](#).

Step 3: Create tables, indexes, and sample data

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

When your Amazon QLDB ledger is active and accepts connections, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the vehicle-registration ledger:

- `VehicleRegistration`
- `Vehicle`
- `Person`
- `DriversLicense`

You also create the following indexes.

Table name	Field
<code>VehicleRegistration</code>	<code>VIN</code>
<code>VehicleRegistration</code>	<code>LicensePlateNumber</code>
<code>Vehicle</code>	<code>VIN</code>
<code>Person</code>	<code>GovId</code>
<code>DriversLicense</code>	<code>LicenseNumber</code>
<code>DriversLicense</code>	<code>PersonId</code>

When inserting sample data, you first insert documents into the `Person` table. Then, you use the system-assigned `id` from each `Person` document to populate the corresponding fields in the appropriate `VehicleRegistration` and `DriversLicense` documents.

 Tip

As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's `VIN`), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table). For more information about views in QLDB, see [Core concepts](#). To learn more about metadata, see [Querying document metadata](#).

To set up the sample data

1. Review the following .java files. These model classes represent documents that you store in the vehicle-registration tables. They're serializable to and from Amazon Ion format.

Note

[Amazon QLDB documents](#) are stored in Ion format, which is a superset of JSON. So, you can use the FasterXML Jackson library to model the data in JSON.

1. DriversLicense.java

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
```

```

import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

import java.time.LocalDate;

/**
 * Represents a driver's license, serializable to (and from) Ion.
 */
public final class DriversLicense implements RevisionData {
    private final String personId;
    private final String licenseNumber;
    private final String licenseType;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate validFromDate;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate validToDate;

    @JsonCreator
    public DriversLicense(@JsonProperty("PersonId") final String personId,
                          @JsonProperty("LicenseNumber") final String
licenseNumber,
                          @JsonProperty("LicenseType") final String licenseType,
                          @JsonProperty("ValidFromDate") final LocalDate
validFromDate,
                          @JsonProperty("ValidToDate") final LocalDate
validToDate) {
        this.personId = personId;
        this.licenseNumber = licenseNumber;
        this.licenseType = licenseType;
        this.validFromDate = validFromDate;
        this.validToDate = validToDate;
    }

    @JsonProperty("PersonId")
    public String getPersonId() {
        return personId;
    }

    @JsonProperty("LicenseNumber")

```

```
public String getLicenseNumber() {
    return licenseNumber;
}

@JsonProperty("LicenseType")
public String getLicenseType() {
    return licenseType;
}

@JsonProperty("ValidFromDate")
public LocalDate getValidFromDate() {
    return validFromDate;
}

@JsonProperty("ValidToDate")
public LocalDate getValidToDate() {
    return validToDate;
}

@Override
public String toString() {
    return "DriversLicense{" +
        "personId='" + personId + '\'' +
        ", licenseNumber='" + licenseNumber + '\'' +
        ", licenseType='" + licenseType + '\'' +
        ", validFromDate=" + validFromDate +
        ", validToDate=" + validToDate +
        '}';
}
}
```

2. Person.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
```

```
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

```
package software.amazon.qldb.tutorial.model;
```

```
import java.time.LocalDate;
```

```
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
```

```
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.model.streams.RevisionData;
```

```
/**
```

```
 * Represents a person, serializable to (and from) Ion.
```

```
*/
```

```
public final class Person implements RevisionData {
```

```
    private final String firstName;
```

```
    private final String lastName;
```

```
    @JsonSerialize(using = IonLocalDateSerializer.class)
```

```
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
```

```
    private final LocalDate dob;
```

```
    private final String govId;
```

```
    private final String govIdType;
```

```
    private final String address;
```

```
    @JsonCreator
```

```
    public Person(@JsonProperty("FirstName") final String firstName,
```

```
        @JsonProperty("LastName") final String lastName,
        @JsonProperty("DOB") final LocalDate dob,
        @JsonProperty("GovId") final String govId,
        @JsonProperty("GovIdType") final String govIdType,
        @JsonProperty("Address") final String address) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.dob = dob;
    this.govId = govId;
    this.govIdType = govIdType;
    this.address = address;
}

@JsonProperty("Address")
public String getAddress() {
    return address;
}

@JsonProperty("DOB")
public LocalDate getDob() {
    return dob;
}

@JsonProperty("FirstName")
public String getFirstName() {
    return firstName;
}

@JsonProperty("LastName")
public String getLastName() {
    return lastName;
}

@JsonProperty("GovId")
public String getGovId() {
    return govId;
}

@JsonProperty("GovIdType")
public String getGovIdType() {
    return govIdType;
}

/**
```



```

    * This returns the unique document ID given a specific government ID.
    *
    * @param txn
    *           A transaction executor object.
    * @param govId
    *           The government ID of a driver.
    * @return the unique document ID.
    */
    public static String getDocumentIdByGovId(final TransactionExecutor txn,
final String govId) {
        return SampleData.getDocumentId(txn, Constants.PERSON_TABLE_NAME,
"GovId", govId);
    }

    @Override
    public String toString() {
        return "Person{" +
            "firstName='" + firstName + '\'' +
            ", lastName='" + lastName + '\'' +
            ", dob=" + dob +
            ", govId='" + govId + '\'' +
            ", govIdType='" + govIdType + '\'' +
            ", address='" + address + '\'' +
            '}';
    }
}

```

3. VehicleRegistration.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 */

```

```

* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

```

```
package software.amazon.qldb.tutorial.model;
```

```

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

```

```

import java.math.BigDecimal;
import java.time.LocalDate;

```

```
/**
```

```

 * Represents a vehicle registration, serializable to (and from) Ion.
 */

```

```
public final class VehicleRegistration implements RevisionData {
```

```

    private final String vin;
    private final String licensePlateNumber;
    private final String state;
    private final String city;
    private final BigDecimal pendingPenaltyTicketAmount;
    private final LocalDate validFromDate;
    private final LocalDate validToDate;
    private final Owners owners;

```

```
    @JsonCreator
```

```

    public VehicleRegistration(@JsonProperty("VIN") final String vin,
                               @JsonProperty("LicensePlateNumber") final String
licensePlateNumber,
                               @JsonProperty("State") final String state,
                               @JsonProperty("City") final String city,

```

```
        @JsonProperty("PendingPenaltyTicketAmount") final
BigDecimal pendingPenaltyTicketAmount,
        @JsonProperty("ValidFromDate") final LocalDate
validFromDate,
        @JsonProperty("ValidToDate") final LocalDate
validToDate,
        @JsonProperty("Owners") final Owners owners) {
    this.vin = vin;
    this.licensePlateNumber = licensePlateNumber;
    this.state = state;
    this.city = city;
    this.pendingPenaltyTicketAmount = pendingPenaltyTicketAmount;
    this.validFromDate = validFromDate;
    this.validToDate = validToDate;
    this.owners = owners;
}

@JsonProperty("City")
public String getCity() {
    return city;
}

@JsonProperty("LicensePlateNumber")
public String getLicensePlateNumber() {
    return licensePlateNumber;
}

@JsonProperty("Owners")
public Owners getOwners() {
    return owners;
}

@JsonProperty("PendingPenaltyTicketAmount")
public BigDecimal getPendingPenaltyTicketAmount() {
    return pendingPenaltyTicketAmount;
}

@JsonProperty("State")
public String getState() {
    return state;
}

@JsonProperty("ValidFromDate")
@JsonSerialize(using = IonLocalDateSerializer.class)
```

```

@JsonDeserialize(using = IonLocalDateDeserializer.class)
public LocalDate getValidFromDate() {
    return validFromDate;
}

@JsonProperty("ValidToDate")
@JsonSerialize(using = IonLocalDateSerializer.class)
@JsonDeserialize(using = IonLocalDateDeserializer.class)
public LocalDate getValidToDate() {
    return validToDate;
}

@JsonProperty("VIN")
public String getVin() {
    return vin;
}

/**
 * Returns the unique document ID of a vehicle given a specific VIN.
 *
 * @param txn
 *           A transaction executor object.
 * @param vin
 *           The VIN of a vehicle.
 * @return the unique document ID of the specified vehicle.
 */
public static String getDocumentIdByVin(final TransactionExecutor txn, final
String vin) {
    return SampleData.getDocumentId(txn,
Constants.VEHICLE_REGISTRATION_TABLE_NAME, "VIN", vin);
}

@Override
public String toString() {
    return "VehicleRegistration{" +
        "vin='" + vin + '\'' +
        ", licensePlateNumber='" + licensePlateNumber + '\'' +
        ", state='" + state + '\'' +
        ", city='" + city + '\'' +
        ", pendingPenaltyTicketAmount=" + pendingPenaltyTicketAmount +
        ", validFromDate=" + validFromDate +
        ", validToDate=" + validToDate +
        ", owners=" + owners +
        '}';
}

```

```
}  
}
```

4. Vehicle.java

```
/*  
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
 * SPDX-License-Identifier: MIT-0  
 *  
 * Permission is hereby granted, free of charge, to any person obtaining a copy  
 of this  
 * software and associated documentation files (the "Software"), to deal in the  
 Software  
 * without restriction, including without limitation the rights to use, copy,  
 modify,  
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,  
 and to  
 * permit persons to whom the Software is furnished to do so.  
 *  
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  
 IMPLIED,  
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A  
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR  
 COPYRIGHT  
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN  
 ACTION  
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE  
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.  
 */  
  
package software.amazon.qldb.tutorial.model;  
  
import com.fasterxml.jackson.annotation.JsonCreator;  
import com.fasterxml.jackson.annotation.JsonProperty;  
import software.amazon.qldb.tutorial.model.streams.RevisionData;  
  
/**  
 * Represents a vehicle, serializable to (and from) Ion.  
 */  
public final class Vehicle implements RevisionData {  
    private final String vin;  
    private final String type;  
    private final int year;  
    private final String make;
```

```
private final String model;
private final String color;

@JsonCreator
public Vehicle(@JsonProperty("VIN") final String vin,
               @JsonProperty("Type") final String type,
               @JsonProperty("Year") final int year,
               @JsonProperty("Make") final String make,
               @JsonProperty("Model") final String model,
               @JsonProperty("Color") final String color) {
    this.vin = vin;
    this.type = type;
    this.year = year;
    this.make = make;
    this.model = model;
    this.color = color;
}

@JsonProperty("Color")
public String getColor() {
    return color;
}

@JsonProperty("Make")
public String getMake() {
    return make;
}

@JsonProperty("Model")
public String getModel() {
    return model;
}

@JsonProperty("Type")
public String getType() {
    return type;
}

@JsonProperty("VIN")
public String getVin() {
    return vin;
}

@JsonProperty("Year")
```

```

    public int getYear() {
        return year;
    }

    @Override
    public String toString() {
        return "Vehicle{" +
            "vin='" + vin + '\'' +
            ", type='" + type + '\'' +
            ", year=" + year +
            ", make='" + make + '\'' +
            ", model='" + model + '\'' +
            ", color='" + color + '\'' +
            '}';
    }
}

```

5. Owner.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

```

```
package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Represents a vehicle owner, serializable to (and from) Ion.
 */
public final class Owner {
    private final String personId;

    public Owner(@JsonProperty("PersonId") final String personId) {
        this.personId = personId;
    }

    @JsonProperty("PersonId")
    public String getPersonId() {
        return personId;
    }

    @Override
    public String toString() {
        return "Owner{" +
            "personId='" + personId + '\'' +
            '}';
    }
}
```

6. Owners.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 */
```



```
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

```
package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonProperty;

import java.util.List;

/**
 * Represents a set of owners for a given vehicle, serializable to (and from)
 * Ion.
 */
public final class Owners {
    private final Owner primaryOwner;
    private final List<Owner> secondaryOwners;

    public Owners(@JsonProperty("PrimaryOwner") final Owner primaryOwner,
                  @JsonProperty("SecondaryOwners") final List<Owner>
secondaryOwners) {
        this.primaryOwner = primaryOwner;
        this.secondaryOwners = secondaryOwners;
    }

    @JsonProperty("PrimaryOwner")
    public Owner getPrimaryOwner() {
        return primaryOwner;
    }

    @JsonProperty("SecondaryOwners")
    public List<Owner> getSecondaryOwners() {
        return secondaryOwners;
    }

    @Override
    public String toString() {
```

```

        return "Owners{" +
            "primaryOwner=" + primaryOwner +
            ", secondaryOwners=" + secondaryOwners +
            '}';
    }
}

```

7. DmlResultDocument.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Contains information about an individual document inserted or modified
 * as a result of DML.
 */
public class DmlResultDocument {

```

```

    private String documentId;

    @JsonCreator
    public DmlResultDocument(@JsonProperty("documentId") final String documentId)
    {
        this.documentId = documentId;
    }

    public String getDocumentId() {
        return documentId;
    }

    @Override
    public String toString() {
        return "DmlResultDocument{"
            + "documentId='" + documentId + '\''
            + '}';
    }
}

```

8. RevisionData.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION

```

```

    * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
    * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
    */

package software.amazon.qldb.tutorial.model.streams;

/**
 * Allows modeling the content of all revisions as a generic revision data. Used
 * in the {@link Revision} and extended by domain models in {@link
 * software.amazon.qldb.tutorial.model} to make it easier to write the {@link
 * Revision.RevisionDataDeserializer} that must deserialize the {@link
 * Revision#data} from different domain models.
 */
public interface RevisionData { }

```

9. RevisionMetadata.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

```

```
import com.amazon.ion.IonInt;
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonTimestamp;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.dataformat.ion.IonTimestampSerializers;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Date;
import java.util.Objects;

/**
 * Represents the metadata field of a QLDB Document
 */
public class RevisionMetadata {
    private static final Logger log =
        LoggerFactory.getLogger(RevisionMetadata.class);
    private final String id;
    private final long version;
    @JsonSerialize(using =
        IonTimestampSerializers.IonTimestampJavaDateSerializer.class)
    private final Date txTime;
    private final String txId;

    @JsonCreator
    public RevisionMetadata(@JsonProperty("id") final String id,
                           @JsonProperty("version") final long version,
                           @JsonProperty("txTime") final Date txTime,
                           @JsonProperty("txId") final String txId) {
        this.id = id;
        this.version = version;
        this.txTime = txTime;
        this.txId = txId;
    }

    /**
     * Gets the unique ID of a QLDB document.
     *
     * @return the document ID.
     */
    public String getId() {
```

```
        return id;
    }

    /**
     * Gets the version number of the document in the document's modification
    history.
     * @return the version number.
     */
    public long getVersion() {
        return version;
    }

    /**
     * Gets the time during which the document was modified.
     *
     * @return the transaction time.
     */
    public Date getTxTime() {
        return txTime;
    }

    /**
     * Gets the transaction ID associated with this document.
     *
     * @return the transaction ID.
     */
    public String getTxId() {
        return txId;
    }

    public static RevisionMetadata fromIon(final IonStruct ionStruct) {
        if (ionStruct == null) {
            throw new IllegalArgumentException("Metadata cannot be null");
        }
        try {
            IonString id = (IonString) ionStruct.get("id");
            IonInt version = (IonInt) ionStruct.get("version");
            IonTimestamp txTime = (IonTimestamp) ionStruct.get("txTime");
            IonString txId = (IonString) ionStruct.get("txId");
            if (id == null || version == null || txTime == null || txId == null)
            {
                throw new IllegalArgumentException("Document is missing required
            fields");
            }
        }
    }
}
```

```

        return new RevisionMetadata(id.stringValue(), version.longValue(),
new Date(txTime.getMillis()), txId.stringValue());
    } catch (ClassCastException e) {
        log.error("Failed to parse ion document");
        throw new IllegalArgumentException("Document members are not of the
correct type", e);
    }
}

/**
 * Converts a {@link RevisionMetadata} object to a string.
 *
 * @return the string representation of the {@link QldbRevision} object.
 */
@Override
public String toString() {
    return "Metadata{"
        + "id='" + id + '\''
        + ", version=" + version
        + ", txTime=" + txTime
        + ", txId='" + txId
        + '\''
        + '}';
}

/**
 * Check whether two {@link RevisionMetadata} objects are equivalent.
 *
 * @return {@code true} if the two objects are equal, {@code false}
otherwise.
 */
@Override
public boolean equals(Object o) {
    if (this == o) { return true; }
    if (o == null || getClass() != o.getClass()) { return false; }
    RevisionMetadata metadata = (RevisionMetadata) o;
    return version == metadata.version
        && id.equals(metadata.id)
        && txTime.equals(metadata.txTime)
        && txId.equals(metadata.txId);
}

/**
 * Generate a hash code for the {@link RevisionMetadata} object.

```

```

        *
        * @return the hash code.
        */
        @Override
        public int hashCode() {
            // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
            properties.
            return Objects.hash(id, version, txTime, txId);
            // CHECKSTYLE:ON
        }
    }
}

```

10QldbRevision.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonBlob;
import com.amazon.ion.IonStruct;
import com.fasterxml.jackson.annotation.JsonCreator;

```



```
import com.fasterxml.jackson.annotation.JsonProperty;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.Verifier;

import java.io.IOException;
import java.util.Arrays;
import java.util.Objects;

/**
 * Represents a QldbRevision including both user data and metadata.
 */
public final class QldbRevision {
    private static final Logger log =
        LoggerFactory.getLogger(QldbRevision.class);

    private final BlockAddress blockAddress;
    private final RevisionMetadata metadata;
    private final byte[] hash;
    private final byte[] dataHash;
    private final IonStruct data;

    @JsonCreator
    public QldbRevision(@JsonProperty("blockAddress") final BlockAddress
        blockAddress,
                        @JsonProperty("metadata") final RevisionMetadata
        metadata,
                        @JsonProperty("hash") final byte[] hash,
                        @JsonProperty("dataHash") final byte[] dataHash,
                        @JsonProperty("data") final IonStruct data) {
        this.blockAddress = blockAddress;
        this.metadata = metadata;
        this.hash = hash;
        this.dataHash = dataHash;
        this.data = data;
    }

    /**
     * Gets the unique ID of a QLDB document.
     *
     * @return the {@link BlockAddress} object.
     */
    public BlockAddress getBlockAddress() {
```

```
        return blockAddress;
    }

    /**
     * Gets the metadata of the revision.
     *
     * @return the {@link RevisionMetadata} object.
     */
    public RevisionMetadata getMetadata() {
        return metadata;
    }

    /**
     * Gets the SHA-256 hash value of the revision.
     * This is equivalent to the hash of the revision metadata and data.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getHash() {
        return hash;
    }

    /**
     * Gets the SHA-256 hash value of the data portion of the revision.
     * This is only present if the revision is redacted.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getDataHash() {
        return dataHash;
    }

    /**
     * Gets the revision data.
     *
     * @return the revision data.
     */
    public IonStruct getData() {
        return data;
    }

    /**
     * Returns true if the revision has been redacted.
     * @return a boolean value representing the redaction status
     */
}
```

```

    * of this revision.
    */
    public Boolean isRedacted() {
        return dataHash != null;
    }

    /**
     * Constructs a new {@link QldbRevision} from an {@link IonStruct}.
     *
     * The specified {@link IonStruct} must include the following fields
     *
     * - blockAddress -- a {@link BlockAddress},
     * - metadata -- a {@link RevisionMetadata},
     * - hash -- the revision's hash calculated by QLDB,
     * - dataHash -- the user data's hash calculated by QLDB (only present if
revision is redacted),
     * - data -- an {@link IonStruct} containing user data in the document.
     *
     * If any of these fields are missing or are malformed, then throws {@link
IllegalArgumentException}.
     *
     * If the document hash calculated from the members of the specified {@link
IonStruct} does not match
     * the hash member of the {@link IonStruct} then throws {@link
IllegalArgumentException}.
     *
     * @param ionStruct
     *             The {@link IonStruct} that contains a {@link QldbRevision}
object.
     * @return the converted {@link QldbRevision} object.
     * @throws IOException if failed to parse parameter {@link IonStruct}.
     */
    public static QldbRevision fromIon(final IonStruct ionStruct) throws
IOException {
        try {
            BlockAddress blockAddress =
Constants.MAPPER.readValue(ionStruct.get("blockAddress"), BlockAddress.class);
            IonBlob revisionHash = (IonBlob) ionStruct.get("hash");
            IonStruct metadataStruct = (IonStruct) ionStruct.get("metadata");
            IonStruct data = ionStruct.get("data") == null ||
ionStruct.get("data").isNullValue() ?
                null : (IonStruct) ionStruct.get("data");
            IonBlob dataHash = ionStruct.get("dataHash") == null ||
ionStruct.get("dataHash").isNullValue() ?

```

```

        null : (IonBlob) ionStruct.get("dataHash");
        if (revisionHash == null || metadataStruct == null) {
            throw new IllegalArgumentException("Document is missing required
fields");
        }
        byte[] dataHashBytes = dataHash != null ? dataHash.getBytes() :
QldbIonUtils.hashIonValue(data);
        verifyRevisionHash(metadataStruct, dataHashBytes,
revisionHash.getBytes());
        RevisionMetadata metadata = RevisionMetadata.fromIon(metadataStruct);
        return new QldbRevision(
            blockAddress,
            metadata,
            revisionHash.getBytes(),
            dataHash != null ? dataHash.getBytes() : null,
            data
        );
    } catch (ClassCastException e) {
        log.error("Failed to parse ion document");
        throw new IllegalArgumentException("Document members are not of the
correct type", e);
    }
}

/**
 * Converts a {@link QldbRevision} object to string.
 *
 * @return the string representation of the {@link QldbRevision} object.
 */
@Override
public String toString() {
    return "QldbRevision{" +
        "blockAddress=" + blockAddress +
        ", metadata=" + metadata +
        ", hash=" + Arrays.toString(hash) +
        ", dataHash=" + Arrays.toString(dataHash) +
        ", data=" + data +
        '}';
}

/**
 * Check whether two {@link QldbRevision} objects are equivalent.
 *

```

```

    * @return {@code true} if the two objects are equal, {@code false}
    otherwise.
    */
    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof QldbRevision)) {
            return false;
        }
        final QldbRevision that = (QldbRevision) o;
        return Objects.equals(getBlockAddress(), that.getBlockAddress())
            && Objects.equals(getMetadata(), that.getMetadata())
            && Arrays.equals(getHash(), that.getHash())
            && Arrays.equals(getDataHash(), that.getDataHash())
            && Objects.equals(getData(), that.getData());
    }

    /**
     * Create a hash code for the {@link QldbRevision} object.
     *
     * @return the hash code.
     */
    @Override
    public int hashCode() {
        // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
        properties.
        int result = Objects.hash(blockAddress, metadata, data);
        // CHECKSTYLE:ON
        result = 31 * result + Arrays.hashCode(hash);
        return result;
    }

    /**
     * Throws an IllegalArgumentException if the hash of the revision data and
     metadata
     * does not match the hash provided by QLDB with the revision.
     */
    public void verifyRevisionHash() {
        // Certain internal-only system revisions only contain a hash which
        cannot be
        // further computed. However, these system hashes still participate to
        validate

```

```

        // the journal block. User revisions will always contain values for all
        fields
        // and can therefore have their hash computed.
        if (blockAddress == null && metadata == null && data == null && dataHash
        == null) {
            return;
        }

        try {
            IonStruct metadataIon = (IonStruct)
            Constants.MAPPER.writeValueAsIonValue(metadata);
            byte[] dataHashBytes = isRedacted() ? dataHash :
            QldbIonUtils.hashIonValue(data);
            verifyRevisionHash(metadataIon, dataHashBytes, hash);
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not encode revision
            metadata to ion.", e);
        }
    }

    private static void verifyRevisionHash(IonStruct metadata, byte[] dataHash,
    byte[] expectedHash) {
        byte[] metadataHash = QldbIonUtils.hashIonValue(metadata);
        byte[] candidateHash = Verifier.dot(metadataHash, dataHash);
        if (!Arrays.equals(candidateHash, expectedHash)) {
            throw new IllegalArgumentException("Hash entry of QLDB revision and
            computed hash "
                + "of QLDB revision do not match");
        }
    }
}

```

11 IonLocalDateDeserializer.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,

```

```

* merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.JsonDeserializer;

import java.io.IOException;
import java.time.LocalDate;

/**
 * Deserializes [java.time.LocalDate] from Ion.
 */
public class IonLocalDateDeserializer extends JsonDeserializer<LocalDate> {

    @Override
    public LocalDate deserialize(JsonParser jp, DeserializationContext ctxt)
        throws IOException {
        return timestampToLocalDate((Timestamp) jp.getEmbeddedObject());
    }

    private LocalDate timestampToLocalDate(Timestamp timestamp) {
        return LocalDate.of(timestamp.getYear(), timestamp.getMonth(),
            timestamp.getDay());
    }
}

```

12 IonLocalDateSerializer.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.SerializerProvider;
import com.fasterxml.jackson.databind.ser.std.StdScalarSerializer;
import com.fasterxml.jackson.dataformat.ion.IonGenerator;

import java.io.IOException;
import java.time.LocalDate;

/**
 * Serializes [java.time.LocalDate] to Ion.
 */
public class IonLocalDateSerializer extends StdScalarSerializer<LocalDate> {

    public IonLocalDateSerializer() {
        super(LocalDate.class);
    }
}
```



```

    }

    @Override
    public void serialize(LocalDate date, JsonGenerator jsonGenerator,
        SerializerProvider serializerProvider) throws IOException {
        Timestamp timestamp = Timestamp.forDay(date.getYear(),
            date.getMonthValue(), date.getDayOfMonth());
        ((IonGenerator) jsonGenerator).writeValue(timestamp);
    }
}

```

2. Review the following file (`SampleData.java`), which represents the sample data that you insert into the vehicle-registration tables.

2.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

```

```
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import java.io.IOException;
import java.math.BigDecimal;
import java.text.ParseException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.ConnectToLedger;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.qldb.DmlResultDocument;
import software.amazon.qldb.tutorial.qldb.QldbRevision;

/**
 * Sample domain objects for use throughout this tutorial.
 */
public final class SampleData {
    public static final DateTimeFormatter DATE_TIME_FORMAT =
        DateTimeFormatter.ofPattern("yyyy-MM-dd");

    public static final List<VehicleRegistration> REGISTRATIONS =
        Collections.unmodifiableList(Arrays.asList(
            new VehicleRegistration("1N4AL11D75C109151", "LEWISR261LL", "WA",
                "Seattle",
                BigDecimal.valueOf(90.25), convertToLocalDate("2017-08-21"),
                convertToLocalDate("2020-05-11"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("KM8SRDHF6EU074761", "CA762X", "WA", "Kent",
                BigDecimal.valueOf(130.75),
                convertToLocalDate("2017-09-14"), convertToLocalDate("2020-06-25"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("3HGGK5G53FM761765", "CD820Z", "WA",
                "Everett",
                BigDecimal.valueOf(442.30),
                convertToLocalDate("2011-03-17"), convertToLocalDate("2021-03-24"),
                new Owners(new Owner(null), Collections.emptyList())));
```

```

        new VehicleRegistration("1HVBBAANXWH544237", "LS477D", "WA",
        "Tacoma",
            BigDecimal.valueOf(42.20), convertToLocalDate("2011-10-26"),
        convertToLocalDate("2023-09-25"),
            new Owners(new Owner(null), Collections.emptyList()),
        new VehicleRegistration("1C4RJFAG0FC625797", "TH393F", "WA",
        "Olympia",
            BigDecimal.valueOf(30.45), convertToLocalDate("2013-09-02"),
        convertToLocalDate("2024-03-19"),
            new Owners(new Owner(null), Collections.emptyList()))
    ));

    public static final List<Vehicle> VEHICLES =
    Collections.unmodifiableList(Arrays.asList(
        new Vehicle("1N4AL11D75C109151", "Sedan", 2011, "Audi", "A5",
        "Silver"),
        new Vehicle("KM8SRDHF6EU074761", "Sedan", 2015, "Tesla", "Model S",
        "Blue"),
        new Vehicle("3HGGK5G53FM761765", "Motorcycle", 2011, "Ducati",
        "Monster 1200", "Yellow"),
        new Vehicle("1HVBBAANXWH544237", "Semi", 2009, "Ford", "F 150",
        "Black"),
        new Vehicle("1C4RJFAG0FC625797", "Sedan", 2019, "Mercedes", "CLK
        350", "White")
    ));

    public static final List<Person> PEOPLE =
    Collections.unmodifiableList(Arrays.asList(
        new Person("Raul", "Lewis", convertToLocalDate("1963-08-19"),
        "LEWISR261LL", "Driver License", "1719 University Street,
        Seattle, WA, 98109"),
        new Person("Brent", "Logan", convertToLocalDate("1967-07-03"),
        "LOGANB486CG", "Driver License", "43 Stockert Hollow Road,
        Everett, WA, 98203"),
        new Person("Alexis", "Pena", convertToLocalDate("1974-02-10"),
        "744 849 301", "SSN", "4058 Melrose Street, Spokane Valley,
        WA, 99206"),
        new Person("Melvin", "Parker", convertToLocalDate("1976-05-22"),
        "P626-168-229-765", "Passport", "4362 Ryder Avenue, Seattle,
        WA, 98101"),
        new Person("Salvatore", "Spencer", convertToLocalDate("1997-11-15"),
        "S152-780-97-415-0", "Passport", "4450 Honeysuckle Lane,
        Seattle, WA, 98101")
    ));

```

```

    public static final List<DriversLicense> LICENSES =
Collections.unmodifiableList(Arrays.asList(
    new DriversLicense(null, "LEWISR261LL", "Learner",
        convertToLocalDate("2016-12-20"),
convertToLocalDate("2020-11-15")),
    new DriversLicense(null, "LOGANB486CG", "Probationary",
        convertToLocalDate("2016-04-06"),
convertToLocalDate("2020-11-15")),
    new DriversLicense(null, "744 849 301", "Full",
        convertToLocalDate("2017-12-06"),
convertToLocalDate("2022-10-15")),
    new DriversLicense(null, "P626-168-229-765", "Learner",
        convertToLocalDate("2017-08-16"),
convertToLocalDate("2021-11-15")),
    new DriversLicense(null, "S152-780-97-415-0", "Probationary",
        convertToLocalDate("2015-08-15"),
convertToLocalDate("2021-08-21"))
));

    private SampleData() { }

    /**
     * Converts a date string with the format 'yyyy-MM-dd' into a {@link
java.util.Date} object.
     *
     * @param date
     *             The date string to convert.
     * @return {@link java.time.LocalDate} or null if there is a {@link
ParseException}
     */
    public static synchronized LocalDate convertToLocalDate(String date) {
        return LocalDate.parse(date, DATE_TIME_FORMAT);
    }

    /**
     * Convert the result set into a list of IonValues.
     *
     * @param result
     *             The result set to convert.
     * @return a list of IonValues.
     */
    public static List<IonValue> toIonValues(Result result) {
        final List<IonValue> valueList = new ArrayList<>();

```

```

        result.iterator().forEachRemaining(valueList::add);
        return valueList;
    }

    /**
     * Get the document ID of a particular document.
     *
     * @param txn
     *         A transaction executor object.
     * @param tableName
     *         Name of the table containing the document.
     * @param identifier
     *         The identifier used to narrow down the search.
     * @param value
     *         Value of the identifier.
     * @return the list of document IDs in the result set.
     */
    public static String getDocumentId(final TransactionExecutor txn, final
String tableName,
                                     final String identifier, final String
value) {
        try {
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(value));
            final String query = String.format("SELECT metadata.id FROM
_ql_committed_%s AS p WHERE p.data.%s = ?",
                tableName, identifier);
            Result result = txn.execute(query, parameters);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document ID
using " + value);
            }
            return getStringValueOfStructField((IonStruct)
result.iterator().next(), "id");
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the document by ID.
     *
     * @param tableName
     *         Name of the table to insert documents into.

```

```

    * @param documentId
    *           The unique ID of a document in the Person table.
    * @return a {@link QldbRevision} object.
    * @throws IllegalStateException if failed to convert parameter into {@link
    IonValue}.
    */
    public static QldbRevision getDocumentById(String tableName, String
documentId) {
        try {
            final IonValue ionValue =
Constants.MAPPER.writeValueAsIonValue(documentId);
            Result result = ConnectToLedger.getDriver().execute(txn -> {
                return txn.execute("SELECT c.* FROM _ql_committed_" + tableName
+ " AS c BY docId "
                                + "WHERE docId = ?", ionValue);
            });
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document by
id " + documentId + " in table " + tableName);
            }
            return Constants.MAPPER.readValue(result.iterator().next(),
QldbRevision.class);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Return a list of modified document IDs as strings from a DML {@link
Result}.
     *
     * @param result
     *           The result set from a DML operation.
     * @return the list of document IDs modified by the operation.
     */
    public static List<String> getDocumentIdsFromDmlResult(final Result result)
{
        final List<String> strings = new ArrayList<>();
        result.iterator().forEachRemaining(row ->
strings.add(getDocumentIdFromDmlResultDocument(row)));
        return strings;
    }

    /**

```

```

    * Convert the given DML result row's document ID to string.
    *
    * @param dmlResultDocument
    *         The {@link IonValue} representing the results of a DML
operation.
    * @return a string of document ID.
    */
    public static String getDocumentIdFromDmlResultDocument(final IonValue
dmlResultDocument) {
        try {
            DmlResultDocument result =
Constants.MAPPER.readValue(dmlResultDocument, DmlResultDocument.class);
            return result.getDocumentId();
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
    * Get the String value of a given {@link IonStruct} field name.
    * @param struct the {@link IonStruct} from which to get the value.
    * @param fieldName the name of the field from which to get the value.
    * @return the String value of the field within the given {@link IonStruct}.
    */
    public static String getStringValueOfStructField(final IonStruct struct,
final String fieldName) {
        return ((IonString) struct.get(fieldName)).stringValue();
    }

    /**
    * Return a copy of the given driver's license with updated person Id.
    *
    * @param oldLicense
    *         The old driver's license to update.
    * @param personId
    *         The PersonId of the driver.
    * @return the updated {@link DriversLicense}.
    */
    public static DriversLicense updatePersonIdDriversLicense(final
DriversLicense oldLicense, final String personId) {
        return new DriversLicense(personId, oldLicense.getLicenseNumber(),
oldLicense.getLicenseType(),
            oldLicense.getValidFromDate(), oldLicense.getValidToDate());
    }

```

```

/**
 * Return a copy of the given vehicle registration with updated person Id.
 *
 * @param oldRegistration
 *           The old vehicle registration to update.
 * @param personId
 *           The PersonId of the driver.
 * @return the updated {@link VehicleRegistration}.
 */
public static VehicleRegistration updateOwnerVehicleRegistration(final
VehicleRegistration oldRegistration,                                     final
String personId) {
    return new VehicleRegistration(oldRegistration.getVin(),
oldRegistration.getLicensePlateNumber(),
                                oldRegistration.getState(), oldRegistration.getCity(),
oldRegistration.getPendingPenaltyTicketAmount(),
                                oldRegistration.getValidFromDate(),
oldRegistration.getValidToDate(),
                                new Owners(new Owner(personId), Collections.emptyList()));
}
}

```

1.x

⚠ Important

For the Amazon Ion package, you must use the namespace `com.amazon.ion` in your application. The AWS SDK for Java depends on another Ion package under the namespace `software.amazon.ion`, but this is a legacy package that is not compatible with the QLDB driver.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this

```



```
* software and associated documentation files (the "Software"), to deal in the
Software
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

```
package software.amazon.qldb.tutorial.model;
```

```
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.qldb.DmlResultDocument;
import software.amazon.qldb.tutorial.qldb.QldbRevision;
```

```
import java.io.IOException;
```

```
import java.math.BigDecimal;
import java.text.ParseException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
```

```
/**
```

```

* Sample domain objects for use throughout this tutorial.
*/
public final class SampleData {
    public static final DateTimeFormatter DATE_TIME_FORMAT =
        DateTimeFormatter.ofPattern("yyyy-MM-dd");

    public static final List<VehicleRegistration> REGISTRATIONS =
        Collections.unmodifiableList(Arrays.asList(
            new VehicleRegistration("1N4AL11D75C109151", "LEWISR261LL", "WA",
                "Seattle",
                BigDecimal.valueOf(90.25), convertToLocalDate("2017-08-21"),
                convertToLocalDate("2020-05-11"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("KM8SRDHF6EU074761", "CA762X", "WA", "Kent",
                BigDecimal.valueOf(130.75),
                convertToLocalDate("2017-09-14"), convertToLocalDate("2020-06-25"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("3HGGK5G53FM761765", "CD820Z", "WA",
                "Everett",
                BigDecimal.valueOf(442.30),
                convertToLocalDate("2011-03-17"), convertToLocalDate("2021-03-24"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1HVBBAANXWH544237", "LS477D", "WA",
                "Tacoma",
                BigDecimal.valueOf(42.20), convertToLocalDate("2011-10-26"),
                convertToLocalDate("2023-09-25"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1C4RJFAG0FC625797", "TH393F", "WA",
                "Olympia",
                BigDecimal.valueOf(30.45), convertToLocalDate("2013-09-02"),
                convertToLocalDate("2024-03-19"),
                new Owners(new Owner(null), Collections.emptyList()))
        ));

    public static final List<Vehicle> VEHICLES =
        Collections.unmodifiableList(Arrays.asList(
            new Vehicle("1N4AL11D75C109151", "Sedan", 2011, "Audi", "A5",
                "Silver"),
            new Vehicle("KM8SRDHF6EU074761", "Sedan", 2015, "Tesla", "Model S",
                "Blue"),
            new Vehicle("3HGGK5G53FM761765", "Motorcycle", 2011, "Ducati",
                "Monster 1200", "Yellow"),
            new Vehicle("1HVBBAANXWH544237", "Semi", 2009, "Ford", "F 150",
                "Black"),
        ));

```

```

        new Vehicle("1C4RJFAG0FC625797", "Sedan", 2019, "Mercedes", "CLK
350", "White")
    ));

    public static final List<Person> PEOPLE =
Collections.unmodifiableList(Arrays.asList(
        new Person("Raul", "Lewis", convertToLocalDate("1963-08-19"),
            "LEWISR261LL", "Driver License", "1719 University Street,
Seattle, WA, 98109"),
        new Person("Brent", "Logan", convertToLocalDate("1967-07-03"),
            "LOGANB486CG", "Driver License", "43 Stockert Hollow Road,
Everett, WA, 98203"),
        new Person("Alexis", "Pena", convertToLocalDate("1974-02-10"),
            "744 849 301", "SSN", "4058 Melrose Street, Spokane Valley,
WA, 99206"),
        new Person("Melvin", "Parker", convertToLocalDate("1976-05-22"),
            "P626-168-229-765", "Passport", "4362 Ryder Avenue, Seattle,
WA, 98101"),
        new Person("Salvatore", "Spencer", convertToLocalDate("1997-11-15"),
            "S152-780-97-415-0", "Passport", "4450 Honeysuckle Lane,
Seattle, WA, 98101")
    ));

    public static final List<DriversLicense> LICENSES =
Collections.unmodifiableList(Arrays.asList(
        new DriversLicense(null, "LEWISR261LL", "Learner",
            convertToLocalDate("2016-12-20"),
convertToLocalDate("2020-11-15")),
        new DriversLicense(null, "LOGANB486CG", "Probationary",
            convertToLocalDate("2016-04-06"),
convertToLocalDate("2020-11-15")),
        new DriversLicense(null, "744 849 301", "Full",
            convertToLocalDate("2017-12-06"),
convertToLocalDate("2022-10-15")),
        new DriversLicense(null, "P626-168-229-765", "Learner",
            convertToLocalDate("2017-08-16"),
convertToLocalDate("2021-11-15")),
        new DriversLicense(null, "S152-780-97-415-0", "Probationary",
            convertToLocalDate("2015-08-15"),
convertToLocalDate("2021-08-21"))
    ));

    private SampleData() { }

```

```

/**
 * Converts a date string with the format 'yyyy-MM-dd' into a {@link
 java.util.Date} object.
 *
 * @param date
 *           The date string to convert.
 * @return {@link LocalDate} or null if there is a {@link ParseException}
 */
public static synchronized LocalDate convertToLocalDate(String date) {
    return LocalDate.parse(date, DATE_TIME_FORMAT);
}

/**
 * Convert the result set into a list of IonValues.
 *
 * @param result
 *           The result set to convert.
 * @return a list of IonValues.
 */
public static List<IonValue> toIonValues(Result result) {
    final List<IonValue> valueList = new ArrayList<>();
    result.iterator().forEachRemaining(valueList::add);
    return valueList;
}

/**
 * Get the document ID of a particular document.
 *
 * @param txn
 *           A transaction executor object.
 * @param tableName
 *           Name of the table containing the document.
 * @param identifier
 *           The identifier used to narrow down the search.
 * @param value
 *           Value of the identifier.
 * @return the list of document IDs in the result set.
 */
public static String getDocumentId(final TransactionExecutor txn, final
String tableName,
                                   final String identifier, final String
value) {
    try {

```

```

        final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(value));
        final String query = String.format("SELECT metadata.id FROM
_ql_committed_%s AS p WHERE p.data.%s = ?",
            tableName, identifier);
        Result result = txn.execute(query, parameters);
        if (result.isEmpty()) {
            throw new IllegalStateException("Unable to retrieve document ID
using " + value);
        }
        return getStringValueOfStructField((IonStruct)
result.iterator().next(), "id");
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

/**
 * Get the document by ID.
 *
 * @param qlldbSession
 *      A QLDB session.
 * @param tableName
 *      Name of the table to insert documents into.
 * @param documentId
 *      The unique ID of a document in the Person table.
 * @return a {@link QldbRevision} object.
 * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
 */
public static QldbRevision getDocumentById(QldbSession qlldbSession, String
tableName, String documentId) {
    try {
        final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(documentId));
        final String query = String.format("SELECT c.* FROM _ql_committed_%s
AS c BY docId WHERE docId = ?", tableName);
        Result result = qlldbSession.execute(query, parameters);
        if (result.isEmpty()) {
            throw new IllegalStateException("Unable to retrieve document by
id " + documentId + " in table " + tableName);
        }
        return Constants.MAPPER.readValue(result.iterator().next(),
QldbRevision.class);
    }
}

```

```

        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Return a list of modified document IDs as strings from a DML {@link
     Result}.
     *
     * @param result
     *         The result set from a DML operation.
     * @return the list of document IDs modified by the operation.
     */
    public static List<String> getDocumentIdsFromDmlResult(final Result result)
    {
        final List<String> strings = new ArrayList<>();
        result.iterator().forEachRemaining(row ->
strings.add(getDocumentIdFromDmlResultDocument(row)));
        return strings;
    }

    /**
     * Convert the given DML result row's document ID to string.
     *
     * @param dmlResultDocument
     *         The {@link IonValue} representing the results of a DML
     operation.
     * @return a string of document ID.
     */
    public static String getDocumentIdFromDmlResultDocument(final IonValue
dmlResultDocument) {
        try {
            DmlResultDocument result =
Constants.MAPPER.readValue(dmlResultDocument, DmlResultDocument.class);
            return result.getDocumentId();
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the String value of a given {@link IonStruct} field name.
     * @param struct the {@link IonStruct} from which to get the value.
     * @param fieldName the name of the field from which to get the value.

```

```

    * @return the String value of the field within the given {@link IonStruct}.
    */
    public static String getStringValueOfStructField(final IonStruct struct,
final String fieldName) {
        return ((IonString) struct.get(fieldName)).stringValue();
    }

    /**
     * Return a copy of the given driver's license with updated person Id.
     *
     * @param oldLicense
     *             The old driver's license to update.
     * @param personId
     *             The PersonId of the driver.
     * @return the updated {@link DriversLicense}.
     */
    public static DriversLicense updatePersonIdDriversLicense(final
DriversLicense oldLicense, final String personId) {
        return new DriversLicense(personId, oldLicense.getLicenseNumber(),
oldLicense.getLicenseType(),
            oldLicense.getValidFromDate(), oldLicense.getValidToDate());
    }

    /**
     * Return a copy of the given vehicle registration with updated person Id.
     *
     * @param oldRegistration
     *             The old vehicle registration to update.
     * @param personId
     *             The PersonId of the driver.
     * @return the updated {@link VehicleRegistration}.
     */
    public static VehicleRegistration updateOwnerVehicleRegistration(final
VehicleRegistration oldRegistration,
                                                                    final
String personId) {
        return new VehicleRegistration(oldRegistration.getVin(),
oldRegistration.getLicensePlateNumber(),
            oldRegistration.getState(), oldRegistration.getCity(),
oldRegistration.getPendingPenaltyTicketAmount(),
            oldRegistration.getValidFromDate(),
oldRegistration.getValidToDate(),
            new Owners(new Owner(personId), Collections.emptyList()));
    }

```

```
}
```

Note

- This class uses Ion libraries to provide helper methods that convert your data to and from Ion format.
- The `getDocumentId` method runs a query on a table with the prefix `_ql_committed_`. This is a reserved prefix signifying that you want to query the *committed* view of a table. In this view, your data is nested in the `data` field, and metadata is nested in the `metadata` field.

3. Compile and run the following program (`CreateTable.java`) to create the previously mentioned tables.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```



```
*/

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create tables in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class CreateTable {
    public static final Logger log = LoggerFactory.getLogger(CreateTable.class);

    private CreateTable() { }

    /**
     * Registrations, vehicles, owners, and licenses tables being created in a
     * single transaction.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *           Name of the table to be created.
     * @return the number of tables created.
     */
    public static int createTable(final TransactionExecutor txn, final String
tableName) {
        log.info("Creating the '{}' table...", tableName);
        final String createTable = String.format("CREATE TABLE %s", tableName);
        final Result result = txn.execute(createTable);
        log.info("{} table created successfully.", tableName);
        return SampleData.toIonValues(result).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
```

```

        createTable(txn, Constants.DRIVERS_LICENSE_TABLE_NAME);
        createTable(txn, Constants.PERSON_TABLE_NAME);
        createTable(txn, Constants.VEHICLE_TABLE_NAME);
        createTable(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME);
    });
}
}

```

1.x

```

/*
 * Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;

```

```
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create tables in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class CreateTable {
    public static final Logger log = LoggerFactory.getLogger(CreateTable.class);

    private CreateTable() { }

    /**
     * Registrations, vehicles, owners, and licenses tables being created in a
     * single transaction.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *           Name of the table to be created.
     * @return the number of tables created.
     */
    public static int createTable(final TransactionExecutor txn, final String
tableName) {
        log.info("Creating the '{}' table...", tableName);
        final String createTable = String.format("CREATE TABLE %s", tableName);
        final Result result = txn.execute(createTable);
        log.info("{} table created successfully.", tableName);
        return SampleData.toIonValues(result).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createTable(txn, Constants.DRIVERS_LICENSE_TABLE_NAME);
            createTable(txn, Constants.PERSON_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
}
```

Note

This program demonstrates how to pass a `TransactionExecutor` lambda to the `execute` method. In this example, you run multiple `CREATE TABLE PartiQL` statements in a single transaction by using a lambda expression.

The `execute` method implicitly starts a transaction, runs all of the statements in the lambda, and then auto-commits the transaction.

4. Compile and run the following program (`CreateIndex.java`) to create indexes on the tables, as previously described.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create indexes on tables in a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class CreateIndex {
    public static final Logger log = LoggerFactory.getLogger(CreateIndex.class);

    private CreateIndex() { }

    /**
     * In this example, create indexes for registrations and vehicles tables.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *           Name of the table to be created.
     * @param indexAttribute
     *           The index attribute to use.
     * @return the number of tables created.
     */
    public static int createIndex(final TransactionExecutor txn, final String
tableName, final String indexAttribute) {
        log.info("Creating an index on {}...", indexAttribute);
        final String createIndex = String.format("CREATE INDEX ON %s (%s)",
tableName, indexAttribute);
        final Result r = txn.execute(createIndex);
        return SampleData.toIonValues(r).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
```

```

        createIndex(txn, Constants.PERSON_TABLE_NAME,
Constants.PERSON_GOV_ID_INDEX_NAME);
        createIndex(txn, Constants.VEHICLE_TABLE_NAME,
Constants.VIN_INDEX_NAME);
        createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
Constants.DRIVER_LICENSE_NUMBER_INDEX_NAME);
        createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
Constants.DRIVER_LICENSE_PERSONID_INDEX_NAME);
        createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Constants.VIN_INDEX_NAME);
        createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Constants.VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME);
    });
    log.info("Indexes created successfully!");
}
}

```

1.x

```

/*
 * Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE

```

```
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create indexes on tables in a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class CreateIndex {
    public static final Logger log = LoggerFactory.getLogger(CreateIndex.class);

    private CreateIndex() { }

    /**
     * In this example, create indexes for registrations and vehicles tables.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *           Name of the table to be created.
     * @param indexAttribute
     *           The index attribute to use.
     * @return the number of tables created.
     */
    public static int createIndex(final TransactionExecutor txn, final String
tableName, final String indexAttribute) {
        log.info("Creating an index on {}...", indexAttribute);
        final String createIndex = String.format("CREATE INDEX ON %s (%s)",
tableName, indexAttribute);
        final Result r = txn.execute(createIndex);
        return SampleData.toIonValues(r).size();
    }
}
```

```

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createIndex(txn, Constants.PERSON_TABLE_NAME,
Constants.PERSON_GOV_ID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_TABLE_NAME,
Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
Constants.DRIVER_LICENSE_NUMBER_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
Constants.DRIVER_LICENSE_PERSONID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Constants.VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Indexes created successfully!");
    }
}

```

5. Compile and run the following program (`InsertDocument.java`) to insert the sample data into your tables.

2.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A

```



```
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.DriversLicense;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

/**
 * Insert documents into a table in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class InsertDocument {
    public static final Logger log =
        LoggerFactory.getLogger(InsertDocument.class);

    private InsertDocument() { }

    /**
     * Insert the given list of documents into the specified table and return
     * the document IDs of the inserted documents.
     *
     * @param txn
```

```

    *           The {@link TransactionExecutor} for lambda execute.
    * @param tableName
    *           Name of the table to insert documents into.
    * @param documents
    *           List of documents to insert into the specified table.
    * @return a list of document IDs.
    * @throws IllegalStateException if failed to convert documents into an
    {@link IonValue}.
    */
    public static List<String> insertDocuments(final TransactionExecutor txn,
    final String tableName,
                                           final List documents) {
        log.info("Inserting some documents in the {} table...", tableName);
        try {
            final String query = String.format("INSERT INTO %s ?", tableName);
            final IonValue ionDocuments =
    Constants.MAPPER.writeValueAsIonValue(documents);

            return SampleData.getDocumentIdsFromDmlResult(txn.execute(query,
    ionDocuments));
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
    * Update PersonIds in driver's licenses and in vehicle registrations using
    document IDs.
    *
    * @param documentIds
    *           List of document IDs representing the PersonIds in
    DriversLicense and PrimaryOwners in VehicleRegistration.
    * @param licenses
    *           List of driver's licenses to update.
    * @param registrations
    *           List of registrations to update.
    */
    public static void updatePersonId(final List<String> documentIds, final
    List<DriversLicense> licenses,
                                           final List<VehicleRegistration>
    registrations) {
        for (int i = 0; i < documentIds.size(); ++i) {
            DriversLicense license = SampleData.LICENSES.get(i);
            VehicleRegistration registration = SampleData.REGISTRATIONS.get(i);

```

```

        licenses.add(SampleData.updatePersonIdDriversLicense(license,
documentIds.get(i)));

registrations.add(SampleData.updateOwnerVehicleRegistration(registration,
documentIds.get(i)));
    }
}

public static void main(final String... args) {
    final List<DriversLicense> newDriversLicenses = new ArrayList<>();
    final List<VehicleRegistration> newVehicleRegistrations = new
ArrayList<>();
    ConnectToLedger.getDriver().execute(txn -> {
        List<String> documentIds = insertDocuments(txn,
Constants.PERSON_TABLE_NAME, SampleData.PEOPLE);
        updatePersonId(documentIds, newDriversLicenses,
newVehicleRegistrations);
        insertDocuments(txn, Constants.VEHICLE_TABLE_NAME,
SampleData.VEHICLES);
        insertDocuments(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Collections.unmodifiableList(newVehicleRegistrations));
        insertDocuments(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
Collections.unmodifiableList(newDriversLicenses));
    });
    log.info("Documents inserted successfully!");
}
}

```

1.x

```

/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
of this
 * software and associated documentation files (the "Software"), to deal in the
Software
 * without restriction, including without limitation the rights to use, copy,
modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
 * permit persons to whom the Software is furnished to do so.

```

```
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
* COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
* ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
* THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonValue;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.DriversLicense;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Insert documents into a table in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class InsertDocument {
    public static final Logger log =
        LoggerFactory.getLogger(InsertDocument.class);

    private InsertDocument() { }

    /**
```

```

    * Insert the given list of documents into the specified table and return
    the document IDs of the inserted documents.
    *
    * @param txn
    *           The {@link TransactionExecutor} for lambda execute.
    * @param tableName
    *           Name of the table to insert documents into.
    * @param documents
    *           List of documents to insert into the specified table.
    * @return a list of document IDs.
    * @throws IllegalStateException if failed to convert documents into an
    {@link IonValue}.
    */
    public static List<String> insertDocuments(final TransactionExecutor txn,
    final String tableName,
    final List documents) {
        log.info("Inserting some documents in the {} table...", tableName);
        try {
            final String statement = String.format("INSERT INTO %s ?",
    tableName);
            final IonValue ionDocuments =
    Constants.MAPPER.writeValueAsIonValue(documents);
            final List<IonValue> parameters =
    Collections.singletonList(ionDocuments);
            return SampleData.getDocumentIdsFromDmlResult(txn.execute(statement,
    parameters));
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
    * Update PersonIds in driver's licenses and in vehicle registrations using
    document IDs.
    *
    * @param documentIds
    *           List of document IDs representing the PersonIds in
    DriversLicense and PrimaryOwners in VehicleRegistration.
    * @param licenses
    *           List of driver's licenses to update.
    * @param registrations
    *           List of registrations to update.
    */

```

```

        public static void updatePersonId(final List<String> documentIds, final
List<DriversLicense> licenses,
                                         final List<VehicleRegistration>
registrations) {
            for (int i = 0; i < documentIds.size(); ++i) {
                DriversLicense license = SampleData.LICENSES.get(i);
                VehicleRegistration registration = SampleData.REGISTRATIONS.get(i);
                licenses.add(SampleData.updatePersonIdDriversLicense(license,
documentIds.get(i)));

registrations.add(SampleData.updateOwnerVehicleRegistration(registration,
documentIds.get(i)));
            }
        }

        public static void main(final String... args) {
            final List<DriversLicense> newDriversLicenses = new ArrayList<>();
            final List<VehicleRegistration> newVehicleRegistrations = new
ArrayList<>();
            ConnectToLedger.getDriver().execute(txn -> {
                List<String> documentIds = insertDocuments(txn,
Constants.PERSON_TABLE_NAME, SampleData.PEOPLE);
                updatePersonId(documentIds, newDriversLicenses,
newVehicleRegistrations);
                insertDocuments(txn, Constants.VEHICLE_TABLE_NAME,
SampleData.VEHICLES);
                insertDocuments(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Collections.unmodifiableList(newVehicleRegistrations));
                insertDocuments(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
Collections.unmodifiableList(newDriversLicenses));
            }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
            log.info("Documents inserted successfully!");
        }
    }
}

```

Note

- This program demonstrates how to call the execute method with parameterized values. You can pass data parameters of type `IonValue` in addition to the PartiQL statement that you want to run. Use a question mark (?) as a variable placeholder in your statement string.

- If an INSERT statement succeeds, it returns the `id` of each inserted document.

Next, you can use SELECT statements to read data from the tables in the vehicle-registration ledger. Proceed to [Step 4: Query the tables in a ledger](#).

Step 4: Query the tables in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses [PartiQL](#) as its query language and [Amazon Ion](#) as its document-oriented data model.

PartiQL is an open-source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open-source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use SELECT statements to read data from the tables in the vehicle-registration ledger.

Warning

When you run a query in QLDB without an indexed lookup, it invokes a full table scan. PartiQL supports such queries because it's SQL compatible. However, *don't* run table scans for production use cases in QLDB. Table scans can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator on an indexed field or a document ID; for example, WHERE `indexedField = 123` or WHERE `indexedField IN (456, 789)`. For more information, see [Optimizing query performance](#).

To query the tables

- Compile and run the following program (`FindVehicles.java`) to query all vehicles registered under a person in your ledger.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
```



```

import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find all vehicles registered under a person.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class FindVehicles {
    public static final Logger log =
        LoggerFactory.getLogger(FindVehicles.class);

    private FindVehicles() { }

    /**
     * Find vehicles registered under a driver using their government ID.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param govId
     *           The government ID of the owner.
     * @throws IllegalStateException if failed to convert parameters into {@link
     * IonValue}.
     */
    public static void findVehiclesForOwner(final TransactionExecutor txn, final
String govId) {
        try {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            final String query = "SELECT v FROM Vehicle AS v INNER JOIN
VehicleRegistration AS r "
                + "ON v.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId
= ?";

            final Result result = txn.execute(query,
Constants.MAPPER.writeValueAsIonValue(documentId));
            log.info("List of Vehicles for owner with GovId: {}...", govId);
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
}

```

```

    public static void main(final String... args) {
        final Person person = SampleData.PEOPLE.get(0);
        ConnectToLedger.getDriver().execute(txn -> {
            findVehiclesForOwner(txn, person.getGovId());
        });
    }
}

```

1.x

```

/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

```

```

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find all vehicles registered under a person.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class FindVehicles {
    public static final Logger log =
        LoggerFactory.getLogger(FindVehicles.class);

    private FindVehicles() { }

    /**
     * Find vehicles registered under a driver using their government ID.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param govId
     *           The government ID of the owner.
     * @throws IllegalStateException if failed to convert parameters into {@link
     IonValue}.
     */
    public static void findVehiclesForOwner(final TransactionExecutor txn, final
String govId) {
        try {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            final String query = "SELECT v FROM Vehicle AS v INNER JOIN
VehicleRegistration AS r "
                + "ON v.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId
= ?";

            final Result result = txn.execute(query,
Constants.MAPPER.writeValueAsIonValue(documentId));
            log.info("List of Vehicles for owner with GovId: {}...", govId);
            ScanTable.printDocuments(result);

```

```
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final Person person = SampleData.PEOPLE.get(0);
        ConnectToLedger.getDriver().execute(txn -> {
            findVehiclesForOwner(txn, person.getGovId());
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
}
```

Note

First, this program queries the `Person` table for the document with `GovId` `LEWISR261LL` to get its `id` metadata field.

Then, it uses this document `id` as a foreign key to query the `VehicleRegistration` table by `PrimaryOwner.PersonId`. It also joins `VehicleRegistration` with the `Vehicle` table on the `VIN` field.

To learn about modifying documents in the tables in the vehicle-registration ledger, see [Step 5: Modify documents in a ledger](#).

Step 5: Modify documents in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Now that you have data to work with, you can start making changes to documents in the vehicle-registration ledger in Amazon QLDB. In this step, the following code examples demonstrate how to run data manipulation language (DML) statements. These statements update the primary owner of one vehicle and add a secondary owner to another vehicle.

To modify documents

1. Compile and run the following program (TransferVehicleOwnership.java) to update the primary owner of the vehicle with VIN 1N4AL11D75C109151 in your ledger.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonReaderBuilder;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
```

```

import java.util.LinkedHashMap;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class TransferVehicleOwnership {
    public static final Logger log =
        LoggerFactory.getLogger(TransferVehicleOwnership.class);

    private TransferVehicleOwnership() { }

    /**
     * Query a driver's information using the given ID.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param documentId
     *           The unique ID of a document in the Person table.
     * @return a {@link Person} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
     IonValue}.
     */
    public static Person findPersonFromDocumentId(final TransactionExecutor txn,
        final String documentId) {
        try {
            log.info("Finding person for documentId: {}", documentId);
            final String query = "SELECT p.* FROM Person AS p BY pid WHERE pid
= ?";

```

```

        Result result = txn.execute(query,
Constants.MAPPER.writeValueAsIonValue(documentId));
        if (result.isEmpty()) {
            throw new IllegalStateException("Unable to find person with ID:
" + documentId);
        }

        return Constants.MAPPER.readValue(result.iterator().next(),
Person.class);
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

/**
 * Find the primary owner for the given VIN.
 *
 * @param txn
 *           The {@link TransactionExecutor} for lambda execute.
 * @param vin
 *           Unique VIN for a vehicle.
 * @return a {@link Person} object.
 * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
 */
public static Person findPrimaryOwnerForVehicle(final TransactionExecutor
txn, final String vin) {
    try {
        log.info("Finding primary owner for vehicle with Vin: {}", vin);
        final String query = "SELECT Owners.PrimaryOwner.PersonId FROM
VehicleRegistration AS v WHERE v.VIN = ?";
        final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
        Result result = txn.execute(query, parameters);
        final List<IonStruct> documents = ScanTable.toIonStructs(result);
        ScanTable.printDocuments(documents);
        if (documents.isEmpty()) {
            throw new IllegalStateException("Unable to find registrations
with VIN: " + vin);
        }

        final IonReader reader =
IonReaderBuilder.standard().build(documents.get(0));

```

```

        final String personId = Constants.MAPPER.readValue(reader,
LinkedHashMap.class).get("PersonId").toString();
        return findPersonFromDocumentId(txn, personId);
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

/**
 * Update the primary owner for a vehicle registration with the given
documentId.
 *
 * @param txn
 *           The {@link TransactionExecutor} for lambda execute.
 * @param vin
 *           Unique VIN for a vehicle.
 * @param documentId
 *           New PersonId for the primary owner.
 * @throws IllegalStateException if no vehicle registration was found using
the given document ID and VIN, or if failed
 * to convert parameters into {@link IonValue}.
 */
public static void updateVehicleRegistration(final TransactionExecutor txn,
final String vin, final String documentId) {
    try {
        log.info("Updating primary owner for vehicle with Vin: {}...", vin);
        final String query = "UPDATE VehicleRegistration AS v SET
v.Owners.PrimaryOwner = ? WHERE v.VIN = ?";

        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(Constants.MAPPER.writeValueAsIonValue(new
Owner(documentId)));
        parameters.add(Constants.MAPPER.writeValueAsIonValue(vin));

        Result result = txn.execute(query, parameters);
        ScanTable.printDocuments(result);
        if (result.isEmpty()) {
            throw new IllegalStateException("Unable to transfer vehicle,
could not find registration.");
        } else {
            log.info("Successfully transferred vehicle with VIN '{}' to new
owner.", vin);
        }
    } catch (IOException ioe) {

```



```

        throw new IllegalStateException(ioe);
    }
}

public static void main(final String... args) {
    final String vin = SampleData.VEHICLES.get(0).getVin();
    final String primaryOwnerGovId = SampleData.PEOPLE.get(0).getGovId();
    final String newPrimaryOwnerGovId = SampleData.PEOPLE.get(1).getGovId();

    ConnectToLedger.getDriver().execute(txn -> {
        final Person primaryOwner = findPrimaryOwnerForVehicle(txn, vin);
        if (!primaryOwner.getGovId().equals(primaryOwnerGovId)) {
            // Verify the primary owner.
            throw new IllegalStateException("Incorrect primary owner
identified for vehicle, unable to transfer.");
        }

        final String newOwner = Person.getDocumentIdByGovId(txn,
newPrimaryOwnerGovId);
        updateVehicleRegistration(txn, vin, newOwner);
    });
    log.info("Successfully transferred vehicle ownership!");
}
}

```

1.x

```

/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,

```

```
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonReaderBuilder;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class TransferVehicleOwnership {
    public static final Logger log =
        LoggerFactory.getLogger(TransferVehicleOwnership.class);
```

```

private TransferVehicleOwnership() { }

/**
 * Query a driver's information using the given ID.
 *
 * @param txn
 *           The {@link TransactionExecutor} for lambda execute.
 * @param documentId
 *           The unique ID of a document in the Person table.
 * @return a {@link Person} object.
 * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
 */
public static Person findPersonFromDocumentId(final TransactionExecutor txn,
final String documentId) {
    try {
        log.info("Finding person for documentId: {}", documentId);
        final String query = "SELECT p.* FROM Person AS p BY pid WHERE pid
= ?";

        Result result = txn.execute(query,
Constants.MAPPER.writeValueAsIonValue(documentId));
        if (result.isEmpty()) {
            throw new IllegalStateException("Unable to find person with ID:
" + documentId);
        }

        return Constants.MAPPER.readValue(result.iterator().next(),
Person.class);
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

/**
 * Find the primary owner for the given VIN.
 *
 * @param txn
 *           The {@link TransactionExecutor} for lambda execute.
 * @param vin
 *           Unique VIN for a vehicle.
 * @return a {@link Person} object.
 * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.

```

```

    */
    public static Person findPrimaryOwnerForVehicle(final TransactionExecutor
txn, final String vin) {
        try {
            log.info("Finding primary owner for vehicle with Vin: {}", vin);
            final String query = "SELECT Owners.PrimaryOwner.PersonId FROM
VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            Result result = txn.execute(query, parameters);
            final List<IonStruct> documents = ScanTable.toIonStructs(result);
            ScanTable.printDocuments(documents);
            if (documents.isEmpty()) {
                throw new IllegalStateException("Unable to find registrations
with VIN: " + vin);
            }

            final IonReader reader =
IonReaderBuilder.standard().build(documents.get(0));
            final String personId = Constants.MAPPER.readValue(reader,
LinkedHashMap.class).get("PersonId").toString();
            return findPersonFromDocumentId(txn, personId);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Update the primary owner for a vehicle registration with the given
documentId.
     *
     * @param txn
     *             The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *             Unique VIN for a vehicle.
     * @param documentId
     *             New PersonId for the primary owner.
     * @throws IllegalStateException if no vehicle registration was found using
the given document ID and VIN, or if failed
     * to convert parameters into {@link IonValue}.
     */
    public static void updateVehicleRegistration(final TransactionExecutor txn,
final String vin, final String documentId) {
        try {

```

```

        log.info("Updating primary owner for vehicle with Vin: {}", vin);
        final String query = "UPDATE VehicleRegistration AS v SET
v.Owners.PrimaryOwner = ? WHERE v.VIN = ?";

        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(Constants.MAPPER.writeValueAsIonValue(new
Owner(documentId)));
        parameters.add(Constants.MAPPER.writeValueAsIonValue(vin));

        Result result = txn.execute(query, parameters);
        ScanTable.printDocuments(result);
        if (result.isEmpty()) {
            throw new IllegalStateException("Unable to transfer vehicle,
could not find registration.");
        } else {
            log.info("Successfully transferred vehicle with VIN '{}' to new
owner.", vin);
        }
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

public static void main(final String... args) {
    final String vin = SampleData.VEHICLES.get(0).getVin();
    final String primaryOwnerGovId = SampleData.PEOPLE.get(0).getGovId();
    final String newPrimaryOwnerGovId = SampleData.PEOPLE.get(1).getGovId();

    ConnectToLedger.getDriver().execute(txn -> {
        final Person primaryOwner = findPrimaryOwnerForVehicle(txn, vin);
        if (!primaryOwner.getGovId().equals(primaryOwnerGovId)) {
            // Verify the primary owner.
            throw new IllegalStateException("Incorrect primary owner
identified for vehicle, unable to transfer.");
        }

        final String newOwner = Person.getDocumentIdByGovId(txn,
newPrimaryOwnerGovId);
        updateVehicleRegistration(txn, vin, newOwner);
    }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    log.info("Successfully transferred vehicle ownership!");
}
}

```

2. Compile and run the following program (`AddSecondaryOwner.java`) to add a secondary owner to the vehicle with VIN `KM8SRDHF6EU074761` in your ledger.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;
```

```

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Owners;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Finds and adds secondary owners for a vehicle.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class AddSecondaryOwner {
    public static final Logger log =
        LoggerFactory.getLogger(AddSecondaryOwner.class);

    private AddSecondaryOwner() { }

    /**
     * Check whether a secondary owner has already been registered for the given
     VIN.
     *
     * @param txn
     *           The TransactionExecutor for lambda execute.
     * @param vin
     *           Unique VIN for a vehicle.
     * @param secondaryOwnerId
     *           The secondary owner to add.
     * @return {@code true} if the given secondary owner has already been
     registered, {@code false} otherwise.
     * @throws IllegalStateException if failed to convert VIN to an IonValue.
     */
    public static boolean isSecondaryOwnerForVehicle(final TransactionExecutor
txn, final String vin,
                                                    final String
secondaryOwnerId) {
        try {
            log.info("Finding secondary owners for vehicle with VIN: {}",
vin);

            final String query = "SELECT Owners.SecondaryOwners FROM
VehicleRegistration AS v WHERE v.VIN = ?";

```

```

        final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
        final Result result = txn.execute(query, parameters);
        final Iterator<IonValue> itr = result.iterator();
        if (!itr.hasNext()) {
            return false;
        }

        final Owners owners = Constants.MAPPER.readValue(itr.next(),
Owners.class);
        if (null != owners.getSecondaryOwners()) {
            for (Owner owner : owners.getSecondaryOwners()) {
                if (secondaryOwnerId.equalsIgnoreCase(owner.getPersonId()))
{
                    return true;
                }
            }
        }

        return false;
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

/**
 * Adds a secondary owner for the specified VIN.
 *
 * @param txn
 *           The {@link TransactionExecutor} for lambda execute.
 * @param vin
 *           Unique VIN for a vehicle.
 * @param secondaryOwner
 *           The secondary owner to add.
 * @throws IllegalStateException if failed to convert parameter into an
{@link IonValue}.
 */
public static void addSecondaryOwnerForVin(final TransactionExecutor txn,
final String vin,
final String secondaryOwner) {
    try {
        log.info("Inserting secondary owner for vehicle with VIN: {}...",
vin);
    }
}

```



```

        final String query = String.format("FROM VehicleRegistration AS v
WHERE v.VIN = ?" +
        "INSERT INTO v.Owners.SecondaryOwners VALUE ?");
        final IonValue newOwner = Constants.MAPPER.writeValueAsIonValue(new
Owner(secondaryOwner));
        final IonValue vinAsIonValue =
Constants.MAPPER.writeValueAsIonValue(vin);
        Result result = txn.execute(query, vinAsIonValue, newOwner);
        log.info("VehicleRegistration Document IDs which had secondary
owners added: ");
        ScanTable.printDocuments(result);
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

public static void main(final String... args) {
    final String vin = SampleData.VEHICLES.get(1).getVin();
    final String govId = SampleData.PEOPLE.get(0).getGovId();

    ConnectToLedger.getDriver().execute(txn -> {
        final String documentId = Person.getDocumentIdByGovId(txn, govId);
        if (isSecondaryOwnerForVehicle(txn, vin, documentId)) {
            log.info("Person with ID {} has already been added as a
secondary owner of this vehicle.", govId);
        } else {
            addSecondaryOwnerForVin(txn, vin, documentId);
        }
    });
    log.info("Secondary owners successfully updated.");
}
}

```

1.x

```

/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
of this
 * software and associated documentation files (the "Software"), to deal in the
Software

```

```
* without restriction, including without limitation the rights to use, copy,
* modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
* and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
* COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
* ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
* THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Owners;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Finds and adds secondary owners for a vehicle.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
```

```

public final class AddSecondaryOwner {
    public static final Logger log =
        LoggerFactory.getLogger(AddSecondaryOwner.class);

    private AddSecondaryOwner() { }

    /**
     * Check whether a secondary owner has already been registered for the given
     VIN.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *           Unique VIN for a vehicle.
     * @param secondaryOwnerId
     *           The secondary owner to add.
     * @return {@code true} if the given secondary owner has already been
     registered, {@code false} otherwise.
     * @throws IllegalStateException if failed to convert VIN to an {@link
     IonValue}.
     */
    public static boolean isSecondaryOwnerForVehicle(final TransactionExecutor
txn, final String vin,
                                                    final String
secondaryOwnerId) {
        try {
            log.info("Finding secondary owners for vehicle with VIN: {}",
vin);

            final String query = "SELECT Owners.SecondaryOwners FROM
VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            final Iterator<IonValue> itr = result.iterator();
            if (!itr.hasNext()) {
                return false;
            }

            final Owners owners = Constants.MAPPER.readValue(itr.next(),
Owners.class);
            if (null != owners.getSecondaryOwners()) {
                for (Owner owner : owners.getSecondaryOwners()) {
                    if (secondaryOwnerId.equalsIgnoreCase(owner.getPersonId()))
{

```

```

        return true;
    }
}

    return false;
} catch (IOException ioe) {
    throw new IllegalStateException(ioe);
}
}

/**
 * Adds a secondary owner for the specified VIN.
 *
 * @param txn
 *           The {@link TransactionExecutor} for lambda execute.
 * @param vin
 *           Unique VIN for a vehicle.
 * @param secondaryOwner
 *           The secondary owner to add.
 * @throws IllegalStateException if failed to convert parameter into an
 * {@link IonValue}.
 */
public static void addSecondaryOwnerForVin(final TransactionExecutor txn,
final String vin,
final String secondaryOwner) {
    try {
        log.info("Inserting secondary owner for vehicle with VIN: {}",
vin);
        final String query = String.format("FROM VehicleRegistration AS v
WHERE v.VIN = '%s' " +
            "INSERT INTO v.Owners.SecondaryOwners VALUE ?", vin);
        final IonValue newOwner = Constants.MAPPER.writeValueAsIonValue(new
Owner(secondaryOwner));
        Result result = txn.execute(query, newOwner);
        log.info("VehicleRegistration Document IDs which had secondary
owners added: ");
        ScanTable.printDocuments(result);
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

public static void main(final String... args) {

```

```

        final String vin = SampleData.VEHICLES.get(1).getVin();
        final String govId = SampleData.PEOPLE.get(0).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            if (isSecondaryOwnerForVehicle(txn, vin, documentId)) {
                log.info("Person with ID {} has already been added as a
secondary owner of this vehicle.", govId);
            } else {
                addSecondaryOwnerForVin(txn, vin, documentId);
            }
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Secondary owners successfully updated.");
    }
}

```

To review these changes in the vehicle-registration ledger, see [Step 6: View the revision history for a document](#).

Step 6: View the revision history for a document

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After modifying registration data for a vehicle in the previous step, you can query the history of all its registered owners and any other updated fields. In this step, you query the revision history of a document in the `VehicleRegistration` table in your vehicle-registration ledger.

To view the revision history

1. Review the following program (`QueryHistory.java`).

2.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0

```

```
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
of this
* software and associated documentation files (the "Software"), to deal in the
Software
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

```
package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.time.Instant;
import java.time.temporal.ChronoUnit;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

/**
 * Query a table's history for a particular set of documents.
 *
 * This code expects that you have AWS credentials setup per:
```

```

* http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
*/
public final class QueryHistory {
    public static final Logger log =
    LoggerFactory.getLogger(QueryHistory.class);
    private static final int THREE_MONTHS = 90;

    private QueryHistory() { }

    /**
     * In this example, query the 'VehicleRegistration' history table to find
     all previous primary owners for a VIN.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *           VIN to find previous primary owners for.
     * @param query
     *           The query to find previous primary owners.
     * @throws IllegalStateException if failed to convert document ID to an
     {@link IonValue}.
     */
    public static void previousPrimaryOwners(final TransactionExecutor txn,
    final String vin, final String query) {
        try {
            final String docId = VehicleRegistration.getDocumentIdByVin(txn,
    vin);

            log.info("Querying the 'VehicleRegistration' table's history using
    VIN: {}...", vin);
            final Result result = txn.execute(query,
    Constants.MAPPER.writeValueAsIonValue(docId));
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String threeMonthsAgo = Instant.now().minus(THREE_MONTHS,
    ChronoUnit.DAYS).toString();
        final String query = String.format("SELECT data.Owners.PrimaryOwner,
    metadata.version "

```

```

                                + "FROM history(VehicleRegistration,
                                + "AS h WHERE h.metadata.id = ?",
                                threeMonthsAgo);
                                ConnectToLedger.getDriver().execute(txn -> {
                                    final String vin = SampleData.VEHICLES.get(0).getVin();
                                    previousPrimaryOwners(txn, vin, query);
                                });
                                log.info("Successfully queried history.");
                            }
                        }
                    }
                }
            }
        }
    }
}

```

1.x

```

/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qlldb.tutorial;

import com.amazon.ion.IonValue;

```



```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

import java.io.IOException;
import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.Collections;
import java.util.List;

/**
 * Query a table's history for a particular set of documents.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class QueryHistory {
    public static final Logger log =
        LoggerFactory.getLogger(QueryHistory.class);
    private static final int THREE_MONTHS = 90;

    private QueryHistory() { }

    /**
     * In this example, query the 'VehicleRegistration' history table to find
     * all previous primary owners for a VIN.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *           VIN to find previous primary owners for.
     * @param query
     *           The query to find previous primary owners.
     * @throws IllegalStateException if failed to convert document ID to an
     *                               {@link IonValue}.
     */
    public static void previousPrimaryOwners(final TransactionExecutor txn,
        final String vin, final String query) {
        try {

```

```

        final String docId = VehicleRegistration.getDocumentIdByVin(txn,
vin);
        final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(docId));
        log.info("Querying the 'VehicleRegistration' table's history using
VIN: {}...", vin);
        final Result result = txn.execute(query, parameters);
        ScanTable.printDocuments(result);
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}

public static void main(final String... args) {
    final String threeMonthsAgo = Instant.now().minus(THREE_MONTHS,
ChronoUnit.DAYS).toString();
    final String query = String.format("SELECT data.Owners.PrimaryOwner,
metadata.version "
                                     + "FROM history(VehicleRegistration,
`%s`) "
                                     + "AS h WHERE h.metadata.id = ?",
threeMonthsAgo);
    ConnectToLedger.getDriver().execute(txn -> {
        final String vin = SampleData.VEHICLES.get(0).getVin();
        previousPrimaryOwners(txn, vin, query);
    }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    log.info("Successfully queried history.");
}
}

```

Note

- You can view the revision history of a document by querying the built-in [History function](#) in the following syntax.

```

SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
[ WHERE h.metadata.id = 'id' ]

```

- The *start-time* and *end-time* are both optional. They're Amazon Ion literal values that can be denoted with backticks (``...``). To learn more, see [Querying Ion with PartiQL in Amazon QLDB](#).

- As a best practice, qualify a history query with both a date range (*start-time* and *end-time*) and a document ID (`metadata.id`). QLDB processes SELECT queries in transactions, which are subject to a [transaction timeout limit](#).

QLDB history is indexed by document ID, and you can't create additional history indexes at this time. History queries that include a start time and end time gain the benefit of date range qualification.

2. Compile and run the `QueryHistory.java` program to query the revision history of the `VehicleRegistration` document with VIN `1N4AL11D75C109151`.

To cryptographically verify a document revision in the `vehicle-registration` ledger, proceed to [Step 7: Verify a document in a ledger](#).

Step 7: Verify a document in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. To learn more about how verification and cryptographic hashing work in QLDB, see [Data verification in Amazon QLDB](#).

In this step, you verify a document revision in the `VehicleRegistration` table in your `vehicle-registration` ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To verify a document revision

1. Review the following `.java` files, which represent QLDB objects that are required for verification and utility classes with helper methods for `Ion` and string values.

1. `BlockAddress.java`

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import java.util.Objects;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Represents the BlockAddress field of a QLDB document.
 */
public final class BlockAddress {

    private static final Logger log =
        LoggerFactory.getLogger(BlockAddress.class);
```

```
private final String strandId;
private final long sequenceNo;

@JsonCreator
public BlockAddress(@JsonProperty("strandId") final String strandId,
                    @JsonProperty("sequenceNo") final long sequenceNo) {
    this.strandId = strandId;
    this.sequenceNo = sequenceNo;
}

public long getSequenceNo() {
    return sequenceNo;
}

public String getStrandId() {
    return strandId;
}

@Override
public String toString() {
    return "BlockAddress{"
        + "strandId='" + strandId + '\''
        + ", sequenceNo=" + sequenceNo
        + '}';
}

@Override
public boolean equals(final Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }
    BlockAddress that = (BlockAddress) o;
    return sequenceNo == that.sequenceNo
        && strandId.equals(that.strandId);
}

@Override
public int hashCode() {
    // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
    // properties.
    return Objects.hash(strandId, sequenceNo);
}
```

```
    // CHECKSTYLE:ON
  }
}
```

2. Proof.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;

import java.util.ArrayList;
import java.util.List;

/**
 * A Java representation of the {@link Proof} object.
 */
```

```

* Returned from the {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision(GetRevisionRequest)} api.
*/
public final class Proof {
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();

    private List<byte[]> internalHashes;

    public Proof(final List<byte[]> internalHashes) {
        this.internalHashes = internalHashes;
    }

    public List<byte[]> getInternalHashes() {
        return internalHashes;
    }

    /**
     * Decodes a {@link Proof} from an ion text String. This ion text is returned
in
     * a {@link GetRevisionResult#getProof()}
     *
     * @param ionText
     *             The ion text representing a {@link Proof} object.
     * @return {@link JournalBlock} parsed from the ion text.
     * @throws IllegalStateException if failed to parse the {@link Proof} object
from the given ion text.
     */
    public static Proof fromBlob(final String ionText) {
        try {
            IonReader reader = SYSTEM.newReader(ionText);
            List<byte[]> list = new ArrayList<>();
            reader.next();
            reader.stepIn();
            while (reader.next() != null) {
                list.add(reader.newBytes());
            }
            return new Proof(list);
        } catch (Exception e) {
            throw new IllegalStateException("Failed to parse a Proof from byte
array");
        }
    }
}

```

3. QldbIonUtils.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonValue;
import com.amazon.ionhash.IonHashReader;
import com.amazon.ionhash.IonHashReaderBuilder;
import com.amazon.ionhash.MessageDigestIonHasherProvider;
import software.amazon.qldb.tutorial.Constants;

public class QldbIonUtils {

    private static MessageDigestIonHasherProvider ionHasherProvider = new
    MessageDigestIonHasherProvider("SHA-256");

    private QldbIonUtils() {}
}
```



```

/**
 * Builds a hash value from the given {@link IonValue}.
 *
 * @param ionValue
 *           The {@link IonValue} to hash.
 * @return a byte array representing the hash value.
 */
public static byte[] hashIonValue(final IonValue ionValue) {
    IonReader reader = Constants.SYSTEM.newReader(ionValue);
    IonHashReader hashReader = IonHashReaderBuilder.standard()
        .withHasherProvider(ionHasherProvider)
        .withReader(reader)
        .build();
    while (hashReader.next() != null) { }
    return hashReader.digest();
}
}

```

4. QldbStringUtils.java

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```

```
*/

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonTextWriterBuilder;
import com.amazonaws.services.qldb.model.GetBlockResult;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.ValueHolder;

import java.io.IOException;

/**
 * Helper methods to pretty-print certain QLDB response types.
 */
public class QldbStringUtils {

    private QldbStringUtils() {}

    /**
     * Returns the string representation of a given {@link ValueHolder}.
     * Adapted from the AWS SDK autogenerated {@code toString()} method, with
     sensitive values un-redacted.
     * Additionally, this method pretty-prints any IonText included in the {@link
     ValueHolder}.
     *
     * @param valueHolder the {@link ValueHolder} to convert to a String.
     * @return the String representation of the supplied {@link ValueHolder}.
     */
    public static String toUnredactedString(ValueHolder valueHolder) {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        if (valueHolder.getIonText() != null) {

            sb.append("IonText: ");
            IonWriter prettyWriter = IonTextWriterBuilder.pretty().build(sb);
            try {

                prettyWriter.writeValues(IonReaderBuilder.standard().build(valueHolder.getIonText()));
            } catch (IOException ioe) {
                sb.append("**Exception while printing this IonText**");
            }
        }
    }
}
```

```

        sb.append("}");
        return sb.toString();
    }

    /**
     * Returns the string representation of a given {@link GetBlockResult}.
     * Adapted from the AWS SDK autogenerated {@code toString()} method, with
     sensitive values un-redacted.
     *
     * @param getBlockResult the {@link GetBlockResult} to convert to a String.
     * @return the String representation of the supplied {@link GetBlockResult}.
     */
    public static String toUnredactedString(GetBlockResult getBlockResult) {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        if (getBlockResult.getBlock() != null) {
            sb.append("Block:");
            ").append(toUnredactedString(getBlockResult.getBlock())).append(",");
        }

        if (getBlockResult.getProof() != null) {
            sb.append("Proof:");
            ").append(toUnredactedString(getBlockResult.getProof()));
        }

        sb.append("}");
        return sb.toString();
    }

    /**
     * Returns the string representation of a given {@link GetDigestResult}.
     * Adapted from the AWS SDK autogenerated {@code toString()} method, with
     sensitive values un-redacted.
     *
     * @param getDigestResult the {@link GetDigestResult} to convert to a String.
     * @return the String representation of the supplied {@link GetDigestResult}.
     */
    public static String toUnredactedString(GetDigestResult getDigestResult) {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        if (getDigestResult.getDigest() != null) {
            sb.append("Digest:");
            ").append(getDigestResult.getDigest()).append(",");
        }
    }

```

```

    }

    if (getDigestResult.getDigestTipAddress() != null) {
        sb.append("DigestTipAddress:");
    }.append(toUnredactedString(getDigestResult.getDigestTipAddress()));
    }

    sb.append("}");
    return sb.toString();
}
}

```

5. Verifier.java

2.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this
 * software and associated documentation files (the "Software"), to deal in
 * the Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
 * A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

```

```
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.util.Base64;

import software.amazon.qldb.tutorial.qldb.Proof;

/**
 * Encapsulates the logic to verify the integrity of revisions or blocks in a
 * QLDB ledger.
 *
 * The main entry point is {@link #verify(byte[], byte[], String)}.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class Verifier {
    public static final Logger log = LoggerFactory.getLogger(Verifier.class);
    private static final int HASH_LENGTH = 32;
    private static final int UPPER_BOUND = 8;

    /**
     * Compares two hashes by their signed byte values in little-
     * endian order.
     */
    private static Comparator<byte[]> hashComparator = (h1, h2) -> {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }
        for (int i = h1.length - 1; i >= 0; i--) {
            int byteEqual = Byte.compare(h1[i], h2[i]);

```

```

        if (byteEqual != 0) {
            return byteEqual;
        }

        return 0;
    };

    private Verifier() { }

    /**
     * Verify the integrity of a document with respect to a QLDB ledger
     digest.
     *
     * The verification algorithm includes the following steps:
     *
     * 1. {@link #buildCandidateDigest(Proof, byte[])} build the candidate
     digest from the internal hashes
     * in the {@link Proof}.
     * 2. Check that the {@code candidateLedgerDigest} is equal to the {@code
     ledgerDigest}.
     *
     * @param documentHash
     *             The hash of the document to be verified.
     * @param digest
     *             The QLDB ledger digest. This digest should have been
     retrieved using
     *             {@link com.amazonaws.services.qldb.AmazonQLDB#getDigest}
     * @param proofBlob
     *             The ion encoded bytes representing the {@link Proof}
     associated with the supplied
     *             {@code digestTipAddress} and {@code address} retrieved
     using
     *             {@link
     com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
     * @return {@code true} if the record is verified or {@code false} if it
     is not verified.
     */
    public static boolean verify(
        final byte[] documentHash,
        final byte[] digest,
        final String proofBlob
    ) {
        Proof proof = Proof.fromBlob(proofBlob);

```

```

        byte[] candidateDigest = buildCandidateDigest(proof, documentHash);

        return Arrays.equals(digest, candidateDigest);
    }

    /**
     * Build the candidate digest representing the entire ledger from the
     * internal hashes of the {@link Proof}.
     *
     * @param proof
     *           A Java representation of {@link Proof}
     *           returned from {@link
     com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
     * @param leafHash
     *           Leaf hash to build the candidate digest with.
     * @return a byte array of the candidate digest.
     */
    private static byte[] buildCandidateDigest(final Proof proof, final byte[]
leafHash) {
        return calculateRootHashFromInternalHashes(proof.getInternalHashes(),
leafHash);
    }

    /**
     * Get a new instance of {@link MessageDigest} using the SHA-256
     * algorithm.
     *
     * @return an instance of {@link MessageDigest}.
     * @throws IllegalStateException if the algorithm is not available on the
     current JVM.
     */
    static MessageDigest newMessageDigest() {
        try {
            return MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            log.error("Failed to create SHA-256 MessageDigest", e);
            throw new IllegalStateException("SHA-256 message digest is
unavailable", e);
        }
    }

    /**
     * Takes two hashes, sorts them, concatenates them, and then returns the

```

```

    * hash of the concatenated array.
    *
    * @param h1
    *         Byte array containing one of the hashes to compare.
    * @param h2
    *         Byte array containing one of the hashes to compare.
    * @return the concatenated array of hashes.
    */
    public static byte[] dot(final byte[] h1, final byte[] h2) {
        if (h1.length == 0) {
            return h2;
        }
        if (h2.length == 0) {
            return h1;
        }
        byte[] concatenated = new byte[h1.length + h2.length];
        if (hashComparator.compare(h1, h2) < 0) {
            System.arraycopy(h1, 0, concatenated, 0, h1.length);
            System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
        } else {
            System.arraycopy(h2, 0, concatenated, 0, h2.length);
            System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
        }
        MessageDigest messageDigest = new MessageDigest();
        messageDigest.update(concatenated);

        return messageDigest.digest();
    }

    /**
     * Starting with the provided {@code leafHash} combined with the provided
     * {@code internalHashes}
     * pairwise until only the root hash remains.
     *
     * @param internalHashes
     *         Internal hashes of Merkle tree.
     * @param leafHash
     *         Leaf hashes of Merkle tree.
     * @return the root hash.
     */
    private static byte[] calculateRootHashFromInternalHashes(final
        List<byte[]> internalHashes, final byte[] leafHash) {
        return internalHashes.stream().reduce(leafHash, Verifier::dot);
    }

```



```

/**
 * Flip a single random bit in the given byte array. This method is used
to demonstrate
 * QLDB's verification features.
 *
 * @param original
 *         The original byte array.
 * @return the altered byte array with a single random bit changed.
 */
public static byte[] flipRandomBit(final byte[] original) {
    if (original.length == 0) {
        throw new IllegalArgumentException("Array cannot be empty!");
    }
    int alteredPosition =
ThreadLocalRandom.current().nextInt(original.length);
    int b = ThreadLocalRandom.current().nextInt(UPPER_BOUND);
    byte[] altered = new byte[original.length];
    System.arraycopy(original, 0, altered, 0, original.length);
    altered[alteredPosition] = (byte) (altered[alteredPosition] ^ (1 <<
b));
    return altered;
}

public static String toBase64(byte[] arr) {
    return new String(Base64.encode(arr), StandardCharsets.UTF_8);
}

/**
 * Convert a {@link ByteBuffer} into byte array.
 *
 * @param buffer
 *         The {@link ByteBuffer} to convert.
 * @return the converted byte array.
 */
public static byte[] convertByteBufferToByteArray(final ByteBuffer buffer)
{
    byte[] arr = new byte[buffer.remaining()];
    buffer.get(arr);
    return arr;
}

/**

```

```

    * Calculates the root hash from a list of hashes that represent the base
    of a Merkle tree.
    *
    * @param hashes
    *         The list of byte arrays representing hashes making up base
    of a Merkle tree.
    * @return a byte array that is the root hash of the given list of hashes.
    */
    public static byte[] calculateMerkleTreeRootHash(List<byte[]> hashes) {
        if (hashes.isEmpty()) {
            return new byte[0];
        }

        List<byte[]> remaining = combineLeafHashes(hashes);
        while (remaining.size() > 1) {
            remaining = combineLeafHashes(remaining);
        }
        return remaining.get(0);
    }

    private static List<byte[]> combineLeafHashes(List<byte[]> hashes) {
        List<byte[]> combinedHashes = new ArrayList<>();
        Iterator<byte[]> it = hashes.stream().iterator();

        while (it.hasNext()) {
            byte[] left = it.next();
            if (it.hasNext()) {
                byte[] right = it.next();
                byte[] combined = dot(left, right);
                combinedHashes.add(combined);
            } else {
                combinedHashes.add(left);
            }
        }

        return combinedHashes;
    }
}

```

1.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.

```

```
* SPDX-License-Identifier: MIT-0
*
* Permission is hereby granted, free of charge, to any person obtaining a
* copy of this
* software and associated documentation files (the "Software"), to deal in
* the Software
* without restriction, including without limitation the rights to use, copy,
* modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
* and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
* A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
* COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
* ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
* THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import com.amazonaws.util.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.qldb.Proof;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
import java.util.concurrent.ThreadLocalRandom;

/**
 * Encapsulates the logic to verify the integrity of revisions or blocks in a
 * QLDB ledger.
 *
 * The main entry point is {@link #verify(byte[], byte[], String)}.
```

```

*
* This code expects that you have AWS credentials setup per:
* http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
*/
public final class Verifier {
    public static final Logger log = LoggerFactory.getLogger(Verifier.class);
    private static final int HASH_LENGTH = 32;
    private static final int UPPER_BOUND = 8;

    /**
     * Compares two hashes by their signed byte values in little-endian order.
     */
    private static Comparator<byte[]> hashComparator = (h1, h2) -> {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }
        for (int i = h1.length - 1; i >= 0; i--) {
            int byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                return byteEqual;
            }
        }

        return 0;
    };

    private Verifier() { }

    /**
     * Verify the integrity of a document with respect to a QLDB ledger digest.
     *
     * The verification algorithm includes the following steps:
     *
     * 1. buildCandidateDigest(Proof, byte[]) build the candidate digest from the internal hashes
     *    in the Proof.
     * 2. Check that the candidateLedgerDigest is equal to the ledgerDigest.
     *
     * @param documentHash
     *        The hash of the document to be verified.

```

```

    * @param digest
    *           The QLDB ledger digest. This digest should have been
retrieved using
    *           {@link com.amazonaws.services.qldb.AmazonQLDB#getDigest}
    * @param proofBlob
    *           The ion encoded bytes representing the {@link Proof}
associated with the supplied
    *           {@code digestTipAddress} and {@code address} retrieved
using
    *           {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
    * @return {@code true} if the record is verified or {@code false} if it
is not verified.
    */
    public static boolean verify(
        final byte[] documentHash,
        final byte[] digest,
        final String proofBlob
    ) {
        Proof proof = Proof.fromBlob(proofBlob);

        byte[] candidateDigest = buildCandidateDigest(proof, documentHash);

        return Arrays.equals(digest, candidateDigest);
    }

    /**
     * Build the candidate digest representing the entire ledger from the
internal hashes of the {@link Proof}.
     *
     * @param proof
     *           A Java representation of {@link Proof}
     *           returned from {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
     * @param leafHash
     *           Leaf hash to build the candidate digest with.
     * @return a byte array of the candidate digest.
     */
    private static byte[] buildCandidateDigest(final Proof proof, final byte[]
leafHash) {
        return calculateRootHashFromInternalHashes(proof.getInternalHashes(),
leafHash);
    }

```

```
/**
 * Get a new instance of {@link MessageDigest} using the SHA-256
algorithm.
 *
 * @return an instance of {@link MessageDigest}.
 * @throws IllegalStateException if the algorithm is not available on the
current JVM.
 */
static MessageDigest newMessageDigest() {
    try {
        return MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        log.error("Failed to create SHA-256 MessageDigest", e);
        throw new IllegalStateException("SHA-256 message digest is
unavailable", e);
    }
}

/**
 * Takes two hashes, sorts them, concatenates them, and then returns the
 * hash of the concatenated array.
 *
 * @param h1
 *           Byte array containing one of the hashes to compare.
 * @param h2
 *           Byte array containing one of the hashes to compare.
 * @return the concatenated array of hashes.
 */
public static byte[] dot(final byte[] h1, final byte[] h2) {
    if (h1.length == 0) {
        return h2;
    }
    if (h2.length == 0) {
        return h1;
    }
    byte[] concatenated = new byte[h1.length + h2.length];
    if (hashComparator.compare(h1, h2) < 0) {
        System.arraycopy(h1, 0, concatenated, 0, h1.length);
        System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
    } else {
        System.arraycopy(h2, 0, concatenated, 0, h2.length);
        System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
    }
    MessageDigest messageDigest = newMessageDigest();
```

```

        messageDigest.update(concatenated);

        return messageDigest.digest();
    }

    /**
     * Starting with the provided {@code leafHash} combined with the provided
     * {@code internalHashes}
     * pairwise until only the root hash remains.
     *
     * @param internalHashes
     *         Internal hashes of Merkle tree.
     * @param leafHash
     *         Leaf hashes of Merkle tree.
     * @return the root hash.
     */
    private static byte[] calculateRootHashFromInternalHashes(final
    List<byte[]> internalHashes, final byte[] leafHash) {
        return internalHashes.stream().reduce(leafHash, Verifier::dot);
    }

    /**
     * Flip a single random bit in the given byte array. This method is used
     * to demonstrate
     * QLDB's verification features.
     *
     * @param original
     *         The original byte array.
     * @return the altered byte array with a single random bit changed.
     */
    public static byte[] flipRandomBit(final byte[] original) {
        if (original.length == 0) {
            throw new IllegalArgumentException("Array cannot be empty!");
        }
        int alteredPosition =
    ThreadLocalRandom.current().nextInt(original.length);
        int b = ThreadLocalRandom.current().nextInt(UPPER_BOUND);
        byte[] altered = new byte[original.length];
        System.arraycopy(original, 0, altered, 0, original.length);
        altered[alteredPosition] = (byte) (altered[alteredPosition] ^ (1 <<
    b));
        return altered;
    }

```

```

    public static String toBase64(byte[] arr) {
        return new String(Base64.encode(arr), StandardCharsets.UTF_8);
    }

    /**
     * Convert a {@link ByteBuffer} into byte array.
     *
     * @param buffer
     *           The {@link ByteBuffer} to convert.
     * @return the converted byte array.
     */
    public static byte[] convertByteBufferToByteArray(final ByteBuffer buffer)
    {
        byte[] arr = new byte[buffer.remaining()];
        buffer.get(arr);
        return arr;
    }

    /**
     * Calculates the root hash from a list of hashes that represent the base
     of a Merkle tree.
     *
     * @param hashes
     *           The list of byte arrays representing hashes making up base
     of a Merkle tree.
     * @return a byte array that is the root hash of the given list of hashes.
     */
    public static byte[] calculateMerkleTreeRootHash(List<byte[]> hashes) {
        if (hashes.isEmpty()) {
            return new byte[0];
        }

        List<byte[]> remaining = combineLeafHashes(hashes);
        while (remaining.size() > 1) {
            remaining = combineLeafHashes(remaining);
        }
        return remaining.get(0);
    }

    private static List<byte[]> combineLeafHashes(List<byte[]> hashes) {
        List<byte[]> combinedHashes = new ArrayList<>();
        Iterator<byte[]> it = hashes.stream().iterator();

        while (it.hasNext()) {

```



```

        byte[] left = it.next();
        if (it.hasNext()) {
            byte[] right = it.next();
            byte[] combined = dot(left, right);
            combinedHashes.add(combined);
        } else {
            combinedHashes.add(left);
        }
    }

    return combinedHashes;
}
}

```

2. Use two .java files (GetDigest.java and GetRevision.java) to perform the following steps:

- Request a new digest from the vehicle-registration ledger.
- Request a proof for each revision of a document from the VehicleRegistration table.
- Verify the revisions using the returned digest and proof by recalculating the digest.

The GetDigest.java program contains the following code.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT

```

```
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestRequest;
import com.amazonaws.services.qldb.model.GetDigestResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

/**
 * This is an example for retrieving the digest of a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class GetDigest {
    public static final Logger log = LoggerFactory.getLogger(GetDigest.class);
    public static AmazonQLDB client = CreateLedger.getClient();

    private GetDigest() { }

    /**
     * Calls {@link #getDigest(String)} for a ledger.
     *
     * @param args
     *         Arbitrary command-line arguments.
     * @throws Exception if failed to get a ledger digest.
     */
    public static void main(final String... args) throws Exception {
        try {

            getDigest(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to get a ledger digest!", e);
            throw e;
        }
    }
}
```

```

    }
}

/**
 * Get the digest for the specified ledger.
 *
 * @param ledgerName
 *           The ledger to get digest from.
 * @return {@link GetDigestResult}.
 */
public static GetDigestResult getDigest(final String ledgerName) {
    log.info("Let's get the current digest of the ledger named {}.\"",
ledgerName);
    GetDigestRequest request = new GetDigestRequest()
        .withName(ledgerName);
    GetDigestResult result = client.getDigest(request);
    log.info("Success. LedgerDigest: {}.\"",
QldbStringUtils.toUnredactedString(result));
    return result;
}
}

```

Note

Use the `getDigest` method to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request.

The `GetRevision.java` program contains the following code.

2.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software

```

```
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
```

```
package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.qldb.BlockAddress;
```

```

import software.amazon.qldb.tutorial.qldb.QldbRevision;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class GetRevision {
    public static final Logger log = LoggerFactory.getLogger(GetRevision.class);
    public static AmazonQLDB client = CreateLedger.getClient();
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();

    private GetRevision() { }

    public static void main(String... args) throws Exception {

        final String vin = SampleData.REGISTRATIONS.get(0).getVin();

        verifyRegistration(ConnectToLedger.getDriver(), Constants.LEDGER_NAME,
vin);
    }

    /**
     * Verify each version of the registration for the given VIN.
     *
     * @param driver
     *             A QLDB driver.
     * @param ledgerName
     *             The ledger to get digest from.
     * @param vin
     *             VIN to query the revision history of a specific registration
with.
     * @throws Exception if failed to verify digests.
     * @throws AssertionError if document revision verification failed.
     */
    public static void verifyRegistration(final QldbDriver driver, final String
ledgerName, final String vin)
        throws Exception {
        log.info(String.format("Let's verify the registration with VIN=%s, in
ledger=%s.", vin, ledgerName));
    }
}

```

```

        try {
            log.info("First, let's get a digest.");
            GetDigestResult digestResult = GetDigest.getDigest(ledgerName);

            ValueHolder digestTipAddress = digestResult.getDigestTipAddress();
            byte[] digestBytes =
                Verifier.convertByteBufferToByteArray(digestResult.getDigest());

            log.info("Got a ledger digest. Digest end address={}, digest={}.",
                QldbStringUtils.toUnredactedString(digestTipAddress),
                Verifier.toBase64(digestBytes));

            log.info(String.format("Next, let's query the registration with VIN=
%s. "
                                + "Then we can verify each version of the registration.",
                                vin));
            List<IonStruct> documentsWithMetadataList = new ArrayList<>();
            driver.execute(txn -> {
                documentsWithMetadataList.addAll(queryRegistrationsByVin(txn,
                                vin));
            });
            log.info("Registrations queried successfully!");

            log.info(String.format("Found %s revisions of the registration with
VIN=%s.",
                                documentsWithMetadataList.size(), vin));

            for (IonStruct ionStruct : documentsWithMetadataList) {

                QldbRevision document = QldbRevision.fromIon(ionStruct);
                log.info(String.format("Let's verify the document: %s",
                                document));

                log.info("Let's get a proof for the document.");
                GetRevisionResult proofResult = getRevision(
                    ledgerName,
                    document.getMetadata().getId(),
                    digestTipAddress,
                    document.getBlockAddress()
                );

                final IonValue proof =
                    Constants.MAPPER.writeValueAsIonValue(proofResult.getProof());
            }
        }
    }

```

```
        final IonReader reader =
IonReaderBuilder.standard().build(proof);
        reader.next();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        IonWriter writer = SYSTEM.newBinaryWriter(baos);
        writer.writeValue(reader);
        writer.close();
        baos.flush();
        baos.close();
        byte[] byteProof = baos.toByteArray();

        log.info(String.format("Got back a proof: %s",
Verifier.toBase64(byteProof)));

        boolean verified = Verifier.verify(
            document.getHash(),
            digestBytes,
            proofResult.getProof().getIonText()
        );

        if (!verified) {
            throw new AssertionError("Document revision is not
verified!");
        } else {
            log.info("Success! The document is verified");
        }

        byte[] alteredDigest = Verifier.flipRandomBit(digestBytes);
        log.info(String.format("Flipping one bit in the digest and
assert that the document is NOT verified. "
            + "The altered digest is: %s",
Verifier.toBase64(alteredDigest)));
        verified = Verifier.verify(
            document.getHash(),
            alteredDigest,
            proofResult.getProof().getIonText()
        );

        if (verified) {
            throw new AssertionError("Expected document to not be
verified against altered digest.");
        } else {
            log.info("Success! As expected flipping a bit in the digest
causes verification to fail.");
        }
    }
}
```

```

    }

    byte[] alteredDocumentHash =
Verifier.flipRandomBit(document.getHash());
    log.info(String.format("Flipping one bit in the document's hash
and assert that it is NOT verified. "
        + "The altered document hash is: %s.",
Verifier.toBase64(alteredDocumentHash)));
    verified = Verifier.verify(
        alteredDocumentHash,
        digestBytes,
        proofResult.getProof().getIonText()
    );

    if (verified) {
        throw new AssertionError("Expected altered document hash to
not be verified against digest.");
    } else {
        log.info("Success! As expected flipping a bit in the
document hash causes verification to fail.");
    }
}

} catch (Exception e) {
    log.error("Failed to verify digests.", e);
    throw e;
}

log.info(String.format("Finished verifying the registration with VIN=%s
in ledger=%s.", vin, ledgerName));
}

/**
 * Get the revision of a particular document specified by the given document
ID and block address.
 *
 * @param ledgerName
 *      Name of the ledger containing the document.
 * @param documentId
 *      Unique ID for the document to be verified, contained in the
committed view of the document.
 * @param digestTipAddress
 *      The latest block location covered by the digest.
 * @param blockAddress

```



```

        *           The location of the block to request.
        * @return the requested revision.
        */
    public static GetRevisionResult getRevision(final String ledgerName, final
String documentId,
                                           final ValueHolder
digestTipAddress, final BlockAddress blockAddress) {
        try {
            GetRevisionRequest request = new GetRevisionRequest()
                .WithName(ledgerName)
                .withDigestTipAddress(digestTipAddress)
                .withBlockAddress(new
ValueHolder().withIonText(Constants.MAPPER.writeValueAsIonValue(blockAddress)
                    .toString()))
                .withDocumentId(documentId);
            return client.getRevision(request);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Query the registration history for the given VIN.
     *
     * @param txn
     *           The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *           The unique VIN to query.
     * @return a list of {@link IonStruct} representing the registration
    history.
     * @throws IllegalStateException if failed to convert parameters into {@link
    IonValue}
     */
    public static List<IonStruct> queryRegistrationsByVin(final
TransactionExecutor txn, final String vin) {
        log.info(String.format("Let's query the 'VehicleRegistration' table for
VIN: %s...", vin));
        log.info("Let's query the 'VehicleRegistration' table for VIN: {}...",
vin);
        final String query = String.format("SELECT * FROM _ql_committed_%s WHERE
data.VIN = ?",
            Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        try {

```

```

        final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
        final Result result = txn.execute(query, parameters);
        List<IonStruct> list = ScanTable.toIonStructs(result);
        log.info(String.format("Found %d document(s)!", list.size()));
        return list;
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}
}

```

1.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;

```

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.qldb.BlockAddress;
import software.amazon.qldb.tutorial.qldb.QldbRevision;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public final class GetRevision {
    public static final Logger log = LoggerFactory.getLogger(GetRevision.class);
    public static AmazonQLDB client = CreateLedger.getClient();

    private GetRevision() { }

    public static void main(String... args) throws Exception {

        final String vin = SampleData.REGISTRATIONS.get(0).getVin();

        try (QldbSession qldbSession = ConnectToLedger.createQldbSession()) {
            verifyRegistration(qldbSession, Constants.LEDGER_NAME, vin);
        }
    }
}
```

```

    }
}

/**
 * Verify each version of the registration for the given VIN.
 *
 * @param qlldbSession
 *           A QLDB session.
 * @param ledgerName
 *           The ledger to get digest from.
 * @param vin
 *           VIN to query the revision history of a specific registration
with.
 * @throws Exception if failed to verify digests.
 * @throws AssertionError if document revision verification failed.
 */
public static void verifyRegistration(final QldbSession qlldbSession, final
String ledgerName, final String vin)
    throws Exception {
    log.info(String.format("Let's verify the registration with VIN=%s, in
ledger=%s.", vin, ledgerName));

    try {
        log.info("First, let's get a digest.");
        GetDigestResult digestResult = GetDigest.getDigest(ledgerName);

        ValueHolder digestTipAddress = digestResult.getDigestTipAddress();
        byte[] digestBytes =
Verifier.convertByteBufferToByteArray(digestResult.getDigest());

        log.info("Got a ledger digest. Digest end address={}, digest={}.",
QldbStringUtils.toUnredactedString(digestTipAddress),
Verifier.toBase64(digestBytes));

        log.info(String.format("Next, let's query the registration with VIN=
%s. "
                                + "Then we can verify each version of the registration.",
vin));
        List<IonStruct> documentsWithMetadataList = new ArrayList<>();
        qlldbSession.execute(txn -> {
            documentsWithMetadataList.addAll(queryRegistrationsByVin(txn,
vin));
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Registrations queried successfully!");
    }
}

```

```
        log.info(String.format("Found %s revisions of the registration with  
VIN=%s.",  
                                documentsWithMetadataList.size(), vin));  
  
        for (IonStruct ionStruct : documentsWithMetadataList) {  
  
            QldbRevision document = QldbRevision.fromIon(ionStruct);  
            log.info(String.format("Let's verify the document: %s",  
document));  
  
            log.info("Let's get a proof for the document.");  
            GetRevisionResult proofResult = getRevision(  
                ledgerName,  
                document.getMetadata().getId(),  
                digestTipAddress,  
                document.getBlockAddress()  
            );  
  
            final IonValue proof =  
Constants.MAPPER.writeValueAsIonValue(proofResult.getProof());  
            final IonReader reader =  
IonReaderBuilder.standard().build(proof);  
            reader.next();  
            ByteArrayOutputStream baos = new ByteArrayOutputStream();  
            IonWriter writer = Constants.SYSTEM.newBinaryWriter(baos);  
            writer.writeValue(reader);  
            writer.close();  
            baos.flush();  
            baos.close();  
            byte[] byteProof = baos.toByteArray();  
  
            log.info(String.format("Got back a proof: %s",  
Verifier.toBase64(byteProof)));  
  
            boolean verified = Verifier.verify(  
                document.getHash(),  
                digestBytes,  
                proofResult.getProof().getIonText()  
            );  
  
            if (!verified) {  
                throw new AssertionError("Document revision is not  
verified!");  
            }  
        }  
    }  
}
```

```
        } else {
            log.info("Success! The document is verified");
        }

        byte[] alteredDigest = Verifier.flipRandomBit(digestBytes);
        log.info(String.format("Flipping one bit in the digest and
assert that the document is NOT verified. "
            + "The altered digest is: %s",
Verifier.toBase64(alteredDigest)));
        verified = Verifier.verify(
            document.getHash(),
            alteredDigest,
            proofResult.getProof().getIonText()
        );

        if (verified) {
            throw new AssertionError("Expected document to not be
verified against altered digest.");
        } else {
            log.info("Success! As expected flipping a bit in the digest
causes verification to fail.");
        }

        byte[] alteredDocumentHash =
Verifier.flipRandomBit(document.getHash());
        log.info(String.format("Flipping one bit in the document's hash
and assert that it is NOT verified. "
            + "The altered document hash is: %s.",
Verifier.toBase64(alteredDocumentHash)));
        verified = Verifier.verify(
            alteredDocumentHash,
            digestBytes,
            proofResult.getProof().getIonText()
        );

        if (verified) {
            throw new AssertionError("Expected altered document hash to
not be verified against digest.");
        } else {
            log.info("Success! As expected flipping a bit in the
document hash causes verification to fail.");
        }
    }
}
```

```

        } catch (Exception e) {
            log.error("Failed to verify digests.", e);
            throw e;
        }

        log.info(String.format("Finished verifying the registration with VIN=%s
in ledger=%s.", vin, ledgerName));
    }

    /**
     * Get the revision of a particular document specified by the given document
     ID and block address.
     *
     * @param ledgerName
     *         Name of the ledger containing the document.
     * @param documentId
     *         Unique ID for the document to be verified, contained in the
     committed view of the document.
     * @param digestTipAddress
     *         The latest block location covered by the digest.
     * @param blockAddress
     *         The location of the block to request.
     * @return the requested revision.
     */
    public static GetRevisionResult getRevision(final String ledgerName, final
String documentId,
                                              final ValueHolder
digestTipAddress, final BlockAddress blockAddress) {
        try {
            GetRevisionRequest request = new GetRevisionRequest()
                .withName(ledgerName)
                .withDigestTipAddress(digestTipAddress)
                .withBlockAddress(new
ValueHolder().withIonText(Constants.MAPPER.writeValueAsIonValue(blockAddress)
                    .toString()))
                .withDocumentId(documentId);
            return client.getRevision(request);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Query the registration history for the given VIN.

```

```

    *
    * @param txn
    *           The {@link TransactionExecutor} for lambda execute.
    * @param vin
    *           The unique VIN to query.
    * @return a list of {@link IonStruct} representing the registration
    history.
    * @throws IllegalStateException if failed to convert parameters into {@link
    IonValue}
    */
    public static List<IonStruct> queryRegistrationsByVin(final
    TransactionExecutor txn, final String vin) {
        log.info(String.format("Let's query the 'VehicleRegistration' table for
    VIN: %s...", vin));
        log.info("Let's query the 'VehicleRegistration' table for VIN: {}",
    vin);
        final String query = String.format("SELECT * FROM _ql_committed_%s WHERE
    data.VIN = ?",
            Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        try {
            final List<IonValue> parameters =
    Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            List<IonStruct> list = ScanTable.toIonStructs(result);
            log.info(String.format("Found %d document(s)!", list.size()));
            return list;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
}

```

Note

After the `getRevision` method returns a proof for the specified document revision, this program uses a client-side API to verify that revision. For an overview of the algorithm used by this API, see [Using a proof to recalculate your digest](#).

3. Compile and run the `GetRevision.java` program to cryptographically verify the `VehicleRegistration` document with VIN `1N4AL11D75C109151`.

To export and validate journal data in the vehicle-registration ledger, proceed to [Step 8: Export and validate journal data in a ledger](#).

Step 8: Export and validate journal data in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, you can access the contents of the journal in your ledger for various purposes such as data retention, analytics, and auditing. For more information, see [Exporting journal data from Amazon QLDB](#).

In this step, you export [journal blocks](#) from the vehicle-registration ledger into an Amazon S3 bucket. Then, you use the exported data to validate the hash chain between journal blocks and the individual hash components within each block.

The AWS Identity and Access Management (IAM) principal entity that you use must have sufficient IAM permissions to create an Amazon S3 bucket in your AWS account. For information, see [Policies and Permissions in Amazon S3](#) in the *Amazon S3 User Guide*. You must also have permissions to create an IAM role with an attached permissions policy that allows QLDB to write objects into your Amazon S3 bucket. To learn more, see [Permissions required to access IAM resources](#) in the *IAM User Guide*.

To export and validate journal data

1. Review the following file (`JournalBlock.java`), which represents a journal block and its data contents. It includes a method named `verifyBlockHash()` that demonstrates how to calculate each individual component of a block hash.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
```

```

* software and associated documentation files (the "Software"), to deal in the
Software
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software, and
to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

```

```
package software.amazon.qldb.tutorial.qldb;
```

```

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.dataformat.ion.IonTimestampSerializers;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.Verifier;

```

```

import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

```

```
import static java.nio.ByteBuffer.wrap;
```

```

/**
 * Represents a JournalBlock that was recorded after executing a transaction
 * in the ledger.

```

```

*/
public final class JournalBlock {
    private static final Logger log = LoggerFactory.getLogger(JournalBlock.class);

    private BlockAddress blockAddress;
    private String transactionId;
    @JsonSerialize(using =
IonTimestampSerializers.IonTimestampJavaDateSerializer.class)
    private Date blockTimestamp;
    private byte[] blockHash;
    private byte[] entriesHash;
    private byte[] previousBlockHash;
    private byte[][] entriesHashList;
    private TransactionInfo transactionInfo;
    private RedactionInfo redactionInfo;
    private List<QldbRevision> revisions;

    @JsonCreator
    public JournalBlock(@JsonProperty("blockAddress") final BlockAddress
blockAddress,
                        @JsonProperty("transactionId") final String transactionId,
                        @JsonProperty("blockTimestamp") final Date blockTimestamp,
                        @JsonProperty("blockHash") final byte[] blockHash,
                        @JsonProperty("entriesHash") final byte[] entriesHash,
                        @JsonProperty("previousBlockHash") final byte[]
previousBlockHash,
                        @JsonProperty("entriesHashList") final byte[][]
entriesHashList,
                        @JsonProperty("transactionInfo") final TransactionInfo
transactionInfo,
                        @JsonProperty("redactionInfo") final RedactionInfo
redactionInfo,
                        @JsonProperty("revisions") final List<QldbRevision>
revisions) {
        this.blockAddress = blockAddress;
        this.transactionId = transactionId;
        this.blockTimestamp = blockTimestamp;
        this.blockHash = blockHash;
        this.entriesHash = entriesHash;
        this.previousBlockHash = previousBlockHash;
        this.entriesHashList = entriesHashList;
        this.transactionInfo = transactionInfo;
        this.redactionInfo = redactionInfo;
        this.revisions = revisions;
    }
}

```

```
}

public BlockAddress getBlockAddress() {
    return blockAddress;
}

public String getTransactionId() {
    return transactionId;
}

public Date getBlockTimestamp() {
    return blockTimestamp;
}

public byte[][] getEntriesHashList() {
    return entriesHashList;
}

public TransactionInfo getTransactionInfo() {
    return transactionInfo;
}

public RedactionInfo getRedactionInfo() {
    return redactionInfo;
}

public List<QldbRevision> getRevisions() {
    return revisions;
}

public byte[] getEntriesHash() {
    return entriesHash;
}

public byte[] getBlockHash() {
    return blockHash;
}

public byte[] getPreviousBlockHash() {
    return previousBlockHash;
}

@Override
public String toString() {
```

```
        return "JournalBlock{"
            + "blockAddress=" + blockAddress
            + ", transactionId='" + transactionId + '\''
            + ", blockTimestamp=" + blockTimestamp
            + ", blockHash=" + Arrays.toString(blockHash)
            + ", entriesHash=" + Arrays.toString(entriesHash)
            + ", previousBlockHash=" + Arrays.toString(previousBlockHash)
            + ", entriesHashList=" + Arrays.toString(entriesHashList)
            + ", transactionInfo=" + transactionInfo
            + ", redactionInfo=" + redactionInfo
            + ", revisions=" + revisions
            + '}';
    }

    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof JournalBlock)) {
            return false;
        }

        final JournalBlock that = (JournalBlock) o;

        if (!getBlockAddress().equals(that.getBlockAddress())) {
            return false;
        }
        if (!getTransactionId().equals(that.getTransactionId())) {
            return false;
        }
        if (!getBlockTimestamp().equals(that.getBlockTimestamp())) {
            return false;
        }
        if (!Arrays.equals(getBlockHash(), that.getBlockHash())) {
            return false;
        }
        if (!Arrays.equals(getEntriesHash(), that.getEntriesHash())) {
            return false;
        }
        if (!Arrays.equals(getPreviousBlockHash(), that.getPreviousBlockHash())) {
            return false;
        }
        if (!Arrays.deepEquals(getEntriesHashList(), that.getEntriesHashList())) {
```

```

        return false;
    }
    if (!getTransactionInfo().equals(that.getTransactionInfo())) {
        return false;
    }
    if (getRedactionInfo() != null ? !
getRedactionInfo().equals(that.getRedactionInfo()) : that.getRedactionInfo() !=
null) {
        return false;
    }
    return getRevisions() != null ?
getRevisions().equals(that.getRevisions()) : that.getRevisions() == null;
}

@Override
public int hashCode() {
    int result = getBlockAddress().hashCode();
    result = 31 * result + getTransactionId().hashCode();
    result = 31 * result + getBlockTimestamp().hashCode();
    result = 31 * result + Arrays.hashCode(getBlockHash());
    result = 31 * result + Arrays.hashCode(getEntriesHash());
    result = 31 * result + Arrays.hashCode(getPreviousBlockHash());
    result = 31 * result + Arrays.deepHashCode(getEntriesHashList());
    result = 31 * result + getTransactionInfo().hashCode();
    result = 31 * result + (getRedactionInfo() != null ?
getRedactionInfo().hashCode() : 0);
    result = 31 * result + (getRevisions() != null ?
getRevisions().hashCode() : 0);
    return result;
}

/**
 * This method validates that the hashes of the components of a journal block
 * make up the block
 * hash that is provided with the block itself.
 *
 * The components that contribute to the hash of the journal block consist of
 * the following:
 *
 * - user transaction information (contained in [transactionInfo])
 * - user redaction information (contained in [redactionInfo])
 * - user revisions (contained in [revisions])
 * - hashes of internal-only system metadata (contained in [revisions] and in
 [entriesHashList])
 * - the previous block hash

```

```

    *
    * If any of the computed hashes of user information cannot be validated or any
    of the system
    * hashes do not result in the correct computed values, this method will throw
    an IllegalArgumentException.
    *
    * Internal-only system metadata is represented by its hash, and can be present
    in the form of certain
    * items in the [revisions] list that only contain a hash and no user data, as
    well as some hashes
    * in [entriesHashList].
    *
    * To validate that the hashes of the user data are valid components of the
    [blockHash], this method
    * performs the following steps:
    *
    * 1. Compute the hash of the [transactionInfo] and validate that it is
    included in the [entriesHashList].
    * 2. Compute the hash of the [redactionInfo], if present, and validate that it
    is included in the [entriesHashList].
    * 3. Validate the hash of each user revision was correctly computed and
    matches the hash published
    * with that revision.
    * 4. Compute the hash of the [revisions] by treating the revision hashes as
    the leaf nodes of a Merkle tree
    * and calculating the root hash of that tree. Then validate that hash is
    included in the [entriesHashList].
    * 5. Compute the hash of the [entriesHashList] by treating the hashes as the
    leaf nodes of a Merkle tree
    * and calculating the root hash of that tree. Then validate that hash matches
    [entriesHash].
    * 6. Finally, compute the block hash by computing the hash resulting from
    concatenating the [entriesHash]
    * and previous block hash, and validate that the result matches the
    [blockHash] provided by QLDB with the block.
    *
    * This method is called by ValidateQldbHashChain::verify for each journal
    block to validate its
    * contents before verifying that the hash chain between consecutive blocks is
    correct.
    */
    public void verifyBlockHash() {
        Set<ByteBuffer> entriesHashSet = new HashSet<>();

```

```
Arrays.stream(entriesHashList).forEach(hash ->
entriesHashSet.add(wrap(hash).asReadOnlyBuffer()));

byte[] computedTransactionInfoHash = computeTransactionInfoHash();
if (!
entriesHashSet.contains(wrap(computedTransactionInfoHash).asReadOnlyBuffer())) {
    throw new IllegalArgumentException(
        "Block transactionInfo hash is not contained in the QLDB block
entries hash list.");
}

if (redactionInfo != null) {
    byte[] computedRedactionInfoHash = computeRedactionInfoHash();
    if (!
entriesHashSet.contains(wrap(computedRedactionInfoHash).asReadOnlyBuffer())) {
        throw new IllegalArgumentException(
            "Block redactionInfo hash is not contained in the QLDB
block entries hash list.");
    }
}

if (revisions != null) {
    revisions.forEach(QldbRevision::verifyRevisionHash);
    byte[] computedRevisionsHash = computeRevisionsHash();
    if (!
entriesHashSet.contains(wrap(computedRevisionsHash).asReadOnlyBuffer())) {
        throw new IllegalArgumentException(
            "Block revisions list hash is not contained in the QLDB
block entries hash list.");
    }
}

byte[] computedEntriesHash = computeEntriesHash();
if (!Arrays.equals(computedEntriesHash, entriesHash)) {
    throw new IllegalArgumentException("Computed entries hash does not
match entries hash provided in the block.");
}

byte[] computedBlockHash = Verifier.dot(computedEntriesHash,
previousBlockHash);
if (!Arrays.equals(computedBlockHash, blockHash)) {
    throw new IllegalArgumentException("Computed block hash does not match
block hash provided in the block.");
}
```



```

    }

    private byte[] computeTransactionInfoHash() {
        try {
            return
QldbIonUtils.hashIonValue(Constants.MAPPER.writeValueAsIonValue(transactionInfo));
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not compute transactionInfo
hash to verify block hash.", e);
        }
    }

    private byte[] computeRedactionInfoHash() {
        try {
            return
QldbIonUtils.hashIonValue(Constants.MAPPER.writeValueAsIonValue(redactionInfo));
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not compute redactionInfo
hash to verify block hash.", e);
        }
    }

    private byte[] computeRevisionsHash() {
        return
Verifier.calculateMerkleTreeRootHash(revisions.stream().map(QldbRevision::getHash).collect
    )
    }

    private byte[] computeEntriesHash() {
        return
Verifier.calculateMerkleTreeRootHash(Arrays.asList(entriesHashList));
    }
}

```

2. Compile and run the following program (ValidateQldbHashChain.java) to do the following steps:
 1. Export journal blocks from the vehicle-registration ledger into an Amazon S3 bucket named **qldb-tutorial-journal-export-111122223333** (replace with your AWS account number).
 2. Validate the individual hash components within each block by calling `verifyBlockHash()`.
 3. Validate the hash chain between journal blocks.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.model.ExportJournalToS3Result;
import com.amazonaws.services.qldb.model.S3EncryptionConfiguration;
import com.amazonaws.services.qldb.model.S3ObjectEncryptionType;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import java.time.Instant;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.GetCallerIdentityRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.tutorial.qldb.JournalBlock;
```

```

/**
 * Validate the hash chain of a QLDB ledger by stepping through its S3 export.
 *
 * This code accepts an exportId as an argument, if exportId is passed the code
 * will use that or request QLDB to generate a new export to perform QLDB hash
 * chain validation.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
 * credentials.html
 */
public final class ValidateQldbHashChain {
    public static final Logger log =
        LoggerFactory.getLogger(ValidateQldbHashChain.class);
    private static final int TIME_SKEW = 20;

    private ValidateQldbHashChain() { }

    /**
     * Export journal contents to a S3 bucket.
     *
     * @return the ExportId of the journal export.
     * @throws InterruptedException if the thread is interrupted while waiting for
     * export to complete.
     */
    private static String createExport() throws InterruptedException {
        String accountId = AWSSecurityTokenServiceClientBuilder.defaultClient()
            .getCallerIdentity(new GetCallerIdentityRequest()).getAccount();
        String bucketName = Constants.JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX + "-" +
            accountId;
        String prefix = Constants.LEDGER_NAME + "-" +
            Instant.now().getEpochSecond() + "/";

        S3EncryptionConfiguration encryptionConfiguration = new
            S3EncryptionConfiguration()
                .withObjectEncryptionType(S3ObjectEncryptionType.SSE_S3);
        ExportJournalToS3Result exportJournalToS3Result =

        ExportJournal.createJournalExportAndAwaitCompletion(Constants.LEDGER_NAME,
            bucketName, prefix, null, encryptionConfiguration,
            ExportJournal.DEFAULT_EXPORT_TIMEOUT_MS);

        return exportJournalToS3Result.getExportId();
    }
}

```

```

    }

    /**
     * Validates that the chain hash on the {@link JournalBlock} is valid.
     *
     * @param journalBlocks
     *         {@link JournalBlock} containing hashes to validate.
     * @throws IllegalStateException if previous block hash does not match.
     */
    public static void verify(final List<JournalBlock> journalBlocks) {
        if (journalBlocks.size() == 0) {
            return;
        }

        journalBlocks.stream().reduce(null, (previousJournalBlock, journalBlock) ->
    {
        journalBlock.verifyBlockHash();
        if (previousJournalBlock == null) { return journalBlock; }
        if (!Arrays.equals(previousJournalBlock.getBlockHash(),
journalBlock.getPreviousBlockHash())) {
            throw new IllegalStateException("Previous block hash doesn't
match.");
        }
        byte[] blockHash = Verifier.dot(journalBlock.getEntriesHash(),
previousJournalBlock.getBlockHash());
        if (!Arrays.equals(blockHash, journalBlock.getBlockHash())) {
            throw new IllegalStateException("Block hash doesn't match
entriesHash dot previousBlockHash, the chain is "
+ "broken.");
        }
        return journalBlock;
    });
    }

    public static void main(final String... args) throws InterruptedException {
        try {
            String exportId;
            if (args.length == 1) {
                exportId = args[0];
                log.info("Validating QLDB hash chain for exportId: " + exportId);
            } else {
                log.info("Requesting QLDB to create an export.");
                exportId = createExport();
            }
        }
    }

```

```

        List<JournalBlock> journalBlocks =

        JournalS3ExportReader.readExport(DescribeJournalExport.describeExport(Constants.LEDGER_NAME,
            exportId), AmazonS3ClientBuilder.defaultClient());
        verify(journalBlocks);
    } catch (Exception e) {
        log.error("Unable to perform hash chain verification.", e);
        throw e;
    }
}
}

```

If you no longer need to use the vehicle-registration ledger, proceed to [Step 9 \(optional\): Clean up resources](#).

Step 9 (optional): Clean up resources

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can continue using the vehicle-registration ledger. However, if you no longer need it, you should delete it.

To delete the ledger

1. Compile and run the following program (DeleteLedger.java) to delete your vehicle-registration ledger and all of its contents.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software

```

```

* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software, and
to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

```

```
package software.amazon.qldb.tutorial;
```

```

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.DeleteLedgerRequest;
import com.amazonaws.services.qldb.model.DeleteLedgerResult;
import com.amazonaws.services.qldb.model.ResourceNotFoundException;
import com.amazonaws.services.qldb.model.UpdateLedgerRequest;
import com.amazonaws.services.qldb.model.UpdateLedgerResult;

```

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

```

```

/**
 * Delete a ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */

```

```

public final class DeleteLedger {
    public static final Logger log = LoggerFactory.getLogger(DeleteLedger.class);
    public static final Long LEDGER_DELETION_POLL_PERIOD_MS = 20_000L;
    public static AmazonQLDB client = CreateLedger.getClient();

    private DeleteLedger() { }

    public static void main(String... args) throws Exception {

```

```

        try {
            setDeletionProtection(Constants.LEDGER_NAME, false);

            delete(Constants.LEDGER_NAME);

            waitForDeleted(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to delete the ledger.", e);
            throw e;
        }
    }

    /**
     * Send a request to the QLDB database to delete the specified ledger.
     *
     * @param ledgerName
     *         Name of the ledger to be deleted.
     * @return DeleteLedgerResult.
     */
    public static DeleteLedgerResult delete(final String ledgerName) {
        log.info("Attempting to delete the ledger with name: {}", ledgerName);
        DeleteLedgerRequest request = new
DeleteLedgerRequest().withName(ledgerName);
        DeleteLedgerResult result = client.deleteLedger(request);
        log.info("Success.");
        return result;
    }

    /**
     * Wait for the ledger to be deleted.
     *
     * @param ledgerName
     *         Name of the ledger being deleted.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static void waitForDeleted(final String ledgerName) throws
InterruptedException {
        log.info("Waiting for the ledger to be deleted...");
        while (true) {
            try {
                DescribeLedger.describe(ledgerName);
                log.info("The ledger is still being deleted. Please wait...");
                Thread.sleep(LEDGER_DELETION_POLL_PERIOD_MS);
            }

```

```
        } catch (ResourceNotFoundException ex) {
            log.info("Success. The ledger is deleted.");
            break;
        }
    }
}

public static UpdateLedgerResult setDeletionProtection(String ledgerName,
boolean deletionProtection) {
    log.info("Let's set deletionProtection to {} for the ledger with name {}",
deletionProtection, ledgerName);
    UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withDeletionProtection(deletionProtection);

    UpdateLedgerResult result = client.updateLedger(request);
    log.info("Success. Ledger updated: {}", result);
    return result;
}
}
```

Note

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API.

2. If you exported journal data in the [previous step](#) and no longer need it, use the Amazon S3 console to delete your S3 bucket.

Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Amazon QLDB Node.js tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this implementation of the tutorial sample application, you use the Amazon QLDB driver with the AWS SDK for JavaScript in Node.js to create a QLDB ledger and populate it with sample data.

As you work through this tutorial, you can refer to the [AWS SDK for JavaScript API Reference](#) for management API operations. For transactional data operations, you can refer to the [QLDB Driver for Node.js API Reference](#).

 Note

Where applicable, some tutorial steps have different commands or code examples for each supported major version of the QLDB driver for Node.js.

Topics

- [Installing the Amazon QLDB Node.js sample application](#)
- [Step 1: Create a new ledger](#)
- [Step 2: Test connectivity to the ledger](#)
- [Step 3: Create tables, indexes, and sample data](#)
- [Step 4: Query the tables in a ledger](#)
- [Step 5: Modify documents in a ledger](#)
- [Step 6: View the revision history for a document](#)
- [Step 7: Verify a document in a ledger](#)
- [Step 8 \(optional\): Clean up resources](#)

Installing the Amazon QLDB Node.js sample application

 Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes how to install and run the provided Amazon QLDB sample application for the step-by-step Node.js tutorial. The use case for this sample application is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations.

The DMV sample application for Node.js is open source in the GitHub repository [aws-samples/amazon-qldb-dmv-sample-nodejs](https://github.com/aws-samples/amazon-qldb-dmv-sample-nodejs).

Prerequisites

Before you get started, make sure that you complete the QLDB driver for Node.js [Prerequisites](#). This includes installing Node.js and doing the following:

1. Sign up for AWS.
2. Create a user with the appropriate QLDB permissions.
3. Grant programmatic access for development.

To complete all of the steps in this tutorial, you need full administrative access to your ledger resource through the QLDB API.

Installation

To install the sample application

1. Enter the following command to clone the sample application from GitHub.

2.x

```
git clone https://github.com/aws-samples/amazon-qldb-dmv-sample-nodejs.git
```

1.x

```
git clone -b v1.0.0 https://github.com/aws-samples/amazon-qldb-dmv-sample-nodejs.git
```

The sample application packages the complete source code from this tutorial and its dependencies, including the Node.js driver and the [AWS SDK for JavaScript in Node.js](#). This application is written in TypeScript.

2. Switch to the directory where the `amazon-qldb-dmv-sample-nodejs` package is cloned.

```
cd amazon-qldb-dmv-sample-nodejs
```

3. Do a clean install of the dependencies.

```
npm ci
```

4. Transpile the package.

```
npm run build
```

The transpiled JavaScript files are written in the `./dist` directory.

5. Proceed to [Step 1: Create a new ledger](#) to start the tutorial and create a ledger.

Step 1: Create a new ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`.

To create a new ledger

1. Review the following file (`Constants.ts`), which contains constant values that are used by all of the other programs in this tutorial.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 */
```

```

* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

/**
 * Constant values used throughout this tutorial.
 */
export const LEDGER_NAME = "vehicle-registration";
export const LEDGER_NAME_WITH_TAGS = "tags";

export const DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
export const PERSON_TABLE_NAME = "Person";
export const VEHICLE_REGISTRATION_TABLE_NAME = "VehicleRegistration";
export const VEHICLE_TABLE_NAME = "Vehicle";

export const GOV_ID_INDEX_NAME = "GovId";
export const LICENSE_NUMBER_INDEX_NAME = "LicenseNumber";
export const LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
export const PERSON_ID_INDEX_NAME = "PersonId";
export const VIN_INDEX_NAME = "VIN";

export const RETRY_LIMIT = 4;

export const JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-tutorial-journal-export";
export const USER_TABLES = "information_schema.user_tables";

```

2. Use the following program (`CreateLedger.ts`) to create a ledger named `vehicle-registration`.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
this

```

```

* software and associated documentation files (the "Software"), to deal in the
Software
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software, and
to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

```

```

import { QLDB } from "aws-sdk";
import {
    CreateLedgerRequest,
    CreateLedgerResponse,
    DescribeLedgerRequest,
    DescribeLedgerResponse
} from "aws-sdk/clients/qlldb";

import { LEDGER_NAME } from "./qlldb/Constants";
import { error, log } from "./qlldb/LogUtil";
import { sleep } from "./qlldb/Util";

const LEDGER_CREATION_POLL_PERIOD_MS = 10000;
const ACTIVE_STATE = "ACTIVE";

/**
 * Create a new ledger with the specified name.
 * @param ledgerName Name of the ledger to be created.
 * @param qlldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a CreateLedgerResponse.
 */
export async function createLedger(ledgerName: string, qlldbClient: QLDB):
    Promise<CreateLedgerResponse> {
    log(`Creating a ledger named: ${ledgerName}...`);
    const request: CreateLedgerRequest = {

```

```

        Name: ledgerName,
        PermissionsMode: "ALLOW_ALL"
    }
    const result: CreateLedgerResponse = await
qlldbClient.createLedger(request).promise();
    log(`Success. Ledger state: ${result.State}`);
    return result;
}

/**
 * Wait for the newly created ledger to become active.
 * @param ledgerName Name of the ledger to be checked on.
 * @param qlldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a DescribeLedgerResponse.
 */
export async function waitForActive(ledgerName: string, qlldbClient: QLDB):
Promise<DescribeLedgerResponse> {
    log(`Waiting for ledger ${ledgerName} to become active...`);
    const request: DescribeLedgerRequest = {
        Name: ledgerName
    }
    while (true) {
        const result: DescribeLedgerResponse = await
qlldbClient.describeLedger(request).promise();
        if (result.State === ACTIVE_STATE) {
            log("Success. Ledger is active and ready to be used.");
            return result;
        }
        log("The ledger is still creating. Please wait...");
        await sleep(LEDGER_CREATION_POLL_PERIOD_MS);
    }
}

/**
 * Create a ledger and wait for it to be active.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qlldbClient: QLDB = new QLDB();
        await createLedger(LEDGER_NAME, qlldbClient);
        await waitForActive(LEDGER_NAME, qlldbClient);
    } catch (e) {
        error(`Unable to create the ledger: ${e}`);
    }
}

```

```
    }  
  }  
  
  if (require.main === module) {  
    main();  
  }  
}
```

Note

- In the `createLedger` call, you must specify a ledger name and a permissions mode. We recommend using the STANDARD permissions mode to maximize the security of your ledger data.
- When you create a ledger, *deletion protection* is enabled by default. This is a feature in QLDB that prevents ledgers from being deleted by any user. You have the option of disabling deletion protection on ledger creation using the QLDB API or the AWS Command Line Interface (AWS CLI).
- Optionally, you can also specify tags to attach to your ledger.

3. To run the transpiled program, enter the following command.

```
node dist/CreateLedger.js
```

To verify your connection to the new ledger, proceed to [Step 2: Test connectivity to the ledger](#).

Step 2: Test connectivity to the ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this step, you verify that you can connect to the `vehicle-registration` ledger in Amazon QLDB using the transactional data API endpoint.

To test connectivity to the ledger

1. Use the following program (`ConnectToLedger.ts`) to create a data session connection to the vehicle-registration ledger.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, RetryConfig } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/qldb-session";

import { LEDGER_NAME } from "../qldb/Constants";
import { error, log } from "../qldb/LogUtil";

const qldbDriver: QldbDriver = createQldbDriver();

/**
 * Create a driver for creating sessions.
```



```

    * @param ledgerName The name of the ledger to create the driver on.
    * @param serviceConfigurationOptions The configurations for the AWS SDK client
    that the driver uses.
    * @returns The driver for creating sessions.
    */
export function createQldbDriver(
    ledgerName: string = LEDGER_NAME,
    serviceConfigurationOptions: ClientConfiguration = {}
): QldbDriver {
    const retryLimit = 4;
    const maxConcurrentTransactions = 10;
    //Use driver's default backoff function (and hence, no second parameter
    provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
    const qldbDriver: QldbDriver = new QldbDriver(ledgerName,
    serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);
    return qldbDriver;
}

export function getQldbDriver(): QldbDriver {
    return qldbDriver;
}

/**
 * Connect to a session for a given ledger using default settings.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        log("Listing table names...");
        const tableNames: string[] = await qldbDriver.getTableNames();
        tableNames.forEach((tableName: string): void => {
            log(tableName);
        });
    } catch (e) {
        error(`Unable to create session: ${e}`);
    }
}

if (require.main === module) {
    main();
}

```

1.x

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 * THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/qldb-session";

import { LEDGER_NAME } from "../qldb/Constants";
import { error, log } from "../qldb/LogUtil";

const qldbDriver: QldbDriver = createQldbDriver();

/**
 * Create a driver for creating sessions.
 * @param ledgerName The name of the ledger to create the driver on.
 * @param serviceConfigurationOptions The configurations for the AWS SDK client
 * that the driver uses.
 * @returns The driver for creating sessions.
 */

```

```

export function createQldbDriver(
    ledgerName: string = LEDGER_NAME,
    serviceConfigurationOptions: ClientConfiguration = {}
): QldbDriver {
    const qldbDriver: QldbDriver = new QldbDriver(ledgerName,
    serviceConfigurationOptions);
    return qldbDriver;
}

export function getQldbDriver(): QldbDriver {
    return qldbDriver;
}

/**
 * Connect to a session for a given ledger using default settings.
 * @returns Promise which fulfills with void.
 */
var main = async function(): Promise<void> {
    try {
        log("Listing table names...");
        const tableNames: string[] = await qldbDriver.getTableNames();
        tableNames.forEach((tableName: string): void => {
            log(tableName);
        });
    } catch (e) {
        error(`Unable to create session: ${e}`);
    }
}

if (require.main === module) {
    main();
}

```

Note

To run data transactions on your ledger, you must create a QLDB driver object to connect to a specified ledger. This is a different client object than the `qldbClient` object that you used in the [previous step](#) to create the ledger. That previous client is only used to run the management API operations listed in the [Amazon QLDB API reference](#).

2. To run the transpiled program, enter the following command.

```
node dist/ConnectToLedger.js
```

To create tables in the vehicle-registration ledger, proceed to [Step 3: Create tables, indexes, and sample data](#).

Step 3: Create tables, indexes, and sample data

 **Important**

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

When your Amazon QLDB ledger is active and accepts connections, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the vehicle-registration ledger:

- VehicleRegistration
- Vehicle
- Person
- DriversLicense

You also create the following indexes.

Table name	Field
VehicleRegistration	VIN
VehicleRegistration	LicensePlateNumber
Vehicle	VIN

Table name	Field
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

When inserting sample data, you first insert documents into the `Person` table. Then, you use the system-assigned `id` from each `Person` document to populate the corresponding fields in the appropriate `VehicleRegistration` and `DriversLicense` documents.

Tip

As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table). For more information about views in QLDB, see [Core concepts](#). To learn more about metadata, see [Querying document metadata](#).

To create tables and indexes

1. Use the following program (`CreateTable.ts`) to create the previously mentioned tables.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 */
```

```

* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

import { QldbDriver, Result, TransactionExecutor } from "amazon-qlldb-driver-
nodejs";

import { getQldbDriver } from "./ConnectToLedger";
import {
    DRIVERS_LICENSE_TABLE_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME
} from "./qlldb/Constants";
import { error, log } from "./qlldb/LogUtil";

/**
 * Create multiple tables in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to create.
 * @returns Promise which fulfills with the number of changes to the database.
 */
export async function createTable(txn: TransactionExecutor, tableName: string):
    Promise<number> {
    const statement: string = `CREATE TABLE ${tableName}`;
    return await txn.execute(statement).then((result: Result) => {
        log(`Successfully created table ${tableName}.`);
        return result.getResultList().length;
    });
}

/**
 * Create tables in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {

```

```

const qlldbDriver: QldbDriver = getQldbDriver();
await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
    Promise.all([
        createTable(txn, VEHICLE_REGISTRATION_TABLE_NAME),
        createTable(txn, VEHICLE_TABLE_NAME),
        createTable(txn, PERSON_TABLE_NAME),
        createTable(txn, DRIVERS_LICENSE_TABLE_NAME)
    ]);
});
} catch (e) {
    error(`Unable to create tables: ${e}`);
}
}

if (require.main === module) {
    main();
}

```

Note

This program demonstrates how to use the `executeLambda` function in a QLDB driver instance. In this example, you run multiple `CREATE TABLE` PartiQL statements with a single lambda expression.

This `execute` function implicitly starts a transaction, runs all of the statements in the lambda, and then auto-commits the transaction.

2. To run the transpiled program, enter the following command.

```
node dist/CreateTable.js
```

3. Use the following program (`CreateIndex.ts`) to create indexes on the tables, as previously described.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software

```

```

* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software, and
to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";

import { getQldbDriver } from "./ConnectToLedger";
import {
    DRIVERS_LICENSE_TABLE_NAME,
    GOV_ID_INDEX_NAME,
    LICENSE_NUMBER_INDEX_NAME,
    LICENSE_PLATE_NUMBER_INDEX_NAME,
    PERSON_ID_INDEX_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME,
    VIN_INDEX_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Create an index for a particular table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to add indexes for.
 * @param indexAttribute Index to create on a single attribute.
 * @returns Promise which fulfills with the number of changes to the database.
 */
export async function createIndex(
    txn: TransactionExecutor,
    tableName: string,
    indexAttribute: string

```



```

): Promise<number> {
    const statement: string = `CREATE INDEX on ${tableName} (${indexAttribute})`;
    return await txn.execute(statement).then((result) => {
        log(`Successfully created index ${indexAttribute} on table ${tableName}.`);
        return result.getResultList().length;
    });
}

/**
 * Create indexes on tables in a particular ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qlldbDriver: QldbDriver = getQldbDriver();
        await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            Promise.all([
                createIndex(txn, PERSON_TABLE_NAME, GOV_ID_INDEX_NAME),
                createIndex(txn, VEHICLE_TABLE_NAME, VIN_INDEX_NAME),
                createIndex(txn, VEHICLE_REGISTRATION_TABLE_NAME, VIN_INDEX_NAME),
                createIndex(txn, VEHICLE_REGISTRATION_TABLE_NAME,
                    LICENSE_PLATE_NUMBER_INDEX_NAME),
                createIndex(txn, DRIVERS_LICENSE_TABLE_NAME, PERSON_ID_INDEX_NAME),
                createIndex(txn, DRIVERS_LICENSE_TABLE_NAME,
                    LICENSE_NUMBER_INDEX_NAME)
            ]);
        });
    } catch (e) {
        error(`Unable to create indexes: ${e}`);
    }
}

if (require.main === module) {
    main();
}

```

4. To run the transpiled program, enter the following command.

```
node dist/CreateIndex.js
```

To load data into the tables

1. Review the following .ts files.

1. `SampleData.ts` – Contains the sample data that you insert into the vehicle-registration tables.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Decimal } from "ion-js";

const EMPTY_SECONDARY_OWNERS: object[] = [];
export const DRIVERS_LICENSE = [
  {
    PersonId: "",
    LicenseNumber: "LEWISR261LL",
    LicenseType: "Learner",
    ValidFromDate: new Date("2016-12-20"),
    ValidToDate: new Date("2020-11-15")
  },
];
```

```
{
  PersonId: "",
  LicenseNumber : "LOGANB486CG",
  LicenseType: "Probationary",
  ValidFromDate : new Date("2016-04-06"),
  ValidToDate : new Date("2020-11-15")
},
{
  PersonId: "",
  LicenseNumber : "744 849 301",
  LicenseType: "Full",
  ValidFromDate : new Date("2017-12-06"),
  ValidToDate : new Date("2022-10-15")
},
{
  PersonId: "",
  LicenseNumber : "P626-168-229-765",
  LicenseType: "Learner",
  ValidFromDate : new Date("2017-08-16"),
  ValidToDate : new Date("2021-11-15")
},
{
  PersonId: "",
  LicenseNumber : "S152-780-97-415-0",
  LicenseType: "Probationary",
  ValidFromDate : new Date("2015-08-15"),
  ValidToDate : new Date("2021-08-21")
}
];
export const PERSON = [
  {
    FirstName : "Raul",
    LastName : "Lewis",
    DOB : new Date("1963-08-19"),
    Address : "1719 University Street, Seattle, WA, 98109",
    GovId : "LEWISR261LL",
    GovIdType : "Driver License"
  },
  {
    FirstName : "Brent",
    LastName : "Logan",
    DOB : new Date("1967-07-03"),
    Address : "43 Stockert Hollow Road, Everett, WA, 98203",
    GovId : "LOGANB486CG",
```

```
        GovIdType : "Driver License"
    },
    {
        FirstName : "Alexis",
        LastName : "Pena",
        DOB : new Date("1974-02-10"),
        Address : "4058 Melrose Street, Spokane Valley, WA, 99206",
        GovId : "744 849 301",
        GovIdType : "SSN"
    },
    {
        FirstName : "Melvin",
        LastName : "Parker",
        DOB : new Date("1976-05-22"),
        Address : "4362 Ryder Avenue, Seattle, WA, 98101",
        GovId : "P626-168-229-765",
        GovIdType : "Passport"
    },
    {
        FirstName : "Salvatore",
        LastName : "Spencer",
        DOB : new Date("1997-11-15"),
        Address : "4450 Honeysuckle Lane, Seattle, WA, 98101",
        GovId : "S152-780-97-415-0",
        GovIdType : "Passport"
    }
];
export const VEHICLE = [
    {
        VIN : "1N4AL11D75C109151",
        Type : "Sedan",
        Year : 2011,
        Make : "Audi",
        Model : "A5",
        Color : "Silver"
    },
    {
        VIN : "KM8SRDHF6EU074761",
        Type : "Sedan",
        Year : 2015,
        Make : "Tesla",
        Model : "Model S",
        Color : "Blue"
    },
]
```

```
{
  VIN : "3HGGK5G53FM761765",
  Type : "Motorcycle",
  Year : 2011,
  Make : "Ducati",
  Model : "Monster 1200",
  Color : "Yellow"
},
{
  VIN : "1HVBBAANXWH544237",
  Type : "Semi",
  Year : 2009,
  Make : "Ford",
  Model : "F 150",
  Color : "Black"
},
{
  VIN : "1C4RJFAG0FC625797",
  Type : "Sedan",
  Year : 2019,
  Make : "Mercedes",
  Model : "CLK 350",
  Color : "White"
}
];
export const VEHICLE_REGISTRATION = [
  {
    VIN : "1N4AL11D75C109151",
    LicensePlateNumber : "LEWISR261LL",
    State : "WA",
    City : "Seattle",
    ValidFromDate : new Date("2017-08-21"),
    ValidToDate : new Date("2020-05-11"),
    PendingPenaltyTicketAmount : new Decimal(9025, -2),
    Owners : {
      PrimaryOwner : { PersonId : "" },
      SecondaryOwners : EMPTY_SECONDARY_OWNERS
    }
  },
  {
    VIN : "KM8SRDHF6EU074761",
    LicensePlateNumber : "CA762X",
    State : "WA",
    City : "Kent",
```

```

        PendingPenaltyTicketAmount : new Decimal(13075, -2),
        ValidFromDate : new Date("2017-09-14"),
        ValidToDate : new Date("2020-06-25"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "3HGGK5G53FM761765",
        LicensePlateNumber : "CD820Z",
        State : "WA",
        City : "Everett",
        PendingPenaltyTicketAmount : new Decimal(44230, -2),
        ValidFromDate : new Date("2011-03-17"),
        ValidToDate : new Date("2021-03-24"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "1HVBBAANXWH544237",
        LicensePlateNumber : "LS477D",
        State : "WA",
        City : "Tacoma",
        PendingPenaltyTicketAmount : new Decimal(4220, -2),
        ValidFromDate : new Date("2011-10-26"),
        ValidToDate : new Date("2023-09-25"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "1C4RJFAG0FC625797",
        LicensePlateNumber : "TH393F",
        State : "WA",
        City : "Olympia",
        PendingPenaltyTicketAmount : new Decimal(3045, -2),
        ValidFromDate : new Date("2013-09-02"),
        ValidToDate : new Date("2024-03-19"),
        Owners : {
            PrimaryOwner : { PersonId : "" },

```

```

        SecondaryOwners : EMPTY_SECONDARY_OWNERS
    }
}
];

```

2. `Util.ts` – A utility module that imports from the `ion-js` package to provide helper functions that convert, parse, and print [Amazon Ion](#) data.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Result, TransactionExecutor } from "amazon-qlldb-driver-nodejs";
import { GetBlockResponse, GetDigestResponse, ValueHolder } from "aws-sdk/clients/qlldb";
import {
    Decimal,
    decodeUtf8,
    dom,
    IonTypes,
    makePrettyWriter,
    makeReader,

```

```

    Reader,
    Timestamp,
    toBase64,
    Writer
} from "ion-js";

import { error } from "../LogUtil";

/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given BlockResponse.
 * @param blockResponse The BlockResponse to convert to string.
 * @returns The string representation of the supplied BlockResponse.
 */
export function blockResponseToString(blockResponse: GetBlockResponse): string {
    let stringBuilder: string = "";
    if (blockResponse.Block.IonText) {
        stringBuilder = stringBuilder + "Block: " + blockResponse.Block.IonText +
        ", ";
    }
    if (blockResponse.Proof.IonText) {
        stringBuilder = stringBuilder + "Proof: " + blockResponse.Proof.IonText;
    }
    stringBuilder = "{" + stringBuilder + "}";
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given GetDigestResponse.
 * @param digestResponse The GetDigestResponse to convert to string.
 * @returns The string representation of the supplied GetDigestResponse.
 */
export function digestResponseToString(digestResponse: GetDigestResponse): string
{
    let stringBuilder: string = "";
    if (digestResponse.Digest) {
        stringBuilder += "Digest: " + JSON.stringify(toBase64(<Uint8Array>
digestResponse.Digest)) + ", ";
    }
}

```



```

    }
    if (digestResponse.DigestTipAddress.IonText) {
        stringBuilder += "DigestTipAddress: " +
digestResponse.DigestTipAddress.IonText;
    }
    stringBuilder = "{" + stringBuilder + "}";
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * Get the document IDs from the given table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName The table name to query.
 * @param field A field to query.
 * @param value The key of the given field.
 * @returns Promise which fulfills with the document ID as a string.
 */
export async function getDocumentId(
    txn: TransactionExecutor,
    tableName: string,
    field: string,
    value: string
): Promise<string> {
    const query: string = `SELECT id FROM ${tableName} AS t BY id WHERE t.
${field} = ?`;
    let documentId: string = undefined;
    await txn.execute(query, value).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to retrieve document ID using ${value}.`);
        }
        documentId = resultList[0].get("id").stringValue();

    }).catch((err: any) => {
        error(`Error getting documentId: ${err}`);
    });
    return documentId;
}

/**
 * Sleep for the specified amount of time.

```

```

    * @param ms The amount of time to sleep in milliseconds.
    * @returns Promise which fulfills with void.
    */
export function sleep(ms: number): Promise<void> {
    return new Promise(resolve => setTimeout(resolve, ms));
}

/**
 * Find the value of a given path in an Ion value. The path should contain a blob
 * value.
 * @param value The Ion value that contains the journal block attributes.
 * @param path The path to a certain attribute.
 * @returns Uint8Array value of the blob, or null if the attribute cannot be
 * found in the Ion value
 *
 *          or is not of type Blob
 */
export function getBlobValue(value: dom.Value, path: string): Uint8Array | null {
    const attribute: dom.Value = value.get(path);
    if (attribute !== null && attribute.getType() === IonTypes.BLOB) {
        return attribute.uInt8ArrayValue();
    }
    return null;
}

/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given ValueHolder.
 * @param valueHolder The ValueHolder to convert to string.
 * @returns The string representation of the supplied ValueHolder.
 */
export function valueHolderToString(valueHolder: ValueHolder): string {
    const stringBuilder: string = `{ IonText: ${valueHolder.IonText} }`;
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * Converts a given value to Ion using the provided writer.
 * @param value The value to convert to Ion.
 * @param ionWriter The Writer to pass the value into.
 * @throws Error: If the given value cannot be converted to Ion.
 */

```

```
export function writeValueAsIon(value: any, ionWriter: Writer): void {
  switch (typeof value) {
    case "string":
      ionWriter.writeString(value);
      break;
    case "boolean":
      ionWriter.writeBoolean(value);
      break;
    case "number":
      ionWriter.writeInt(value);
      break;
    case "object":
      if (Array.isArray(value)) {
        // Object is an array.
        ionWriter.stepIn(IonTypes.LIST);

        for (const element of value) {
          writeValueAsIon(element, ionWriter);
        }

        ionWriter.stepOut();
      } else if (value instanceof Date) {
        // Object is a Date.
        ionWriter.writeTimestamp(Timestamp.parse(value.toISOString()));
      } else if (value instanceof Decimal) {
        // Object is a Decimal.
        ionWriter.writeDecimal(value);
      } else if (value === null) {
        ionWriter.writeNull(IonTypes.NULL);
      } else {
        // Object is a struct.
        ionWriter.stepIn(IonTypes.STRUCT);

        for (const key of Object.keys(value)) {
          ionWriter.writeFieldName(key);
          writeValueAsIon(value[key], ionWriter);
        }
        ionWriter.stepOut();
      }
      break;
    default:
      throw new Error(`Cannot convert to Ion for type: ${typeof
value}).`);
  }
}
```

}

Note

The `getDocumentId` function runs a query that returns system-assigned document IDs from a table. To learn more, see [Using the BY clause to query document ID](#).

2. Use the following program (`InsertDocument.ts`) to insert the sample data into your tables.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "../ConnectToLedger";
import { DRIVERS_LICENSE, PERSON, VEHICLE, VEHICLE_REGISTRATION } from "../model/
SampleData";
import {
```

```

    DRIVERS_LICENSE_TABLE_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Insert the given list of documents into a table in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to insert documents into.
 * @param documents List of documents to insert.
 * @returns Promise which fulfills with a {@linkcode Result} object.
 */
export async function insertDocument(
    txn: TransactionExecutor,
    tableName: string,
    documents: object[]
): Promise<Result> {
    const statement: string = `INSERT INTO ${tableName} ?`;
    const result: Result = await txn.execute(statement, documents);
    return result;
}

/**
 * Handle the insertion of documents and updating PersonIds all in a single
 * transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @returns Promise which fulfills with void.
 */
async function updateAndInsertDocuments(txn: TransactionExecutor): Promise<void> {
    log("Inserting multiple documents into the 'Person' table...");
    const documentIds: Result = await insertDocument(txn, PERSON_TABLE_NAME,
PERSON);

    const listOfDocumentIds: dom.Value[] = documentIds.getResultList();
    log("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
'VehicleRegistration'...");
    updatePersonId(listOfDocumentIds);

    log("Inserting multiple documents into the remaining tables...");
    await Promise.all([
        insertDocument(txn, DRIVERS_LICENSE_TABLE_NAME, DRIVERS_LICENSE),
        insertDocument(txn, VEHICLE_REGISTRATION_TABLE_NAME, VEHICLE_REGISTRATION),
    ])
}

```

```

        insertDocument(txn, VEHICLE_TABLE_NAME, VEHICLE)
    });
}

/**
 * Update the PersonId value for DriversLicense records and the PrimaryOwner value
 * for VehicleRegistration records.
 * @param documentIds List of document IDs.
 */
export function updatePersonId(documentIds: dom.Value[]): void {
    documentIds.forEach((value: dom.Value, i: number) => {
        const documentId: string = value.get("documentId").stringValue();
        DRIVERS_LICENSE[i].PersonId = documentId;
        VEHICLE_REGISTRATION[i].Owners.PrimaryOwner.PersonId = documentId;
    });
}

/**
 * Insert documents into a table in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qlldbDriver: QldbDriver = getQldbDriver();
        await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await updateAndInsertDocuments(txn);
        });
    } catch (e) {
        error(`Unable to insert documents: ${e}`);
    }
}

if (require.main === module) {
    main();
}

```

Note

- This program demonstrates how to call the execute function with parameterized values. You can pass data parameters in addition to the PartiQL statement that you want to run. Use a question mark (?) as a variable placeholder in your statement string.

- If an INSERT statement succeeds, it returns the `id` of each inserted document.

3. To run the transpiled program, enter the following command.

```
node dist/InsertDocument.js
```

Next, you can use SELECT statements to read data from the tables in the vehicle-registration ledger. Proceed to [Step 4: Query the tables in a ledger](#).

Step 4: Query the tables in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses [PartiQL](#) as its query language and [Amazon Ion](#) as its document-oriented data model.

PartiQL is an open-source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open-source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use SELECT statements to read data from the tables in the vehicle-registration ledger.

Warning

When you run a query in QLDB without an indexed lookup, it invokes a full table scan. PartiQL supports such queries because it's SQL compatible. However, *don't* run table scans for production use cases in QLDB. Table scans can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator on an indexed field or a document ID; for example, WHERE

`indexedField = 123` or `WHERE indexedField IN (456, 789)`. For more information, see [Optimizing query performance](#).

To query the tables

1. Use the following program (`FindVehicles.ts`) to query all vehicles registered under a person in your ledger.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "../ConnectToLedger";
import { PERSON } from "../model/SampleData";
import { PERSON_TABLE_NAME } from "../qldb/Constants";
import { error, log } from "../qldb/LogUtil";
import { getDocumentId } from "../qldb/Util";
```



```

import { prettyPrintResultList } from "./ScanTable";

/**
 * Query 'Vehicle' and 'VehicleRegistration' tables using a unique document ID in
 * one transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param govId The owner's government ID.
 * @returns Promise which fulfills with void.
 */
async function findVehiclesForOwner(txn: TransactionExecutor, govId: string):
  Promise<void> {
    const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME, "GovId",
govId);
    const query: string = "SELECT Vehicle FROM Vehicle INNER JOIN
VehicleRegistration AS r " +
                        "ON Vehicle.VIN = r.VIN WHERE
r.Owners.PrimaryOwner.PersonId = ?";

    await txn.execute(query, documentId).then((result: Result) => {
      const resultList: dom.Value[] = result.getResultList();
      log(`List of vehicles for owner with GovId: ${govId}`);
      prettyPrintResultList(resultList);
    });
}

/**
 * Find all vehicles registered under a person.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
  try {
    const qlldbDriver: QldbDriver = getQldbDriver();
    await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
      await findVehiclesForOwner(txn, PERSON[0].GovId);
    });
  } catch (e) {
    error(`Error getting vehicles for owner: ${e}`);
  }
}

if (require.main === module) {
  main();
}

```

Note

First, this program queries the `Person` table for the document with `GovId` `LEWISR261LL` to get its `id` metadata field.

Then, it uses this document `id` as a foreign key to query the `VehicleRegistration` table by `PrimaryOwner.PersonId`. It also joins `VehicleRegistration` with the `Vehicle` table on the `VIN` field.

2. To run the transpiled program, enter the following command.

```
node dist/FindVehicles.js
```

To learn about modifying documents in the tables in the `vehicle-registration` ledger, see [Step 5: Modify documents in a ledger](#).

Step 5: Modify documents in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Now that you have data to work with, you can start making changes to documents in the `vehicle-registration` ledger in Amazon QLDB. In this step, the following code examples demonstrate how to run data manipulation language (DML) statements. These statements update the primary owner of one vehicle and add a secondary owner to another vehicle.

To modify documents

1. Use the following program (`TransferVehicleOwnership.ts`) to update the primary owner of the vehicle with VIN `1N4AL11D75C109151` in your ledger.

```
/*  
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
 * SPDX-License-Identifier: MIT-0
```

```

*
* Permission is hereby granted, free of charge, to any person obtaining a copy of
this
* software and associated documentation files (the "Software"), to deal in the
Software
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software, and
to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

import { QldbDriver, Result, TransactionExecutor } from "amazon-qlldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "../ConnectToLedger";
import { PERSON, VEHICLE } from "../model/SampleData";
import { PERSON_TABLE_NAME } from "../qlldb/Constants";
import { error, log } from "../qlldb/LogUtil";
import { getDocumentId } from "../qlldb/Util";

/**
 * Query a driver's information using the given ID.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param documentId The unique ID of a document in the Person table.
 * @returns Promise which fulfills with an Ion value containing the person.
 */
export async function findPersonFromDocumentId(txn: TransactionExecutor,
documentId: string): Promise<dom.Value> {
    const query: string = "SELECT p.* FROM Person AS p BY pid WHERE pid = ?";

    let personId: dom.Value;
    await txn.execute(query, documentId).then((result: Result) => {

```

```

        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to find person with ID: ${documentId}.`);
        }
        personId = resultList[0];
    });
    return personId;
}

/**
 * Find the primary owner for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN to find primary owner for.
 * @returns Promise which fulfills with an Ion value containing the primary owner.
 */
export async function findPrimaryOwnerForVehicle(txn: TransactionExecutor, vin:
string): Promise<dom.Value> {
    log(`Finding primary owner for vehicle with VIN: ${vin}`);
    const query: string = "SELECT Owners.PrimaryOwner.PersonId FROM
VehicleRegistration AS v WHERE v.VIN = ?";

    let documentId: string = undefined;
    await txn.execute(query, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to retrieve document ID using ${vin}.`);
        }
        const PersonIdValue: dom.Value = resultList[0].get("PersonId");
        if (PersonIdValue === null) {
            throw new Error(`Expected field name PersonId not found.`);
        }
        documentId = PersonIdValue.stringValue();
    });
    return findPersonFromDocumentId(txn, documentId);
}

/**
 * Update the primary owner for a vehicle using the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN for the vehicle to operate on.
 * @param documentId New PersonId for the primary owner.
 * @returns Promise which fulfills with void.
 */

```

```

async function updateVehicleRegistration(txn: TransactionExecutor, vin: string,
documentId: string): Promise<void> {
    const statement: string = "UPDATE VehicleRegistration AS r SET
r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?";

    log(`Updating the primary owner for vehicle with VIN: ${vin}...`);
    await txn.execute(statement, documentId, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error("Unable to transfer vehicle, could not find
registration.");
        }
        log(`Successfully transferred vehicle with VIN ${vin} to new owner.`);
    });
}

/**
 * Validate the current owner of the given vehicle and transfer its ownership to a
new owner in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN of the vehicle to transfer ownership of.
 * @param currentOwner The GovId of the current owner of the vehicle.
 * @param newOwner The GovId of the new owner of the vehicle.
 */
export async function validateAndUpdateRegistration(
    txn: TransactionExecutor,
    vin: string,
    currentOwner: string,
    newOwner: string
): Promise<void> {
    const primaryOwner: dom.Value = await findPrimaryOwnerForVehicle(txn, vin);
    const govIdValue: dom.Value = primaryOwner.get("GovId");
    if (govIdValue !== null && govIdValue.stringValue() !== currentOwner) {
        log("Incorrect primary owner identified for vehicle, unable to transfer.");
    }
    else {
        const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME,
"GovId", newOwner);
        await updateVehicleRegistration(txn, vin, documentId);
        log("Successfully transferred vehicle ownership!");
    }
}

/**

```

```

    * Find primary owner for a particular vehicle's VIN.
    * Transfer to another primary owner for a particular vehicle's VIN.
    * @returns Promise which fulfills with void.
    */
const main = async function(): Promise<void> {
    try {
        const qlldbDriver: QldbDriver = getQldbDriver();

        const vin: string = VEHICLE[0].VIN;
        const previousOwnerGovId: string = PERSON[0].GovId;
        const newPrimaryOwnerGovId: string = PERSON[1].GovId;

        await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await validateAndUpdateRegistration(txn, vin, previousOwnerGovId,
            newPrimaryOwnerGovId);
        });
    } catch (e) {
        error(`Unable to connect and run queries: ${e}`);
    }
}

if (require.main === module) {
    main();
}

```

2. To run the transpiled program, enter the following command.

```
node dist/TransferVehicleOwnership.js
```

3. Use the following program (AddSecondaryOwner.ts) to add a secondary owner to the vehicle with VIN KM8SRDHF6EU074761 in your ledger.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,

```

```

* merge, publish, distribute, sublicense, and/or sell copies of the Software, and
to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { PERSON, VEHICLE_REGISTRATION } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";
import { prettyPrintResultList } from "./ScanTable";

/**
 * Add a secondary owner into 'VehicleRegistration' table for a particular VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN of the vehicle to query.
 * @param secondaryOwnerId The secondary owner's person ID.
 * @returns Promise which fulfills with void.
 */
export async function addSecondaryOwner(
    txn: TransactionExecutor,
    vin: string,
    secondaryOwnerId: string
): Promise<void> {
    log(`Inserting secondary owner for vehicle with VIN: ${vin}`);
    const query: string =
        `FROM VehicleRegistration AS v WHERE v.VIN = ? INSERT INTO
v.Owners.SecondaryOwners VALUE ?`;

    const personToInsert = {PersonId: secondaryOwnerId};

```

```

        await txn.execute(query, vin, personToInsert).then(async (result: Result) => {
            const resultList: dom.Value[] = result.getResultList();
            log("VehicleRegistration Document IDs which had secondary owners added: ");
            prettyPrintResultList(resultList);
        });
    }

/**
 * Query for a document ID with a government ID.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param governmentId The government ID to query with.
 * @returns Promise which fulfills with the document ID as a string.
 */
export async function getDocumentIdByGovId(txn: TransactionExecutor, governmentId:
string): Promise<string> {
    const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME, "GovId",
governmentId);
    return documentId;
}

/**
 * Check whether a driver has already been registered for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN of the vehicle to query.
 * @param secondaryOwnerId The secondary owner's person ID.
 * @returns Promise which fulfills with a boolean.
 */
export async function isSecondaryOwnerForVehicle(
    txn: TransactionExecutor,
    vin: string,
    secondaryOwnerId: string
): Promise<boolean> {
    log(`Finding secondary owners for vehicle with VIN: ${vin}`);
    const query: string = "SELECT Owners.SecondaryOwners FROM VehicleRegistration
AS v WHERE v.VIN = ?";

    let doesExist: boolean = false;

    await txn.execute(query, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();

        resultList.forEach((value: dom.Value) => {
            const secondaryOwnersList: dom.Value[] =
value.get("SecondaryOwners").elements();

```



```

        secondaryOwnersList.forEach((secondaryOwner) => {
            const personId: dom.Value = secondaryOwner.get("PersonId");
            if (personId !== null && personId.stringValue() ===
secondaryOwnerId) {
                doesExist = true;
            }
        });
    });
});
return doesExist;
}

/**
 * Finds and adds secondary owners for a vehicle.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qlldbDriver: QldbDriver = getQldbDriver();
        const vin: string = VEHICLE_REGISTRATION[1].VIN;
        const govId: string = PERSON[0].GovId;

        await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            const documentId: string = await getDocumentIdByGovId(txn, govId);

            if (await isSecondaryOwnerForVehicle(txn, vin, documentId)) {
                log(`Person with ID ${documentId} has already been added as a
secondary owner of this vehicle.`);
            } else {
                await addSecondaryOwner(txn, vin, documentId);
            }
        });

        log("Secondary owners successfully updated.");
    } catch (e) {
        error(`Unable to add secondary owner: ${e}`);
    }
}

if (require.main === module) {
    main();
}

```

4. To run the transpiled program, enter the following command.

```
node dist/AddSecondaryOwner.js
```

To review these changes in the vehicle-registration ledger, see [Step 6: View the revision history for a document](#).

Step 6: View the revision history for a document

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After modifying the registration data for a vehicle in the [previous step](#), you can query the history of all its registered owners and any other updated fields. In this step, you query the revision history of a document in the VehicleRegistration table in your vehicle-registration ledger.

To view the revision history

1. Use the following program (QueryHistory.ts) to query the revision history of the VehicleRegistration document with VIN 1N4AL11D75C109151.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 */
```

```

* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

```

```
import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
```

```
import { dom } from "ion-js";
```

```
import { getQldbDriver } from "./ConnectToLedger";
import { VEHICLE_REGISTRATION } from "./model/SampleData";
import { VEHICLE_REGISTRATION_TABLE_NAME } from "./qldb/Constants";
import { prettyPrintResultList } from "./ScanTable";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";
```

```
/**
```

```

* Find previous primary owners for the given VIN in a single transaction.
* @param txn The {@linkcode TransactionExecutor} for lambda execute.
* @param vin The VIN to find previous primary owners for.
* @returns Promise which fulfills with void.
*/

```

```
async function previousPrimaryOwners(txn: TransactionExecutor, vin: string):
```

```
Promise<void> {
```

```
    const documentId: string = await getDocumentId(txn,
VEHICLE_REGISTRATION_TABLE_NAME, "VIN", vin);
```

```
    const todaysDate: Date = new Date();
```

```
    // set todaysDate back one minute to ensure end time is in the past
    // by the time the request reaches our backend
```

```
    todaysDate.setMinutes(todaysDate.getMinutes() - 1);
```

```
    const threeMonthsAgo: Date = new Date(todaysDate);
```

```
    threeMonthsAgo.setMonth(todaysDate.getMonth() - 3);
```

```
    const query: string =
```

```
        `SELECT data.Owners.PrimaryOwner, metadata.version FROM history ` +
        `(${VEHICLE_REGISTRATION_TABLE_NAME}, \`${threeMonthsAgo.toISOString()}\`,
\`${todaysDate.toISOString()}\`) ` +
        `AS h WHERE h.metadata.id = ?`;

```

```

    await txn.execute(query, documentId).then((result: Result) => {
        log(`Querying the 'VehicleRegistration' table's history using VIN:
${vin}.`);
        const resultList: dom.Value[] = result.getResultList();
        prettyPrintResultList(resultList);
    });
}

/**
 * Query a table's history for a particular set of documents.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qlldbDriver: QldbDriver = getQldbDriver();
        const vin: string = VEHICLE_REGISTRATION[0].VIN;
        await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await previousPrimaryOwners(txn, vin);
        });
    } catch (e) {
        error(`Unable to query history to find previous owners: ${e}`);
    }
}

if (require.main === module) {
    main();
}

```

Note

- You can view the revision history of a document by querying the built-in [History function](#) in the following syntax.

```

SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
[ WHERE h.metadata.id = 'id' ]

```

- The *start-time* and *end-time* are both optional. They're Amazon Ion literal values that can be denoted with backticks (``...``). For more information, see [Querying Ion with PartiQL in Amazon QLDB](#).

- As a best practice, qualify a history query with both a date range (*start-time* and *end-time*) and a document ID (`metadata.id`). QLDB processes SELECT queries in transactions, which are subject to a [transaction timeout limit](#).

QLDB history is indexed by document ID, and you can't create additional history indexes at this time. History queries that include a start time and end time gain the benefit of date range qualification.

2. To run the transpiled program, enter the following command.

```
node dist/QueryHistory.js
```

To cryptographically verify a document revision in the vehicle-registration ledger, proceed to [Step 7: Verify a document in a ledger](#).

Step 7: Verify a document in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. To learn more about how verification and cryptographic hashing work in QLDB, see [Data verification in Amazon QLDB](#).

In this step, you verify a document revision in the `VehicleRegistration` table in your vehicle-registration ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To verify a document revision

1. Review the following `.ts` files, which contain the QLDB objects that are required for verification.

1. BlockAddress.ts

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { ValueHolder } from "aws-sdk/clients/qldb";
import { dom, IonTypes } from "ion-js";

export class BlockAddress {
  _strandId: string;
  _sequenceNo: number;

  constructor(strandId: string, sequenceNo: number) {
    this._strandId = strandId;
    this._sequenceNo = sequenceNo;
  }
}

/**
 * Convert a block address from an Ion value into a ValueHolder.
```

```

    * Shape of the ValueHolder must be: {'IonText': "{strandId: <"strandId">,
sequenceNo: <sequenceNo>}"}
    * @param value The Ion value that contains the block address values to convert.
    * @returns The ValueHolder that contains the strandId and sequenceNo.
    */
export function blockAddressToValueHolder(value: dom.Value): ValueHolder {
    const blockAddressValue : dom.Value = getBlockAddressValue(value);
    const strandId: string = getStrandId(blockAddressValue);
    const sequenceNo: number = getSequenceNo(blockAddressValue);
    const valueHolder: string = `{strandId: "${strandId}", sequenceNo:
${sequenceNo}}`;
    const blockAddress: ValueHolder = {IonText: valueHolder};
    return blockAddress;
}

/**
 * Helper method that to get the Metadata ID.
 * @param value The Ion value.
 * @returns The Metadata ID.
 */
export function getMetadataId(value: dom.Value): string {
    const metaDataId: dom.Value = value.get("id");
    if (metaDataId === null) {
        throw new Error(`Expected field name id, but not found.`);
    }
    return metaDataId.stringValue();
}

/**
 * Helper method to get the Sequence No.
 * @param value The Ion value.
 * @returns The Sequence No.
 */
export function getSequenceNo(value : dom.Value): number {
    const sequenceNo: dom.Value = value.get("sequenceNo");
    if (sequenceNo === null) {
        throw new Error(`Expected field name sequenceNo, but not found.`);
    }
    return sequenceNo.numberValue();
}

/**
 * Helper method to get the Strand ID.
 * @param value The Ion value.

```

```

    * @returns The Strand ID.
    */
export function getStrandId(value: dom.Value): string {
    const strandId: dom.Value = value.get("strandId");
    if (strandId === null) {
        throw new Error(`Expected field name strandId, but not found.`);
    }
    return strandId.stringValue();
}

export function getBlockAddressValue(value: dom.Value) : dom.Value {
    const type = value.getType();
    if (type !== IonTypes.STRUCT) {
        throw new Error(`Unexpected format: expected struct, but got IonType:
${type.name}`);
    }
    const blockAddress: dom.Value = value.get("blockAddress");
    if (blockAddress == null) {
        throw new Error(`Expected field name blockAddress, but not found.`);
    }
    return blockAddress;
}

```

2. Verifier.ts

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT

```



```

* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

import { Digest, ValueHolder } from "aws-sdk/clients/qldb";
import { createHash } from "crypto";
import { dom, toBase64 } from "ion-js";

import { getBlobValue } from "../Util";

const HASH_LENGTH: number = 32;
const UPPER_BOUND: number = 8;

/**
 * Build the candidate digest representing the entire ledger from the Proof
 hashes.
 * @param proof The Proof object.
 * @param leafHash The revision hash to pair with the first hash in the Proof
 hashes list.
 * @returns The calculated root hash.
 */
function buildCandidateDigest(proof: ValueHolder, leafHash: Uint8Array):
  Uint8Array {
    const parsedProof: Uint8Array[] = parseProof(proof);
    const rootHash: Uint8Array = calculateRootHashFromInternalHash(parsedProof,
leafHash);
    return rootHash;
}

/**
 * Combine the internal hashes and the leaf hash until only one root hash
 remains.
 * @param internalHashes An array of hash values.
 * @param leafHash The revision hash to pair with the first hash in the Proof
 hashes list.
 * @returns The root hash constructed by combining internal hashes.
 */
function calculateRootHashFromInternalHash(internalHashes: Uint8Array[],
leafHash: Uint8Array): Uint8Array {
    const rootHash: Uint8Array = internalHashes.reduce(joinHashesPairwise,
leafHash);
    return rootHash;
}

```

```

}

/**
 * Compare two hash values by converting each Uint8Array byte, which is unsigned
 * by default,
 * into a signed byte, assuming they are little endian.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 * the first pair of non-matching bytes.
 */
function compareHashValues(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_LENGTH || hash2.length !== HASH_LENGTH) {
        throw new Error("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of array to concatenate, in the order provided.
 * @returns The concatenated array.
 */
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}

/**

```

```

    * Flip a single random bit in the given hash value.
    * This method is intended to be used for purpose of demonstrating the QLDB
    verification features only.
    * @param original The hash value to alter.
    * @returns The altered hash with a single random bit changed.
    */
export function flipRandomBit(original: any): Uint8Array {
    if (original.length === 0) {
        throw new Error("Array cannot be empty!");
    }
    const bytePos: number = Math.floor(Math.random() * original.length);
    const bitShift: number = Math.floor(Math.random() * UPPER_BOUND);
    const alteredHash: Uint8Array = original;

    alteredHash[bytePos] = alteredHash[bytePos] ^ (1 << bitShift);
    return alteredHash;
}

/**
    * Take two hash values, sort them, concatenate them, and generate a new hash
    value from the concatenated values.
    * @param h1 Byte array containing one of the hashes to compare.
    * @param h2 Byte array containing one of the hashes to compare.
    * @returns The concatenated array of hashes.
    */
export function joinHashesPairwise(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }
    let concat: Uint8Array;
    if (compareHashValues(h1, h2) < 0) {
        concat = concatenate(h1, h2);
    } else {
        concat = concatenate(h2, h1);
    }
    const hash = createHash('sha256');
    hash.update(concat);
    const newDigest: Uint8Array = hash.digest();
    return newDigest;
}

```

```

/**
 * Parse the Block object returned by QLDB and retrieve block hash.
 * @param valueHolder A structure containing an Ion string value.
 * @returns The block hash.
 */
export function parseBlock(valueHolder: ValueHolder): Uint8Array {
    const block: dom.Value = dom.load(valueHolder.IonText);
    const blockHash: Uint8Array = getBlobValue(block, "blockHash");
    return blockHash;
}

/**
 * Parse the Proof object returned by QLDB into an iterator.
 * The Proof object returned by QLDB is a dictionary like the following:
 * {'IonText': '[[{<hash>}],[{<hash>}]]'}
 * @param valueHolder A structure containing an Ion string value.
 * @returns A list of hash values.
 */
function parseProof(valueHolder: ValueHolder): Uint8Array[] {
    const proofs : dom.Value = dom.load(valueHolder.IonText);
    return proofs.elements().map(proof => proof.uInt8ArrayValue());
}

/**
 * Verify document revision against the provided digest.
 * @param documentHash The SHA-256 value representing the document revision to be
    verified.
 * @param digest The SHA-256 hash value representing the ledger digest.
 * @param proof The Proof object retrieved from GetRevision.getRevision.
 * @returns If the document revision verifies against the ledger digest.
 */
export function verifyDocument(documentHash: Uint8Array, digest: Digest, proof:
    ValueHolder): boolean {
    const candidateDigest = buildCandidateDigest(proof, documentHash);
    return (toBase64(<Uint8Array> digest) === toBase64(candidateDigest));
}

```

2. Use two .ts programs (GetDigest.ts and GetRevision.ts) to perform the following steps:
 - Request a new digest from the vehicle-registration ledger.
 - Request a proof for each revision of the document with VIN 1N4AL11D75C109151 from the VehicleRegistration table.

- Verify the revisions using the returned digest and proof by recalculating the digest.

The `GetDigest.ts` program contains the following code.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QLDB } from "aws-sdk";
import { GetDigestRequest, GetDigestResponse } from "aws-sdk/clients/qldb";

import { LEDGER_NAME } from "../qlldb/Constants";
import { error, log } from "../qlldb/LogUtil";
import { digestResponseToString } from "../qlldb/Util";

/**
 * Get the digest of a ledger's journal.
 * @param ledgerName Name of the ledger to operate on.
 * @param qlldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a GetDigestResponse.
 */
```

```

export async function getDigestResult(ledgerName: string, qlldbClient: QLDB):
  Promise<GetDigestResponse> {
  const request: GetDigestRequest = {
    Name: ledgerName
  };
  const result: GetDigestResponse = await
qlldbClient.getDigest(request).promise();
  return result;
}

/**
 * This is an example for retrieving the digest of a particular ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
  try {
    const qlldbClient: QLDB = new QLDB();
    log(`Retrieving the current digest for ledger: ${LEDGER_NAME}.`);
    const digest: GetDigestResponse = await getDigestResult(LEDGER_NAME,
qlldbClient);
    log(`Success. Ledger digest: \n${digestResponseToString(digest)}.`);
  } catch (e) {
    error(`Unable to get a ledger digest: ${e}`);
  }
}

if (require.main === module) {
  main();
}

```

Note

Use the `getDigest` function to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request.

The `GetRevision.ts` program contains the following code.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0

```

```

*
* Permission is hereby granted, free of charge, to any person obtaining a copy of
this
* software and associated documentation files (the "Software"), to deal in the
Software
* without restriction, including without limitation the rights to use, copy,
modify,
* merge, publish, distribute, sublicense, and/or sell copies of the Software, and
to
* permit persons to whom the Software is furnished to do so.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
* INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT
* HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
* OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { QLDB } from "aws-sdk";
import { Digest, GetDigestResponse, GetRevisionRequest, GetRevisionResponse,
  ValueHolder } from "aws-sdk/clients/qldb";
import { dom, toBase64 } from "ion-js";

import { getQldbDriver } from "../ConnectToLedger";
import { getDigestResult } from '../GetDigest';
import { VEHICLE_REGISTRATION } from "../model/SampleData"
import { blockAddressToValueHolder, getMetadataId } from '../qldb/BlockAddress';
import { LEDGER_NAME } from '../qldb/Constants';
import { error, log } from "../qldb/LogUtil";
import { getBlobValue, valueHolderToString } from "../qldb/Util";
import { flipRandomBit, verifyDocument } from "../qldb/Verifier";

/**
 * Get the revision data object for a specified document ID and block address.
 * Also returns a proof of the specified revision for verification.
 * @param ledgerName Name of the ledger containing the document to query.
 * @param documentId Unique ID for the document to be verified, contained in the
committed view of the document.
 * @param blockAddress The location of the block to request.

```

```

    * @param digestTipAddress The latest block location covered by the digest.
    * @param qlldbClient The QLDB control plane client to use.
    * @returns Promise which fulfills with a GetRevisionResponse.
    */
    async function getRevision(
        ledgerName: string,
        documentId: string,
        blockAddress: ValueHolder,
        digestTipAddress: ValueHolder,
        qlldbClient: QLDB
    ): Promise<GetRevisionResponse> {
        const request: GetRevisionRequest = {
            Name: ledgerName,
            BlockAddress: blockAddress,
            DocumentId: documentId,
            DigestTipAddress: digestTipAddress
        };
        const result: GetRevisionResponse = await
        qlldbClient.getRevision(request).promise();
        return result;
    }

    /**
     * Query the table metadata for a particular vehicle for verification.
     * @param txn The {@linkcode TransactionExecutor} for lambda execute.
     * @param vin VIN to query the table metadata of a specific registration with.
     * @returns Promise which fulfills with a list of Ion values that contains the
     results of the query.
     */
    export async function lookupRegistrationForVin(txn: TransactionExecutor, vin:
    string): Promise<dom.Value[]> {
        log(`Querying the 'VehicleRegistration' table for VIN: ${vin}...`);
        let resultList: dom.Value[];
        const query: string = "SELECT blockAddress, metadata.id FROM
        _ql_committed_VehicleRegistration WHERE data.VIN = ?";

        await txn.execute(query, vin).then(function(result) {
            resultList = result.getResultList();
        });
        return resultList;
    }

    /**
     * Verify each version of the registration for the given VIN.

```



```

* @param txn The {@linkcode TransactionExecutor} for lambda execute.
* @param ledgerName The ledger to get the digest from.
* @param vin VIN to query the revision history of a specific registration with.
* @param qlldbClient The QLDB control plane client to use.
* @returns Promise which fulfills with void.
* @throws Error: When verification fails.
*/
export async function verifyRegistration(
    txn: TransactionExecutor,
    ledgerName: string,
    vin: string,
    qlldbClient: QLDB
): Promise<void> {
    log(`Let's verify the registration with VIN = ${vin}, in ledger =
    ${ledgerName}.`);
    const digest: GetDigestResponse = await getDigestResult(ledgerName,
    qlldbClient);
    const digestBytes: Digest = digest.Digest;
    const digestTipAddress: ValueHolder = digest.DigestTipAddress;

    log(
        `Got a ledger digest: digest tip address = \n
    ${valueHolderToString(digestTipAddress)},
        digest = \n${toBase64(<Uint8Array> digestBytes)}.`
    );
    log(`Querying the registration with VIN = ${vin} to verify each version of the
    registration...`);
    const resultList: dom.Value[] = await lookupRegistrationForVin(txn, vin);
    log("Getting a proof for the document.");

    for (const result of resultList) {
        const blockAddress: ValueHolder = blockAddressToValueHolder(result);
        const documentId: string = getMetadataId(result);

        const revisionResponse: GetRevisionResponse = await getRevision(
            ledgerName,
            documentId,
            blockAddress,
            digestTipAddress,
            qlldbClient
        );

        const revision: dom.Value = dom.load(revisionResponse.Revision.IonText);
        const documentHash: Uint8Array = getBlobValue(revision, "hash");
    }
}

```

```

    const proof: ValueHolder = revisionResponse.Proof;
    log(`Got back a proof: ${valueHolderToString(proof)}.`);

    let verified: boolean = verifyDocument(documentHash, digestBytes, proof);
    if (!verified) {
        throw new Error("Document revision is not verified.");
    } else {
        log("Success! The document is verified.");
    }
    const alteredDocumentHash: Uint8Array = flipRandomBit(documentHash);

    log(
        `Flipping one bit in the document's hash and assert that the document
is NOT verified.
        The altered document hash is: ${toBase64(alteredDocumentHash)}`
    );
    verified = verifyDocument(alteredDocumentHash, digestBytes, proof);

    if (verified) {
        throw new Error("Expected altered document hash to not be verified
against digest.");
    } else {
        log("Success! As expected flipping a bit in the document hash causes
verification to fail.");
    }
    log(`Finished verifying the registration with VIN = ${vin} in ledger =
${ledgerName}.`);
}
}

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qlldbClient: QLDB = new QLDB();
        const qlldbDriver: QldbDriver = getQldbDriver();

        const registration = VEHICLE_REGISTRATION[0];
        const vin: string = registration.VIN;

        await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await verifyRegistration(txn, LEDGER_NAME, vin, qlldbClient);
        });
    }
}

```

```
    });  
    } catch (e) {  
        error(`Unable to verify revision: ${e}`);  
    }  
}  
  
if (require.main === module) {  
    main();  
}
```

Note

After the `getRevision` function returns a proof for the specified document revision, this program uses a client-side API to verify that revision.

3. To run the transpiled program, enter the following command.

```
node dist/GetRevision.js
```

If you no longer need to use the `vehicle-registration` ledger, proceed to [Step 8 \(optional\): Clean up resources](#).

Step 8 (optional): Clean up resources

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can continue using the `vehicle-registration` ledger. However, if you no longer need it, you should delete it.

To delete the ledger

1. Use the following program (`DeleteLedger.ts`) to delete your `vehicle-registration` ledger and all of its contents.

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { isResourceNotFoundException } from "amazon-qlldb-driver-nodejs";
import { AWSError, QLDB } from "aws-sdk";
import { DeleteLedgerRequest, DescribeLedgerRequest } from "aws-sdk/clients/qlldb";

import { setDeletionProtection } from "./DeletionProtection";
import { LEDGER_NAME } from "./qlldb/Constants";
import { error, log } from "./qlldb/LogUtil";
import { sleep } from "./qlldb/Util";

const LEDGER_DELETION_POLL_PERIOD_MS = 20000;

/**
 * Send a request to QLDB to delete the specified ledger.
 * @param ledgerName Name of the ledger to be deleted.
 * @param qlldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */

```

```

export async function deleteLedger(ledgerName: string, qlldbClient: QLDB):
  Promise<void> {
  log(`Attempting to delete the ledger with name: ${ledgerName}`);
  const request: DeleteLedgerRequest = {
    Name: ledgerName
  };
  await qlldbClient.deleteLedger(request).promise();
  log("Success.");
}

/**
 * Wait for the ledger to be deleted.
 * @param ledgerName Name of the ledger to be deleted.
 * @param qlldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */
export async function waitForDeleted(ledgerName: string, qlldbClient: QLDB):
  Promise<void> {
  log("Waiting for the ledger to be deleted...");
  const request: DescribeLedgerRequest = {
    Name: ledgerName
  };
  let isDeleted: boolean = false;
  while (true) {
    await qlldbClient.describeLedger(request).promise().catch((error: AWSError)
=> {
      if (isResourceNotFoundException(error)) {
        isDeleted = true;
        log("Success. Ledger is deleted.");
      }
    });
    if (isDeleted) {
      break;
    }
    log("The ledger is still being deleted. Please wait...");
    await sleep(LEDGER_DELETION_POLL_PERIOD_MS);
  }
}

/**
 * Delete a ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {

```

```
    try {
      const qlldbClient: QLDB = new QLDB();
      await setDeletionProtection(LEDGER_NAME, qlldbClient, false);
      await deleteLedger(LEDGER_NAME, qlldbClient);
      await waitForDeleted(LEDGER_NAME, qlldbClient);
    } catch (e) {
      error(`Unable to delete the ledger: ${e}`);
    }
  }

  if (require.main === module) {
    main();
  }
}
```

Note

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API.

2. To run the transpiled program, enter the following command.

```
node dist/DeleteLedger.js
```

Amazon QLDB Python tutorial

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this implementation of the tutorial sample application, you use the Amazon QLDB driver with the AWS SDK for Python (Boto3) to create a QLDB ledger and populate it with sample data.

As you work through this tutorial, you can refer to [QLDB low-level client](#) in the *AWS SDK for Python (Boto3) API Reference* for management API operations. For transactional data operations, you can refer to the [QLDB Driver for Python API Reference](#).

Note

Where applicable, some tutorial steps have different commands or code examples for each supported major version of the QLDB driver for Python.

Topics

- [Installing the Amazon QLDB Python sample application](#)
- [Step 1: Create a new ledger](#)
- [Step 2: Test connectivity to the ledger](#)
- [Step 3: Create tables, indexes, and sample data](#)
- [Step 4: Query the tables in a ledger](#)
- [Step 5: Modify documents in a ledger](#)
- [Step 6: View the revision history for a document](#)
- [Step 7: Verify a document in a ledger](#)
- [Step 8 \(optional\): Clean up resources](#)

Installing the Amazon QLDB Python sample application**⚠ Important**

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes how to install and run the provided Amazon QLDB sample application for the step-by-step Python tutorial. The use case for this sample application is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations.

The DMV sample application for Python is open source in the GitHub repository [aws-samples/amazon-qldb-dmv-sample-python](#).

Prerequisites

Before you get started, make sure that you complete the QLDB driver for Python [Prerequisites](#). This includes installing Python and doing the following:

1. Sign up for AWS.
2. Create a user with the appropriate QLDB permissions.
3. Grant programmatic access for development.

To complete all of the steps in this tutorial, you need full administrative access to your ledger resource through the QLDB API.

Installation

To install the sample application

1. Enter the following pip command to clone and install the sample application from GitHub.

3.x

```
pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-python.git
```

2.x

```
pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-python.git@v1.0.0
```

The sample application packages the complete source code from this tutorial and its dependencies, including the Python driver and the [AWS SDK for Python \(Boto3\)](#).

2. Before you start running code at the command line, switch your current working directory to the location where the `pyqldb samples` package is installed. Enter the following command.

```
cd $(python -c "import pyqldbsamples; print(pyqldbsamples.__path__[0])")
```

3. Proceed to [Step 1: Create a new ledger](#) to start the tutorial and create a ledger.

Step 1: Create a new ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`.

To create a new ledger

1. Review the following file (`constants.py`), which contains constant values that are used by all of the other programs in this tutorial.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

class Constants:
    """
    Constant values used throughout this tutorial.
```

```

"""
LEDGER_NAME = "vehicle-registration"

VEHICLE_REGISTRATION_TABLE_NAME = "VehicleRegistration"
VEHICLE_TABLE_NAME = "Vehicle"
PERSON_TABLE_NAME = "Person"
DRIVERS_LICENSE_TABLE_NAME = "DriversLicense"

LICENSE_NUMBER_INDEX_NAME = "LicenseNumber"
GOV_ID_INDEX_NAME = "GovId"
VEHICLE_VIN_INDEX_NAME = "VIN"
LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber"
PERSON_ID_INDEX_NAME = "PersonId"

JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-tutorial-journal-export"
USER_TABLES = "information_schema.user_tables"
S3_BUCKET_ARN_TEMPLATE = "arn:aws:s3:::"
LEDGER_NAME_WITH_TAGS = "tags"

RETRY_LIMIT = 4

```

2. Use the following program (`create_ledger.py`) to create a ledger named `vehicle-registration`.

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION

```

```
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO
from time import sleep

from boto3 import client

from pyqldb.samples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

LEDGER_CREATION_POLL_PERIOD_SEC = 10
ACTIVE_STATE = "ACTIVE"

def create_ledger(name):
    """
    Create a new ledger with the specified name.

    :type name: str
    :param name: Name for the ledger to be created.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info("Let's create the ledger named: {}".format(name))
    result = qldb_client.create_ledger(Name=name, PermissionsMode='ALLOW_ALL')
    logger.info('Success. Ledger state: {}'.format(result.get('State')))
    return result

def wait_for_active(name):
    """
    Wait for the newly created ledger to become active.

    :type name: str
    :param name: The ledger to check on.

    :rtype: dict
```

```
:return: Result from the request.
"""
logger.info('Waiting for ledger to become active...')
while True:
    result = qlldb_client.describe_ledger(Name=name)
    if result.get('State') == ACTIVE_STATE:
        logger.info('Success. Ledger is active and ready to use.')
        return result
    logger.info('The ledger is still creating. Please wait...')
    sleep(LEDGER_CREATION_POLL_PERIOD_SEC)

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create a ledger and wait for it to be active.
    """
    try:
        create_ledger(ledger_name)
        wait_for_active(ledger_name)
    except Exception as e:
        logger.exception('Unable to create the ledger!')
        raise e

if __name__ == '__main__':
    main()
```

Note

- In the `create_ledger` call, you must specify a ledger name and a permissions mode. We recommend using the STANDARD permissions mode to maximize the security of your ledger data.
- When you create a ledger, *deletion protection* is enabled by default. This is a feature in QLDB that prevents ledgers from being deleted by any user. You have the option of disabling deletion protection on ledger creation using the QLDB API or the AWS Command Line Interface (AWS CLI).
- Optionally, you can also specify tags to attach to your ledger.

3. To run the program, enter the following command.

```
python create_ledger.py
```

To verify your connection to the new ledger, proceed to [Step 2: Test connectivity to the ledger](#).

Step 2: Test connectivity to the ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this step, you verify that you can connect to the vehicle-registration ledger in Amazon QLDB using the transactional data API endpoint.

To test connectivity to the ledger

1. Use the following program (`connect_to_ledger.py`) to create a data session connection to the vehicle-registration ledger.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```

# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from botocore.exceptions import ClientError

from pyqldb.driver.qldb_driver import QldbDriver
from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)

def create_qldb_driver(ledger_name=Constants.LEDGER_NAME, region_name=None,
                      endpoint_url=None, boto3_session=None):
    """
    Create a QLDB driver for executing transactions.

    :type ledger_name: str
    :param ledger_name: The QLDB ledger name.

    :type region_name: str
    :param region_name: See [1].

    :type endpoint_url: str
    :param endpoint_url: See [1].

    :type boto3_session: :py:class:`boto3.session.Session`
    :param boto3_session: The boto3 session to create the client with (see [1]).

    :rtype: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :return: A QLDB driver object.

    [1]: `Boto3 Session.client` Reference <https://
    boto3.amazonaws.com/v1/documentation/api/latest/reference/core/
    session.html#boto3.session.Session.client>`.
    """

```

```

        qlldb_driver = QldbDriver(ledger_name=ledger_name, region_name=region_name,
                                  endpoint_url=endpoint_url,
                                  boto3_session=boto3_session)

        return qlldb_driver

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Connect to a given ledger using default settings.
    """
    try:
        with create_qlldb_driver(ledger_name) as driver:
            logger.info('Listing table names ')
            for table in driver.list_tables():
                logger.info(table)
    except ClientError as ce:
        logger.exception('Unable to list tables.')
        raise ce

if __name__ == '__main__':
    main()

```

Note

- To run data transactions on your ledger, you must create a QLDB driver object to connect to a specific ledger. This is a different client object than the `qlldb_client` object that you used in the previous step to create the ledger. That previous client is only used to run the management API operations listed in the [Amazon QLDB API reference](#).
- You must specify a ledger name when you create this driver object.

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
of this

```

```

# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from botocore.exceptions import ClientError

from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)

def create_qldb_driver(ledger_name=Constants.LEDGER_NAME, region_name=None,
                      endpoint_url=None, boto3_session=None):
    """
    Create a QLDB driver for creating sessions.

    :type ledger_name: str
    :param ledger_name: The QLDB ledger name.

    :type region_name: str
    :param region_name: See [1].

    :type endpoint_url: str
    :param endpoint_url: See [1].

```



```

:type boto3_session: :py:class:`boto3.session.Session`
:param boto3_session: The boto3 session to create the client with (see [1]).

:rtype: :py:class:`pyqldb.driver.pooled_qldb_driver.PooledQldbDriver`
:return: A pooled QLDB driver object.

[1]: `Boto3 Session.client Reference <https://
boto3.amazonaws.com/v1/documentation/api/latest/reference/core/
session.html#boto3.session.Session.client>`.
"""
    qldb_driver = PooledQldbDriver(ledger_name=ledger_name,
region_name=region_name, endpoint_url=endpoint_url,
                                boto3_session=boto3_session)

    return qldb_driver

def create_qldb_session():
    """
    Retrieve a QLDB session object.

    :rtype: :py:class:`pyqldb.session.pooled_qldb_session.PooledQldbSession`
    :return: A pooled QLDB session object.
    """
    qldb_session = pooled_qldb_driver.get_session()
    return qldb_session

pooled_qldb_driver = create_qldb_driver()

if __name__ == '__main__':
    """
    Connect to a session for a given ledger using default settings.
    """
    try:
        qldb_session = create_qldb_session()
        logger.info('Listing table names ')
        for table in qldb_session.list_tables():
            logger.info(table)
    except ClientError:
        logger.exception('Unable to create session.')

```

Note

- To run data transactions on your ledger, you must create a QLDB driver object to connect to a specific ledger. This is a different client object than the `qldb_client` object that you used in the previous step to create the ledger. That previous client is only used to run the management API operations listed in the [Amazon QLDB API reference](#).
- First, create a pooled QLDB driver object. You must specify a ledger name when you create this driver.
- Then, you can create sessions from this pooled driver object.

2. To run the program, enter the following command.

```
python connect_to_ledger.py
```

To create tables in the `vehicle-registration` ledger, proceed to [Step 3: Create tables, indexes, and sample data](#).

Step 3: Create tables, indexes, and sample data

⚠ Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

When your Amazon QLDB ledger is active and accepts connections, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the `vehicle-registration` ledger:

- `VehicleRegistration`
- `Vehicle`
- `Person`

- DriversLicense

You also create the following indexes.

Table name	Field
VehicleRegistration	VIN
VehicleRegistration	LicensePlateNumber
Vehicle	VIN
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

When inserting sample data, you first insert documents into the Person table. Then, you use the system-assigned id from each Person document to populate the corresponding fields in the appropriate VehicleRegistration and DriversLicense documents.

 Tip

As a best practice, use a document's system-assigned id as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its id. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table). For more information about views in QLDB, see [Core concepts](#). To learn more about metadata, see [Querying document metadata](#).

To create tables and indexes

1. Use the following program (`create_table.py`) to create the previously mentioned tables.

3.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

def create_table(driver, table_name):
    """
    Create a table with the specified name.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

```

```

        :type table_name: str
        :param table_name: Name of the table to create.

        :rtype: int
        :return: The number of changes to the database.
        """
        logger.info("Creating the '{}' table...".format(table_name))
        statement = 'CREATE TABLE {}'.format(table_name)
        cursor = driver.execute_lambda(lambda executor:
        executor.execute_statement(statement))
        logger.info('{} table created successfully.'.format(table_name))
        return len(list(cursor))

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create registrations, vehicles, owners, and licenses tables.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            create_table(driver, Constants.DRIVERS_LICENSE_TABLE_NAME)
            create_table(driver, Constants.PERSON_TABLE_NAME)
            create_table(driver, Constants.VEHICLE_TABLE_NAME)
            create_table(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME)
            logger.info('Tables created successfully.')
    except Exception as e:
        logger.exception('Errors creating tables.')
        raise e

if __name__ == '__main__':
    main()

```

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software

```

```

# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)

def create_table(transaction_executor, table_name):
    """
    Create a table with the specified name using an Executor object.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to create.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Creating the '{}' table...".format(table_name))
    statement = 'CREATE TABLE {}'.format(table_name)
    cursor = transaction_executor.execute_statement(statement)

```

```
logger.info('{} table created successfully.'.format(table_name))
return len(list(cursor))

if __name__ == '__main__':
    """
    Create registrations, vehicles, owners, and licenses tables in a single
    transaction.
    """
    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda x: create_table(x,
Constants.DRIVERS_LICENSE_TABLE_NAME) and
                                create_table(x, Constants.PERSON_TABLE_NAME)
and
                                create_table(x, Constants.VEHICLE_TABLE_NAME)
and
                                create_table(x,
Constants.VEHICLE_REGISTRATION_TABLE_NAME),
                                lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Tables created successfully.')
    except Exception:
        logger.exception('Errors creating tables.')
```

Note

This program demonstrates how to use the `execute_lambda` function. In this example, you run multiple `CREATE TABLE PartiQL` statements with a single lambda expression.

This execute function implicitly starts a transaction, runs all of the statements in the lambda, and then auto-commits the transaction.

2. To run the program, enter the following command.

```
python create_table.py
```

3. Use the following program (`create_index.py`) to create indexes on the tables, as previously described.

3.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

def create_index(driver, table_name, index_attribute):
    """
    Create an index for a particular table.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

```



```

:type table_name: str
:param table_name: Name of the table to add indexes for.

:type index_attribute: str
:param index_attribute: Index to create on a single attribute.

:rtype: int
:return: The number of changes to the database.
"""
logger.info("Creating index on '{}...'".format(index_attribute))
statement = 'CREATE INDEX on {} ({} )'.format(table_name, index_attribute)
cursor = driver.execute_lambda(lambda executor:
executor.execute_statement(statement))
return len(list(cursor))

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create indexes on tables in a particular ledger.
    """
    logger.info('Creating indexes on all tables...')
    try:
        with create_qldb_driver(ledger_name) as driver:
            create_index(driver, Constants.PERSON_TABLE_NAME,
Constants.GOV_ID_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_TABLE_NAME,
Constants.VEHICLE_VIN_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Constants.LICENSE_PLATE_NUMBER_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
Constants.VEHICLE_VIN_INDEX_NAME)
            create_index(driver, Constants.DRIVERS_LICENSE_TABLE_NAME,
Constants.PERSON_ID_INDEX_NAME)
            create_index(driver, Constants.DRIVERS_LICENSE_TABLE_NAME,
Constants.LICENSE_NUMBER_INDEX_NAME)
            logger.info('Indexes created successfully.')
    except Exception as e:
        logger.exception('Unable to create indexes.')
        raise e

if __name__ == '__main__':
    main()

```

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)

def create_index(transaction_executor, table_name, index_attribute):
    """
    Create an index for a particular table.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

```

```

:type table_name: str
:param table_name: Name of the table to add indexes for.

:type index_attribute: str
:param index_attribute: Index to create on a single attribute.

:rtype: int
:return: The number of changes to the database.
"""
logger.info("Creating index on '{}...'".format(index_attribute))
statement = 'CREATE INDEX on {} ({} )'.format(table_name, index_attribute)
cursor = transaction_executor.execute_statement(statement)
return len(list(cursor))

if __name__ == '__main__':
    """
    Create indexes on tables in a particular ledger.
    """
    logger.info('Creating indexes on all tables in a single transaction...')
    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda x: create_index(x,
Constants.PERSON_TABLE_NAME,

Constants.GOV_ID_INDEX_NAME)
                                and create_index(x,
Constants.VEHICLE_TABLE_NAME,

Constants.VEHICLE_VIN_INDEX_NAME)
                                and create_index(x,
Constants.VEHICLE_REGISTRATION_TABLE_NAME,

Constants.LICENSE_PLATE_NUMBER_INDEX_NAME)
                                and create_index(x,
Constants.VEHICLE_REGISTRATION_TABLE_NAME,

Constants.VEHICLE_VIN_INDEX_NAME)
                                and create_index(x,
Constants.DRIVERS_LICENSE_TABLE_NAME,

Constants.PERSON_ID_INDEX_NAME)

```

```

                                and create_index(x,
Constants.DRIVERS_LICENSE_TABLE_NAME,

Constants.LICENSE_NUMBER_INDEX_NAME),
                                lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
                                logger.info('Indexes created successfully.')
except Exception:
    logger.exception('Unable to create indexes.')

```

4. To run the program, enter the following command.

```
python create_index.py
```

To load data into the tables

1. Review the following file (`sample_data.py`), which represents the sample data that you insert into the vehicle-registration tables. This file also imports from the `amazon.ion` package to provide helper functions that convert, parse, and print [Amazon Ion](#) data.

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
from datetime import datetime

```

```
from decimal import Decimal
from logging import basicConfig, getLogger, INFO

from amazon.ion.simple_types import IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict,
    IonPyFloat, IonPyInt, IonPyList, \
    IonPyNull, IonPySymbol, IonPyText, IonPyTimestamp
from amazon.ion.simpleion import dumps, loads

logger = getLogger(__name__)
basicConfig(level=INFO)
IonValue = (IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict, IonPyFloat, IonPyInt,
    IonPyList, IonPyNull, IonPySymbol,
    IonPyText, IonPyTimestamp)

class SampleData:
    """
    Sample domain objects for use throughout this tutorial.
    """
    DRIVERS_LICENSE = [
        {
            'PersonId': '',
            'LicenseNumber': 'LEWISR261LL',
            'LicenseType': 'Learner',
            'ValidFromDate': datetime(2016, 12, 20),
            'ValidToDate': datetime(2020, 11, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': 'LOGANB486CG',
            'LicenseType': 'Probationary',
            'ValidFromDate': datetime(2016, 4, 6),
            'ValidToDate': datetime(2020, 11, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': '744 849 301',
            'LicenseType': 'Full',
            'ValidFromDate': datetime(2017, 12, 6),
            'ValidToDate': datetime(2022, 10, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': 'P626-168-229-765',
```

```
        'LicenseType': 'Learner',
        'ValidFromDate': datetime(2017, 8, 16),
        'ValidToDate': datetime(2021, 11, 15)
    },
    {
        'PersonId': '',
        'LicenseNumber': 'S152-780-97-415-0',
        'LicenseType': 'Probationary',
        'ValidFromDate': datetime(2015, 8, 15),
        'ValidToDate': datetime(2021, 8, 21)
    }
]
PERSON = [
    {
        'FirstName': 'Raul',
        'LastName': 'Lewis',
        'Address': '1719 University Street, Seattle, WA, 98109',
        'DOB': datetime(1963, 8, 19),
        'GovId': 'LEWISR261LL',
        'GovIdType': 'Driver License'
    },
    {
        'FirstName': 'Brent',
        'LastName': 'Logan',
        'DOB': datetime(1967, 7, 3),
        'Address': '43 Stockert Hollow Road, Everett, WA, 98203',
        'GovId': 'LOGANB486CG',
        'GovIdType': 'Driver License'
    },
    {
        'FirstName': 'Alexis',
        'LastName': 'Pena',
        'DOB': datetime(1974, 2, 10),
        'Address': '4058 Melrose Street, Spokane Valley, WA, 99206',
        'GovId': '744 849 301',
        'GovIdType': 'SSN'
    },
    {
        'FirstName': 'Melvin',
        'LastName': 'Parker',
        'DOB': datetime(1976, 5, 22),
        'Address': '4362 Ryder Avenue, Seattle, WA, 98101',
        'GovId': 'P626-168-229-765',
        'GovIdType': 'Passport'
    }
]
```

```
    },
    {
        'FirstName': 'Salvatore',
        'LastName': 'Spencer',
        'DOB': datetime(1997, 11, 15),
        'Address': '4450 Honeysuckle Lane, Seattle, WA, 98101',
        'GovId': 'S152-780-97-415-0',
        'GovIdType': 'Passport'
    }
]
VEHICLE = [
    {
        'VIN': '1N4AL11D75C109151',
        'Type': 'Sedan',
        'Year': 2011,
        'Make': 'Audi',
        'Model': 'A5',
        'Color': 'Silver'
    },
    {
        'VIN': 'KM8SRDHF6EU074761',
        'Type': 'Sedan',
        'Year': 2015,
        'Make': 'Tesla',
        'Model': 'Model S',
        'Color': 'Blue'
    },
    {
        'VIN': '3HGGK5G53FM761765',
        'Type': 'Motorcycle',
        'Year': 2011,
        'Make': 'Ducati',
        'Model': 'Monster 1200',
        'Color': 'Yellow'
    },
    {
        'VIN': '1HVBBAANXWH544237',
        'Type': 'Semi',
        'Year': 2009,
        'Make': 'Ford',
        'Model': 'F 150',
        'Color': 'Black'
    },
    {
```

```

        'VIN': '1C4RJFAG0FC625797',
        'Type': 'Sedan',
        'Year': 2019,
        'Make': 'Mercedes',
        'Model': 'CLK 350',
        'Color': 'White'
    }
]
VEHICLE_REGISTRATION = [
    {
        'VIN': '1N4AL11D75C109151',
        'LicensePlateNumber': 'LEWISR261LL',
        'State': 'WA',
        'City': 'Seattle',
        'ValidFromDate': datetime(2017, 8, 21),
        'ValidToDate': datetime(2020, 5, 11),
        'PendingPenaltyTicketAmount': Decimal('90.25'),
        'Owners': {
            'PrimaryOwner': {'PersonId': ''},
            'SecondaryOwners': []
        }
    },
    {
        'VIN': 'KM8SRDHF6EU074761',
        'LicensePlateNumber': 'CA762X',
        'State': 'WA',
        'City': 'Kent',
        'PendingPenaltyTicketAmount': Decimal('130.75'),
        'ValidFromDate': datetime(2017, 9, 14),
        'ValidToDate': datetime(2020, 6, 25),
        'Owners': {
            'PrimaryOwner': {'PersonId': ''},
            'SecondaryOwners': []
        }
    },
    {
        'VIN': '3HGGK5G53FM761765',
        'LicensePlateNumber': 'CD820Z',
        'State': 'WA',
        'City': 'Everett',
        'PendingPenaltyTicketAmount': Decimal('442.30'),
        'ValidFromDate': datetime(2011, 3, 17),
        'ValidToDate': datetime(2021, 3, 24),
        'Owners': {

```



```

        'PrimaryOwner': {'PersonId': ''},
        'SecondaryOwners': []
    }
},
{
    'VIN': '1HVBBAANXWH544237',
    'LicensePlateNumber': 'LS477D',
    'State': 'WA',
    'City': 'Tacoma',
    'PendingPenaltyTicketAmount': Decimal('42.20'),
    'ValidFromDate': datetime(2011, 10, 26),
    'ValidToDate': datetime(2023, 9, 25),
    'Owners': {
        'PrimaryOwner': {'PersonId': ''},
        'SecondaryOwners': []
    }
},
{
    'VIN': '1C4RJFAG0FC625797',
    'LicensePlateNumber': 'TH393F',
    'State': 'WA',
    'City': 'Olympia',
    'PendingPenaltyTicketAmount': Decimal('30.45'),
    'ValidFromDate': datetime(2013, 9, 2),
    'ValidToDate': datetime(2024, 3, 19),
    'Owners': {
        'PrimaryOwner': {'PersonId': ''},
        'SecondaryOwners': []
    }
}
]

```

```

def convert_object_to_ion(py_object):
    """
    Convert a Python object into an Ion object.

    :type py_object: object
    :param py_object: The object to convert.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyValue`
    :return: The converted Ion object.
    """
    ion_object = loads(dumps(py_object))

```

```

    return ion_object

def to_ion_struct(key, value):
    """
    Convert the given key and value into an Ion struct.

    :type key: str
    :param key: The key which serves as an unique identifier.

    :type value: str
    :param value: The value associated with a given key.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The Ion dictionary object.
    """
    ion_struct = dict()
    ion_struct[key] = value
    return loads(str(ion_struct))

def get_document_ids(transaction_executor, table_name, field, value):
    """
    Gets the document IDs from the given table.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type table_name: str
    :param table_name: The table name to query.

    :type field: str
    :param field: A field to query.

    :type value: str
    :param value: The key of the given field.

    :rtype: list
    :return: A list of document IDs.
    """
    query = "SELECT id FROM {} AS t BY id WHERE t.{} = {}".format(table_name, field, value)
    cursor = transaction_executor.execute_statement(query,
    convert_object_to_ion(value))

```

```

    return list(map(lambda table: table.get('id'), cursor))

def get_document_ids_from_dml_results(result):
    """
    Return a list of modified document IDs as strings from DML results.

    :type result: :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :param result: The result set from DML operation.

    :rtype: list
    :return: List of document IDs.
    """
    ret_val = list(map(lambda x: x.get('documentId'), result))
    return ret_val

def print_result(cursor):
    """
    Pretty print the result set. Returns the number of documents in the result set.

    :type cursor: :py:class:`pyqldb.cursor.stream_cursor.StreamCursor`/
                  :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :param cursor: An instance of the StreamCursor or BufferedCursor class.

    :rtype: int
    :return: Number of documents in the result set.
    """
    result_counter = 0
    for row in cursor:
        # Each row would be in Ion format.
        print_ion(row)
        result_counter += 1
    return result_counter

def print_ion(ion_value):
    """
    Pretty print an Ion Value.

    :type ion_value: :py:class:`amazon.ion.simple_types.IonPySymbol`
    :param ion_value: Any Ion Value to be pretty printed.
    """

```

```
logger.info(dumps(ion_value, binary=False, indent=' ',
omit_version_marker=True))
```

Note

The `get_document_ids` function runs a query that returns system-assigned document IDs from a table. To learn more, see [Using the BY clause to query document ID](#).

2. Use the following program (`insert_document.py`) to insert the sample data into your tables.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.constants import Constants
```

```

from pyqldb.samples.model.sample_data import convert_object_to_ion, SampleData,
    get_document_ids_from_dml_results
from pyqldb.samples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

def update_person_id(document_ids):
    """
    Update the PersonId value for DriversLicense records and the PrimaryOwner
    value for VehicleRegistration records.

    :type document_ids: list
    :param document_ids: List of document IDs.

    :rtype: list
    :return: Lists of updated DriversLicense records and updated
    VehicleRegistration records.
    """
    new_drivers_licenses = SampleData.DRIVERS_LICENSE.copy()
    new_vehicle_registrations = SampleData.VEHICLE_REGISTRATION.copy()
    for i in range(len(SampleData.PERSON)):
        drivers_license = new_drivers_licenses[i]
        registration = new_vehicle_registrations[i]
        drivers_license.update({'PersonId': str(document_ids[i])})
        registration['Owners']['PrimaryOwner'].update({'PersonId':
str(document_ids[i])})
    return new_drivers_licenses, new_vehicle_registrations

def insert_documents(driver, table_name, documents):
    """
    Insert the given list of documents into a table in a single transaction.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to insert documents into.

    :type documents: list
    :param documents: List of documents to insert.

```

```

        :rtype: list
        :return: List of documents IDs for the newly inserted documents.
        """
        logger.info('Inserting some documents in the {}
table...'.format(table_name))
        statement = 'INSERT INTO {} ?'.format(table_name)
        cursor = driver.execute_lambda(lambda executor:
executor.execute_statement(statement,

convert_object_to_ion(documents)))
        list_of_document_ids = get_document_ids_from_dml_results(cursor)

        return list_of_document_ids

def update_and_insert_documents(driver):
    """
    Handle the insertion of documents and updating PersonIds.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.
    """
    list_ids = insert_documents(driver, Constants.PERSON_TABLE_NAME,
SampleData.PERSON)

    logger.info("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
'VehicleRegistration'...")
    new_licenses, new_registrations = update_person_id(list_ids)

    insert_documents(driver, Constants.VEHICLE_TABLE_NAME, SampleData.VEHICLE)
    insert_documents(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
new_registrations)
    insert_documents(driver, Constants.DRIVERS_LICENSE_TABLE_NAME, new_licenses)

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Insert documents into a table in a QLDB ledger.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            # An INSERT statement creates the initial revision of a document
            with a version number of zero.

```

```

        # QLDB also assigns a unique document identifier in GUID format as
        part of the metadata.
        update_and_insert_documents(driver)
        logger.info('Documents inserted successfully!')
    except Exception as e:
        logger.exception('Error inserting or updating documents.')
        raise e

if __name__ == '__main__':
    main()

```

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbconstants import Constants

```

```

from pyqldb.samples.model.sample_data import convert_object_to_ion, SampleData,
    get_document_ids_from_dml_results
from pyqldb.samples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)

def update_person_id(document_ids):
    """
    Update the PersonId value for DriversLicense records and the PrimaryOwner
    value for VehicleRegistration records.

    :type document_ids: list
    :param document_ids: List of document IDs.

    :rtype: list
    :return: Lists of updated DriversLicense records and updated
    VehicleRegistration records.
    """
    new_drivers_licenses = SampleData.DRIVERS_LICENSE.copy()
    new_vehicle_registrations = SampleData.VEHICLE_REGISTRATION.copy()
    for i in range(len(SampleData.PERSON)):
        drivers_license = new_drivers_licenses[i]
        registration = new_vehicle_registrations[i]
        drivers_license.update({'PersonId': str(document_ids[i])})
        registration['Owners']['PrimaryOwner'].update({'PersonId':
str(document_ids[i])})
    return new_drivers_licenses, new_vehicle_registrations

def insert_documents(transaction_executor, table_name, documents):
    """
    Insert the given list of documents into a table in a single transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to insert documents into.

    :type documents: list
    :param documents: List of documents to insert.

```



```

        :rtype: list
        :return: List of documents IDs for the newly inserted documents.
        """
        logger.info('Inserting some documents in the {}'.format(table_name))
        statement = 'INSERT INTO {} ?'.format(table_name)
        cursor = transaction_executor.execute_statement(statement,
            convert_object_to_ion(documents))
        list_of_document_ids = get_document_ids_from_dml_results(cursor)

        return list_of_document_ids

def update_and_insert_documents(transaction_executor):
    """
    Handle the insertion of documents and updating PersonIds all in a single
    transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.
    """
    list_ids = insert_documents(transaction_executor,
        Constants.PERSON_TABLE_NAME, SampleData.PERSON)

    logger.info("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
    'VehicleRegistration'...")
    new_licenses, new_registrations = update_person_id(list_ids)

    insert_documents(transaction_executor, Constants.VEHICLE_TABLE_NAME,
        SampleData.VEHICLE)
    insert_documents(transaction_executor,
        Constants.VEHICLE_REGISTRATION_TABLE_NAME, new_registrations)
    insert_documents(transaction_executor, Constants.DRIVERS_LICENSE_TABLE_NAME,
        new_licenses)

if __name__ == '__main__':
    """
    Insert documents into a table in a QLDB ledger.
    """
    try:
        with create_qldb_session() as session:

```

```
# An INSERT statement creates the initial revision of a document
with a version number of zero.
# QLDB also assigns a unique document identifier in GUID format as
part of the metadata.
session.execute_lambda(lambda executor:
update_and_insert_documents(executor),
                        lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
    logger.info('Documents inserted successfully!')
except Exception:
    logger.exception('Error inserting or updating documents.')
```

Note

- This program demonstrates how to call the `execute_statement` function with parameterized values. You can pass data parameters in addition to the PartiQL statement that you want to run. Use a question mark (?) as a variable placeholder in your statement string.
- If an INSERT statement succeeds, it returns the `id` of each inserted document.

3. To run the program, enter the following command.

```
python insert_document.py
```

Next, you can use SELECT statements to read data from the tables in the vehicle-registration ledger. Proceed to [Step 4: Query the tables in a ledger](#).

Step 4: Query the tables in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses [PartiQL](#) as its query language and [Amazon Ion](#) as its document-oriented data model.

PartiQL is an open-source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open-source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use SELECT statements to read data from the tables in the vehicle-registration ledger.

Warning

When you run a query in QLDB without an indexed lookup, it invokes a full table scan. PartiQL supports such queries because it's SQL compatible. However, *don't* run table scans for production use cases in QLDB. Table scans can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator on an indexed field or a document ID; for example, WHERE indexedField = 123 or WHERE indexedField IN (456, 789). For more information, see [Optimizing query performance](#).

To query the tables

1. Use the following program (`find_vehicles.py`) to query all vehicles registered under a person in your ledger.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
```

```

# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.model.sample_data import get_document_ids, print_result,
    SampleData
from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

def find_vehicles_for_owner(driver, gov_id):
    """
    Find vehicles registered under a driver using their government ID.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type gov_id: str
    :param gov_id: The owner's government ID.
    """
    document_ids = driver.execute_lambda(lambda executor:
        get_document_ids(executor, Constants.PERSON_TABLE_NAME,
            'GovId', gov_id))

    query = "SELECT Vehicle FROM Vehicle INNER JOIN VehicleRegistration AS r " \
        "ON Vehicle.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId = ?"

```

```

        for ids in document_ids:
            cursor = driver.execute_lambda(lambda executor:
executor.execute_statement(query, ids))
            logger.info('List of Vehicles for owner with GovId:
{}'.format(gov_id))
            print_result(cursor)

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Find all vehicles registered under a person.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            # Find all vehicles registered under a person.
            gov_id = SampleData.PERSON[0]['GovId']
            find_vehicles_for_owner(driver, gov_id)
    except Exception as e:
        logger.exception('Error getting vehicles for owner.')
        raise e

if __name__ == '__main__':
    main()

```

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,

```

```

# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.model.sample_data import get_document_ids, print_result,
    SampleData
from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)

def find_vehicles_for_owner(transaction_executor, gov_id):
    """
    Find vehicles registered under a driver using their government ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type gov_id: str
    :param gov_id: The owner's government ID.
    """
    document_ids = get_document_ids(transaction_executor,
    Constants.PERSON_TABLE_NAME, 'GovId', gov_id)

    query = "SELECT Vehicle FROM Vehicle INNER JOIN VehicleRegistration AS r " \
    "ON Vehicle.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId = ?"

    for ids in document_ids:
        cursor = transaction_executor.execute_statement(query, ids)
        logger.info('List of Vehicles for owner with GovId:
        {}...'.format(gov_id))
        print_result(cursor)

```

```
if __name__ == '__main__':  
    """  
    Find all vehicles registered under a person.  
    """  
    try:  
        with create_qldb_session() as session:  
            # Find all vehicles registered under a person.  
            gov_id = SampleData.PERSON[0]['GovId']  
            session.execute_lambda(lambda executor:  
find_vehicles_for_owner(executor, gov_id),  
                                lambda retry_attempt: logger.info('Retrying  
due to OCC conflict...'))  
    except Exception:  
        logger.exception('Error getting vehicles for owner.')
```

Note

First, this program queries the Person table for the document with GovId LEWISR261LL to get its id metadata field.

Then, it uses this document id as a foreign key to query the VehicleRegistration table by PrimaryOwner.PersonId. It also joins VehicleRegistration with the Vehicle table on the VIN field.

2. To run the program, enter the following command.

```
python find_vehicles.py
```

To learn about modifying documents in the tables in the vehicle-registration ledger, see [Step 5: Modify documents in a ledger](#).

Step 5: Modify documents in a ledger

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Now that you have data to work with, you can start making changes to documents in the vehicle-registration ledger in Amazon QLDB. In this step, the following code examples demonstrate how to run data manipulation language (DML) statements. These statements update the primary owner of one vehicle and add a secondary owner to another vehicle.

To modify documents

1. Use the following program (`transfer_vehicle_ownership.py`) to update the primary owner of the vehicle with VIN 1N4AL11D75C109151 in your ledger.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.add_secondary_owner import get_document_ids, print_result,
    SampleData
from pyqldb.samples.constants import Constants
from pyqldb.samples.model.sample_data import convert_object_to_ion
```



```

from pyqldb.samples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

def find_person_from_document_id(transaction_executor, document_id):
    """
    Query a driver's information using the given ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: The document ID required to query for the person.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    query = 'SELECT p.* FROM Person AS p BY pid WHERE pid = ?'
    cursor = transaction_executor.execute_statement(query, document_id)
    return next(cursor)

def find_primary_owner_for_vehicle(driver, vin):
    """
    Find the primary owner of a vehicle given its VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN to find primary owner for.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    logger.info('Finding primary owner for vehicle with VIN: {}'.format(vin))
    query = "SELECT Owners.PrimaryOwner.PersonId FROM VehicleRegistration AS v
    WHERE v.VIN = ?"
    cursor = driver.execute_lambda(lambda executor:
    executor.execute_statement(query, convert_object_to_ion(vin)))
    try:

```

```

        return driver.execute_lambda(lambda executor:
find_person_from_document_id(executor,

next(cursor).get('PersonId'))))
    except StopIteration:
        logger.error('No primary owner registered for this vehicle.')
        return None

def update_vehicle_registration(driver, vin, document_id):
    """
    Update the primary owner for a vehicle using the given VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN for the vehicle to operate on.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: New PersonId for the primary owner.

    :raises RuntimeError: If no vehicle registration was found using the given
document ID and VIN.
    """
    logger.info('Updating the primary owner for vehicle with Vin:
{}'.format(vin))
    statement = "UPDATE VehicleRegistration AS r SET
r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?"
    cursor = driver.execute_lambda(lambda executor:
executor.execute_statement(statement, document_id,

convert_object_to_ion(vin)))
    try:
        print_result(cursor)
        logger.info('Successfully transferred vehicle with VIN: {} to new
owner.'.format(vin))
    except StopIteration:
        raise RuntimeError('Unable to transfer vehicle, could not find
registration.')

def validate_and_update_registration(driver, vin, current_owner, new_owner):
    """

```

Validate the current owner of the given vehicle and transfer its ownership to a new owner.

```
:type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
:param driver: An instance of the QldbDriver class.

:type vin: str
:param vin: The VIN of the vehicle to transfer ownership of.

:type current_owner: str
:param current_owner: The GovId of the current owner of the vehicle.

:type new_owner: str
:param new_owner: The GovId of the new owner of the vehicle.

:raises RuntimeError: If unable to verify primary owner.
"""
primary_owner = find_primary_owner_for_vehicle(driver, vin)
if primary_owner is None or primary_owner['GovId'] != current_owner:
    raise RuntimeError('Incorrect primary owner identified for vehicle,
unable to transfer.')

    document_ids = driver.execute_lambda(lambda executor:
get_document_ids(executor, Constants.PERSON_TABLE_NAME,

'GovId', new_owner))
    update_vehicle_registration(driver, vin, document_ids[0])

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Find primary owner for a particular vehicle's VIN.
    Transfer to another primary owner for a particular vehicle's VIN.
    """
    vehicle_vin = SampleData.VEHICLE[0]['VIN']
    previous_owner = SampleData.PERSON[0]['GovId']
    new_owner = SampleData.PERSON[1]['GovId']

    try:
        with create_qldb_driver(ledger_name) as driver:
            validate_and_update_registration(driver, vehicle_vin,
previous_owner, new_owner)
    except Exception as e:
        logger.exception('Error updating VehicleRegistration.')
```

```

        raise e

if __name__ == '__main__':
    main()

```

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.add_secondary_owner import get_document_ids, print_result,
    SampleData
from pyqldb.samples.constants import Constants
from pyqldb.samples.model.sample_data import convert_object_to_ion
from pyqldb.samples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)

```

```

def find_person_from_document_id(transaction_executor, document_id):
    """
    Query a driver's information using the given ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: The document ID required to query for the person.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    query = 'SELECT p.* FROM Person AS p BY pid WHERE pid = ?'
    cursor = transaction_executor.execute_statement(query, document_id)
    return next(cursor)

def find_primary_owner_for_vehicle(transaction_executor, vin):
    """
    Find the primary owner of a vehicle given its VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type vin: str
    :param vin: The VIN to find primary owner for.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    logger.info('Finding primary owner for vehicle with VIN: {}'.format(vin))
    query = "SELECT Owners.PrimaryOwner.PersonId FROM VehicleRegistration AS v
    WHERE v.VIN = ?"
    cursor = transaction_executor.execute_statement(query,
    convert_object_to_ion(vin))
    try:
        return find_person_from_document_id(transaction_executor,
    next(cursor).get('PersonId'))
    except StopIteration:

```

```

        logger.error('No primary owner registered for this vehicle.')
        return None

def update_vehicle_registration(transaction_executor, vin, document_id):
    """
    Update the primary owner for a vehicle using the given VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.

    :type vin: str
    :param vin: The VIN for the vehicle to operate on.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: New PersonId for the primary owner.

    :raises RuntimeError: If no vehicle registration was found using the given
    document ID and VIN.
    """
    logger.info('Updating the primary owner for vehicle with Vin:
    {}...'.format(vin))
    statement = "UPDATE VehicleRegistration AS r SET
    r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?"
    cursor = transaction_executor.execute_statement(statement, document_id,
    convert_object_to_ion(vin))
    try:
        print_result(cursor)
        logger.info('Successfully transferred vehicle with VIN: {} to new
        owner.'.format(vin))
    except StopIteration:
        raise RuntimeError('Unable to transfer vehicle, could not find
        registration.')

def validate_and_update_registration(transaction_executor, vin, current_owner,
    new_owner):
    """
    Validate the current owner of the given vehicle and transfer its ownership
    to a new owner in a single transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`

```

```

:param transaction_executor: An Executor object allowing for execution of
statements within a transaction.

:type vin: str
:param vin: The VIN of the vehicle to transfer ownership of.

:type current_owner: str
:param current_owner: The GovId of the current owner of the vehicle.

:type new_owner: str
:param new_owner: The GovId of the new owner of the vehicle.

:raises RuntimeError: If unable to verify primary owner.
"""
primary_owner = find_primary_owner_for_vehicle(transaction_executor, vin)
if primary_owner is None or primary_owner['GovId'] != current_owner:
    raise RuntimeError('Incorrect primary owner identified for vehicle,
unable to transfer.')

    document_id = next(get_document_ids(transaction_executor,
Constants.PERSON_TABLE_NAME, 'GovId', new_owner))

    update_vehicle_registration(transaction_executor, vin, document_id)

if __name__ == '__main__':
    """
    Find primary owner for a particular vehicle's VIN.
    Transfer to another primary owner for a particular vehicle's VIN.
    """
    vehicle_vin = SampleData.VEHICLE[0]['VIN']
    previous_owner = SampleData.PERSON[0]['GovId']
    new_owner = SampleData.PERSON[1]['GovId']

    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda executor:
validate_and_update_registration(executor, vehicle_vin,

            previous_owner, new_owner),

                                retry_indicator=lambda retry_attempt:
logger.info('Retrying due to OCC conflict...'))
    except Exception:

```

```
logger.exception('Error updating VehicleRegistration.')
```

2. To run the program, enter the following command.

```
python transfer_vehicle_ownership.py
```

3. Use the following program (`add_secondary_owner.py`) to add a secondary owner to the vehicle with VIN `KM8SRDHF6EU074761` in your ledger.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.model.sample_data import to_ion_struct, get_document_ids,
    print_result, SampleData, \
        convert_object_to_ion
from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_driver
```



```

logger = getLogger(__name__)
basicConfig(level=INFO)

def get_document_id_by_gov_id(driver, government_id):
    """
    Find a driver's person ID using the given government ID.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type government_id: str
    :param government_id: A driver's government ID.

    :rtype: list
    :return: A list of document IDs.
    """
    logger.info("Finding secondary owner's person ID using given government ID:
    {}".format(government_id))
    return driver.execute_lambda(lambda executor: get_document_ids(executor,
    Constants.PERSON_TABLE_NAME, 'GovId',
    government_id))

def is_secondary_owner_for_vehicle(driver, vin, secondary_owner_id):
    """
    Check whether a secondary owner has already been registered for the given
    VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to query.

    :type secondary_owner_id: str
    :param secondary_owner_id: The secondary owner's person ID.

    :rtype: bool
    :return: If the driver has already been registered.
    """

```

```

        logger.info('Finding secondary owners for vehicle with VIN:
        {}'.format(vin))
        query = 'SELECT Owners.SecondaryOwners FROM VehicleRegistration AS v WHERE
        v.VIN = ?'
        rows = driver.execute_lambda(lambda executor:
        executor.execute_statement(query, convert_object_to_ion(vin)))

        for row in rows:
            secondary_owners = row.get('SecondaryOwners')
            person_ids = map(lambda owner: owner.get('PersonId').text,
            secondary_owners)
            if secondary_owner_id in person_ids:
                return True
        return False

def add_secondary_owner_for_vin(driver, vin, parameter):
    """
    Add a secondary owner into `VehicleRegistration` table for a particular VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to add a secondary owner for.

    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to
    Ion for filling in parameters of the
        statement.
    """
    logger.info('Inserting secondary owner for vehicle with VIN:
    {}'.format(vin))
    statement = "FROM VehicleRegistration AS v WHERE v.VIN = ? INSERT INTO
    v.Owners.SecondaryOwners VALUE ?"

    cursor = driver.execute_lambda(lambda executor:
    executor.execute_statement(statement, convert_object_to_ion(vin),
    parameter))
    logger.info('VehicleRegistration Document IDs which had secondary owners
    added: ')
    print_result(cursor)

```

```

def register_secondary_owner(driver, vin, gov_id):
    """
    Register a secondary owner for a vehicle if they are not already registered.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to register a secondary owner for.

    :type gov_id: str
    :param gov_id: The government ID of the owner.
    """
    logger.info('Finding the secondary owners for vehicle with VIN:
    {}'.format(vin))

    document_ids = get_document_id_by_gov_id(driver, gov_id)

    for document_id in document_ids:
        if is_secondary_owner_for_vehicle(driver, vin, document_id):
            logger.info('Person with ID {} has already been added as a secondary
            owner of this vehicle.'.format(gov_id))
        else:
            add_secondary_owner_for_vin(driver, vin, to_ion_struct('PersonId',
            document_id))

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Finds and adds secondary owners for a vehicle.
    """
    vin = SampleData.VEHICLE[1]['VIN']
    gov_id = SampleData.PERSON[0]['GovId']
    try:
        with create_qldb_driver(ledger_name) as driver:
            register_secondary_owner(driver, vin, gov_id)
            logger.info('Secondary owners successfully updated.')
    except Exception as e:
        logger.exception('Error adding secondary owner.')
        raise e

if __name__ == '__main__':

```

```
main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.model.sample_data import to_ion_struct, get_document_ids,
    print_result, SampleData, \
        convert_object_to_ion
from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)

def get_document_id_by_gov_id(transaction_executor, government_id):
    """
```

```

        Find a driver's person ID using the given government ID.
        :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
        :param transaction_executor: An Executor object allowing for execution of
statements within a transaction.
        :type government_id: str
        :param government_id: A driver's government ID.
        :rtype: list
        :return: A list of document IDs.
        """
        logger.info("Finding secondary owner's person ID using given government ID:
{}".format(government_id))
        return get_document_ids(transaction_executor, Constants.PERSON_TABLE_NAME,
'GovId', government_id)

def is_secondary_owner_for_vehicle(transaction_executor, vin,
secondary_owner_id):
    """
    Check whether a secondary owner has already been registered for the given
VIN.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to query.
    :type secondary_owner_id: str
    :param secondary_owner_id: The secondary owner's person ID.
    :rtype: bool
    :return: If the driver has already been registered.
    """
    logger.info('Finding secondary owners for vehicle with VIN:
{}...'.format(vin))
    query = 'SELECT Owners.SecondaryOwners FROM VehicleRegistration AS v WHERE
v.VIN = ?'
    rows = transaction_executor.execute_statement(query,
convert_object_to_ion(vin))

    for row in rows:
        secondary_owners = row.get('SecondaryOwners')
        person_ids = map(lambda owner: owner.get('PersonId').text,
secondary_owners)
        if secondary_owner_id in person_ids:
            return True
    return False

```

```

def add_secondary_owner_for_vin(transaction_executor, vin, parameter):
    """
    Add a secondary owner into `VehicleRegistration` table for a particular VIN.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to add a secondary owner for.
    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to
    Ion for filling in parameters of the
                        statement.
    """
    logger.info('Inserting secondary owner for vehicle with VIN:
    {}.format(vin))
    statement = "FROM VehicleRegistration AS v WHERE v.VIN = '{}' INSERT INTO
    v.Owners.SecondaryOwners VALUE ?" \
        .format(vin)

    cursor = transaction_executor.execute_statement(statement, parameter)
    logger.info('VehicleRegistration Document IDs which had secondary owners
    added: ')
    print_result(cursor)

def register_secondary_owner(transaction_executor, vin, gov_id):
    """
    Register a secondary owner for a vehicle if they are not already registered.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
    statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to register a secondary owner for.
    :type gov_id: str
    :param gov_id: The government ID of the owner.
    """
    logger.info('Finding the secondary owners for vehicle with VIN:
    {}.format(vin))
    document_ids = get_document_id_by_gov_id(transaction_executor, gov_id)

    for document_id in document_ids:

```

```
        if is_secondary_owner_for_vehicle(transaction_executor, vin,
document_id):
            logger.info('Person with ID {} has already been added as a secondary
owner of this vehicle.'.format(gov_id))
        else:
            add_secondary_owner_for_vin(transaction_executor, vin,
to_ion_struct('PersonId', document_id))

if __name__ == '__main__':
    """
    Finds and adds secondary owners for a vehicle.
    """
    vin = SampleData.VEHICLE[1]['VIN']
    gov_id = SampleData.PERSON[0]['GovId']
    try:
        with create_qlldb_session() as session:
            session.execute_lambda(lambda executor:
register_secondary_owner(executor, vin, gov_id),
                                lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Secondary owners successfully updated.')
    except Exception:
        logger.exception('Error adding secondary owner.')
```

4. To run the program, enter the following command.

```
python add_secondary_owner.py
```

To review these changes in the vehicle-registration ledger, see [Step 6: View the revision history for a document](#).

Step 6: View the revision history for a document

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After modifying registration data for a vehicle in the previous step, you can query the history of all its registered owners and any other updated fields. In this step, you query the revision history of a document in the `VehicleRegistration` table in your vehicle-registration ledger.

To view the revision history

1. Use the following program (`query_history.py`) to query the revision history of the `VehicleRegistration` document with VIN 1N4AL11D75C109151.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime, timedelta
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.model.sample_data import print_result, get_document_ids,
    SampleData
from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_driver
```



```

logger = getLogger(__name__)
basicConfig(level=INFO)

def format_date_time(date_time):
    """
    Format the given date time to a string.

    :type date_time: :py:class:`datetime.datetime`
    :param date_time: The date time to format.

    :rtype: str
    :return: The formatted date time.
    """
    return date_time.strftime('%Y-%m-%dT%H:%M:%S.%fZ')

def previous_primary_owners(driver, vin):
    """
    Find previous primary owners for the given VIN in a single transaction.
    In this example, query the `VehicleRegistration` history table to find all
    previous primary owners for a VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN to find previous primary owners for.
    """
    person_ids = driver.execute_lambda(lambda executor:
get_document_ids(executor,

Constants.VEHICLE_REGISTRATION_TABLE_NAME,

'VIN',

vin))

    todays_date = datetime.utcnow() - timedelta(seconds=1)
    three_months_ago = todays_date - timedelta(days=90)
    query = 'SELECT data.Owners.PrimaryOwner, metadata.version FROM history({},
{}, {}) AS h WHERE h.metadata.id = ?'.\
        format(Constants.VEHICLE_REGISTRATION_TABLE_NAME,
format_date_time(three_months_ago),
format_date_time(todays_date))

```

```

        for ids in person_ids:
            logger.info("Querying the 'VehicleRegistration' table's history using
VIN: {}".format(vin))
            cursor = driver.execute_lambda(lambda executor:
executor.execute_statement(query, ids))
            if not (print_result(cursor)) > 0:
                logger.info('No modification history found within the given time
frame for document ID: {}'.format(ids))

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Query a table's history for a particular set of documents.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            vin = SampleData.VEHICLE_REGISTRATION[0]['VIN']
            previous_primary_owners(driver, vin)
            logger.info('Successfully queried history.')
    except Exception as e:
        logger.exception('Unable to query history to find previous owners.')
        raise e

if __name__ == '__main__':
    main()

```

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
of this
# software and associated documentation files (the "Software"), to deal in the
Software
# without restriction, including without limitation the rights to use, copy,
modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
# permit persons to whom the Software is furnished to do so.
#

```

```

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime, timedelta
from logging import basicConfig, getLogger, INFO

from pyqldb.samples.model.sample_data import print_result, get_document_ids,
    SampleData
from pyqldb.samples.constants import Constants
from pyqldb.samples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)

def format_date_time(date_time):
    """
    Format the given date time to a string.

    :type date_time: :py:class:`datetime.datetime`
    :param date_time: The date time to format.

    :rtype: str
    :return: The formatted date time.
    """
    return date_time.strftime('%Y-%m-%dT%H:%M:%S.%fZ')

def previous_primary_owners(transaction_executor, vin):
    """
    Find previous primary owners for the given VIN in a single transaction.
    In this example, query the `VehicleRegistration` history table to find all
    previous primary owners for a VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`

```

```

:param transaction_executor: An Executor object allowing for execution of
statements within a transaction.

:type vin: str
:param vin: VIN to find previous primary owners for.
"""

person_ids = get_document_ids(transaction_executor,
Constants.VEHICLE_REGISTRATION_TABLE_NAME, 'VIN', vin)

todays_date = datetime.utcnow() - timedelta(seconds=1)
three_months_ago = todays_date - timedelta(days=90)
query = 'SELECT data.Owners.PrimaryOwner, metadata.version FROM history({},
{}, {}) AS h WHERE h.metadata.id = ?'.\
        format(Constants.VEHICLE_REGISTRATION_TABLE_NAME,
format_date_time(three_months_ago),
format_date_time(todays_date))

for ids in person_ids:
    logger.info("Querying the 'VehicleRegistration' table's history using
VIN: {}".format(vin))
    cursor = transaction_executor.execute_statement(query, ids)
    if not (print_result(cursor)) > 0:
        logger.info('No modification history found within the given time
frame for document ID: {}'.format(ids))

if __name__ == '__main__':
    """
    Query a table's history for a particular set of documents.
    """
    try:
        with create_qldb_session() as session:
            vin = SampleData.VEHICLE_REGISTRATION[0]['VIN']
            session.execute_lambda(lambda lambda_executor:
previous_primary_owners(lambda_executor, vin),
                                lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Successfully queried history.')
    except Exception:
        logger.exception('Unable to query history to find previous owners.')

```

Note

- You can view the revision history of a document by querying the built-in [History function](#) in the following syntax.

```
SELECT * FROM history( table_name [, 'start-time' [, 'end-time' ] ] ) AS h
[ WHERE h.metadata.id = 'id' ]
```

- The *start-time* and *end-time* are both optional. They're Amazon Ion literal values that can be denoted with backticks (``...``). To learn more, see [Querying Ion with PartiQL in Amazon QLDB](#).
- As a best practice, qualify a history query with both a date range (*start-time* and *end-time*) and a document ID (`metadata.id`). QLDB processes SELECT queries in transactions, which are subject to a [transaction timeout limit](#).

QLDB history is indexed by document ID, and you can't create additional history indexes at this time. History queries that include a start time and end time gain the benefit of date range qualification.

2. To run the program, enter the following command.

```
python query_history.py
```

To cryptographically verify a document revision in the vehicle-registration ledger, proceed to [Step 7: Verify a document in a ledger](#).

Step 7: Verify a document in a ledger

⚠ Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. To learn more about how verification and cryptographic hashing work in QLDB, see [Data verification in Amazon QLDB](#).

In this step, you verify a document revision in the `VehicleRegistration` table in your vehicle-registration ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To verify a document revision

1. Review the following `.py` files, which represent QLDB objects that are required for verification and a utility module with helper functions to convert QLDB response types to strings.

1. `block_address.py`

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

def block_address_to_dictionary(ion_dict):
    """
```

```

    Convert a block address from IonPyDict into a dictionary.
    Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">,
sequenceNo: <sequenceNo>}"}

:type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
:param ion_dict: The block address value to convert.

:rtype: dict
:return: The converted dict.
"""
    block_address = {'IonText': {}}
    if not isinstance(ion_dict, str):
        py_dict = '{{strandId: "{}", sequenceNo:
{}}}'.format(ion_dict['strandId'], ion_dict['sequenceNo'])
        ion_dict = py_dict
    block_address['IonText'] = ion_dict
    return block_address

```

2. verifier.py

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:

```

```
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from array import array
from base64 import b64encode
from functools import reduce
from hashlib import sha256
from random import randrange

from amazon.ion.simpleion import loads

HASH_LENGTH = 32
UPPER_BOUND = 8

def parse_proof(value_holder):
    """
    Parse the Proof object returned by QLDB into an iterator.

    The Proof object returned by QLDB is a dictionary like the following:
    {'IonText': '[[{<hash>}],[{<hash>}]]'}

    :type value_holder: dict
    :param value_holder: A structure containing an Ion string value.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyList`
    :return: A list of hash values.
    """
    value_holder = value_holder.get('IonText')
    proof_list = loads(value_holder)
    return proof_list

def parse_block(value_holder):
    """
    Parse the Block object returned by QLDB and retrieve block hash.

    :type value_holder: dict
    :param value_holder: A structure containing an Ion string value.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyBytes`
    :return: The block hash.
    """
    value_holder = value_holder.get('IonText')
    block = loads(value_holder)
    block_hash = block.get('blockHash')
```



```

    return block_hash

def flip_random_bit(original):
    """
    Flip a single random bit in the given hash value.
    This method is used to demonstrate QLDB's verification features.

    :type original: bytes
    :param original: The hash value to alter.

    :rtype: bytes
    :return: The altered hash with a single random bit changed.
    """
    assert len(original) != 0, 'Invalid bytes.'

    altered_position = randrange(len(original))
    bit_shift = randrange(UPPER_BOUND)
    altered_hash = bytearray(original).copy()

    altered_hash[altered_position] = altered_hash[altered_position] ^ (1 <<
bit_shift)
    return bytes(altered_hash)

def compare_hash_values(hash1, hash2):
    """
    Compare two hash values by converting them into byte arrays, assuming they
    are little endian.

    :type hash1: bytes
    :param hash1: The hash value to compare.

    :type hash2: bytes
    :param hash2: The hash value to compare.

    :rtype: int
    :return: Zero if the hash values are equal, otherwise return the difference
of the first pair of non-matching bytes.
    """
    assert len(hash1) == HASH_LENGTH
    assert len(hash2) == HASH_LENGTH

    hash_array1 = array('b', hash1)

```

```

hash_array2 = array('b', hash2)

for i in range(len(hash_array1) - 1, -1, -1):
    difference = hash_array1[i] - hash_array2[i]
    if difference != 0:
        return difference
return 0

def join_hash_pairwise(hash1, hash2):
    """
    Take two hash values, sort them, concatenate them, and generate a new hash
    value from the concatenated values.

    :type hash1: bytes
    :param hash1: Hash value to concatenate.

    :type hash2: bytes
    :param hash2: Hash value to concatenate.

    :rtype: bytes
    :return: The new hash value generated from concatenated hash values.
    """
    if len(hash1) == 0:
        return hash2
    if len(hash2) == 0:
        return hash1

    concatenated = hash1 + hash2 if compare_hash_values(hash1, hash2) < 0 else
hash2 + hash1
    new_hash_lib = sha256()
    new_hash_lib.update(concatenated)
    new_digest = new_hash_lib.digest()
    return new_digest

def calculate_root_hash_from_internal_hashes(internal_hashes, leaf_hash):
    """
    Combine the internal hashes and the leaf hash until only one root hash
    remains.

    :type internal_hashes: map
    :param internal_hashes: An iterable over a list of hash values.

```

```

        :type leaf_hash: bytes
        :param leaf_hash: The revision hash to pair with the first hash in the Proof
        hashes list.

        :rtype: bytes
        :return: The root hash constructed by combining internal hashes.
        """
        root_hash = reduce(join_hash_pairwise, internal_hashes, leaf_hash)
        return root_hash

def build_candidate_digest(proof, leaf_hash):
    """
    Build the candidate digest representing the entire ledger from the Proof
    hashes.

    :type proof: dict
    :param proof: The Proof object.

    :type leaf_hash: bytes
    :param leaf_hash: The revision hash to pair with the first hash in the Proof
    hashes list.

    :rtype: bytes
    :return: The calculated root hash.
    """
    parsed_proof = parse_proof(proof)
    root_hash = calculate_root_hash_from_internal_hashes(parsed_proof, leaf_hash)
    return root_hash

def verify_document(document_hash, digest, proof):
    """
    Verify document revision against the provided digest.

    :type document_hash: bytes
    :param document_hash: The SHA-256 value representing the document revision to
    be verified.

    :type digest: bytes
    :param digest: The SHA-256 hash value representing the ledger digest.

    :type proof: dict

```

```

        :param proof: The Proof object retrieved
        from :func:`pyqldb.samples.get_revision.get_revision`.

        :rtype: bool
        :return: If the document revision verify against the ledger digest.
        """
        candidate_digest = build_candidate_digest(proof, document_hash)
        return digest == candidate_digest

def to_base_64(input):
    """
    Encode input in base64.

    :type input: bytes
    :param input: Input to be encoded.

    :rtype: string
    :return: Return input that has been encoded in base64.
    """
    encoded_value = b64encode(input)
    return str(encoded_value, 'UTF-8')

```

3. qlldb_string_utils.py

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT

```

```

# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
from amazon.ion.simpleion import dumps, loads

def value_holder_to_string(value_holder):
    """
    Returns the string representation of a given `value_holder`.

    :type value_holder: dict
    :param value_holder: The `value_holder` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `value_holder`.
    """
    ret_val = dumps(loads(value_holder), binary=False, indent=' ',
omit_version_marker=True)
    val = '{{ IonText: {}}}'.format(ret_val)
    return val

def block_response_to_string(block_response):
    """
    Returns the string representation of a given `block_response`.

    :type block_response: dict
    :param block_response: The `block_response` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `block_response`.
    """
    string = ''
    if block_response.get('Block', {}).get('IonText') is not None:
        string += 'Block: ' + value_holder_to_string(block_response['Block']
['IonText']) + ', '

    if block_response.get('Proof', {}).get('IonText') is not None:
        string += 'Proof: ' + value_holder_to_string(block_response['Proof']
['IonText'])

    return '{' + string + '}'

```

```
def digest_response_to_string(digest_response):
    """
    Returns the string representation of a given `digest_response`.

    :type digest_response: dict
    :param digest_response: The `digest_response` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `digest_response`.
    """
    string = ''
    if digest_response.get('Digest') is not None:
        string += 'Digest: ' + str(digest_response['Digest']) + ', '

    if digest_response.get('DigestTipAddress', {}).get('IonText') is not None:
        string += 'DigestTipAddress: ' +
        value_holder_to_string(digest_response['DigestTipAddress']['IonText'])

    return '{' + string + '}'
```

2. Use two .py programs (get_digest.py and get_revision.py) to perform the following steps:
 - Request a new digest from the vehicle-registration ledger.
 - Request a proof for each revision of the document with VIN 1N4AL11D75C109151 from the VehicleRegistration table.
 - Verify the revisions using the returned digest and proof by recalculating the digest.

The get_digest.py program contains the following code.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
```

```

# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from boto3 import client

from pyqldb.samples.constants import Constants
from pyqldb.samples.qldb.qldb_string_utils import digest_response_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

def get_digest_result(name):
    """
    Get the digest of a ledger's journal.

    :type name: str
    :param name: Name of the ledger to operate on.

    :rtype: dict
    :return: The digest in a 256-bit hash value and a block address.
    """
    logger.info("Let's get the current digest of the ledger named {}".format(name))
    result = qldb_client.get_digest(Name=name)
    logger.info('Success. LedgerDigest:
    {}'.format(digest_response_to_string(result)))
    return result

def main(ledger_name=Constants.LEDGER_NAME):
    """

```

```
This is an example for retrieving the digest of a particular ledger.
"""
try:
    get_digest_result(ledger_name)
except Exception as e:
    logger.exception('Unable to get a ledger digest!')
    raise e

if __name__ == '__main__':
    main()
```

Note

Use the `get_digest_result` function to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request.

The `get_revision.py` program contains the following code.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
# and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
```



```

# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from amazon.ion.simpleion import loads
from boto3 import client

from pyqldb.samples.constants import Constants
from pyqldb.samples.get_digest import get_digest_result
from pyqldb.samples.model.sample_data import SampleData, convert_object_to_ion
from pyqldb.samples.qldb.block_address import block_address_to_dictionary
from pyqldb.samples.verifier import verify_document, flip_random_bit, to_base_64
from pyqldb.samples.connect_to_ledger import create_qldb_driver
from pyqldb.samples.qldb.qldb_string_utils import value_holder_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

def get_revision(ledger_name, document_id, block_address, digest_tip_address):
    """
    Get the revision data object for a specified document ID and block address.
    Also returns a proof of the specified revision for verification.

    :type ledger_name: str
    :param ledger_name: Name of the ledger containing the document to query.

    :type document_id: str
    :param document_id: Unique ID for the document to be verified, contained in
    the committed view of the document.

    :type block_address: dict
    :param block_address: The location of the block to request.

    :type digest_tip_address: dict
    :param digest_tip_address: The latest block location covered by the digest.

    :rtype: dict

```

```

        :return: The response of the request.
        """
        result = qlldb_client.get_revision(Name=ledger_name,
                                           BlockAddress=block_address, DocumentId=document_id,
                                           DigestTipAddress=digest_tip_address)

        return result

def lookup_registration_for_vin(driver, vin):
    """
    Query revision history for a particular vehicle for verification.

    :type driver: :py:class:`pyqlldb.driver.qlldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
    with.

    :rtype: :py:class:`pyqlldb.cursor.buffered_cursor.BufferedCursor`
    :return: Cursor on the result set of the statement query.
    """
    logger.info("Querying the 'VehicleRegistration' table for VIN:
    {}".format(vin))
    query = 'SELECT * FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?'
    return driver.execute_lambda(lambda txn: txn.execute_statement(query,
    convert_object_to_ion(vin)))

def verify_registration(driver, ledger_name, vin):
    """
    Verify each version of the registration for the given VIN.

    :type driver: :py:class:`pyqlldb.driver.qlldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type ledger_name: str
    :param ledger_name: The ledger to get digest from.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
    with.

    :raises AssertionError: When verification failed.

```

```

"""
    logger.info("Let's verify the registration with VIN = {}, in ledger =
    {}".format(vin, ledger_name))
    digest = get_digest_result(ledger_name)
    digest_bytes = digest.get('Digest')
    digest_tip_address = digest.get('DigestTipAddress')

    logger.info('Got a ledger digest: digest tip address = {}, digest =
    {}'.format(
        value_holder_to_string(digest_tip_address.get('IonText')),
        to_base_64(digest_bytes)))

    logger.info('Querying the registration with VIN = {} to verify each version
    of the registration...'.format(vin))
    cursor = lookup_registration_for_vin(driver, vin)
    logger.info('Getting a proof for the document.')

    for row in cursor:
        block_address = row.get('blockAddress')
        document_id = row.get('metadata').get('id')

        result = get_revision(ledger_name, document_id,
        block_address_to_dictionary(block_address), digest_tip_address)
        revision = result.get('Revision').get('IonText')
        document_hash = loads(revision).get('hash')

        proof = result.get('Proof')
        logger.info('Got back a proof: {}'.format(proof))

        verified = verify_document(document_hash, digest_bytes, proof)
        if not verified:
            raise AssertionError('Document revision is not verified.')
        else:
            logger.info('Success! The document is verified.')

        altered_document_hash = flip_random_bit(document_hash)
        logger.info("Flipping one bit in the document's hash and assert that the
        document is NOT verified. "
            "The altered document hash is:
        {}".format(to_base_64(altered_document_hash)))
        verified = verify_document(altered_document_hash, digest_bytes, proof)
        if verified:
            raise AssertionError('Expected altered document hash to not be
            verified against digest.')

```

```

        else:
            logger.info('Success! As expected flipping a bit in the document
hash causes verification to fail.')

            logger.info('Finished verifying the registration with VIN = {} in ledger
= {}'.format(vin, ledger_name))

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Verify the integrity of a document revision in a QLDB ledger.
    """
    registration = SampleData.VEHICLE_REGISTRATION[0]
    vin = registration['VIN']
    try:
        with create_qldb_driver(ledger_name) as driver:
            verify_registration(driver, ledger_name, vin)
    except Exception as e:
        logger.exception('Unable to verify revision.')
        raise e

if __name__ == '__main__':
    main()

```

2.x

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
of this
# software and associated documentation files (the "Software"), to deal in the
Software
# without restriction, including without limitation the rights to use, copy,
modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A

```

```

# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from amazon.ion.simpleion import loads
from boto3 import client

from pyqldb.samples.constants import Constants
from pyqldb.samples.get_digest import get_digest_result
from pyqldb.samples.model.sample_data import SampleData, convert_object_to_ion
from pyqldb.samples.qldb.block_address import block_address_to_dictionary
from pyqldb.samples.verifier import verify_document, flip_random_bit, to_base_64
from pyqldb.samples.connect_to_ledger import create_qldb_session
from pyqldb.samples.qldb.qldb_string_utils import value_holder_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

def get_revision(ledger_name, document_id, block_address, digest_tip_address):
    """
    Get the revision data object for a specified document ID and block address.
    Also returns a proof of the specified revision for verification.

    :type ledger_name: str
    :param ledger_name: Name of the ledger containing the document to query.

    :type document_id: str
    :param document_id: Unique ID for the document to be verified, contained in
    the committed view of the document.

    :type block_address: dict
    :param block_address: The location of the block to request.

    :type digest_tip_address: dict
    :param digest_tip_address: The latest block location covered by the digest.

```

```

        :rtype: dict
        :return: The response of the request.
        """
        result = qlldb_client.get_revision(Name=ledger_name,
                                           BlockAddress=block_address, DocumentId=document_id,
                                           DigestTipAddress=digest_tip_address)

        return result

def lookup_registration_for_vin(qlldb_session, vin):
    """
    Query revision history for a particular vehicle for verification.

    :type qlldb_session: :py:class:`pyqlldb.session.qlldb_session.QldbSession`
    :param qlldb_session: An instance of the QldbSession class.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
    with.

    :rtype: :py:class:`pyqlldb.cursor.buffered_cursor.BufferedCursor`
    :return: Cursor on the result set of the statement query.
    """
    logger.info("Querying the 'VehicleRegistration' table for VIN:
    {}".format(vin))
    query = 'SELECT * FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?'
    parameters = [convert_object_to_ion(vin)]
    cursor = qlldb_session.execute_statement(query, parameters)
    return cursor

def verify_registration(qlldb_session, ledger_name, vin):
    """
    Verify each version of the registration for the given VIN.

    :type qlldb_session: :py:class:`pyqlldb.session.qlldb_session.QldbSession`
    :param qlldb_session: An instance of the QldbSession class.

    :type ledger_name: str
    :param ledger_name: The ledger to get digest from.

    :type vin: str

```

```

        :param vin: VIN to query the revision history of a specific registration
        with.

        :raises AssertionError: When verification failed.
        """
        logger.info("Let's verify the registration with VIN = {}, in ledger =
        {}".format(vin, ledger_name))
        digest = get_digest_result(ledger_name)
        digest_bytes = digest.get('Digest')
        digest_tip_address = digest.get('DigestTipAddress')

        logger.info('Got a ledger digest: digest tip address = {}, digest =
        {}'.format(
            value_holder_to_string(digest_tip_address.get('IonText')),
            to_base_64(digest_bytes)))

        logger.info('Querying the registration with VIN = {} to verify each version
        of the registration...'.format(vin))
        cursor = lookup_registration_for_vin(qlldb_session, vin)
        logger.info('Getting a proof for the document.')

        for row in cursor:
            block_address = row.get('blockAddress')
            document_id = row.get('metadata').get('id')

            result = get_revision(ledger_name, document_id,
            block_address_to_dictionary(block_address), digest_tip_address)
            revision = result.get('Revision').get('IonText')
            document_hash = loads(revision).get('hash')

            proof = result.get('Proof')
            logger.info('Got back a proof: {}'.format(proof))

            verified = verify_document(document_hash, digest_bytes, proof)
            if not verified:
                raise AssertionError('Document revision is not verified.')
            else:
                logger.info('Success! The document is verified.')

            altered_document_hash = flip_random_bit(document_hash)
            logger.info("Flipping one bit in the document's hash and assert that the
            document is NOT verified. "
                "The altered document hash is:
            {}".format(to_base_64(altered_document_hash)))

```

```
        verified = verify_document(altered_document_hash, digest_bytes, proof)
        if verified:
            raise AssertionError('Expected altered document hash to not be
verified against digest.')
        else:
            logger.info('Success! As expected flipping a bit in the document
hash causes verification to fail.')

            logger.info('Finished verifying the registration with VIN = {} in ledger
= {}'.format(vin, ledger_name))

if __name__ == '__main__':
    """
    Verify the integrity of a document revision in a QLDB ledger.
    """
    registration = SampleData.VEHICLE_REGISTRATION[0]
    vin = registration['VIN']
    try:
        with create_qldb_session() as session:
            verify_registration(session, Constants.LEDGER_NAME, vin)
    except Exception:
        logger.exception('Unable to verify revision.')
```

Note

After the `get_revision` function returns a proof for the specified document revision, this program uses a client-side API to verify that revision.

3. To run the program, enter the following command.

```
python get_revision.py
```

If you no longer need to use the vehicle-registration ledger, proceed to [Step 8 \(optional\): Clean up resources](#).

Step 8 (optional): Clean up resources

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can continue using the vehicle-registration ledger. However, if you no longer need it, you should delete it.

To delete the ledger

1. Use the following program (`delete_ledger.py`) to delete your vehicle-registration ledger and all of its contents.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO
```

```
from time import sleep

from boto3 import client

from pyqldb.samples.constants import Constants
from pyqldb.samples.describe_ledger import describe_ledger

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

LEDGER_DELETION_POLL_PERIOD_SEC = 20

def delete_ledger(ledger_name):
    """
    Send a request to QLDB to delete the specified ledger.

    :type ledger_name: str
    :param ledger_name: Name for the ledger to be deleted.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info('Attempting to delete the ledger with name:
    {}'.format(ledger_name))
    result = qldb_client.delete_ledger(Name=ledger_name)
    logger.info('Success.')
    return result

def wait_for_deleted(ledger_name):
    """
    Wait for the ledger to be deleted.

    :type ledger_name: str
    :param ledger_name: The ledger to check on.
    """
    logger.info('Waiting for the ledger to be deleted...')
    while True:
        try:
            describe_ledger(ledger_name)
            logger.info('The ledger is still being deleted. Please wait...')
            sleep(LEDGER_DELETION_POLL_PERIOD_SEC)
```

```

        except qlldb_client.exceptions.ResourceNotFoundException:
            logger.info('Success. The ledger is deleted.')
            break

def set_deletion_protection(ledger_name, deletion_protection):
    """
    Update an existing ledger's deletion protection.

    :type ledger_name: str
    :param ledger_name: Name of the ledger to update.

    :type deletion_protection: bool
    :param deletion_protection: Enable or disable the deletion protection.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info("Let's set deletion protection to {} for the ledger with name
    {}.".format(deletion_protection,

                ledger_name))
    result = qlldb_client.update_ledger(Name=ledger_name,
    DeletionProtection=deletion_protection)
    logger.info('Success. Ledger updated: {}'.format(result))

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Delete a ledger.
    """
    try:
        set_deletion_protection(ledger_name, False)
        delete_ledger(ledger_name)
        wait_for_deleted(ledger_name)
    except Exception as e:
        logger.exception('Unable to delete the ledger.')
        raise e

if __name__ == '__main__':
    main()

```

Note

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API.

The `delete_ledger.py` file also has a dependency on the following program (`describe_ledger.py`).

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy,
# modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
# to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
# COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from boto3 import client

from pyqldbconstants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)
```

```
qldb_client = client('qldb')

def describe_ledger(ledger_name):
    """
    Describe a ledger.

    :type ledger_name: str
    :param ledger_name: Name of the ledger to describe.
    """
    logger.info('describe ledger with name: {}'.format(ledger_name))
    result = qldb_client.describe_ledger(Name=ledger_name)
    result.pop('ResponseMetadata')
    logger.info('Success. Ledger description: {}'.format(result))
    return result

def main(ledger_name=Constants.LEDGER_NAME):
    """
    Describe a QLDB ledger.
    """
    try:
        describe_ledger(ledger_name)
    except Exception as e:
        logger.exception('Unable to describe a ledger.')
        raise e

if __name__ == '__main__':
    main()
```

2. To run the program, enter the following command.

```
python delete_ledger.py
```

Working with Amazon Ion data types in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB stores its data in the Amazon Ion format. To work with data in QLDB, you must use an [Ion library](#) as a dependency for a supported programming language.

In this section, learn how to convert data from native types to their Ion equivalents, and the other way around. This reference guide shows code examples that use the QLDB driver to process Ion data in a QLDB ledger. It includes code examples for Java, .NET (C#), Go, Node.js (TypeScript), and Python.

Topics

- [Prerequisites](#)
- [Bool](#)
- [Int](#)
- [Float](#)
- [Decimal](#)
- [Timestamp](#)
- [String](#)
- [Blob](#)
- [List](#)
- [Struct](#)
- [Null values and dynamic types](#)
- [Down-converting to JSON](#)

Prerequisites

The following code examples assume that you have a QLDB driver instance that is connected to an active ledger with a table named `ExampleTable`. The table contains a single existing document that has the following eight fields:

- `ExampleBool`
- `ExampleInt`
- `ExampleFloat`
- `ExampleDecimal`
- `ExampleTimestamp`
- `ExampleString`
- `ExampleBlob`
- `ExampleList`

Note

For the purposes of this reference, assume that the type stored in each field matches its name. In practice, QLDB doesn't enforce schema or data type definitions for document fields.

Bool

The following code examples show how to process the Ion boolean type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

boolean exampleBoolean = driver.execute((txn) -> {
    // Transforming a Java boolean to Ion
    boolean aBoolean = true;
    IonValue ionBool = ionSystem.newBool(aBoolean);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleBool = ?", ionBool);
});
```

```

// Fetching from QLDB
Result result = txn.execute("SELECT VALUE ExampleBool from ExampleTable");
// Assume there is only one document in ExampleTable
for (IonValue ionValue : result)
{
    // Transforming Ion to a Java boolean
    // Cast IonValue to IonBool first
    aBoolean = ((IonBool)ionValue).booleanValue();
}

// exampleBoolean is now the value fetched from QLDB
return aBoolean;
});

```

.NET

```

using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# bool to Ion.
bool nativeBool = true;
IIonValue ionBool = valueFactory.NewBool(nativeBool);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleBool = ?", ionBool);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleBool from ExampleTable");
});

bool? retrievedBool = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# bool.
    retrievedBool = ionValue.BoolValue;
}

```


Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```
exampleBool, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aBool := true

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleBool = ?", aBool)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleBool FROM ExampleTable")
    if err != nil {
        return nil, err
    }

    // Assume there is only one document in ExampleTable
    if result.Next(txn) {
        var decodedResult bool
        err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
        if err != nil {
            return nil, err
        }

        // exampleBool is now the value fetched from QLDB
        return decodedResult, nil
    }
    return nil, result.Err()
})
```

Node.js

```
async function queryIonBoolean(driver: QldbDriver): Promise<boolean> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
```

```

        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleBool = ?", true);

        // Fetching from QLDB
        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
ExampleBool FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Transforming Ion to a TypeScript Boolean
        const boolValue: boolean = ionValue.booleanValue();
        return boolValue;
    })
};
}

```

Python

```

def update_and_query_ion_bool(txn):
    # QLDB can take in a Python bool
    a_bool = True

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleBool = ?", a_bool)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleBool FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python bool is a child class of Python bool
        a_bool = ion_value

    # example_bool is now the value fetched from QLDB
    return a_bool

example_bool = driver.execute_lambda(lambda txn: update_and_query_ion_bool(txn))

```

Int

The following code examples show how to process the Ion integer type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

int exampleInt = driver.execute((txn) -> {
    // Transforming a Java int to Ion
    int aInt = 256;
    IonValue ionInt = ionSystem.newInt(aInt);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleInt = ?", ionInt);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleInt from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java int
        // Cast IonValue to IonInt first
        aInt = ((IonInt)ionValue).intValue();
    }

    // exampleInt is now the value fetched from QLDB
    return aInt;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# int to Ion.
int nativeInt = 256;
IIonValue ionInt = valueFactory.NewInt(nativeInt);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleInt = ?", ionInt);
});
```

```
// Fetching from QLDB.
return await txn.Execute("SELECT VALUE ExampleInt from ExampleTable");
});

int? retrievedInt = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# int.
    retrievedInt = ionValue.IntValue;
}
```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```
exampleInt, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aInt := 256

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleInt = ?", aInt)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleInt FROM ExampleTable")
    if err != nil {
        return nil, err
    }

    // Assume there is only one document in ExampleTable
    if result.Next(txn) {
        var decodedResult int
        err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
        if err != nil {
```

```

    return nil, err
}

// exampleInt is now the value fetched from QLDB
return decodedResult, nil
}
return nil, result.Err()
})

```

Node.js

```

async function queryIonInt(driver: QldbDriver): Promise<number> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleInt = ?", 256);

        // Fetching from QLDB
        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
ExampleInt FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Transforming Ion to a TypeScript Number
        const intValue: number = ionValue.numberValue();
        return intValue;
    }));
}

```

Python

```

def update_and_query_ion_int(txn):
    # QLDB can take in a Python int
    a_int = 256

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleInt = ?", a_int)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleInt FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:

```

```
# Ion Python int is a child class of Python int
a_int = ion_value

# example_int is now the value fetched from QLDB
return a_int

example_int = driver.execute_lambda(lambda txn: update_and_query_ion_int(txn))
```

Float

The following code examples show how to process the Ion float type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

float exampleFloat = driver.execute((txn) -> {
    // Transforming a Java float to Ion
    float aFloat = 256;
    IonValue ionFloat = ionSystem.newFloat(aFloat);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleFloat from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java float
        // Cast IonValue to IonFloat first
        aFloat = ((IonFloat)ionValue).floatValue();
    }

    // exampleFloat is now the value fetched from QLDB
    return aFloat;
});
```

.NET

Using C# float

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# float to Ion.
float nativeFloat = 256;
IIonValue ionFloat = valueFactory.NewFloat(nativeFloat);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleFloat from ExampleTable");
});

float? retrievedFloat = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# float. We cast ionValue.DoubleValue to a float
    // but be cautious, this is a down-cast and can lose precision.
    retrievedFloat = (float)ionValue.DoubleValue;
}
```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Using C# double

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...
```

```

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# double to Ion.
double nativeDouble = 256;
IIonValue ionFloat = valueFactory.NewFloat(nativeDouble);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleFloat from ExampleTable");
});

double? retrievedDouble = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# double.
    retrievedDouble = ionValue.DoubleValue;
}

```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```

exampleFloat, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aFloat := float32(256)

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", aFloat)
    if err != nil {
        return nil, err
    }
}

```



```

// Fetching from QLDB
result, err := txn.Execute("SELECT VALUE ExampleFloat FROM ExampleTable")
if err != nil {
    return nil, err
}

// Assume there is only one document in ExampleTable
if result.Next(txn) {
    // float64 would work as well
    var decodedResult float32
    err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
    if err != nil {
        return nil, err
    }

    // exampleFloat is now the value fetched from QLDB
    return decodedResult, nil
}
return nil, result.Err()
})

```

Node.js

```

async function queryIonFloat(driver: QldbDriver): Promise<number> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleFloat = ?", 25.6);

        // Fetching from QLDB
        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
ExampleFloat FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Transforming Ion to a TypeScript Number
        const floatValue: number = ionValue.numberValue();
        return floatValue;
    }));
};
}

```

Python

```
def update_and_query_ion_float(txn):
    # QLDB can take in a Python float
    a_float = float(256)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleFloat = ?", a_float)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleFloat FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python float is a child class of Python float
        a_float = ion_value

    # example_float is now the value fetched from QLDB
    return a_float

example_float = driver.execute_lambda(lambda txn: update_and_query_ion_float(txn))
```

Decimal

The following code examples show how to process the Ion decimal type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

double exampleDouble = driver.execute((txn) -> {
    // Transforming a Java double to Ion
    double aDouble = 256;
    IonValue ionDecimal = ionSystem.newDecimal(aDouble);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleDecimal = ?", ionDecimal);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleDecimal from ExampleTable");
```

```

    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java double
        // Cast IonValue to IonDecimal first
        aDouble = ((IonDecimal)ionValue).doubleValue();
    }

    // exampleDouble is now the value fetched from QLDB
    return aDouble;
});

```

.NET

```

using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# decimal to Ion.
decimal nativeDecimal = 256.8723m;
IIonValue ionDecimal = valueFactory.NewDecimal(nativeDecimal);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleDecimal = ?", ionDecimal);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleDecimal from ExampleTable");
});

decimal? retrievedDecimal = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# decimal.
    retrievedDecimal = ionValue.DecimalValue;
}

```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```
exampleDecimal, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aDecimal, err := ion.ParseDecimal("256")
    if err != nil {
        return nil, err
    }

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleDecimal = ?", aDecimal)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleDecimal FROM ExampleTable")
    if err != nil {
        return nil, err
    }

    // Assume there is only one document in ExampleTable
    if result.Next(txn) {
        var decodedResult ion.Decimal
        err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
        if err != nil {
            return nil, err
        }

        // exampleDecimal is now the value fetched from QLDB
        return decodedResult, nil
    }
    return nil, result.Err()
})
```

Node.js

```

async function queryIonDecimal(driver: QldbDriver): Promise<Decimal> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        // Creating a Decimal value. Decimal is an Ion Type with high precision
        let ionDecimal: Decimal = dom.load("2.5d-6").decimalValue();
        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleDecimal = ?",
            ionDecimal);

        // Fetching from QLDB
        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
            ExampleDecimal FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Get the Ion Decimal
        ionDecimal = ionValue.decimalValue();
        return ionDecimal;
    }));
}

```

Note

You can also use `ionValue.numberValue()` to transform an Ion decimal to a JavaScript number. Using `ionValue.numberValue()` has better performance, but it's less precise.

Python

```

def update_and_query_ion_decimal(txn):
    # QLDB can take in a Python decimal
    a_decimal = Decimal(256)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleDecimal = ?", a_decimal)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleDecimal FROM ExampleTable")

```

```
# Assume there is only one document in ExampleTable
for ion_value in cursor:
    # Ion Python decimal is a child class of Python decimal
    a_decimal = ion_value

# example_decimal is now the value fetched from QLDB
return a_decimal

example_decimal = driver.execute_lambda(lambda txn:
    update_and_query_ion_decimal(txn))
```

Timestamp

The following code examples show how to process the Ion timestamp type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

Date exampleDate = driver.execute((txn) -> {
    // Transforming a Java Date to Ion
    Date aDate = new Date();
    IonValue ionTimestamp = ionSystem.newUtcTimestamp(aDate);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleTimestamp = ?", ionTimestamp);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleTimestamp from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java Date
        // Cast IonValue to IonTimestamp first
        aDate = ((IonTimestamp)ionValue).dateValue();
    }

    // exampleDate is now the value fetched from QLDB
    return aDate;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using Amazon.IonDotnet;
...

IValueFactory valueFactory = new ValueFactory();

// Convert C# native DateTime to Ion.
DateTime nativeDateTime = DateTime.Now;

// First convert it to a timestamp object from Ion.
Timestamp timestamp = new Timestamp(nativeDateTime);
// Then convert to Ion timestamp.
IIonValue ionTimestamp = valueFactory.NewTimestamp(timestamp);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleTimestamp = ?", ionTimestamp);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleTimestamp from ExampleTable");
});

DateTime? retrievedDateTime = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# DateTime.
    retrievedDateTime = ionValue.TimestampValue.DateTimeValue;
}
```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```

exampleTimestamp, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aTimestamp := time.Date(2006, time.May, 20, 12, 30, 0, 0, time.UTC)
    if err != nil {
        return nil, err
    }

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleTimestamp = ?", aTimestamp)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleTimestamp FROM ExampleTable")
    if err != nil {
        return nil, err
    }

    // Assume there is only one document in ExampleTable
    if result.Next(txn) {
        var decodedResult ion.Timestamp
        err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
        if err != nil {
            return nil, err
        }

        // exampleTimestamp is now the value fetched from QLDB
        return decodedResult.GetDateTime(), nil
    }
    return nil, result.Err()
})

```

Node.js

```

async function queryIonTimestamp(driver: QldbDriver): Promise<Date> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        let exampleDateTime: Date = new Date(Date.now());
        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleTimestamp = ?",
exampleDateTime);

```



```

        // Fetching from QLDB
        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
ExampleTimestamp FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Transforming Ion to a TypeScript Date
        exampleDateTime = ionValue.timestampValue().getDate();
        return exampleDateTime;
    })
};
}

```

Python

```

def update_and_query_ion_timestamp(txn):
    # QLDB can take in a Python timestamp
    a_datetime = datetime.now()

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleTimestamp = ?",
a_datetime)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleTimestamp FROM
ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python timestamp is a child class of Python datetime
        a_timestamp = ion_value

    # example_timestamp is now the value fetched from QLDB
    return a_timestamp

example_timestamp = driver.execute_lambda(lambda txn:
update_and_query_ion_timestamp(txn))

```

String

The following code examples show how to process the Ion string type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

String exampleString = driver.execute((txn) -> {
    // Transforming a Java String to Ion
    String aString = "Hello world!";
    IonValue ionString = ionSystem.newString(aString);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleString = ?", ionString);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleString from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java String
        // Cast IonValue to IonString first
        aString = ((IonString)ionValue).stringValue();
    }

    // exampleString is now the value fetched from QLDB
    return aString;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Convert C# string to Ion.
String nativeString = "Hello world!";
IIonValue ionString = valueFactory.NewString(nativeString);

IAsyncResult selectResult = await driver.Execute(async txn =>
```

```

{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleString = ?", ionString);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleString from ExampleTable");
});

String retrievedString = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# string.
    retrievedString = ionValue.StringValue;
}

```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```

exampleString, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aString := "Hello World!"

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleString = ?", aString)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleString FROM ExampleTable")
    if err != nil {
        return nil, err
    }

    // Assume there is only one document in ExampleTable

```

```

if result.Next(txn) {
    var decodedResult string
    err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
    if err != nil {
        return nil, err
    }

    // exampleString is now the value fetched from QLDB
    return decodedResult, nil
}
return nil, result.Err()
})

```

Node.js

```

async function queryIonString(driver: QldbDriver): Promise<string> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleString = ?", "Hello
World!");

        // Fetching from QLDB
        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
ExampleString FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Transforming Ion to a TypeScript String
        const stringValue: string = ionValue.stringValue();
        return stringValue;
    }));
}

```

Python

```

def update_and_query_ion_string(txn):
    # QLDB can take in a Python string
    a_string = "Hello world!"

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleString = ?", a_string)

```

```
# Fetching from QLDB
cursor = txn.execute_statement("SELECT VALUE ExampleString FROM ExampleTable")

# Assume there is only one document in ExampleTable
for ion_value in cursor:
    # Ion Python string is a child class of Python string
    a_string = ion_value

# example_string is now the value fetched from QLDB
return a_string

example_string = driver.execute_lambda(lambda txn: update_and_query_ion_string(txn))
```

Blob

The following code examples show how to process the Ion blob type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

byte[] exampleBytes = driver.execute((txn) -> {
    // Transforming a Java byte array to Ion
    // Transform any arbitrary data to a byte array to store in QLDB
    String aString = "Hello world!";
    byte[] aByteArray = aString.getBytes();
    IonValue ionBlob = ionSystem.newBlob(aByteArray);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleBlob = ?", ionBlob);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleBlob from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java byte array
        // Cast IonValue to IonBlob first
        aByteArray = ((IonBlob)ionValue).getBytes();
    }
}
```

```
    // exampleBytes is now the value fetched from QLDB
    return aByteArray;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using System.Text;
...

IValueFactory valueFactory = new ValueFactory();

// Transform any arbitrary data to a byte array to store in QLDB.
string nativeString = "Hello world!";
byte[] nativeByteArray = Encoding.UTF8.GetBytes(nativeString);
// Transforming a C# byte array to Ion.
IIonValue ionBlob = valueFactory.NewBlob(nativeByteArray);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleBlob = ?", ionBlob);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleBlob from ExampleTable");
});

byte[] retrievedByteArray = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# byte array.
    retrievedByteArray = ionValue.Bytes().ToArray();
}
```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```

exampleBlob, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aBlob := []byte("Hello World!")

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleBlob = ?", aBlob)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleBlob FROM ExampleTable")
    if err != nil {
        return nil, err
    }

    // Assume there is only one document in ExampleTable
    if result.Next(txn) {
        var decodedResult []byte
        err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
        if err != nil {
            return nil, err
        }

        // exampleBlob is now the value fetched from QLDB
        return decodedResult, nil
    }
    return nil, result.Err()
})

```

Node.js

```

async function queryIonBlob(driver: QldbDriver): Promise<Uint8Array> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        const enc = new TextEncoder();
        let blobValue: Uint8Array = enc.encode("Hello World!");
        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleBlob = ?", blobValue);

        // Fetching from QLDB

```

```

        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
ExampleBlob FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Transforming Ion to TypeScript Uint8Array
        blobValue = ionValue.uInt8ArrayValue();
        return blobValue;
    })
};
}

```

Python

```

def update_and_query_ion_blob(txn):
    # QLDB can take in a Python byte array
    a_string = "Hello world!"
    a_byte_array = str.encode(a_string)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleBlob = ?", a_byte_array)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleBlob FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python blob is a child class of Python byte array
        a_blob = ion_value

    # example_blob is now the value fetched from QLDB
    return a_blob

example_blob = driver.execute_lambda(lambda txn: update_and_query_ion_blob(txn))

```

List

The following code examples show how to process the Ion list type.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

List<Integer> exampleList = driver.execute((txn) -> {
    // Transforming a Java List to Ion
    List<Integer> aList = new ArrayList<>();
    // Add 5 Integers to the List for the sake of example
    for (int i = 0; i < 5; i++) {
        aList.add(i);
    }
    // Create an empty Ion List
    IonList ionList = ionSystem.newEmptyList();
    // Add the 5 Integers to the Ion List
    for (Integer i : aList) {
        // Convert each Integer to Ion ints first to add it to the Ion List
        ionList.add(ionSystem.newInt(i));
    }

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleList = ?", (IonValue) ionList);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleList from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Iterate through the Ion List to map it to a Java List
        List<Integer> intList = new ArrayList<>();
        for (IonValue ionInt : (IonList)ionValue) {
            // Convert the 5 Ion ints to Java Integers
            intList.add(((IonInt)ionInt).intValue());
        }
        aList = intList;
    }

    // exampleList is now the value fetched from QLDB
    return aList;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using System.Collections.Generic;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# list to Ion.
IIonValue ionList = valueFactory.NewEmptyList();
foreach (int i in new List<int> {0, 1, 2, 3, 4})
{
    // Convert to Ion int and add to Ion list.
    ionList.Add(valueFactory.NewInt(i));
}

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleList = ?", ionList);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleList from ExampleTable");
});

List<int> retrievedList = new List<int>();
await foreach (IIonValue ionValue in selectResult)
{
    // Iterate through the Ion List to map it to a C# list.
    foreach (IIonValue ionInt in ionValue)
    {
        retrievedList.Add(ionInt.IntValue);
    }
}
```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```

exampleList, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aList := []int{1, 2, 3, 4, 5}

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleList = ?", aList)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleList FROM ExampleTable")
    if err != nil {
        return nil, err
    }

    // Assume there is only one document in ExampleTable
    if result.Next(txn) {
        var decodedResult []int
        err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
        if err != nil {
            return nil, err
        }

        // exampleList is now the value fetched from QLDB
        return decodedResult, nil
    }
    return nil, result.Err()
})

```

Node.js

```

async function queryIonList(driver: QldbDriver): Promise<number[]> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        let listOfNumbers: number[] = [1, 2, 3, 4, 5];
        // Updating QLDB
        await txn.execute("UPDATE ExampleTable SET ExampleList = ?",
listOfNumbers);

        // Fetching from QLDB

```

```

        const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
ExampleList FROM ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // Get Ion List
        const ionList: dom.Value[] = ionValue.elements();
        // Iterate through the Ion List to map it to a JavaScript Array
        let intList: number[] = [];
        ionList.forEach(item => {
            // Transforming Ion to a TypeScript Number
            const intValue: number = item.numberValue();
            intList.push(intValue);
        });
        listOfNumbers = intList;
        return listOfNumbers;
    })
};
}

```

Python

```

def update_and_query_ion_list(txn):
    # QLDB can take in a Python list
    a_list = list()
    for i in range(0, 5):
        a_list.append(i)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleList = ?", a_list)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleList FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python blob is a child class of Python list
        a_list = ion_value

    # example_list is now the value fetched from QLDB
    return a_list

```

```
example_list = driver.execute_lambda(lambda txn: update_and_query_ion_list(txn))
```

Struct

In QLDB, struct data types are particularly unique from other Ion types. Top-level documents that you insert into a table must be of the struct type. A document field can also store a nested struct.

For simplicity, the following examples define a document that has the `ExampleString` and `ExampleInt` fields only.

Java

```
class ExampleStruct {
    public String exampleString;
    public int exampleInt;

    public ExampleStruct(String exampleString, int exampleInt) {
        this.exampleString = exampleString;
        this.exampleInt = exampleInt;
    }
}

// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

ExampleStruct examplePojo = driver.execute((txn) -> {
    // Transforming a POJO to Ion
    ExampleStruct aPojo = new ExampleStruct("Hello world!", 256);
    // Create an empty Ion struct
    IonStruct ionStruct = ionSystem.newEmptyStruct();
    // Map the fields of the POJO to Ion values and put them in the Ion struct
    ionStruct.add("ExampleString", ionSystem.newString(aPojo.exampleString));
    ionStruct.add("ExampleInt", ionSystem.newInt(aPojo.exampleInt));

    // Insertion into QLDB
    txn.execute("INSERT INTO ExampleTable ?", ionStruct);

    // Fetching from QLDB
    Result result = txn.execute("SELECT * from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
```

```

{
    // Map the fields of the Ion struct to Java values and construct a new POJO
    ionStruct = (IonStruct)ionValue;
    IonString exampleString = (IonString)ionStruct.get("ExampleString");
    IonInt exampleInt = (IonInt)ionStruct.get("ExampleInt");
    aPojo = new ExampleStruct(exampleString.stringValue(),
exampleInt.intValue());
}

// examplePojo is now the document fetched from QLDB
return aPojo;
});

```

Alternatively, you can use the [Jackson library](#) to map data types to and from Ion. The library supports the other Ion data types, but this example focuses on the struct type.

```

class ExampleStruct {
    public String exampleString;
    public int exampleInt;

    @JsonCreator
    public ExampleStruct(@JsonProperty("ExampleString") String exampleString,
        @JsonProperty("ExampleInt") int exampleInt) {
        this.exampleString = exampleString;
        this.exampleInt = exampleInt;
    }

    @JsonProperty("ExampleString")
    public String getExampleString() {
        return this.exampleString;
    }

    @JsonProperty("ExampleInt")
    public int getExampleInt() {
        return this.exampleInt;
    }
}

// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();
// Instantiate an IonObjectMapper from the Jackson library
IonObjectMapper ionMapper = new IonValueMapper(ionSystem);

```

```

ExampleStruct examplePojo = driver.execute((txn) -> {
    // Transforming a POJO to Ion
    ExampleStruct aPojo = new ExampleStruct("Hello world!", 256);
    IonValue ionStruct;
    try {
        // Use the mapper to convert Java objects into Ion
        ionStruct = ionMapper.writeValueAsIonValue(aPojo);
    } catch (IOException e) {
        // Wrap the exception and throw it for the sake of simplicity in this
example
        throw new RuntimeException(e);
    }

    // Insertion into QLDB
    txn.execute("INSERT INTO ExampleTable ?", ionStruct);

    // Fetching from QLDB
    Result result = txn.execute("SELECT * from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Use the mapper to convert Ion to Java objects
        try {
            aPojo = ionMapper.readValue(ionValue, ExampleStruct.class);
        } catch (IOException e) {
            // Wrap the exception and throw it for the sake of simplicity in this
example
            throw new RuntimeException(e);
        }
    }

    // examplePojo is now the document fetched from QLDB
    return aPojo;
});

```

.NET

Use the [Amazon.QLDB.Driver.Serialization](#) library to map native C# data types to and from Ion. The library supports the other Ion data types, but this example focuses on the struct type.

```

using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;
...

```

```

IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
    .WithLedger("vehicle-registration")
    // Add Serialization library
    .WithSerializer(new ObjectSerializer())
    .Build();

// Creating a C# POCO.
ExampleStruct exampleStruct = new ExampleStruct
{
    ExampleString = "Hello world!",
    ExampleInt = 256
};

IAsyncResult<ExampleStruct> selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute(txn.Query<Document>("UPDATE ExampleTable SET ExampleStruct
= ?", exampleStruct));

    // Fetching from QLDB.
    return await txn.Execute(txn.Query<ExampleStruct>("SELECT VALUE ExampleStruct
from ExampleTable"));
});

await foreach (ExampleStruct row in selectResult)
{
    Console.WriteLine(row.ExampleString);
    Console.WriteLine(row.ExampleInt);
}

```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Alternatively, you can use the [Amazon.IonDotnet.Builders](#) library to process Ion data types.

```

using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

```



```

IValueFactory valueFactory = new ValueFactory();

// Creating Ion struct.
IIonValue ionStruct = valueFactory.NewEmptyStruct();
ionStruct.SetField("ExampleString", valueFactory.NewString("Hello world!"));
ionStruct.SetField("ExampleInt", valueFactory.NewInt(256));

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleStruct = ?", ionStruct);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleStruct from ExampleTable");
});

string retrievedString = null;
int? retrievedInt = null;

await foreach (IIonValue ionValue in selectResult)
{
    retrievedString = ionValue.GetField("ExampleString").StringValue;
    retrievedInt = ionValue.GetField("ExampleInt").IntValue;
}

```

Go

```

exampleStruct, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
    aStruct := map[string]interface{} {
        "ExampleString": "Hello World!",
        "ExampleInt": 256,
    }

    // Insertion into QLDB
    _, err = txn.Execute("UPDATE ExampleTable SET ExampleStruct = ?", aStruct)
    if err != nil {
        return nil, err
    }

    // Fetching from QLDB
    result, err := txn.Execute("SELECT VALUE ExampleStruct FROM ExampleTable")
    if err != nil {

```

```

    return nil, err
}

// Assume there is only one document in ExampleTable
if result.Next(txn) {
    var decodedResult map[string]interface{}
    err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
    if err != nil {
        return nil, err
    }

    // exampleStruct is now the value fetched from QLDB
    return decodedResult, nil
}
return nil, result.Err()
})

```

Node.js

```

async function queryIonStruct(driver: QldbDriver): Promise<any> {
    let exampleStruct: any = {stringValue: "Hello World!", intValue: 256};
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
        // Inserting into QLDB
        await txn.execute("INSERT INTO ExampleTable ?", exampleStruct);

        // Fetching from QLDB
        const resultList: dom.Value[] = (await txn.execute("SELECT * FROM
ExampleTable")).getResultList();

        // Assume there is only one document in ExampleTable
        const ionValue: dom.Value = resultList[0];
        // We can get all the keys of Ion struct and their associated values
        const ionFieldNames: string[] = ionValue.fieldNames();

        // Getting key and value of Ion struct to TypeScript String and Number
        const nativeStringVal: string =
ionValue.get(ionFieldNames[0]).stringValue();
        const nativeIntVal: number =
ionValue.get(ionFieldNames[1]).numberValue();
        // Alternatively, we can access to Ion struct fields, using their
literal field names:
        // const nativeStringVal = ionValue.get("stringValue").stringValue();
        // const nativeIntVal = ionValue.get("intValue").numberValue();
    }));
}

```

```

        exampleStruct = {[ionFieldNames[0]]: nativeStringVal,
[ionFieldNames[1]]: nativeIntVal};
        return exampleStruct;
    })
);
}

```

Python

```

def update_and_query_ion_struct(txn):
    # QLDB can take in a Python struct
    a_struct = {"ExampleString": "Hello world!", "ExampleInt": 256}

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleStruct = ?", a_struct)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleStruct FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python struct is a child class of Python struct
        a_struct = ion_value

    # example_struct is now the value fetched from QLDB
    return a_struct

example_struct = driver.execute_lambda(lambda txn: update_and_query_ion_struct(txn))

```

Null values and dynamic types

QLDB supports open content and doesn't enforce schema or data type definitions for document fields. You can also store Ion null values in a QLDB document. All of the prior examples assume that each returned data type is known and is not null. The following examples show how to work with Ion when the data type isn't known or is possibly null.

Java

```
// Empty variables
```

```
String exampleString = null;
Integer exampleInt = null;

// Assume ionValue is some queried data from QLDB
IonValue ionValue = null;

// Check the value type and assign it to the variable if it is not null
if (ionValue.getType() == IonType.STRING) {
    if (ionValue.isNullValue()) {
        exampleString = null;
    } else {
        exampleString = ((IonString)ionValue).stringValue();
    }
} else if (ionValue.getType() == IonType.INT) {
    if (ionValue.isNullValue()) {
        exampleInt = null;
    } else {
        exampleInt = ((IonInt)ionValue).intValue();
    }
};

// Creating null values
IonSystem ionSystem = IonSystemBuilder.standard().build();

// A null value still has an Ion type
IonString ionString;
if (exampleString == null) {
    // Specifically a null string
    ionString = ionSystem.newNullString();
} else {
    ionString = ionSystem.newString(exampleString);
}

IonInt ionInt;
if (exampleInt == null) {
    // Specifically a null int
    ionInt = ionSystem.newNullInt();
} else {
    ionInt = ionSystem.newInt(exampleInt);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
// The above null values still have the types 'String' and 'Int'
```

```
IonValue specialNull = ionSystem.newNull();
if (specialNull.getType() == IonType.NULL) {
    // This is true!
}
if (specialNull.isNullValue()) {
    // This is also true!
}
if (specialNull.getType() == IonType.STRING || specialNull.getType() == IonType.INT)
{
    // This is false!
}
```

.NET

```
// Empty variables.
string exampleString = null;
int? exampleInt = null;

// Assume ionValue is some queried data from QLDB.
IIonValue ionValue;

if (ionValue.Type() == IonType.String)
{
    exampleString = ionValue.StringValue;
}
else if (ionValue.Type() == IonType.Int)
{
    if (ionValue.IsNull)
    {
        exampleInt = null;
    }
    else
    {
        exampleInt = ionValue.IntValue;
    }
}
};

// Creating null values.
IValueFactory valueFactory = new ValueFactory();

// A null value still has an Ion type.
IIonValue ionString = valueFactory.NewString(exampleString);
```

```

IIonValue ionInt;
if (exampleInt == null)
{
    // Specifically a null int.
    ionInt = valueFactory.NewNullInt();
}
else
{
    ionInt = valueFactory.NewInt(exampleInt.Value);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
IIonValue specialNull = valueFactory.NewNull();
if (specialNull.Type() == IonType.Null) {
    // This is true!
}
if (specialNull.IsNull) {
    // This is also true!
}

```

Go

Marshalling limitations

In Go, `nil` values don't retain their type when you marshal and then unmarshal them. A `nil` value marshals to Ion null, but Ion null unmarshals to a zero value rather than `nil`.

```

ionNull, err := ion.MarshalText(nil) // ionNull is set to ion null
if err != nil {
    return
}

var result int
err = ion.Unmarshal(ionNull, &result) // result unmarshals to 0
if err != nil {
    return
}

```

Node.js

```

// Empty variables
let exampleString: string;

```

```
let exampleInt: number;

// Assume ionValue is some queried data from QLDB
// Check the value type and assign it to the variable if it is not null
if (ionValue.getType() === IonTypes.STRING) {
  if (ionValue.isNull()) {
    exampleString = null;
  } else {
    exampleString = ionValue.stringValue();
  }
} else if (ionValue.getType() === IonTypes.INT) {
  if (ionValue.isNull()) {
    exampleInt = null;
  } else {
    exampleInt = ionValue.numberValue();
  }
}

// Creating null values
if (exampleString === null) {
  ionString = dom.load('null.string');
} else {
  ionString = dom.load.of(exampleString);
}

if (exampleInt === null) {
  ionInt = dom.load('null.int');
} else {
  ionInt = dom.load.of(exampleInt);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
// The above null values still have the types 'String' and 'Int'
specialNull: dom.Value = dom.load("null.null");
if (specialNull.getType() === IonType.NULL) {
  // This is true!
}
if (specialNull.getType() === IonType.STRING || specialNull.getType() ===
  IonType.INT) {
  // This is false!
}
```

Python

```
# Empty variables
example_string = None
example_int = None

# Assume ion_value is some queried data from QLDB
# Check the value type and assign it to the variable if it is not null
if ion_value.ion_type == IonType.STRING:
    if isinstance(ion_value, IonPyNull):
        example_string = None
    else:
        example_string = ion_value
elif ion_value.ion_type == IonType.INT:
    if isinstance(ion_value, IonPyNull):
        example_int = None
    else:
        example_int = ion_value

# Creating Ion null values
if example_string is None:
    # Specifically a null string
    ion_string = loads("null.string")
else:
    # QLDB can take in Python string
    ion_string = example_string
if example_int is None:
    # Specifically a null int
    ion_int = loads("null.int")
else:
    # QLDB can take in Python int
    ion_int = example_int

# Special case regarding null!
# There is a generic null type which has Ion type 'Null'.
# The above null values still have the types 'String' and 'Int'
special_null = loads("null.null")
if special_null.ion_type == IonType.NULL:
    # This is true!
if special_null.ion_type == IonType.STRING or special_null.ion_type == IonType.INT:
    # This is false!
```


Down-converting to JSON

If your application requires JSON compatibility, you can down-convert Amazon Ion data to JSON. However, converting Ion to JSON is lossy in certain cases where your data uses the rich Ion types that don't exist in JSON.

For details about Ion to JSON conversion rules, see [Down-converting to JSON](#) in the *Amazon Ion Cookbook*.

Working with data and history in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following topics provide basic examples of *create*, *read*, *update*, and *delete* (CRUD) statements. You can manually run these statements by using the *PartiQL editor* on the [QLDB console](#), or the [QLDB shell](#). This guide also walks you through the process of how QLDB handles your data as you make changes in your ledger.

QLDB supports the [PartiQL](#) query language.

For code examples that show how to programmatically run similar statements using the QLDB driver, see the tutorials in [Getting started with the driver](#).

Tip

The following is a short summary of tips and best practices for working with PartiQL in QLDB:

- **Understand concurrency and transaction limits** – All statements, including SELECT queries, are subject to [optimistic concurrency control \(OCC\)](#) conflicts and [transaction limits](#), including a 30-second transaction timeout.
- **Use indexes** – Use high-cardinality indexes and run targeted queries to optimize your statements and avoid full table scans. To learn more, see [Optimizing query performance](#).
- **Use equality predicates** – Indexed lookups require an *equality* operator (= or IN). Inequality operators (<, >, LIKE, BETWEEN) don't qualify for indexed lookups and result in full table scans.
- **Use inner joins only** – QLDB supports inner joins only. As a best practice, join on fields that are indexed for each table that you're joining. Choose high-cardinality indexes for both the join criteria and the equality predicates.

Topics

- [Creating tables with indexes and inserting documents](#)
- [Querying your data](#)
- [Querying document metadata](#)
- [Using the BY clause to query document ID](#)
- [Updating and deleting documents](#)
- [Querying revision history](#)
- [Redacting document revisions](#)
- [Optimizing query performance](#)
- [Getting PartiQL statement statistics](#)
- [Querying the system catalog](#)
- [Managing tables](#)
- [Managing indexes](#)
- [Unique IDs in Amazon QLDB](#)

Creating tables with indexes and inserting documents

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

After creating an Amazon QLDB ledger, your first step is to create a table with a basic [CREATE TABLE](#) statement. Tables consist of [QLDB documents](#), which are datasets in [Amazon Ion](#) struct format.

Topics

- [Creating tables and indexes](#)
- [Inserting documents](#)

Creating tables and indexes

Tables have simple, case-sensitive names with no namespaces. QLDB supports open content and doesn't enforce schema, so you don't define attributes or data types when creating tables.

```
CREATE TABLE VehicleRegistration
```

```
CREATE TABLE Vehicle
```

A `CREATE TABLE` statement returns the system-assigned ID of the new table. All [system-assigned IDs](#) in QLDB are universally unique identifiers (UUID) that are each represented in a Base62-encoded string.

Note

Optionally, you can define tags for a table resource while you're creating the table. To learn how, see [Tagging tables on creation](#).

You can also create indexes on tables to optimize query performance.

```
CREATE INDEX ON VehicleRegistration (VIN)
```

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

Important

QLDB requires an index to efficiently look up a document. Without an index, QLDB needs to do a full table scan when reading documents. This can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a `WHERE` predicate clause using an *equality* operator (`=` or `IN`) on an indexed field or a document ID. For more information, see [Optimizing query performance](#).

Note the following constraints when creating indexes:

- An index can only be created on a single top-level field. Composite, nested, unique, and function-based indexes are not supported.
- You can create an index on any [Ion data types](#), including `list` and `struct`. However, you can only do the indexed lookup by equality of the whole Ion value regardless of the Ion type. For example, when using a `list` type as an index, you can't do an indexed lookup by one item inside the list.
- Query performance is improved only when you use an equality predicate; for example, `WHERE indexedField = 123` or `WHERE indexedField IN (456, 789)`.

QLDB doesn't honor inequalities in query predicates. As a result, range filtered scans are not implemented.

- Names of indexed fields are case sensitive and can have a maximum of 128 characters.
- Index creation in QLDB is asynchronous. The amount of time it takes to finish building an index on a non-empty table varies depending on the table size. For more information, see [Managing indexes](#).

Inserting documents

Then you can insert documents into your tables. QLDB documents are stored in Amazon Ion format. The following PartiQL [INSERT](#) statements include a subset of the vehicle registration sample data used in [Getting started with the Amazon QLDB console](#).

```
INSERT INTO VehicleRegistration
<< {
  'VIN' : '1N4AL11D75C109151',
  'LicensePlateNumber' : 'LEWISR261LL',
  'State' : 'WA',
  'City' : 'Seattle',
  'PendingPenaltyTicketAmount' : 90.25,
  'ValidFromDate' : `2017-08-21T`,
  'ValidToDate' : `2020-05-11T`,
  'Owners' : {
    'PrimaryOwner' : { 'PersonId' : '294jJ3YUoH1IEEm8GSab0s' },
    'SecondaryOwners' : [ { 'PersonId' : '5Ufgdlnj06gF5CWc0Iu64s' } ]
  }
},
{
```

```

'VIN' : 'KM8SRDHF6EU074761',
'LicensePlateNumber' : 'CA762X',
'State' : 'WA',
'City' : 'Kent',
'PendingPenaltyTicketAmount' : 130.75,
'ValidFromDate' : `2017-09-14T`,
'ValidToDate' : `2020-06-25T`,
'Owners' : {
  'PrimaryOwner' : { 'PersonId': 'IN7MvYtUjKp1GMZu0F6CG9' },
  'SecondaryOwners' : []
}
} >>

```

```

INSERT INTO Vehicle
<< {
  'VIN' : '1N4AL11D75C109151',
  'Type' : 'Sedan',
  'Year' : 2011,
  'Make' : 'Audi',
  'Model' : 'A5',
  'Color' : 'Silver'
} ,
{
  'VIN' : 'KM8SRDHF6EU074761',
  'Type' : 'Sedan',
  'Year' : 2015,
  'Make' : 'Tesla',
  'Model' : 'Model S',
  'Color' : 'Blue'
} >>

```

PartiQL syntax and semantics

- Field names are enclosed in single quotation marks (' . . . ').
- String values are also enclosed in single quotation marks (' . . . ').
- Timestamps are enclosed in backticks (` . . . `). Backticks can be used to denote any Ion literals.
- Integers and decimals are literal values that don't need to be denoted.

For more details on the syntax and semantics of PartiQL, see [Querying Ion with PartiQL in Amazon QLDB](#).

An INSERT statement creates the initial revision of a document with a version number of zero. To uniquely identify each document, QLDB assigns a *document ID* as part of the metadata. Insert statements return the ID of each document that is inserted.

Important

Because QLDB doesn't enforce schema, you can insert the same document into a table multiple times. Each insert statement commits a separate document entry to the journal, and QLDB assigns each document a unique ID.

To learn how to query the documents you inserted into your table, proceed to [Querying your data](#).

Querying your data

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The *user view* returns the latest non-deleted revision of your user data only. This is the default view in Amazon QLDB. This means that no special qualifiers are needed when you want to query only your data.

For details on the syntax and parameters of the following query examples, see [SELECT](#) in the *Amazon QLDB PartiQL reference*.

Topics

- [Basic queries](#)
- [Projections and filters](#)
- [Joins](#)
- [Nested data](#)

Basic queries

Basic SELECT queries return the documents that you inserted into the table.

Warning

When you run a query in QLDB without an indexed lookup, it invokes a full table scan. PartiQL supports such queries because it's SQL compatible. However, *don't* run table scans for production use cases in QLDB. Table scans can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator on an indexed field or a document ID; for example, WHERE indexedField = 123 or WHERE indexedField IN (456, 789). For more information, see [Optimizing query performance](#).

The following queries show results for the vehicle registration documents that you previously inserted in [Creating tables with indexes and inserting documents](#). The order of the results is not specific and can vary for each SELECT query. You shouldn't rely on the results order for any query in QLDB.

```
SELECT * FROM VehicleRegistration
WHERE LicensePlateNumber IN ('LEWISR261LL', 'CA762X')
```

```
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  PendingPenaltyTicketAmount: 90.25,
  ValidFromDate: 2017-08-21T,
  ValidToDate: 2020-05-11T,
  Owners: {
    PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSab0s" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWc0Iu64s" }]
  }
},
{
  VIN: "KM8SRDHF6EU074761",
```



```
LicensePlateNumber: "CA762X",
State: "WA",
City: "Kent",
PendingPenaltyTicketAmount: 130.75,
ValidFromDate: 2017-09-14T,
ValidToDate: 2020-06-25T,
Owners: {
  PrimaryOwner: { PersonId: "IN7MvYtUjKp1GMZu0F6CG9" },
  SecondaryOwners: []
}
}
```

```
SELECT * FROM Vehicle
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
  VIN: "1N4AL11D75C109151",
  Type: "Sedan",
  Year: 2011,
  Make: "Audi",
  Model: "A5",
  Color: "Silver"
},
{
  VIN: "KM8SRDHF6EU074761",
  Type: "Sedan",
  Year: 2015,
  Make: "Tesla",
  Model: "Model S",
  Color: "Blue"
}
```

Important

In PartiQL, you use single quotation marks to denote strings in data manipulation language (DML) or query statements. But the QLDB console and the QLDB shell return query results in Amazon Ion text format, so you see strings enclosed in double quotation marks. This syntax allows the PartiQL query language to maintain SQL compatibility, and the Amazon Ion text format to maintain JSON compatibility.

Projections and filters

You can do projections (targeted SELECT) and other standard filters (WHERE clauses). The following query returns a subset of document fields from the `VehicleRegistration` table. It filters for vehicles with the following criteria:

- **String filter** – It's registered in Seattle.
- **Decimal filter** – It has a pending penalty ticket amount less than `100.0`.
- **Date filter** – It has a registration date that is valid on or after September 4, 2019.

```
SELECT r.VIN, r.PendingPenaltyTicketAmount, r.Owners
FROM VehicleRegistration AS r
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
AND r.City = 'Seattle' --string
AND r.PendingPenaltyTicketAmount < 100.0 --decimal
AND r.ValidToDate >= '2019-09-04T' --timestamp with day precision
```

```
{
  VIN: "1N4AL11D75C109151",
  PendingPenaltyTicketAmount: 90.25,
  Owners: {
    PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSab0s" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWc0Iu64s" }]
  }
}
```

Joins

You can also write inner join queries. The following example shows an implicit inner join query that returns all registration documents along with attributes of the registered vehicles.

```
SELECT * FROM VehicleRegistration AS r, Vehicle AS v
WHERE r.VIN = v.VIN
AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
```

```

    City: "Seattle",
    PendingPenaltyTicketAmount: 90.25,
    ValidFromDate: 2017-08-21T,
    ValidToDate: 2020-05-11T,
    Owners: {
      PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSab0s" },
      SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5Cwc0Iu64s" }]
    },
    Type: "Sedan",
    Year: 2011,
    Make: "Audi",
    Model: "A5",
    Color: "Silver"
  },
  {
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFromDate: 2017-09-14T,
    ValidToDate: 2020-06-25T,
    Owners: {
      PrimaryOwner: { PersonId: "IN7MvYtUj1Gp1GMZu0F6CG9" },
      SecondaryOwners: []
    },
    Type: "Sedan",
    Year: 2015,
    Make: "Tesla",
    Model: "Model S",
    Color: "Blue"
  }
}

```

Or, you can write the same inner join query in the explicit syntax as follows.

```

SELECT * FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')

```

Nested data

You can use PartiQL in QLDB to query nested data in documents. The following example shows a correlated subquery that flattens nested data. The @ character is technically optional here. But

it explicitly indicates that you want the Owners structure within VehicleRegistration, not a different collection named Owners (if one existed).

```
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
  VIN: "1N4AL11D75C109151",
  SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWc0Iu64s" }]
},
{
  VIN: "KM8SRDHF6EU074761",
  SecondaryOwners: []
}
```

The following shows a subquery in the SELECT list that projects nested data, in addition to an inner join.

```
SELECT
    v.Make,
    v.Model,
    (SELECT VALUE o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM
    VehicleRegistration AS r, Vehicle AS v
WHERE
    r.VIN = v.VIN AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
  Make: "Audi",
  Model: "A5",
  PrimaryOwner: ["294jJ3YUoH1IEEm8GSab0s"]
},
{
  Make: "Tesla",
  Model: "Model S",
  PrimaryOwner: ["IN7MvYtUj1GMZu0F6CG9"]
}
```

```
}
```

The following query returns the `PersonId` and index (ordinal) number of each person in the `Owners.SecondaryOwners` list for a `VehicleRegistration` document.

```
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = '1N4AL11D75C109151'
```

```
{
  PersonId: "5Ufgdlnj06gF5CWc0Iu64s",
  owner_idx: 0
}
```

To learn how to query your document metadata, proceed to [Querying document metadata](#).

Querying document metadata

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

An `INSERT` statement creates the initial revision of a document with a version number of zero. To uniquely identify each document, Amazon QLDB assigns a *document ID* as part of the metadata.

In addition to the document ID and version number, QLDB stores other system-generated metadata for each document in a table. This metadata includes transaction information, journal attributes, and the document's hash value.

All system-assigned IDs are universally unique identifiers (UUID) that are each represented in a Base62-encoded string. For more information, see [Unique IDs in Amazon QLDB](#).

Topics

- [Committed view](#)
- [Joining the committed and user views](#)

Committed view

You can access document metadata by querying the *committed view*. This view returns documents from the system-defined table that directly corresponds to your user table. It includes the latest committed, non-deleted revision of both your data and the system-generated metadata. To query this view, add the prefix `_ql_committed_` to the table name in your query. (The prefix `_ql_` is reserved in QLDB for system objects.)

```
SELECT * FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

Using the data previously inserted in [Creating tables with indexes and inserting documents](#), the output of this query shows the system contents of each non-deleted document's latest revision. The system document has metadata nested in the `metadata` field, and your user data nested in the `data` field.

```
{
  blockAddress:{
    strandId:"JdxjkR9bSYB5jMHWcI464T",
    sequenceNo:14
  },
  hash:{wCsmM6qD4STxz0WYmE+47nZvWtcCz9D6zNtCiM5GoWg=}},
  data:{
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    PendingPenaltyTicketAmount: 90.25,
    ValidFromDate: 2017-08-21T,
    ValidToDate: 2020-05-11T,
    Owners: {
      PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSab0s" },
      SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWc0Iu64s" }]
    }
  },
  metadata:{
    id:"3Qv67yjXEwB9SjmvkuG6Cp",
    version:0,
    txTime:2019-06-05T20:53:321d-3Z,
    txId:"HgXAKLjAtV0HQ4lNYdzX60"
  }
}
```

```
},
{
  blockAddress:{
    strandId:"JdxjkR9bSYB5jMHwCI464T",
    sequenceNo:14
  },
  hash:{{wPuwH60TtcCvg/23BFp+redRXuCALkbDihkEvCX22Jk=}},
  data:{
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFromDate: 2017-09-14T,
    ValidToDate: 2020-06-25T,
    Owners: {
      PrimaryOwner: { PersonId: "IN7MvYtUjKp1GMZu0F6CG9" },
      SecondaryOwners: []
    }
  },
  metadata:{
    id:"J0zfB3lWqGU727mpPeWyxg",
    version:0,
    txTime:2019-06-05T20:53:321d-3Z,
    txId:"HgXAkLjAtV0HQ4lNYdzX60"
  }
}
```

Committed view fields

- **blockAddress** – The location of the block in your ledger's journal where the document revision was committed. An address, which can be used for cryptographic verification, has the following two fields.
 - **strandId** – The unique ID of the journal strand that contains the block.
 - **sequenceNo** – An index number that specifies the location of the block within the strand.

Note

Both documents in this example have an identical **blockAddress** with the same **sequenceNo**. Because these documents were inserted within a single transaction (and in this case, in a single statement), they were committed in the same block.

- **hash** – The SHA-256 Ion hash value that uniquely represents the document revision. The hash covers the revision's data and metadata fields and can be used for [cryptographic verification](#).
- **data** – The document's user data attributes.

If you redact a revision, this data structure is replaced by a `dataHash` field, whose value is the Ion hash of the removed data structure.

- **metadata** – The document's metadata attributes.
 - **id** – The system-assigned unique ID of the document.
 - **version** – The version number of the document. This is a zero-based integer that increments with each document revision.
 - **txTime** – The timestamp when the document revision was committed to the journal.
 - **txId** – The unique ID of the transaction that committed the document revision.

Joining the committed and user views

You can write queries that join a table in the committed view with a table in the user view. For example, you might want to join the document `id` of one table with a user-defined field of another table.

The following query joins two tables named `DriversLicense` and `Person` on their `PersonId` and document `id` fields respectively, using the committed view for the latter.

```
SELECT * FROM DriversLicense AS d INNER JOIN _ql_committed_Person AS p
ON d.PersonId = p.metadata.id
WHERE p.metadata.id = '1CWScY2qHYI9G88C2SjvtH'
```

To learn how to query the document ID field in the default user view, proceed to [Using the BY clause to query document ID](#).

Using the BY clause to query document ID

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is the `id` metadata field, as described in [Inserting documents](#). For this reason, you can use the `id` field to create relationships between tables.

The document `id` field is directly accessible in the committed view only, but you can also project it in the default user view by using the `BY` clause. For an example, see the following query and its results.

```
SELECT r_id, r.VIN, r.LicensePlateNumber, r.State, r.City, r.Owners
FROM VehicleRegistration AS r BY r_id
WHERE r_id = '3Qv67yjXEwB9SjmvkuG6Cp'
```

```
{
  r_id: "3Qv67yjXEwB9SjmvkuG6Cp",
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  Owners: {
    PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSab0s" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWc0Iu64s" }]
  }
}
```

In this query, `r_id` is a user-defined alias that is declared in the `FROM` clause, using the `BY` keyword. This `r_id` alias binds to the `id` metadata field for each document in the query's result set. You can use this alias in the `SELECT` clause and also in the `WHERE` clause of a query in the *user view*.

To access other metadata attributes, however, you must query the committed view.

Joining on document ID

Suppose that you're using the document `id` of one table as a foreign key in a user-defined field of another table. You can use the `BY` clause to write an inner join query for the two tables on these fields (similar to [Joining the committed and user views](#) in the previous topic).

The following example joins two tables named `DriversLicense` and `Person` on their `PersonId` and document `id` fields respectively, using the `BY` clause for the latter.

```
SELECT * FROM DriversLicense AS d INNER JOIN Person AS p BY pid
```

```
ON d.PersonId = pid
WHERE pid = '1CWScY2qHYI9G88C2SjvtH'
```

To learn how to make changes to a document in your table, proceed to [Updating and deleting documents](#).

Updating and deleting documents

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, a *document revision* is an Amazon Ion structure that represents a single version of a sequence of documents that are identified by a unique document ID. Every revision contains the document's full dataset, including both your user data and system-generated metadata. Each revision is uniquely identified by a combination of the document ID and a zero-based version number.

When you update a document, QLDB creates a new revision with the same document ID and an incremented version number. The lifecycle of a document ends when you delete it from a table. This means that no document revision with the same document ID can be created again.

Making document revisions

For example, the following statements insert a new vehicle registration, update the registration city, and then delete the registration. This results in three revisions of a document.

```
INSERT INTO VehicleRegistration
{
  'VIN' : '1HVBBAANXWH544237',
  'LicensePlateNumber' : 'LS477D',
  'State' : 'WA',
  'City' : 'Tacoma',
  'PendingPenaltyTicketAmount' : 42.20,
  'ValidFromDate' : `2011-10-26T`,
```

```
'ValidToDate' : `2023-09-25T`,
'Owners' : {
  'PrimaryOwner' : { 'PersonId': 'KmA3XPKKFqYCP2zhR3d0Ho' },
  'SecondaryOwners' : []
}
}
```

Note

Insert statements and other DML statements return the ID of each affected document. Before you continue, save this ID because you need it for the history function in the next topic. You can also find the document ID with the following query.

```
SELECT r_id FROM VehicleRegistration AS r BY r_id
WHERE r.VIN = '1HVBBAANXWH544237'
```

```
UPDATE VehicleRegistration AS r
SET r.City = 'Bellevue'
WHERE r.VIN = '1HVBBAANXWH544237'
```

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

For more examples and information about the syntax of these DML statements, see [UPDATE](#) and [DELETE](#) in the *Amazon QLDB PartiQL reference*.

To insert and remove specific elements within a document, you can use UPDATE statements or other DML statements that start with the FROM keyword. For information and examples, see the [FROM \(INSERT, REMOVE, or SET\)](#) reference.

After you delete a document, you can no longer query it in the committed or user views. To learn how to query the revision history of this document using the built-in history function, proceed to [Querying revision history](#).

Querying revision history

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB stores the complete history of every document in a table. You can see all three revisions of the vehicle registration document you previously inserted, updated, and deleted in [Updating and deleting documents](#) by querying the built-in history function.

Topics

- [History function](#)
- [History query example](#)

History function

The history function in QLDB is a PartiQL extension that returns revisions from the system-defined view of your table. So, it includes both your data and the associated metadata in the same schema as the committed view.

Syntax

```
SELECT * FROM history( table_name | 'table_id' [, 'start-time' [, 'end-time' ] ] ) AS h
[ WHERE h.metadata.id = 'id' ]
```

Arguments

***table_name* | '*table_id*'**

Either the table name or table ID. A table name is a PartiQL identifier that you can denote with double quotation marks or no quotation marks. A table ID is a string literal that must be enclosed in single quotation marks. To learn more about using table IDs, see [Querying the history of inactive tables](#).

``start-time`, `end-time``

(Optional) Specifies the time range during which any revisions were active. *These parameters don't specify the time range during which revisions were committed to the journal in a transaction.*

The start and end times are Ion timestamp literals that can be denoted with backticks (``...``). To learn more, see [Querying Ion with PartiQL in Amazon QLDB](#).

These time parameters have the following behavior:

- The *start-time* and *end-time* are both inclusive. They must be in [ISO 8601](#) date and time format and in Coordinated Universal Time (UTC).
- The *start-time* must be less than or equal to *end-time* and can be any arbitrary date in the past.
- The *end-time* must be less than or equal to the current UTC date and time.
- If you specify a *start-time* but not an *end-time*, your query defaults the *end-time* to the current date and time. If you specify neither, your query returns the entire history.

`'id'`

(Optional) The document ID for which you want to query the revision history, denoted by single quotation marks.

Tip

As a best practice, qualify a history query with both a date range (*start-time* and *end-time*) and a document ID (`metadata.id`). In QLDB, every SELECT query is processed in a transaction and is subject to a [transaction timeout limit](#).

History queries don't use the indexes that you create on a table. QLDB history is indexed by document ID only, and you can't create additional history indexes at this time. History queries that include a start time and end time gain the benefit of date range qualification.

History query example

To query the vehicle registration document's history, use the `id` that you previously saved in [Updating and deleting documents](#). For example, the following history query returns

any revisions for document ID ADR2L11fGsU4Jr4EqTdnQF that were ever active between 2019-06-05T00:00:00Z and 2019-06-05T23:59:59Z.

Note

Remember that the start and end time parameters *don't* specify the time range when revisions were committed to the journal in a transaction. For example, if a revision was committed before 2019-06-05T00:00:00Z and remained active past that start time, this example query will return that revision in the results.

Be sure to replace the `id`, start time, and end time with your own values as appropriate.

```
SELECT * FROM history(VehicleRegistration, `2019-06-05T00:00:00Z`,
`2019-06-05T23:59:59Z`) AS h
WHERE h.metadata.id = 'ADR2L11fGsU4Jr4EqTdnQF' --replace with your id
```

Your query results should look similar to the following.

```
{
  blockAddress:{
    strandId:"Jdxjkr9bSYB5jMHwcI464T",
    sequenceNo:14
  },
  hash:{{B2wYwrHK0WsmIBmxUgPRrTx9lv36tMlod2xVvWNiTbo=}},
  data: {
    VIN: "1HVBBAANXWH544237",
    LicensePlateNumber: "LS477D",
    State: "WA",
    City: "Tacoma",
    PendingPenaltyTicketAmount: 42.20,
    ValidFromDate: 2011-10-26T,
    ValidToDate: 2023-09-25T,
    Owners: {
      PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
      SecondaryOwners: []
    }
  },
  metadata:{
    id:"ADR2L11fGsU4Jr4EqTdnQF",
    version:0,
    txTime:2019-06-05T20:53:321d-3Z,
```

```

        txId:"HgXAkLjAtV0HQ4lNYdzX60"
    },
    {
        blockAddress:{
            strandId:"JdxjkR9bSYB5jMHwCI464T",
            sequenceNo:17
        },
        hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuM0mFCj8pEd6Mp0=}},
        data: {
            VIN: "1HVBBAANXWH544237",
            LicensePlateNumber: "LS477D",
            State: "WA",
            PendingPenaltyTicketAmount: 42.20,
            ValidFromDate: 2011-10-26T,
            ValidToDate: 2023-09-25T,
            Owners: {
                PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
                SecondaryOwners: []
            },
            City: "Bellevue"
        },
        metadata:{
            id:"ADR2Ll1fGsU4Jr4EqTdnQF",
            version:1,
            txTime:2019-06-05T21:01:442d-3Z,
            txId:"9cArhIQV5xf5Tf5vtsPwPq"
        }
    },
    {
        blockAddress:{
            strandId:"JdxjkR9bSYB5jMHwCI464T",
            sequenceNo:19
        },
        hash:{{7bm5DUwpqJFGrmZpb7h9wAxtvggYLPcXq+LAobi9fDg=}},
        metadata:{
            id:"ADR2Ll1fGsU4Jr4EqTdnQF",
            version:2,
            txTime:2019-06-05T21:03:76d-3Z,
            txId:"9GslbtDtpVHAgYghR5FXbZ"
        }
    }
}

```

The output includes metadata attributes that provide details on when each item was modified, and by which transaction. From this data, you can see the following:

- The document is uniquely identified by its system-assigned `id`: `ADR2L11fGsU4Jr4EqTdnQF`. This is a UUID that is represented in a Base62-encoded string.
- An `INSERT` statement creates the initial revision of a document (version `0`).
- Each subsequent update creates a new revision with the same document `id` and an incremented version number.
- The `txId` field indicates the transaction that committed each revision, and `txTime` shows when each was committed.
- A `DELETE` statement creates a new, but final revision of a document. This final revision has metadata only.

To learn how to permanently delete a revision, proceed to [Redacting document revisions](#).

Redacting document revisions

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, a `DELETE` statement only logically deletes a document by creating a new revision that marks it as deleted. QLDB also supports a *data redaction* operation that lets you permanently delete inactive document revisions in the history of a table.

Note

Any ledgers that were created before July 22, 2021 are currently not eligible for redaction. You can view the creation time of your ledger on the Amazon QLDB console.

The redaction operation deletes only the user data in the specified revision and leaves the journal sequence and the document metadata unchanged. This maintains the overall data integrity of your ledger.

Before you get started with data redaction in QLDB, make sure that you review [Redaction considerations and limitations](#) in the *Amazon QLDB PartiQL reference*.

Topics

- [Redaction stored procedure](#)
- [Checking whether a redaction is complete](#)
- [Redaction example](#)
- [Deleting and redacting an active revision](#)
- [Redacting a particular field within a revision](#)

Redaction stored procedure

You can use the [REDACT_REVISION](#) stored procedure to permanently delete an individual, inactive revision in a ledger. This stored procedure deletes all of the user data in the specified revision in both indexed storage and journal storage. However, it leaves the journal sequence and the document metadata, including the document ID and hash, unchanged. *This operation is irreversible.*

The specified document revision must be an inactive revision in history. The latest active revision of a document is not eligible for redaction.

To redact multiple revisions, you must run the stored procedure once for each revision. You can redact one revision per transaction.

Syntax

```
EXEC REDACT_REVISION `block-address`, 'table-id', 'document-id'
```

Arguments

``block-address``

The journal block location of the document revision to be redacted. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

This is an Ion literal value that is denoted by backticks. For example:

```
`{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`
```

'table-id'

The unique ID of the table whose document revision you want to redact, denoted by single quotation marks.

'document-id'

The unique document ID of the revision to be redacted, denoted by single quotation marks.

Checking whether a redaction is complete

When you submit a redaction request by running the stored procedure, QLDB processes the redaction of data asynchronously. Upon completion, the user data in the revision (represented by the data structure) is removed permanently. To check whether a redaction request has completed, you can use one of the following:

- [Journal export](#)
- [Journal stream](#)
- [GetBlock API operation](#)
- [GetRevision API operation](#)
- [History function](#) – **Note:** After a redaction is completed in the journal, it can take some time before history queries show the result of the redaction. You might see some revisions redacted before others as asynchronous redaction is completed, but history queries will show the completed results eventually.

After a revision redaction is complete, the revision's data structure is replaced by a new dataHash field. The value of this field is the Ion hash of the removed data structure, as shown in the following example. As a result, the ledger maintains its overall data integrity and remains cryptographically verifiable through the existing verification API operations. To learn more about verification, see [Data verification in Amazon QLDB](#).

Redaction example

Consider the vehicle registration document that you previously reviewed in [Querying revision history](#). Suppose that you want to redact the second revision (version:1). The following query

example shows this revision before redaction. In the query results, the data structure that will be redacted is highlighted in *red italics*.

```
SELECT * FROM history(VehicleRegistration) AS h
WHERE h.metadata.id = 'ADR2L11fGsU4Jr4EqTdnQF' --replace with your id
AND h.metadata.version = 1
```

```
{
  blockAddress:{
    strandId:"JdxjkR9bSYB5jMHwCI464T",
    sequenceNo:17
  },
  hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuM0mFCj8pEd6Mp0=}},
  data: {
    VIN: "1HVBBAANXWH544237",
    LicensePlateNumber: "LS477D",
    State: "WA",
    PendingPenaltyTicketAmount: 42.20,
    ValidFromDate: 2011-10-26T,
    ValidToDate: 2023-09-25T,
    Owners: {
      PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
      SecondaryOwners: []
    },
    City: "Bellevue"
  },
  metadata:{
    id:"ADR2L11fGsU4Jr4EqTdnQF",
    version:1,
    txTime:2019-06-05T21:01:44Z,
    txId:"9cArhIQV5xf5Tf5vtsPwPq"
  }
}
```

Note the blockAddress in the query results because you need to pass this value to the REDACT_REVISION stored procedure. Then, find the unique ID of the VehicleRegistration table by querying the [system catalog](#), as follows.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

Use this table ID along with the document ID and block address to run REDACT_REVISION. The table ID and document ID are string literals that must be enclosed in single quotation marks, and the block address is an Ion literal that is enclosed in backticks. Be sure to replace these arguments with your own values as appropriate.

```
EXEC REDACT_REVISION `{strandId:"JdxjkR9bSYB5jMHwCI464T", sequenceNo:17}`,  
'5PLf9SXwndd63lPaSIa006', 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

Tip

When you use the QLDB console or the QLDB shell to query for a table ID or document ID (or any string literal value), the returned value is enclosed in *double* quotation marks. However, when you specify the table ID and document ID arguments of the REDACT_REVISION stored procedure, you must enclose the values in *single* quotation marks.

This is because you write statements in PartiQL format, but QLDB returns results in Amazon Ion format. For details on the syntax and semantics of PartiQL in QLDB, see [Querying Ion with PartiQL](#).

A valid redaction request returns an Ion structure that represents the document revision that you are redacting, as follows.

```
{  
  blockAddress: {  
    strandId: "JdxjkR9bSYB5jMHwCI464T",  
    sequenceNo: 17  
  },  
  tableId: "5PLf9SXwndd63lPaSIa006",  
  documentId: "ADR2Ll1fGsU4Jr4EqTdnQF",  
  version: 1  
}
```

When you run this stored procedure, QLDB processes your redaction request asynchronously. Upon completion of the redaction, the data structure is permanently removed and replaced by a new *dataHash* field. The value of this field is the Ion hash of the removed data structure, as follows.

Note

This dataHash example is provided for informational purposes only and isn't a real calculated hash value.

```
{
  blockAddress:{
    strandId:"JdxjkR9bSYB5jMHWcI464T",
    sequenceNo:17
  },
  hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuM0mFCj8pEd6Mp0=}},
  dataHash: {{s83jd7sfhsdfhksj7hskjdfjfpIPP/DP2hvionas2d4=}},
  metadata:{
    id:"ADR2L11fGsU4Jr4EqTdnQF",
    version:1,
    txTime:2019-06-05T21:01:44Z,
    txId:"9cArhIQV5xf5Tf5vtsPwPq"
  }
}
```

Deleting and redacting an active revision

Active document revisions (that is, the latest non-deleted revisions of each document) are not eligible for data redaction. Before you can redact an active revision, you must first update or delete it. This moves the previously active revision to history and makes it eligible for redaction.

If your use case requires the entire document to be marked as deleted, you first use a [DELETE](#) statement. For example, the following statement logically deletes the VehicleRegistration document with a VIN of 1HVBBAANXWH544237.

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

Then, redact the previous revision before this deletion, as described previously. If required, you can also individually redact any prior revisions.

If your use case requires the document to remain active, you first use an [UPDATE](#) or [FROM](#) statement to obscure or remove the fields that you want to redact. This process is described in the following section.

Redacting a particular field within a revision

QLDB doesn't support the redaction of a particular field within a document revision. To do so, you can first use an [UPDATE-REMOVE](#) or [FROM-REMOVE](#) statement to remove an existing field from a revision. For example, the following statement removes the `LicensePlateNumber` field from the `VehicleRegistration` document with a VIN of `1HVBBAANXWH544237`.

```
UPDATE VehicleRegistration AS r
REMOVE r.LicensePlateNumber
WHERE r.VIN = '1HVBBAANXWH544237'
```

Then, redact the previous revision before this removal, as described previously. If required, you can also individually redact any prior revisions that include this now removed field.

To learn how to optimize your queries, proceed to [Optimizing query performance](#).

Optimizing query performance

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. This means that QLDB is optimized for a specific set of query patterns, even though it supports SQL-like query capabilities. It's critical to design applications and their data models to work with these query patterns. Otherwise, as your tables grow, you will encounter significant performance problems, including query latency, transaction timeouts, and concurrency conflicts.

This section describes query constraints in QLDB and provides guidance for writing optimal queries given these constraints.

Topics

- [Transaction timeout limit](#)
- [Concurrency conflicts](#)

- [Optimal query patterns](#)
- [Query patterns to avoid](#)
- [Monitoring performance](#)

Transaction timeout limit

In QLDB, every PartiQL statement (including every SELECT query) is processed in a transaction and is subject to a [transaction timeout limit](#). A transaction can run for up to **30 seconds** before being committed. After this limit, QLDB rejects any work done on the transaction and discards the [session](#) that ran the transaction. This limit protects the service's client from leaking sessions by starting transactions and not committing or canceling them.

Concurrency conflicts

QLDB implements concurrency control by using *optimistic concurrency control* (OCC). Suboptimal queries can also lead to more OCC conflicts. For information about OCC, see [Amazon QLDB concurrency model](#).

Optimal query patterns

As a best practice, you should run statements with a WHERE predicate clause that filters on an indexed field or a document ID. QLDB requires an *equality* operator (= or IN) on an indexed field to efficiently look up a document.

The following are examples of optimal query patterns in the [user view](#).

```
--Indexed field (VIN) lookup using the = operator
SELECT * FROM VehicleRegistration
WHERE VIN = '1N4AL11D75C109151'

--Indexed field (VIN) AND non-indexed field (City) lookup
SELECT * FROM VehicleRegistration
WHERE VIN = '1N4AL11D75C109151' AND City = 'Seattle'

--Indexed field (VIN) lookup using the IN operator
SELECT * FROM VehicleRegistration
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')

--Document ID (r_id) lookup using the BY clause
SELECT * FROM VehicleRegistration BY r_id
```

```
WHERE r_id = '3Qv67yjXEwB9SjmvkuG6Cp'
```

Any query that doesn't follow these patterns invokes a *full table scan*. Table scans can cause transaction timeouts for queries on large tables or queries that return large result sets. They can also [lead to OCC conflicts with competing transactions](#).

High-cardinality indexes

We recommend indexing fields that contain high-cardinality values. For example, the VIN and LicensePlateNumber fields in the VehicleRegistration table are indexed fields that are intended to be unique.

Avoid indexing low-cardinality fields such as status codes, address states or provinces, and postal codes. If you index such a field, your queries can produce large result sets that are *more likely to result in transaction timeouts or cause unintended OCC conflicts*.

Committed view queries

Queries that you run in the [committed view](#) follow the same optimization guidelines as user view queries. Indexes that you create on a table are also used for queries in the committed view.

History function queries

[History function](#) queries don't use the indexes that you create on a table. QLDB history is indexed by document ID only, and you can't create additional history indexes at this time.

As a best practice, qualify a history query with both a date range (*start time* and *end time*) and a document ID (`metadata.id`). History queries that include a start time and end time gain the benefit of date range qualification.

Inner join queries

For inner join queries, use join criteria that includes at least an indexed field for the table on the right side of the join. Without a join index, a join query invokes multiple table scans—for every document in the left table of the join, the query fully scans the right table. The best practice is to join on fields that are indexed for each table that you're joining, in addition to specifying a WHERE equality predicate for at least one table.

For example, the following query joins the VehicleRegistration and Vehicle tables on their respective VIN fields, which are both indexed. This query also has an equality predicate on VehicleRegistration.VIN.


```
SELECT * FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

Choose high-cardinality indexes for both the join criteria and the equality predicates in your join queries.

Query patterns to avoid

The following are some examples of **suboptimal** statements that don't scale well for larger tables in QLDB. We strongly recommend that you don't rely on these types of queries for tables that grow over time because your queries will eventually result in transaction timeouts. Because tables contain documents that vary in size, it's difficult to define precise limits for non-indexed queries.

```
--No predicate clause
SELECT * FROM Vehicle

--COUNT() is not an optimized function
SELECT COUNT(*) FROM Vehicle

--Low-cardinality predicate
SELECT * FROM Vehicle WHERE Color = 'Silver'

--Inequality (>) does not qualify for indexed lookup
SELECT * FROM Vehicle WHERE "Year" > 2019

--Inequality (LIKE)
SELECT * FROM Vehicle WHERE VIN LIKE '1N4AL%'

--Inequality (BETWEEN)
SELECT SUM(PendingPenaltyTicketAmount) FROM VehicleRegistration
WHERE ValidToDate BETWEEN `2020-01-01T` AND `2020-07-01T`

--No predicate clause
DELETE FROM Vehicle

--No document id, and no date range for the history() function
SELECT * FROM history(Vehicle)
```

In general, we *don't recommend* running the following types of query patterns for production use cases in QLDB:

- Online analytical processing (OLAP) queries
- Exploratory queries without a predicate clause
- Reporting queries
- Text search

Instead, we recommend streaming your data to a purpose-built database service that is optimized for analytical use cases. For example, you can stream QLDB data to Amazon OpenSearch Service to provide full text search capabilities over documents. For a sample application that demonstrates this use case, see the GitHub repository [aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python](https://github.com/aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python). For information about QLDB streams, see [Streaming journal data from Amazon QLDB](#).

Monitoring performance

The QLDB driver provides consumed I/O usage and timing information in the result object of a statement. You can use these metrics to identify inefficient PartiQL statements. To learn more, proceed to [Getting PartiQL statement statistics](#).

You can also use Amazon CloudWatch to track your ledger's performance for data operations. Monitor the CommandLatency metric for a specified LedgerName and CommandType. For more information, see [Monitoring with Amazon CloudWatch](#). To learn how QLDB uses commands to manage data operations, see [Session management with the driver](#).

Getting PartiQL statement statistics

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB provides statement execution statistics that can help you optimize your usage of QLDB by running more efficient PartiQL statements. QLDB returns these statistics along with the results of the statement. They include metrics that quantify consumed I/O usage and server-side processing time, which you can use to identify inefficient statements.

This feature is currently available in the *PartiQL editor* on the [QLDB console](#), the [QLDB shell](#), and the latest version of the [QLDB driver](#) for all supported languages. You can also view statement statistics for your query history on the console.

Topics

- [I/O usage](#)
- [Timing information](#)

I/O usage

The I/O usage metric describes the number of read I/O requests. If the number of read I/O requests is higher than expected, it indicates that the statement isn't optimized, such as the lack of an index. We recommend that you review [Optimal query patterns](#) in the previous topic, *Optimizing query performance*.

Note

When you run a `CREATE INDEX` statement on a non-empty table, the I/O usage metric includes read requests for the synchronous index creation call only. QLDB builds the index for any existing documents in the table asynchronously. These asynchronous read requests aren't included in the I/O usage metric from your statement results. Asynchronous read requests are charged separately and are added to your total read I/Os after the index build is completed.

Using the QLDB console

To get a statement's read I/O usage by using the QLDB console, do the following steps:

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **PartiQL editor**.
3. Choose a ledger from the dropdown list of ledgers.
4. In the query editor window, enter any statement of your choice, and then choose **Run**. The following is a query example.

```
SELECT * FROM testTable WHERE firstName = 'Jim'
```

To run a statement, you can also use the keyboard shortcut **Ctrl+Enter** for Windows, or **Cmd+Return** for macOS. For more keyboard shortcuts, see [PartiQL editor keyboard shortcuts](#).

5. Below the query editor window, your query results include *read I/Os*, which is the number of read requests that were made by the statement.

You can also view the read I/Os of your query history by doing the following steps:

1. In navigation pane, choose **Recent queries** under **PartiQL editor**.
2. The **Read I/Os** column displays the number of read requests that were made by each statement.

Using the QLDB driver

To get a statement's I/O usage by using the QLDB driver, call the `getConsumedIOs` operation of the result's stream cursor or buffered cursor.

The following code examples show how to get read I/Os from the *stream cursor* of a statement result.

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.IOUsage;
import software.amazon.qldb.Result;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

driver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM testTable WHERE firstName = ?",
    ionFirstName);

    for (IonValue ionValue : result) {
        // User code here to handle results
    }

    IOUsage ioUsage = result.getConsumedIOs();
}
```

```
    long readIOs = ioUsage.getReadIOs();
  });
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

// This is one way of creating Ion values. We can also use a ValueFactory.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionFirstName = IonLoader.Default.Load("Jim");

await driver.Execute(async txn =>
{
    IAsyncResult result = await txn.Execute("SELECT * FROM testTable WHERE firstName = ?", ionFirstName);

    // Iterate through stream cursor to accumulate read IOs.
    await foreach (IIonValue ionValue in result)
    {
        // User code here to handle results.
        // Warning: It is bad practice to rely on results within a lambda block,
        unless
        // it is to check the state of a result. This is because lambdas are
        retryable.
    }

    var ioUsage = result.GetConsumedIOs();
    var readIOs = ioUsage?.ReadIOs;
});
```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qlldb-driver-go/v2/qlldbdriver"
)

driver.Execute(context.Background(), func(txn qlldbdriver.Transaction) (interface{},
error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?", "Jim")

    if err != nil {
        panic(err)
    }

    for result.Next(txn) {
        // User code here to handle results
    }

    ioUsage := result.GetConsumedIOs()
    readIOs := *ioUsage.GetReadIOs()
    fmt.Println(readIOs)
    return nil, nil
})
```

Node.js

```
import { IOUsage, ResultReadable, TransactionExecutor } from "amazon-qlldb-driver-nodejs";

await driver.executeLambda(async (txn: TransactionExecutor) => {
    const result: ResultReadable = await txn.executeAndStreamResults("SELECT * FROM
testTable WHERE firstName = ?", "Jim");

    for await (const chunk of result) {
        // User code here to handle results
    }

    const ioUsage: IOUsage = result.getConsumedIOs();
    const readIOs: number = ioUsage.getReadIOs();
});
```

Python

```
def get_read_ios(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM testTable WHERE
    firstName = ?", "Jim")

    for row in cursor:
        # User code here to handle results
        pass

    consumed_ios = cursor.get_consumed_ios()
    read_ios = consumed_ios.get('ReadIOs')

qlldb_driver.execute_lambda(lambda txn: get_read_ios(txn))
```

The following code examples show how to get read I/Os from the *buffered cursor* of a statement result. This returns the total read I/Os from ExecuteStatement and FetchPage requests.

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qlldb.IOUsage;
import software.amazon.qlldb.Result;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

Result result = driver.execute(txn -> {
    return txn.execute("SELECT * FROM testTable WHERE firstName = ?", ionFirstName);
});

IOUsage ioUsage = result.getConsumedIOs();
long readIOs = ioUsage.getReadIOs();
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
```

```

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
        ionFirstName);
});

var ioUsage = result.GetConsumedIOs();
var readIOs = ioUsage?.ReadIOs;

```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```

import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qlbdbdriver"
)

result, err := driver.Execute(context.Background(), func(txn qlbdbdriver.Transaction)
(interface{}, error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
        "Jim")
    if err != nil {
        return nil, err
    }
    return txn.BufferResult(result)
})

if err != nil {
    panic(err)
}

qlbdbResult := result.(*qlbdbdriver.BufferedResult)
ioUsage := qlbdbResult.GetConsumedIOs()

```



```
readIOs := *ioUsage.GetReadIOs()
fmt.Println(readIOs)
```

Node.js

```
import { IOUsage, Result, TransactionExecutor } from "amazon-qlldb-driver-nodejs";

const result: Result = await driver.executeLambda(async (txn: TransactionExecutor)
=> {
    return await txn.execute("SELECT * FROM testTable WHERE firstName = ?", "Jim");
});

const ioUsage: IOUsage = result.getConsumedIOs();
const readIOs: number = ioUsage.getReadIOs();
```

Python

```
cursor = qlldb_driver.execute_lambda(
    lambda txn: txn.execute_statement("SELECT * FROM testTable WHERE firstName = ?",
    "Jim"))

consumed_ios = cursor.get_consumed_ios()
read_ios = consumed_ios.get('ReadIOs')
```

Note

The stream cursor is stateful because it paginates the result set. Therefore, the `getConsumedIOs` and `getTimingInformation` operations return the accumulated metrics from the time that you call them.

The buffered cursor buffers the result set in memory and returns the total accumulated metrics.

Timing information

The timing information metric describes the server-side processing time in milliseconds. Server-side processing time is defined as the amount of time that QLDB spends on processing a statement. This doesn't include time spent on network calls or pauses. This metric disambiguates the processing time on the QLDB service side from the processing time on the client side.

Using the QLDB console

To get a statement's timing information by using the QLDB console, do the following steps:

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **PartiQL editor**.
3. Choose a ledger from the dropdown list of ledgers.
4. In the query editor window, enter any statement of your choice, and then choose **Run**. The following is a query example.

```
SELECT * FROM testTable WHERE firstName = 'Jim'
```

To run a statement, you can also use the keyboard shortcut **Ctrl+Enter** for Windows, or **Cmd+Return** for macOS. For more keyboard shortcuts, see [PartiQL editor keyboard shortcuts](#).

5. Below the query editor window, your query results include *server-side latency*, which is the amount of time between when QLDB received the statement request, and when it sent the response. This is a subset of the total query duration.

You can also view the timing information of your query history by doing the following steps:

1. In navigation pane, choose **Recent queries** under **PartiQL editor**.
2. The **Execution time (ms)** column displays this timing information for each statement.

Using the QLDB driver

To get a statement's timing information by using the QLDB driver, call the `getTimingInformation` operation of the result's stream cursor or buffered cursor.

The following code examples show how to get the processing time from the *stream cursor* of a statement result.

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.Result;
import software.amazon.qldb.TimingInformation;
```

```

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

driver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM testTable WHERE firstName = ?",
    ionFirstName);

    for (IonValue ionValue : result) {
        // User code here to handle results
    }

    TimingInformation timingInformation = result.getTimingInformation();
    long processingTimeMilliseconds =
    timingInformation.getProcessingTimeMilliseconds();
});

```

.NET

```

using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

await driver.Execute(async txn =>
{
    IAsyncResult result = await txn.Execute("SELECT * FROM testTable WHERE firstName
    = ?", ionFirstName);

    // Iterate through stream cursor to accumulate processing time.
    await foreach(IIonValue ionValue in result)
    {
        // User code here to handle results.
        // Warning: It is bad practice to rely on results within a lambda block,
        unless
        // it is to check the state of a result. This is because lambdas are
        retryable.
    }

    var timingInformation = result.GetTimingInformation();
    var processingTimeMilliseconds = timingInformation?.ProcessingTimeMilliseconds;

```

```
});
```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qlldb-driver-go/v2/qlddbdriver"
)

driver.Execute(context.Background(), func(txn qlddbdriver.Transaction) (interface{},
error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?", "Jim")

    if err != nil {
        panic(err)
    }

    for result.Next(txn) {
        // User code here to handle results
    }

    timingInformation := result.GetTimingInformation()
    processingTimeMilliseconds := *timingInformation.GetProcessingTimeMilliseconds()
    fmt.Println(processingTimeMilliseconds)
    return nil, nil
})
```

Node.js

```
import { ResultReadable, TimingInformation, TransactionExecutor } from "amazon-qlldb-driver-nodejs";

await driver.executeLambda(async (txn: TransactionExecutor) => {
    const result: ResultReadable = await txn.executeAndStreamResults("SELECT * FROM
testTable WHERE firstName = ?", "Jim");
```

```

    for await (const chunk of result) {
        // User code here to handle results
    }

    const timingInformation: TimingInformation = result.getTimingInformation();
    const processingTimeMilliseconds: number =
        timingInformation.getProcessingTimeMilliseconds();
    });

```

Python

```

def get_processing_time_milliseconds(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM testTable WHERE
    firstName = ?", "Jim")

    for row in cursor:
        # User code here to handle results
        pass

    timing_information = cursor.get_timing_information()
    processing_time_milliseconds =
        timing_information.get('ProcessingTimeMilliseconds')

qlldb_driver.execute_lambda(lambda txn: get_processing_time_milliseconds(txn))

```

The following code examples show how to get the processing time from the *buffered cursor* of a statement result. This returns the total processing time from `ExecuteStatement` and `FetchPage` requests.

Java

```

import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qlldb.Result;
import software.amazon.qlldb.TimingInformation;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

Result result = driver.execute(txn -> {

```

```

        return txn.execute("SELECT * FROM testTable WHERE firstName = ?", ionFirstName);
    });

    TimingInformation timingInformation = result.getTimingInformation();
    long processingTimeMilliseconds = timingInformation.getProcessingTimeMilliseconds();

```

.NET

```

using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
        ionFirstName);
});

var timingInformation = result.GetTimingInformation();
var processingTimeMilliseconds = timingInformation?.ProcessingTimeMilliseconds;

```

Note

To convert to synchronous code, remove the `await` and `async` keywords, and change the `IAsyncResult` type to `IResult`.

Go

```

import (
    "context"
    "fmt"
    "github.com/aws-labs/amazon-qlldb-driver-go/v2/qlddbdriver"
)

result, err := driver.Execute(context.Background(), func(txn qlddbdriver.Transaction)
(interface{}, error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
        "Jim")

```

```

    if err != nil {
        return nil, err
    }
    return txn.BufferResult(result)
})

if err != nil {
    panic(err)
}

qldbResult := result.(*qldbdriver.BufferedResult)
timingInformation := qldbResult.GetTimingInformation()
processingTimeMilliseconds := *timingInformation.GetProcessingTimeMilliseconds()
fmt.Println(processingTimeMilliseconds)

```

Node.js

```

import { Result, TimingInformation, TransactionExecutor } from "amazon-qldb-driver-nodejs";

const result: Result = await driver.executeLambda(async (txn: TransactionExecutor)
=> {
    return await txn.execute("SELECT * FROM testTable WHERE firstName = ?", "Jim");
});

const timingInformation: TimingInformation = result.getTimingInformation();
const processingTimeMilliseconds: number =
    timingInformation.getProcessingTimeMilliseconds();

```

Python

```

cursor = qldb_driver.execute_lambda(
    lambda txn: txn.execute_statement("SELECT * FROM testTable WHERE firstName = ?",
    "Jim"))

timing_information = cursor.get_timing_information()
processing_time_milliseconds = timing_information.get('ProcessingTimeMilliseconds')

```

Note

The stream cursor is stateful because it paginates the result set. Therefore, the `getConsumedIOs` and `getTimingInformation` operations return the accumulated metrics from the time that you call them.

The buffered cursor buffers the result set in memory and returns the total accumulated metrics.

To learn how to query the system catalog, proceed to [Querying the system catalog](#).

Querying the system catalog

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Each table that you create in an Amazon QLDB ledger has a system-assigned unique ID. You can find a table's ID, its list of indexes, and other metadata by querying the system catalog table `information_schema.user_tables`.

All system-assigned IDs are universally unique identifiers (UUID) that are each represented in a Base62-encoded string. For more information, see [Unique IDs in Amazon QLDB](#).

The following example shows the results of a query that returns metadata attributes of the `VehicleRegistration` table.

```
SELECT * FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

```
{
  tableId: "5PLf9SXwndd631PaSIa006",
  name: "VehicleRegistration",
  indexes: [
    { indexId: "Djg2nt0yIs2GY0T29Kud1z", expr: "[VIN]", status: "ONLINE" },
```



```
{ indexId: "4tPW3fUhaVhDinRgKRLhGU", expr: "[LicensePlateNumber]", status:
"BUILDING" }
],
status: "ACTIVE"
}
```

Table metadata fields

- **tableId** – The unique ID of the table.
- **name** – The table name.
- **indexes** – The list of indexes on the table.
 - **indexId** – The unique ID of the index.
 - **expr** – The indexed document path. This field is a string in the form: `[fieldName]`.
 - **status** – The index's current status (BUILDING, FINALIZING, ONLINE, FAILED, or DELETING). QLDB doesn't use the index in queries until the status is ONLINE.
 - **message** – The error message that describes the reason that the index has a FAILED status. This field is only included for failed indexes.
- **status** – The table's current status (ACTIVE or INACTIVE). A table becomes INACTIVE when you DROP it.

To learn how to manage tables using the `DROP TABLE` and `UNDROP TABLE` statements, proceed to [Managing tables](#).

Managing tables

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes how to manage tables using the `DROP TABLE` and `UNDROP TABLE` statements in Amazon QLDB. It also describes how to tag tables while you're creating them. The quotas for the number of active tables and total tables that you can create are defined in [Quotas and limits in Amazon QLDB](#).

Topics

- [Tagging tables on creation](#)
- [Dropping tables](#)
- [Querying the history of inactive tables](#)
- [Reactivating tables](#)

Tagging tables on creation

Note

Tagging tables on creation is currently supported for ledgers in the STANDARD permissions mode only.

You can tag your table resources. To manage tags for existing tables, use the AWS Management Console or the API operations `TagResource`, `UntagResource`, and `ListTagsForResource`. For more information, see [Tagging Amazon QLDB resources](#).

You can also define table tags while you're creating the table by using the QLDB console, or by specifying them in a `CREATE TABLE PartiQL` statement. The following example creates a table named `Vehicle` with the tag `environment=production`.

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

By tagging resources while they're being created, you can eliminate the need to run custom tagging scripts after resource creation. After a table is tagged, you can control access to the table based on those tags. For example, you can grant full access only to tables that have a specific tag. For a JSON policy example, see [Full access to all actions based on table tags](#).

Dropping tables

To drop a table, use a basic [DROP TABLE](#) statement. When you drop a table in QLDB, you're just deactivating it.

For example, the following statement deactivates the `VehicleRegistration` table.

```
DROP TABLE VehicleRegistration
```

A `DROP TABLE` statement returns the system-assigned ID of the table. The status of `VehicleRegistration` should now be `INACTIVE` in the system catalog table [information_schema.user_tables](#).

```
SELECT status FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

Querying the history of inactive tables

In addition to a table name, you can also query the QLDB [History function](#) with a table ID as the first input argument. You must use the table ID to query the history of an inactive table. After a table is deactivated, you can no longer query its history with the table name.

First, find the table ID by querying the system catalog table. For example, the following query returns the `tableId` of the `VehicleRegistration` table.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

Then, you can use this ID to run the same history query from [Querying revision history](#). The following is an example that queries the history of document ID `ADR2L11fGsU4Jr4EqTdnQF` from table ID `5PLf9SXwndd631PaSIa006`. The table ID is a string literal that must be enclosed in single quotation marks.

```
--replace both the table and document IDs with your values
SELECT * FROM history('5PLf9SXwndd631PaSIa006', `2000T`, `2019-06-05T23:59:59Z`) AS h
WHERE h.metadata.id = 'ADR2L11fGsU4Jr4EqTdnQF'
```

Reactivating tables

After you deactivate a table in QLDB, you can use the [UNDROP TABLE](#) statement to reactivate it.

First, find the table ID from `information_schema.user_tables`. For example, the following query returns the `tableId` of the `VehicleRegistration` table. The status should be `INACTIVE`.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

Then, use this ID to reactivate the table. The following is an example that *undrops* table ID 5PLf9SXwndd631PaSIa006. In this case, the table ID is a unique identifier that you enclose in double quotation marks.

```
UNDROP TABLE "5PLf9SXwndd631PaSIa006"
```

The status of `VehicleRegistration` should now be `ACTIVE`.

To learn how to create, describe, and drop indexes, proceed to [Managing indexes](#).

Managing indexes

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes how to create, describe, and drop indexes in Amazon QLDB. The quota for the number of indexes per table that you can create is defined in [Quotas and limits in Amazon QLDB](#).

Topics

- [Creating indexes](#)
- [Describing indexes](#)
- [Dropping indexes](#)
- [Common errors](#)

Creating indexes

As also described in [Creating tables and indexes](#), you can use the [CREATE INDEX](#) statement to create an index on a table for a specified top-level field, as follows. The table name and the indexed field name are both case sensitive.

```
CREATE INDEX ON VehicleRegistration (VIN)
```

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

Each index that you create on a table has a system-assigned unique ID. To find this index ID, see the following section [Describing indexes](#).

Important

QLDB requires an index to efficiently look up a document. Without an index, QLDB needs to do a full table scan when reading documents. This can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a `WHERE` predicate clause using an *equality* operator (`=` or `IN`) on an indexed field or a document ID. For more information, see [Optimizing query performance](#).

Note the following constraints when creating indexes:

- An index can only be created on a single top-level field. Composite, nested, unique, and function-based indexes are not supported.
- You can create an index on any [Ion data types](#), including `list` and `struct`. However, you can only do the indexed lookup by equality of the whole Ion value regardless of the Ion type. For example, when using a `list` type as an index, you can't do an indexed lookup by one item inside the list.
- Query performance is improved only when you use an equality predicate; for example, `WHERE indexedField = 123` or `WHERE indexedField IN (456, 789)`.

QLDB doesn't honor inequalities in query predicates. As a result, range filtered scans are not implemented.

- Names of indexed fields are case sensitive and can have a maximum of 128 characters.
- Index creation in QLDB is asynchronous. The amount of time it takes to finish building an index on a non-empty table varies depending on the table size. For more information, see [Managing indexes](#).

Describing indexes

Index creation in QLDB is asynchronous. The amount of time it takes to finish building an index on a non-empty table varies depending on the table size. To check the status of an index build, you can query the system catalog table [information_schema.user_tables](#).

For example, the following statement queries the system catalog for all indexes on the `VehicleRegistration` table.

```
SELECT VALUE indexes
FROM information_schema.user_tables info, info.indexes indexes
WHERE info.name = 'VehicleRegistration'
```

```
{
  indexId: "Djg2nt0yIs2GY0T29Kud1z",
  expr: "[VIN]",
  status: "ONLINE"
},
{
  indexId: "4tPW3fUhaVhDinRgKRLhGU",
  expr: "[LicensePlateNumber]",
  status: "FAILED",
  message: "aws.ledger.errors.InvalidEntityError: Document contains multiple values
for indexed field: LicensePlateNumber"
}
```

Index fields

- `indexId` – The unique ID of the index.
- `expr` – The indexed document path. This field is a string in the form: `[fieldName]`.
- `status` – The index's current status. The status of an index can be one of the following values:
 - `BUILDING` – Is actively building the index for the table.
 - `FINALIZING` – Has finished building the index and is starting to activate it for use.
 - `ONLINE` – Is active and ready to use in queries. QLDB doesn't use the index in queries until the status is online.
 - `FAILED` – Is unable to build the index due to an unrecoverable error. Indexes in this state still count against your quota of indexes per table. For more information, see [Common errors](#).
 - `DELETING` – Is actively deleting the index after a user dropped it.

- **message** – The error message that describes the reason that the index has a FAILED status. This field is only included for failed indexes.

Using the console

You can also use the AWS Management Console to check an index's status.

To check the status of an index (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. In the list of **Ledgers**, choose the ledger name whose indexes you want to manage.
4. On the ledger details page, under the **Tables** tab, choose the table name whose index you want to check.
5. On the table details page, locate the **Indexed fields** card. The **Index status** column displays the current status of each index on the table.

Dropping indexes

Use the [DROP INDEX](#) statement to drop an index. When you drop an index, it's permanently deleted from the table.

First, find the index ID from `information_schema.user_tables`. For example, the following query returns the `indexId` of the indexed `LicensePlateNumber` field on the `VehicleRegistration` table.

```
SELECT indexes.indexId
FROM information_schema.user_tables info, info.indexes indexes
WHERE info.name = 'VehicleRegistration' and indexes.expr = '[LicensePlateNumber]'
```

Then, use this ID to drop the index. The following is an example that drops index ID `4tPW3fUhaVhDinRgKRLhGU`. The index ID is a unique identifier that should be enclosed in double quotation marks.

```
DROP INDEX "4tPW3fUhaVhDinRgKRLhGU" ON VehicleRegistration WITH (purge = true)
```

Note

The clause `WITH (purge = true)` is required for all `DROP INDEX` statements, and `true` is currently the only supported value.

The keyword `purge` is case sensitive and must be all lowercase.

Using the console

You can also use the AWS Management Console to drop an index.

To drop an index (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. In the list of **Ledgers**, choose the ledger name whose indexes you want to manage.
4. On the ledger details page, under the **Tables** tab, choose the table name whose index you want to drop.
5. On the table details page, locate the **Indexed fields** card. Select the index that you want to drop, and then choose **Drop index**.

Common errors

This section describes common errors that you might encounter when creating indexes, and suggests possible solutions.

Note

Indexes that have a status of `FAILED` still count against your quota of indexes per table. A failed index also prevents you from modifying or deleting any documents that caused the index creation to fail on the table.

You must explicitly [drop](#) the index to remove it from the quota.

Document contains multiple values for indexed field: *fieldName*.

QLDB is unable to build an index for the specified field name because the table contains a document with multiple values for the same field (that is, duplicate field names).

You must first drop the failed index. Then, make sure that all of the documents in the table have only one value for each field name before retrying the index creation. You can also create an index for another field that has no duplicates.

QLDB also returns this error if you try to insert a document that contains multiple values for a field that is already indexed on the table.

Exceeded indexes limit: Table *tableName* already has *n* indexes, and cannot create more.

QLDB enforces a limit of five indexes per table, including failed indexes. You must drop an existing index before creating a new one.

No defined index with identifier: *indexId*.

You tried to drop an index that doesn't exist for the specified table and index ID combination. To learn how to check existing indexes, see [Describing indexes](#).

Unique IDs in Amazon QLDB

** Important**

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes the properties and usage guidelines of system-assigned unique identifiers in Amazon QLDB. It also provides some examples of QLDB unique IDs.

Topics

- [Properties](#)
- [Usage](#)
- [Examples](#)

Properties

All system-assigned IDs in QLDB are universally unique identifiers (UUID). Each ID has the following properties:

- 128-bit UUID number
- Represented in Base62-encoded text
- Fixed length alphanumeric string of 22 characters (for example: 3Qv67yjXEwB9SjmvkuG6Cp)

Usage

When using QLDB unique IDs in your application, note the following guidelines:

Do

- Treat the ID as a string.

Don't

- Try to decode the string.
- Ascribe semantic meaning to the string (such as deriving a time component).
- Sort the strings in a semantic order.

Examples

The following attributes are some examples of unique IDs in QLDB:

- Document ID
- Index ID
- Strand ID
- Table ID
- Transaction ID

Amazon QLDB concurrency model

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. QLDB supports SQL-like query capabilities, and delivers full ACID transactions. In addition, QLDB data items are documents, delivering schema flexibility and intuitive data modeling. With a journal at the core, you can use QLDB to access the complete and verifiable history of all changes to your data, and stream coherent transactions to other data services as needed.

Topics

- [Optimistic concurrency control](#)
- [Using indexes to avoid full table scans](#)
- [Insertion OCC conflicts](#)
- [Making transactions idempotent](#)
- [Redaction OCC conflicts](#)
- [Managing concurrent sessions](#)

Optimistic concurrency control

In QLDB, concurrency control is implemented using *optimistic concurrency control* (OCC). OCC operates on the principle that multiple transactions can frequently complete without interfering with each other.

Using OCC, transactions in QLDB don't acquire locks on database resources and operate with full serializable isolation. QLDB runs concurrent transactions in a serial manner, such that it produces the same effect as if those transactions were started serially.

Before committing, each transaction performs a validation check to ensure that no other committed transaction has modified the data that it's accessing. If this check reveals conflicting

modifications, or the state of the data changes, the committing transaction is rejected. However, the transaction can be restarted.

When a transaction writes to QLDB, the validation checks of the OCC model are implemented by QLDB itself. If a transaction can't be written to the journal due to a failure in the verification phase of OCC, QLDB returns an `OccConflictException` to the application layer. The application software is responsible for ensuring that the transaction is restarted. The application should abort the rejected transaction and then retry the whole transaction from the start.

To learn how the QLDB driver handles and retries OCC conflicts and other transient exceptions, see [Understanding retry policy with the driver in Amazon QLDB](#).

Using indexes to avoid full table scans

In QLDB, every PartiQL statement (including every `SELECT` query) is processed in a transaction and is subject to a [transaction timeout limit](#).

As a best practice, you should run statements with a `WHERE` predicate clause that filters on an indexed field or a document ID. QLDB requires an *equality* operator on an indexed field to efficiently look up a document; for example, `WHERE indexedField = 123` or `WHERE indexedField IN (456, 789)`.

Without this indexed lookup, QLDB needs to do a *full table scan* when reading documents. This can cause query latency and transaction timeouts, and also increases the chances of an OCC conflict with competing transactions.

For example, consider a table named `Vehicle` that has an index on the `VIN` field only. It contains the following documents.

VIN	Make	Model	Color
"1N4AL11D 75C109151"	"Audi"	"A5"	"Silver"
"KM8SRDHF 6EU074761"	"Tesla"	"Model S"	"Blue"
"3HGGK5G5 3FM761765"	"Ducati"	"Monster 1200"	"Yellow"

VIN	Make	Model	Color
"1HVBBAAN XWH544237"	"Ford"	"F 150"	"Black"
"1C4RJFAG 0FC625797"	"Mercedes"	"CLK 350"	"White"

Two concurrent users named Alice and Bob are working with the same table in a ledger. They want to update two different documents, as follows.

Alice:

```
UPDATE Vehicle AS v
SET v.Color = 'Blue'
WHERE v.VIN = '1N4AL11D75C109151'
```

Bob:

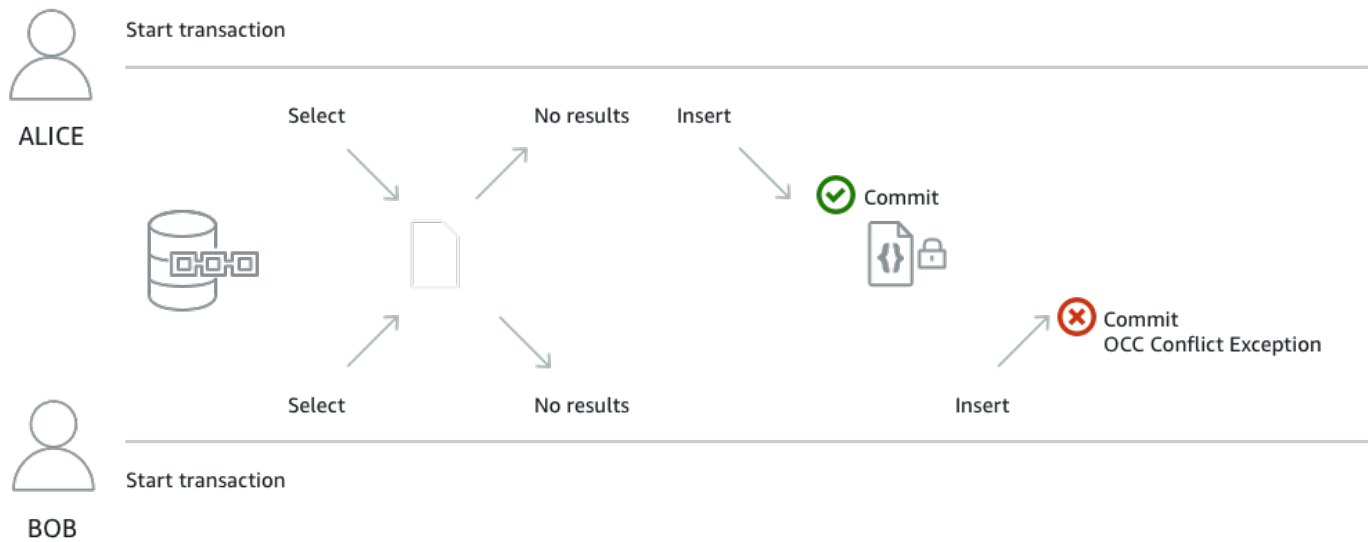
```
UPDATE Vehicle AS v
SET v.Color = 'Red'
WHERE v.Make = 'Tesla' AND v.Model = 'Model S'
```

Suppose that Alice and Bob start their transactions at the same time. Alice's UPDATE statement does an indexed lookup on the VIN field, so it only needs to read that one document. Alice finishes and successfully commits her transaction first.

Bob's statement filters on non-indexed fields, so it does a table scan and encounters an `OccConflictException`. This is because Alice's committed transaction modified the data that Bob's statement is accessing, which includes every document in the table—not only the document that Bob is updating.

Insertion OCC conflicts

OCC conflicts can include documents that are newly inserted—not only documents that previously existed. Consider the following diagram, in which two concurrent users (Alice and Bob) are working with the same table in a ledger. They both want to insert a new document only under the condition that a predicate value does not yet exist.



In this example, both Alice and Bob run the following `SELECT` and `INSERT` statements within a single transaction. Their application runs the `INSERT` statement only if the `SELECT` statement returns no results.

```
SELECT * FROM Vehicle v WHERE v.VIN = 'ABCDE12345EXAMPLE'
```

```
INSERT INTO Vehicle VALUE
{
  'VIN' : 'ABCDE12345EXAMPLE',
  'Type' : 'Wagon',
  'Year' : 2019,
  'Make' : 'Subaru',
  'Model' : 'Outback',
  'Color' : 'Gray'
}
```

Suppose that Alice and Bob start their transactions at the same time. Both of their `SELECT` queries return no existing document with a VIN of `ABCDE12345EXAMPLE`. So, their applications proceed with the `INSERT` statement.

Alice finishes and successfully commits her transaction first. Then, Bob tries to commit his transaction, but QLDB rejects it and throws an `OccConflictException`. This is because Alice's committed transaction modified the result set of Bob's `SELECT` query, and OCC detects this conflict before committing Bob's transaction.

The SELECT query is required for this transaction example to be [idempotent](#). Bob can then retry his whole transaction from the start. But his next SELECT query will return the document that Alice inserted, so Bob's application won't run the INSERT.

Making transactions idempotent

The insert transaction in the [previous section](#) is also an example of an *idempotent* transaction. In other words, running the same transaction multiple times produces identical results. If Bob runs the INSERT without first checking if a particular VIN already exists, the table might end up with documents that have duplicate VIN values.

Consider other retry scenarios in addition to OCC conflicts. For example, it's possible that QLDB successfully commits a transaction on the server side, but the client times out while waiting for a response. As a best practice, make your write transactions idempotent to avoid any unexpected side effects in the case of concurrency or retries.

Redaction OCC conflicts

QLDB prevents concurrent [redactions of revisions](#) on the same journal block. Consider an example where two concurrent users (Alice and Bob) want to redact two different document revisions that are committed on the same block in a ledger. First, Alice requests the redaction of one revision by running the REDACT_REVISION stored procedure, as follows.

```
EXEC REDACT_REVISION `{strandId:"Jdxjkr9bSYB5jMHwCI464T", sequenceNo:17}`,  
'5PLf9SXwndd631PaSIa006', 'ADR2L11fGsU4Jr4EqTdnQF'
```

Then, while Alice's request is still processing, Bob requests the redaction of another revision, as follows.

```
EXEC REDACT_REVISION `{strandId:"Jdxjkr9bSYB5jMHwCI464T", sequenceNo:17}`,  
'8F0TPCmdNQ6JTRpiLj2TmW', '05K8zpGYWynDlE0K5afDRc'
```

QLDB rejects Bob's request with an `OccConflictException` even though they're trying to redact two different document revisions. This is because Bob's revision is located on the same block as the revision that Alice is redacting. After Alice's request finishes processing, Bob can then retry his redaction request.

Similarly, if two concurrent transactions try to redact the same revision, only one request can be processed. The other request fails with an OCC conflict exception until the redaction is complete.

Afterwards, any requests to redact the same revision will result in an error that indicates the revision is already redacted.

Managing concurrent sessions

If you have experience using a relational database management system (RDBMS), you might be familiar with concurrent connection limits. QLDB doesn't have the same concept of a traditional RDBMS connection because transactions are run with HTTP request and response messages.

In QLDB, the analogous concept is an *active session*. A session is conceptually similar to a user login—it manages information about your data transaction requests to a ledger. An active session is one that is actively running a transaction. It can also be a session that recently finished a transaction where the service anticipates it will start another transaction immediately. QLDB supports one actively running transaction per session.

The limit of concurrent active sessions per ledger is defined in [Quotas and limits in Amazon QLDB](#). After this limit is reached, any session that tries to start a transaction will result in an error (`LimitExceededException`).

For information about the lifecycle of a session and how the QLDB driver handles sessions when running data transactions, see [Session management with the driver](#). For best practices for configuring a session pool in your application using the QLDB driver, see [Configuring the QldbDriver object](#) in *Amazon QLDB driver recommendations*.

Data verification in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

With Amazon QLDB, you can trust that the history of changes to your application data is accurate. QLDB uses an immutable transactional log, known as a *journal*, for data storage. The journal tracks every change to your committed data and maintains a complete and verifiable history of changes over time.

QLDB uses the SHA-256 hash function with a Merkle tree-based model to generate a cryptographic representation of your journal, known as a *digest*. The digest acts as a unique signature of your data's entire change history as of a point in time. You use the digest to verify the integrity of your document revisions relative to that signature.

Topics

- [What kind of data can you verify in QLDB?](#)
- [What does data integrity mean?](#)
- [How does verification work?](#)
- [Verification example](#)
- [How does data redaction affect verification?](#)
- [Getting started with verification](#)
- [Step 1: Requesting a digest in QLDB](#)
- [Step 2: Verifying your data in QLDB](#)
- [Verification results](#)
- [Tutorial: Verifying data using an AWS SDK](#)
- [Common errors for verification](#)

What kind of data can you verify in QLDB?

In QLDB, each ledger has exactly one journal. A journal can have multiple *strands*, which are partitions of the journal.

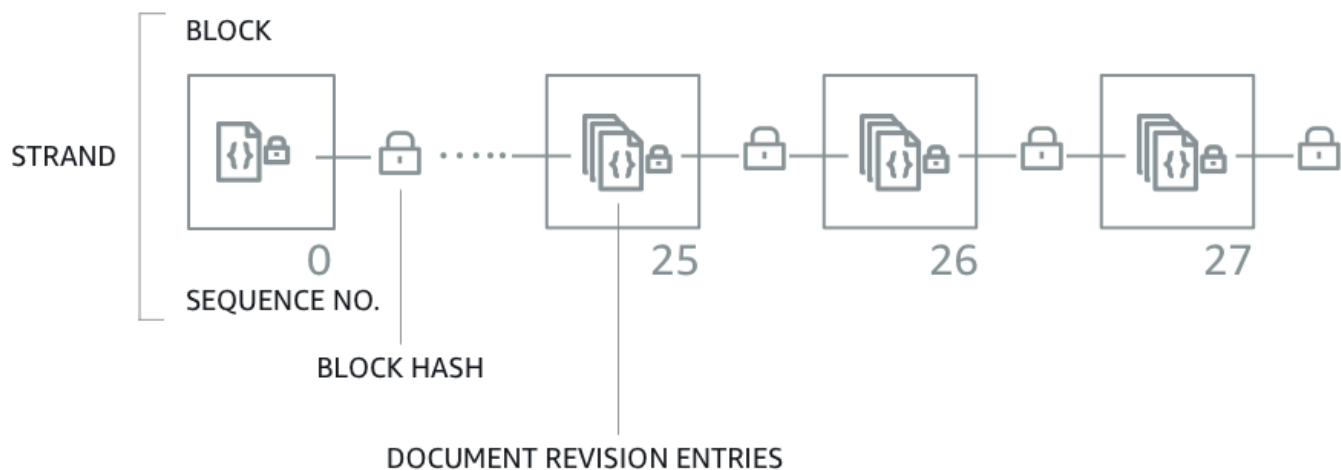
Note

QLDB currently supports journals with a single strand only.

A *block* is an object that is committed to the journal strand during a transaction. This block contains *entry* objects, which represent the document revisions that resulted from the transaction. You can verify either an individual revision or an entire journal block in QLDB.

The following diagram illustrates this journal structure.

QLDB JOURNAL



The diagram shows that transactions are committed to the journal as blocks that contain document revision entries. It also shows that each block is hash-chained to subsequent blocks and has a sequence number to specify its address within the strand.

For information about the data contents in a block, see [Journal contents in Amazon QLDB](#).

What does data integrity mean?

Data integrity in QLDB means that your ledger's journal is in fact immutable. In other words, your data (specifically, each document revision) is in a state where the following are true:

1. It exists at the same location in your journal where it was first written.
2. It hasn't been altered in any way since it was written.

How does verification work?

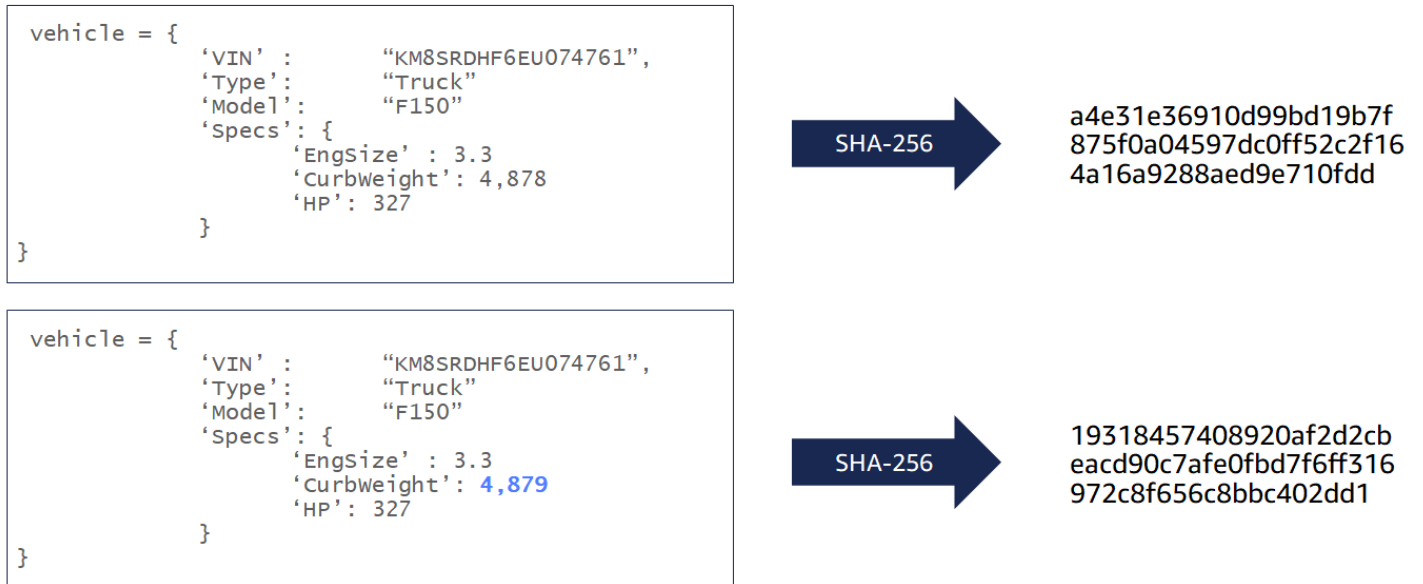
To understand how verification works in Amazon QLDB, you can break down the concept into four basic components.

- [Hashing](#)
- [Digest](#)
- [Merkle tree](#)
- [Proof](#)

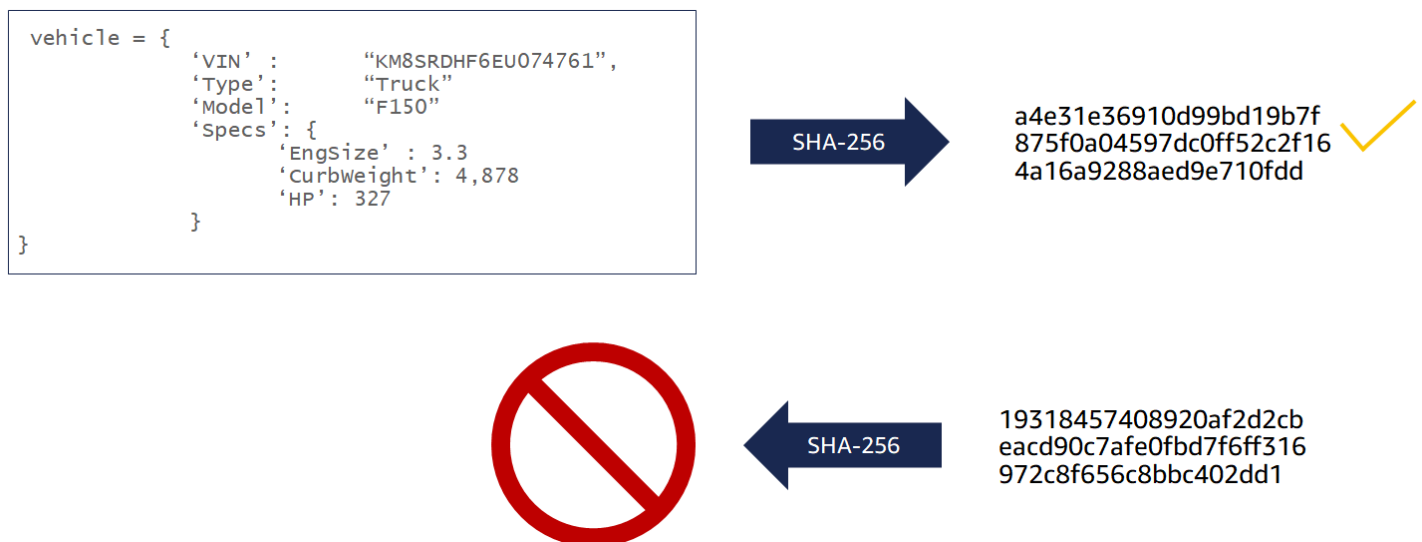
Hashing

QLDB uses the SHA-256 cryptographic hash function to create 256-bit hash values. A hash acts as a unique, fixed-length signature of any arbitrary amount of input data. If you change any part of the input—even a single character or bit—then the output hash changes completely.

The following diagram shows that the SHA-256 hash function creates completely unique hash values for two QLDB documents that differ by only a single digit.



The SHA-256 hash function is one way, which means that it's not mathematically feasible to compute the input when given an output. The following diagram shows that it's not feasible to compute the input QLDB document when given an output hash value.



The following data inputs are hashed in QLDB for verification purposes:

- Document revisions
- PartiQL statements
- Revision entries
- Journal blocks

Digest

A *digest* is a cryptographic representation of your ledger's entire journal at a point in time. A journal is append-only, and journal blocks are sequenced and hash-chained similar to blockchains.

You can request a digest for a ledger at any time. QLDB generates the digest and returns it to you as a secure output file. Then you use that digest to verify the integrity of document revisions that were committed at a prior point in time. If you recalculate hashes by starting with a revision and ending with the digest, you prove that your data has not been altered in between.

Merkle tree

As the size of your ledger grows, it becomes increasingly inefficient to recalculate the journal's full hash chain for verification. QLDB uses a Merkle tree model to address this inefficiency.

A *Merkle tree* is a tree data structure in which each leaf node represents a hash of a data block. Each non-leaf node is a hash of its child nodes. Commonly used in blockchains, a Merkle tree helps you efficiently verify large datasets with an audit proof mechanism. For more information about Merkle trees, see the [Merkle tree Wikipedia page](#). To learn more about Merkle audit proofs and for an example use case, see [How Log Proofs Work](#) on the Certificate Transparency site.

The QLDB implementation of the Merkle tree is constructed from a journal's full hash chain. In this model, the leaf nodes are the set of all individual document revision hashes. The root node represents the digest of the entire journal as of a point in time.

Using a Merkle audit proof, you can verify a revision by checking only a small subset of your ledger's revision history. You do this by traversing the tree from a given leaf node (revision) to its root (digest). Along this traversal path, you recursively hash sibling pairs of nodes to compute their parent hash until you end with the digest. This traversal has a time complexity of $\log(n)$ nodes in the tree.

Proof

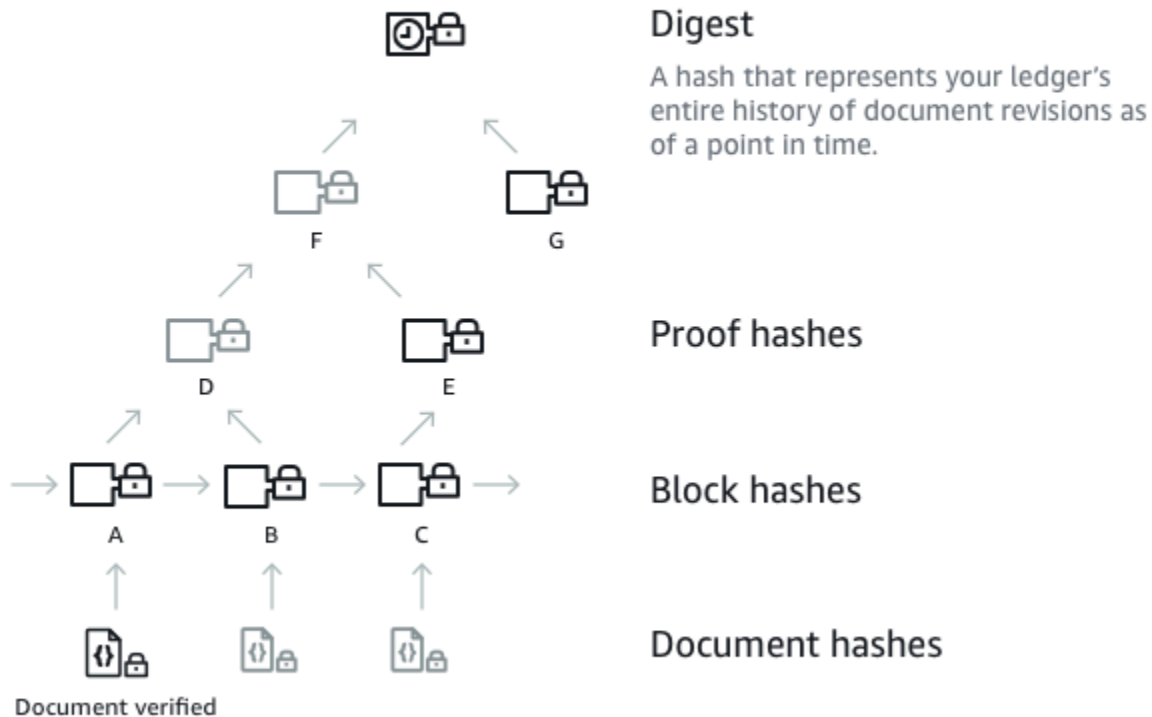
A *proof* is the ordered list of node hashes that QLDB returns for a given digest and document revision. It consists of the hashes that are required by a Merkle tree model to chain the given leaf node hash (a revision) to the root hash (the digest).

Changing any committed data between a revision and a digest breaks your journal's hash chain and makes it impossible to generate a proof.

Verification example

The following diagram illustrates the Amazon QLDB hash tree model. It shows a set of block hashes that rolls up to the top root node, which represents the digest of a journal strand. In a ledger with a single-strand journal, this root node is also the digest of the entire ledger.

PROOF



Suppose that node **A** is the block that contains the document revision whose hash you want to verify. The following nodes represent the ordered list of hashes that QLDB provides in your proof: **B, E, G**. These hashes are required to recalculate the digest from hash **A**.

To recalculate the digest, do the following:

1. Start with hash **A** and concatenate it with hash **B**. Then, hash the result to compute **D**.
2. Use **D** and **E** to compute **F**.
3. Use **F** and **G** to compute the digest.

The verification is successful if your recalculated digest matches the expected value. Given a revision hash and a digest, it's not feasible to reverse engineer the hashes in a proof. Therefore, this exercise proves that your revision was indeed written in this journal location relative to the digest.

How does data redaction affect verification?

In Amazon QLDB, a DELETE statement only logically deletes a document by creating a new revision that marks it as deleted. QLDB also supports a *data redaction* operation that lets you permanently delete inactive document revisions in the history of a table.

The redaction operation deletes only the user data in the specified revision, and leaves the journal sequence and the document metadata unchanged. After a revision is redacted, the user data in the revision (represented by the data structure) is replaced by a new dataHash field. The value of this field is the [Amazon Ion](#) hash of the removed data structure. For more information and an example of a redaction operation, see [Redacting document revisions](#).

As a result, the ledger maintains its overall data integrity and remains cryptographically verifiable through the existing verification API operations. You can still use these API operations as expected to request a digest ([GetDigest](#)), request a proof ([GetBlock](#) or [GetRevision](#)), and then run your verification algorithm using the returned objects.

Recalculating a revision hash

If you plan to verify an individual document revision by recalculating its hash, you must conditionally check whether the revision was redacted. If the revision was redacted, you can use the hash value that is provided in the dataHash field. If it wasn't redacted, you can recalculate the hash by using the data field.

By doing this conditional check, you can identify redacted revisions and take the appropriate action. For example, you can log data manipulation events for monitoring purposes.

Getting started with verification

Before you can verify data, you must request a digest from your ledger and save it for later. Any document revision that is committed before the latest block covered by the digest is eligible for verification against that digest.

Then, you request a proof from Amazon QLDB for an eligible revision that you want to verify. Using this proof, you call a client-side API to recalculate the digest, starting with your revision hash. As

long as the previously saved digest is *known and trusted outside of QLDB*, the integrity of your document is proven if your recalculated digest hash matches the saved digest hash.

Important

- What you're specifically proving is that the document revision wasn't altered between the time that you saved this digest and when you run the verification. You can request and save a digest as soon as a revision that you want to verify later is committed to the journal.
- As a best practice, we recommend that you request digests on a regular basis and store them away from the ledger. Determine the frequency that you request digests based on how often you commit revisions in your ledger.

For a detailed AWS blog post that discusses the value of cryptographic verification in the context of a realistic use case, see [Real-world cryptographic verification with Amazon QLDB](#).

For step-by-step guides on how to request a digest from your ledger and then verify your data, see the following:

- [Step 1: Requesting a digest in QLDB](#)
- [Step 2: Verifying your data in QLDB](#)

Step 1: Requesting a digest in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB provides an API to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request. You can use the AWS Management Console, an AWS SDK, or the AWS Command Line Interface (AWS CLI) to get a digest.

Topics

- [AWS Management Console](#)
- [QLDB API](#)

AWS Management Console

Follow these steps to request a digest using the QLDB console.

To request a digest (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select the ledger name for which you want to request a digest.
4. Choose **Get digest**. The **Get digest** dialog box displays the following digest details:
 - **Digest** – The SHA-256 hash value of the digest that you requested.
 - **Digest tip address** – The latest block location in the journal covered by the digest that you requested. An address has the following two fields:
 - `strandId` – The unique ID of the journal strand that contains the block.
 - `sequenceNo` – The index number that specifies the location of the block within the strand.
 - **Ledger** – The ledger name for which you requested a digest.
 - **Date** – The timestamp when you requested the digest.
5. Review the digest information. Then choose **Save**. You can keep the default file name, or enter a new name.

Note

You might notice that your digest hash and tip address values change even when you don't modify any data in your ledger. This is because the console retrieves the ledger's system catalog each time that you run a query in the *PartiQL editor*. This is a read transaction that gets committed to the journal and causes the latest block address to change.

This step saves a plaintext file with contents in [Amazon Ion](#) format. The file has a file name extension of `.ion.txt` and contains all the digest information that was listed on the preceding dialog box. The following is an example of a digest file's contents. The order of the fields can vary depending on your browser.

```
{
  "digest": "42zaJ0fV8iGutVGNaIuzQWhD5Xb/5B9lScHnvxPXm9E=",
  "digestTipAddress": "{strandId:\"B1FTjlSXze9BIh1K0szcE3\",sequenceNo:73}",
  "ledger": "my-ledger",
  "date": "2019-04-17T16:57:26.749Z"
}
```

6. Save this file where you can access it in the future. Later, you can use this file to verify a document revision against.

Important

The document revision that you verify later must be covered by the digest that you saved. That is, the sequence number of the document's address must be less than or equal to the sequence number of the **Digest tip address**.

QLDB API

You can also request a digest from your ledger by using the Amazon QLDB API with an AWS SDK or the AWS CLI. The QLDB API provides the following operation for use by application programs:

- [GetDigest](#) – Returns the digest of a ledger at the latest committed block in the journal. The response includes a 256-bit hash value and a block address.

For information about requesting a digest using the AWS CLI, see the [get-digest](#) command in the *AWS CLI Command Reference*.

Sample application

For Java code examples, see the GitHub repository [aws-samples/amazon-qldb-dmv-sample-java](#). For instructions on how to download and install this sample application, see [Installing the Amazon](#)

[QLDB Java sample application](#). Before requesting a digest, make sure that you follow Steps 1–3 in the [Java tutorial](#) to create a sample ledger and load it with sample data.

The tutorial code in class [GetDigest](#) provides an example of requesting a digest from the vehicle-registration sample ledger.

To verify a document revision using the digest that you saved, proceed to [Step 2: Verifying your data in QLDB](#).

Step 2: Verifying your data in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB provides an API to request a proof for a specified document ID and its associated block. You must also provide the tip address of a digest that you previously saved, as described in [Step 1: Requesting a digest in QLDB](#). You can use the AWS Management Console, an AWS SDK, or the AWS CLI to get a proof.

Then, you can use the proof returned by QLDB to verify the document revision against the saved digest, using a client-side API. This gives you control over the algorithm that you use to verify your data.

Topics

- [AWS Management Console](#)
- [QLDB API](#)

AWS Management Console

This section describes the steps to verify a document revision against a previously saved digest using the Amazon QLDB console.

Before you start, make sure that you follow the steps in [Step 1: Requesting a digest in QLDB](#). Verification requires a previously saved digest that covers the revision that you want to verify.

To verify a document revision (console)

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. First, query your ledger for the `id` and `blockAddress` of the revision that you want to verify. These fields are included in the document's metadata, which you can query in the *committed view*.

The document `id` is a system-assigned unique ID string. The `blockAddress` is an `Ion` structure that specifies the block location where the revision was committed.

In the navigation pane, choose **PartiQL editor**.

3. Choose the ledger name in which you want to verify a revision.
4. In the query editor window, enter a SELECT statement in the following syntax, and then choose **Run**.

```
SELECT metadata.id, blockAddress FROM _ql_committed_table_name
WHERE criteria
```

For example, the following query returns a document from the `VehicleRegistration` table in the sample ledger created in [Getting started with the Amazon QLDB console](#).

```
SELECT r.metadata.id, r.blockAddress FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN = 'KM8SRDHF6EU074761'
```

5. Copy and save the `id` and `blockAddress` values that your query returns. Be sure to omit the double quotes for the `id` field. In [Amazon Ion](#), string data types are delimited with double quotes. For example, you must copy only the alphanumeric text in the following snippet.

```
"LtMNJYNjSwzBLgf7sLifrG"
```

6. Now that you have a document revision selected, you can start the process of verifying it.

In the navigation pane, choose **Verification**.

7. On the **Verify document** form, under **Specify the document that you want to verify**, enter the following input parameters:
 - **Ledger** – The ledger in which you want to verify a revision.
 - **Block address** – The `blockAddress` value returned by your query in step 4.
 - **Document ID** – The `id` value returned by your query in step 4.

8. Under **Specify the digest to use for verification**, select the digest that you previously saved by choosing **Choose digest**. If the file is valid, this auto-populates all the digest fields on your console. Or, you can manually copy and paste the following values directly from your digest file:
 - **Digest** – The digest value from your digest file.
 - **Digest tip address** – The `digestTipAddress` value from your digest file.
9. Review your document and digest input parameters, and then choose **Verify**.

The console automates two steps for you:

- a. Request a proof from QLDB for your specified document.
- b. Use the proof returned by QLDB to call a client-side API, which verifies your document revision against the provided digest. To examine this verification algorithm, see the following section [QLDB API](#) to download the code example.

The console displays the results of your request in the **Verification results** card. For more information, see [Verification results](#).

QLDB API

You can also verify a document revision by using the Amazon QLDB API with an AWS SDK or the AWS CLI. The QLDB API provides the following operations for use by application programs:

- **GetDigest** – Returns the digest of a ledger at the latest committed block in the journal. The response includes a 256-bit hash value and a block address.
- **GetBlock** – Returns a block object at a specified address in a journal. Also returns a proof of the specified block for verification if `DigestTipAddress` is provided.
- **GetRevision** – Returns a revision data object for a specified document ID and block address. Also returns a proof of the specified revision for verification if `DigestTipAddress` is provided.

For complete descriptions of these API operations, see the [Amazon QLDB API reference](#).

For information about verifying data using the AWS CLI, see the [AWS CLI Command Reference](#).

Sample application

For Java code examples, see the GitHub repository [aws-samples/amazon-qldb-dmv-sample-java](#). For instructions on how to download and install this sample application, see [Installing the Amazon QLDB Java sample application](#). Before doing a verification, make sure that you follow Steps 1–3 in the [Java tutorial](#) to create a sample ledger and load it with sample data.

The tutorial code in class [GetRevision](#) provides an example of requesting a proof for a document revision and then verifying that revision. This class runs the following steps:

1. Requests a new digest from the sample ledger `vehicle-registration`.
2. Requests a proof for a sample document revision from the `VehicleRegistration` table in the `vehicle-registration` ledger.
3. Verifies the sample revision using the returned digest and proof.

Verification results

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes the results returned by an Amazon QLDB data verification request on the AWS Management Console. For detailed steps on how to submit a verification request, see [Step 2: Verifying your data in QLDB](#).

On the **Verification** page of the QLDB console, the results of your request are displayed in the **Verification results** card. The **Proof** tab shows the contents of the proof returned by QLDB for your specified document revision and digest. It includes the following details:

- **Revision hash** – The SHA-256 value that uniquely represents the document revision that you're verifying.
- **Proof hashes** – The ordered list of hashes provided by QLDB that are used to recalculate the specified digest. The console starts with the **Revision hash** and sequentially combines it with each proof hash until it ends with a recalculated digest.

The list is collapsed by default, so you can expand it to reveal the hash values. Optionally, you can try the hash calculations yourself by following the steps as described in [Using a proof to recalculate your digest](#).

- **Digest calculated** – The hash that resulted from the series of **Hash calculations** that were done on the **Revision hash**. If this value matches your previously saved **Digest**, the verification is successful.

The **Block** tab shows the contents of the block that contains the revision you're verifying. It includes the following details:

- **Transaction ID** – The unique ID of the transaction that committed this block.
- **Transaction time** – The timestamp when this block was committed to the strand.
- **Block hash** – The SHA-256 value that uniquely represents this block and all of its contents.
- **Block address** – The location in your ledger's journal where this block was committed. An address has the following two fields:
 - **Strand ID** – The unique ID of the journal strand that contains this block.
 - **Sequence number** – The index number that specifies the location of this block within the strand.
- **Statements** – The PartiQL statements that were performed to commit entries in this block.

 Note

If you run parameterized statements programmatically, they're recorded in your journal blocks with bind parameters instead of the literal data. For example, you might see the following statement in a journal block, where the question mark (?) is a variable placeholder for the document contents.

```
INSERT INTO Vehicle ?
```

- **Document entries** – The document revisions that were committed in this block.

If your request failed to verify the document revision, see [Common errors for verification](#) for information about possible causes.

Using a proof to recalculate your digest

After QLDB returns a proof for your document verification request, you can try doing the hash calculations yourself. This section describes the high-level steps to recalculate your digest using the proof that is provided.

First, pair your **Revision hash** with the first hash in the **Proof hashes** list. Then, do the following steps.

1. Sort the two hashes. Compare the hashes by their *signed* byte values in little-endian order.
2. Concatenate the two hashes in sorted order.
3. Hash the concatenated pair with an SHA-256 hash generator.
4. Pair your new hash with the next hash in the proof and repeat steps 1–3. After you process the last proof hash, your new hash is your recalculated digest.

If your recalculated digest matches your previously saved digest, your document is successfully verified.

For a step-by-step tutorial with code examples that demonstrate these verification steps, proceed to [Tutorial: Verifying data using an AWS SDK](#).

Tutorial: Verifying data using an AWS SDK

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In this tutorial, you verify a document revision hash and a journal block hash in an Amazon QLDB ledger by using the QLDB API through an AWS SDK. You also use the QLDB driver to query the document revision.

Consider an example where you have a document revision that contains data for a vehicle with a vehicle identification number (VIN) of KM8SRDHF6EU074761. The document revision is in a

VehicleRegistration table that is in a ledger named `vehicle-registration`. Suppose that you want to verify the integrity of both the document revision for this vehicle and the journal block that contains the revision.

Note

For a detailed AWS blog post that discusses the value of cryptographic verification in the context of a realistic use case, see [Real-world cryptographic verification with Amazon QLDB](#).

Topics

- [Prerequisites](#)
- [Step 1: Request a digest](#)
- [Step 2: Query the document revision](#)
- [Step 3: Request a proof for the revision](#)
- [Step 4: Recalculate the digest from the revision](#)
- [Step 5: Request a proof for the journal block](#)
- [Step 6: Recalculate the digest from the block](#)
- [Run the full code example](#)

Prerequisites

Before you get started, make sure that you do the following:

1. Set up the QLDB driver for a language of your choice by completing the respective prerequisites under [Getting started with the Amazon QLDB driver](#). This includes signing up for AWS, granting programmatic access for development, and configuring your development environment.
2. Follow steps 1–2 in [Getting started with the Amazon QLDB console](#) to create a ledger named `vehicle-registration` and load it with predefined sample data.

Next, review the following steps to learn how verification works, and then run the full code example from start to end.

Step 1: Request a digest

Before you can verify data, you must first request a digest from your ledger vehicle-registration for use later.

Java

```
// Get a digest
GetDigestRequest digestRequest = new GetDigestRequest().withName(ledgerName);
GetDigestResult digestResult = client.getDigest(digestRequest);

java.nio.ByteBuffer digest = digestResult.getDigest();

// expectedDigest is the buffer we will use later to compare against our calculated
// digest
byte[] expectedDigest = new byte[digest.remaining()];
digest.get(expectedDigest);
```

.NET

```
// Get a digest
GetDigestRequest getDigestRequest = new GetDigestRequest
{
    Name = ledgerName
};
GetDigestResponse getDigestResponse =
    client.GetDigestAsync(getDigestRequest).Result;

// expectedDigest is the buffer we will use later to compare against our calculated
// digest
MemoryStream digest = getDigestResponse.Digest;
byte[] expectedDigest = digest.ToArray();
```

Go

```
// Get a digest
currentLedgerName := ledgerName
input := qlldb.GetDigestInput{Name: &currentLedgerName}
digestOutput, err := client.GetDigest(&input)
if err != nil {
    panic(err)
}
```

```
// expectedDigest is the buffer we will later use to compare against our calculated
digest
expectedDigest := digestOutput.Digest
```

Node.js

```
// Get a digest
const getDigestRequest: GetDigestRequest = {
  Name: ledgerName
};
const getDigestResponse: GetDigestResponse = await
  qlldbClient.getDigest(getDigestRequest).promise();

// expectedDigest is the buffer we will later use to compare against our calculated
digest
const expectedDigest: Uint8Array = <Uint8Array>getDigestResponse.Digest;
```

Python

```
# Get a digest
get_digest_response = qlldb_client.get_digest(Name=ledger_name)

# expected_digest is the buffer we will later use to compare against our calculated
digest
expected_digest = get_digest_response.get('Digest')
digest_tip_address = get_digest_response.get('DigestTipAddress')
```

Step 2: Query the document revision

Use the QLDB driver to query the block addresses, hashes, and document IDs that are associated with VIN KM8SRDHF6EU074761.

Java

```
// Retrieve info for the given vin's document revisions
Result result = driver.execute(txn -> {
    final String query = String.format("SELECT blockAddress, hash, metadata.id FROM
    _ql_committed_%s WHERE data.VIN = '%s'", tableName, vin);
    return txn.execute(query);
});
```

.NET

```
// Retrieve info for the given vin's document revisions
var result = driver.Execute(txn => {
    string query = $"SELECT blockAddress, hash, metadata.id FROM
    _ql_committed_{tableName} WHERE data.VIN = '{vin}'";
    return txn.Execute(query);
});
```

Go

```
// Retrieve info for the given vin's document revisions
result, err := driver.Execute(context.Background(), func(txn qlbdbdriver.Transaction)
(interface{}, error) {
    statement := fmt.Sprintf(
        "SELECT blockAddress, hash, metadata.id FROM _ql_committed_%s WHERE
data.VIN = '%s'",
        tableName,
        vin)
    result, err := txn.Execute(statement)
    if err != nil {
        return nil, err
    }

    results := make([]map[string]interface{}, 0)

    // Convert the result set into a map
    for result.Next(txn) {
        var doc map[string]interface{}
        err := ion.Unmarshal(result.GetCurrentData(), &doc)
        if err != nil {
            return nil, err
        }
        results = append(results, doc)
    }
    return results, nil
})
if err != nil {
    panic(err)
}
resultSlice := result.([]map[string]interface{}))
```

Node.js

```
const result: dom.Value[] = await driver.executeLambda(async (txn:
  TransactionExecutor): Promise<dom.Value[]> => {
    const query: string = `SELECT blockAddress, hash, metadata.id FROM
      _ql_committed_${tableName} WHERE data.VIN = '${vin}'`;
    const queryResult: Result = await txn.execute(query);
    return queryResult.getResultList();
  });
```

Python

```
def query_doc_revision(txn):
    query = "SELECT blockAddress, hash, metadata.id FROM _ql_committed_{table_name} WHERE
      data.VIN = '{vin}'".format(table_name, vin)
    return txn.execute_statement(query)

# Retrieve info for the given vin's document revisions
result = qlldb_driver.execute_lambda(query_doc_revision)
```

Step 3: Request a proof for the revision

Iterate through the query results and use each block address and document ID along with the ledger name to submit a `GetRevision` request. *To get a proof for the revision, you must also provide the tip address from the previously saved digest.* This API operation returns an object that includes the document revision and the proof for the revision.

For information about the revision structure and its contents, see [Querying document metadata](#).

Java

```
for (IonValue ionValue : result) {
    IonStruct ionStruct = (IonStruct)ionValue;

    // Get the requested fields
    IonValue blockAddress = ionStruct.get("blockAddress");
    IonBlob hash = (IonBlob)ionStruct.get("hash");
    String metadataId = ((IonString)ionStruct.get("id")).stringValue();

    System.out.printf("Verifying document revision for id '%s'\n", metadataId);
```

```

String blockAddressText = blockAddress.toString();

// Submit a request for the revision
GetRevisionRequest revisionRequest = new GetRevisionRequest()
    .WithName(ledgerName)
    .WithBlockAddress(new ValueHolder().withIonText(blockAddressText))
    .WithDocumentId(metadataId)
    .WithDigestTipAddress(digestResult.getDigestTipAddress());

// Get a result back
GetRevisionResult revisionResult = client.getRevision(revisionRequest);

...
}

```

.NET

```

foreach (IIonValue ionValue in result)
{
    IIonStruct ionStruct = ionValue;

    // Get the requested fields
    IIonValue blockAddress = ionStruct.GetField("blockAddress");
    IIonBlob hash = ionStruct.GetField("hash");
    String metadataId = ionStruct.GetField("id").StringValue;

    Console.WriteLine($"Verifying document revision for id '{metadataId}'");

    // Use an Ion Reader to convert block address to text
    IIonReader reader = IonReaderBuilder.Build(blockAddress);
    StringWriter sw = new StringWriter();
    IIonWriter textWriter = IonTextWriterBuilder.Build(sw);
    textWriter.WriteValues(reader);
    string blockAddressText = sw.ToString();

    // Submit a request for the revision
    GetRevisionRequest revisionRequest = new GetRevisionRequest
    {
        Name = ledgerName,
        BlockAddress = new ValueHolder
        {
            IonText = blockAddressText
        },
    },

```

```

        DocumentId = metadataId,
        DigestTipAddress = getDigestResponse.DigestTipAddress
    };

    // Get a response back
    GetRevisionResponse revisionResponse =
client.GetRevisionAsync(revisionRequest).Result;

    ...
}

```

Go

```

for _, value := range resultSlice {
    // Get the requested fields
    ionBlockAddress, err := ion.MarshalText(value["blockAddress"])
    if err != nil {
        panic(err)
    }
    blockAddress := string(ionBlockAddress)
    metadataId := value["id"].(string)
    documentHash := value["hash"].([]byte)

    fmt.Printf("Verifying document revision for id '%s'\n", metadataId)

    // Submit a request for the revision
    revisionInput := qlldb.GetRevisionInput{
        BlockAddress:      &qlldb.ValueHolder{IonText: &blockAddress},
        DigestTipAddress: digestOutput.DigestTipAddress,
        DocumentId:        &metadataId,
        Name:              &currentLedgerName,
    }

    // Get a result back
    revisionOutput, err := client.GetRevision(&revisionInput)
    if err != nil {
        panic(err)
    }

    ...
}

```

Node.js

```

for (let value of result) {
    // Get the requested fields
    const blockAddress: dom.Value = value.get("blockAddress");
    const hash: dom.Value = value.get("hash");
    const metadataId: string = value.get("id").stringValue();

    console.log(`Verifying document revision for id '${metadataId}'`);

    // Submit a request for the revision
    const revisionRequest: GetRevisionRequest = {
        Name: ledgerName,
        BlockAddress: {
            IonText: dumpText(blockAddress)
        },
        DocumentId: metadataId,
        DigestTipAddress: getDigestResponse.DigestTipAddress
    };

    // Get a response back
    const revisionResponse: GetRevisionResponse = await
qlldbClient.getRevision(revisionRequest).promise();

    ...
}

```

Python

```

for value in result:
    # Get the requested fields
    block_address = value['blockAddress']
    document_hash = value['hash']
    metadata_id = value['id']

    print("Verifying document revision for id '{}'.format(metadata_id))

    # Submit a request for the revision and get a result back
    proof_response = qlldb_client.get_revision(Name=ledger_name,

BlockAddress=block_address_to_dictionary(block_address),
                                           DocumentId=metadata_id,
                                           DigestTipAddress=digest_tip_address)

```


Then, retrieve the proof for the requested revision.

The QLDB API returns the proof as a string representation of the ordered list of node hashes. To convert this string into a list of the binary representation of the node hashes, you can use an `Ion` reader from the Amazon `Ion` library. For more information about using the `Ion` library, see the [Amazon Ion Cookbook](#).

Java

In this example, you use `IonReader` to do the binary conversion.

```
String proofText = revisionResult.getProof().getIonText();

// Take the proof and convert it to a list of byte arrays
List<byte[]> internalHashes = new ArrayList<>();
IonReader reader = SYSTEM.newReader(proofText);
reader.next();
reader.stepIn();
while (reader.next() != null) {
    internalHashes.add(reader.newBytes());
}
```

.NET

In this example, you use `IonLoader` to load the proof into an `Ion` datagram.

```
string proofText = revisionResponse.Proof.IonText;
IIonDatagram proofValue = IonLoader.Default.Load(proofText);
```

Go

In this example, you use an `Ion` reader to convert the proof to binary and to iterate through the proof's list of node hashes.

```
proofText := revisionOutput.Proof.IonText

// Use ion.Reader to iterate over the proof's node hashes
reader := ion.NewReaderString(*proofText)
// Enter the struct containing node hashes
reader.Next()
if err := reader.StepIn(); err != nil {
    panic(err)
}
```

```
}
```

Node.js

In this example, you use the `load` function to do the binary conversion.

```
let proofValue: dom.Value = load(revisionResponse.Proof.IonText);
```

Python

In this example, you use the `loads` function to do the binary conversion.

```
proof_text = proof_response.get('Proof').get('IonText')
proof_hashes = loads(proof_text)
```

Step 4: Recalculate the digest from the revision

Use the proof's list of hashes to recalculate the digest, starting with the revision hash. As long as the previously saved digest is known and trusted outside of QLDB, the integrity of the document revision is proven if the recalculated digest hash matches the saved digest hash.

Java

```
// Calculate digest
byte[] calculatedDigest = internalHashes.stream().reduce(hash.getBytes(),
    BlockHashVerification::dot);

boolean verified = Arrays.equals(expectedDigest, calculatedDigest);

if (verified) {
    System.out.printf("Successfully verified document revision for id '%s'!\n",
        metadataId);
} else {
    System.out.printf("Document revision for id '%s' verification failed!\n",
        metadataId);
    return;
}
```

.NET

```
byte[] documentHash = hash.Bytes().ToArray();
```

```

foreach (IIonValue proofHash in proofValue.GetElementAt(0))
{
    // Calculate the digest
    documentHash = Dot(documentHash, proofHash.Bytes().ToArray());
}

bool verified = expectedDigest.SequenceEqual(documentHash);

if (verified)
{
    Console.WriteLine($"Successfully verified document revision for id
'{metadataId}'!");
}
else
{
    Console.WriteLine($"Document revision for id '{metadataId}' verification
failed!");
    return;
}

```

Go

```

// Going through nodes and calculate digest
for reader.Next() {
    val, _ := reader.ByteValue()
    documentHash, err = dot(documentHash, val)
}

// Compare documentHash with the expected digest
verified := reflect.DeepEqual(documentHash, expectedDigest)

if verified {
    fmt.Printf("Successfully verified document revision for id '%s'!\n", metadataId)
} else {
    fmt.Printf("Document revision for id '%s' verification failed!\n", metadataId)
    return
}

```

Node.js

```

let documentHash: Uint8Array = hash.uInt8ArrayValue();
proofValue.elements().forEach((proofHash: dom.Value) => {
    // Calculate the digest

```

```

        documentHash = dot(documentHash, proofHash.uint8ArrayValue());
    });

    let verified: boolean = isEqual(expectedDigest, documentHash);

    if (verified) {
        console.log(`Successfully verified document revision for id '${metadataId}'!`);
    } else {
        console.log(`Document revision for id '${metadataId}' verification failed!`);
        return;
    }
}

```

Python

```

# Calculate digest
calculated_digest = reduce(dot, proof_hashes, document_hash)

verified = calculated_digest == expected_digest
if verified:
    print("Successfully verified document revision for id
    '{}!'.format(metadata_id))
else:
    print("Document revision for id '{}' verification failed!".format(metadata_id))

```

Step 5: Request a proof for the journal block

Next, you verify the journal block that contains the document revision.

Use the block address and the tip address from the digest that you saved in [Step 1](#) to submit a GetBlock request. Similar to the GetRevision request in [Step 2](#), *you must again provide the tip address from the saved digest to get a proof for the block*. This API operation returns an object that includes the block and the proof for the block.

For information about the journal block structure and its contents, see [Journal contents in Amazon QLDB](#).

Java

```

// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest()
    .withName(ledgerName)

```

```

        .withBlockAddress(new ValueHolder().withIonText(blockAddressText))
        .withDigestTipAddress(digestResult.getDigestTipAddress());

// Get a result back
GetBlockResult getBlockResult = client.getBlock(getBlockRequest);

```

.NET

```

// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest
{
    Name = ledgerName,
    BlockAddress = new ValueHolder
    {
        IonText = blockAddressText
    },
    DigestTipAddress = getDigestResponse.DigestTipAddress
};

// Get a response back
GetBlockResponse getBlockResponse = client.GetBlockAsync(getBlockRequest).Result;

```

Go

```

// Submit a request for the block
blockInput := qlldb.GetBlockInput{
    Name:          &currentLedgerName,
    BlockAddress:   &qlldb.ValueHolder{IonText: &blockAddress},
    DigestTipAddress: digestOutput.DigestTipAddress,
}

// Get a result back
blockOutput, err := client.GetBlock(&blockInput)
if err != nil {
    panic(err)
}

```

Node.js

```

// Submit a request for the block
const getBlockRequest: GetBlockRequest = {
    Name: ledgerName,

```

```

    BlockAddress: {
        IonText: dumpText(blockAddress)
    },
    DigestTipAddress: getDigestResponse.DigestTipAddress
};

// Get a response back
const getBlockResponse: GetBlockResponse = await
    qlldbClient.getBlock(getBlockRequest).promise();

```

Python

```

def block_address_to_dictionary(ion_dict):
    """
    Convert a block address from IonPyDict into a dictionary.
    Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">, sequenceNo:
    <sequenceNo>}"}

    :type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
    :param ion_dict: The block address value to convert.

    :rtype: dict
    :return: The converted dict.
    """
    block_address = {'IonText': {}}
    if not isinstance(ion_dict, str):
        py_dict = '{{strandId: "{}", sequenceNo:{}}}'.format(ion_dict['strandId'],
            ion_dict['sequenceNo'])
        ion_dict = py_dict
    block_address['IonText'] = ion_dict
    return block_address

# Submit a request for the block and get a result back
block_response = qlldb_client.get_block(Name=ledger_name,
    BlockAddress=block_address_to_dictionary(block_address),
    DigestTipAddress=digest_tip_address)

```

Then, retrieve the block hash and the proof from the result.

Java

In this example, you use `IonLoader` to load the block object into an `IonDatagram` container.

```
String blockText = getBlockResult.getBlock().getIonText();

IonDatagram datagram = SYSTEM.getLoader().load(blockText);
IonStruct ionStruct = (IonStruct)datagram.get(0);

final byte[] blockHash = ((IonBlob)ionStruct.get("blockHash")).getBytes();
```

You also use `IonLoader` to load the proof into an `IonDatagram`.

```
proofText = getBlockResult.getProof().getIonText();

// Take the proof and create a list of hash binary data
datagram = SYSTEM.getLoader().load(proofText);
ListIterator<IonValue> listIter =
    ((IonList)datagram.iterator().next()).listIterator();

internalHashes.clear();
while (listIter.hasNext()) {
    internalHashes.add(((IonBlob)listIter.next()).getBytes());
}
```

.NET

In this example, you use `IonLoader` to load the block and the proof into an `Ion datagram` for each.

```
string blockText = getBlockResponse.Block.IonText;
IIonDatagram blockValue = IonLoader.Default.Load(blockText);

// blockValue is a IonDatagram, and the first value is an IonStruct containing the
// blockHash
byte[] blockHash =
    blockValue.GetElementAt(0).GetField("blockHash").Bytes().ToArray();

proofText = getBlockResponse.Proof.IonText;
proofValue = IonLoader.Default.Load(proofText);
```

Go

In this example, you use an `Ion reader` to convert the proof to binary and to iterate through the proof's list of node hashes.

```
proofText = blockOutput.Proof.IonText

block := new(map[string]interface{})
err = ion.UnmarshalString(*blockOutput.Block.IonText, block)
if err != nil {
    panic(err)
}

blockHash := (*block)["blockHash"].([]byte)

// Use ion.Reader to iterate over the proof's node hashes
reader = ion.NewReaderString(*proofText)
// Enter the struct containing node hashes
reader.Next()
if err := reader.StepIn(); err != nil {
    panic(err)
}
```

Node.js

In this example, you use the `load` function to convert the block and the proof to binary.

```
const blockValue: dom.Value = load(getBlockResponse.Block.IonText)
let blockHash: Uint8Array = blockValue.get("blockHash").uInt8ArrayValue();

proofValue = load(getBlockResponse.Proof.IonText);
```

Python

In this example, you use the `loads` function to convert the block and the proof to binary.

```
block_text = block_response.get('Block').get('IonText')
block = loads(block_text)

block_hash = block.get('blockHash')

proof_text = block_response.get('Proof').get('IonText')
proof_hashes = loads(proof_text)
```


Step 6: Recalculate the digest from the block

Use the proof's list of hashes to recalculate the digest, starting with the block hash. As long as the previously saved digest is known and trusted outside of QLDB, the integrity of the block is proven if the recalculated digest hash matches the saved digest hash.

Java

```
// Calculate digest
calculatedDigest = internalHashes.stream().reduce(blockHash,
    BlockHashVerification::dot);

verified = Arrays.equals(expectedDigest, calculatedDigest);

if (verified) {
    System.out.printf("Block address '%s' successfully verified!\n",
        blockAddressText);
} else {
    System.out.printf("Block address '%s' verification failed!\n",
        blockAddressText);
}
```

.NET

```
foreach (IIonValue proofHash in proofValue.GetElementAt(0))
{
    // Calculate the digest
    blockHash = Dot(blockHash, proofHash.Bytes().ToArray());
}

verified = expectedDigest.SequenceEqual(blockHash);

if (verified)
{
    Console.WriteLine($"Block address '{blockAddressText}' successfully verified!");
}
else
{
    Console.WriteLine($"Block address '{blockAddressText}' verification failed!");
}
```

Go

```
// Going through nodes and calculate digest
for reader.Next() {
    val, err := reader.ByteValue()
    if err != nil {
        panic(err)
    }
    blockHash, err = dot(blockHash, val)
}

// Compare blockHash with the expected digest
verified = reflect.DeepEqual(blockHash, expectedDigest)

if verified {
    fmt.Printf("Block address '%s' successfully verified!\n", blockAddress)
} else {
    fmt.Printf("Block address '%s' verification failed!\n", blockAddress)
    return
}
```

Node.js

```
proofValue.elements().forEach((proofHash: dom.Value) => {
    // Calculate the digest
    blockHash = dot(blockHash, proofHash.uInt8ArrayValue());
});

verified = isEqual(expectedDigest, blockHash);

if (verified) {
    console.log(`Block address '${dumpText(blockAddress)}' successfully verified!`);
} else {
    console.log(`Block address '${dumpText(blockAddress)}' verification failed!`);
}
```

Python

```
# Calculate digest
calculated_digest = reduce(dot, proof_hashes, block_hash)

verified = calculated_digest == expected_digest
if verified:
```

```

        print("Block address '{}' successfully verified!".format(dumps(block_address,
                                                                    binary=False,
                                                                    omit_version_marker=True)))
    else:
        print("Block address '{}' verification failed!".format(block_address))

```

The previous code examples use the following dot function when recalculating the digest. This function takes an input of two hashes, sorts them, concatenates them, and then returns the hash of the concatenated array.

Java

```

/**
 * Takes two hashes, sorts them, concatenates them, and then returns the
 * hash of the concatenated array.
 *
 * @param h1
 *         Byte array containing one of the hashes to compare.
 * @param h2
 *         Byte array containing one of the hashes to compare.
 * @return the concatenated array of hashes.
 */
public static byte[] dot(final byte[] h1, final byte[] h2) {
    if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
        throw new IllegalArgumentException("Invalid hash.");
    }

    int byteEqual = 0;
    for (int i = h1.length - 1; i >= 0; i--) {
        byteEqual = Byte.compare(h1[i], h2[i]);
        if (byteEqual != 0) {
            break;
        }
    }

    byte[] concatenated = new byte[h1.length + h2.length];
    if (byteEqual < 0) {
        System.arraycopy(h1, 0, concatenated, 0, h1.length);
        System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
    } else {
        System.arraycopy(h2, 0, concatenated, 0, h2.length);
    }
}

```

```

        System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
    }

    MessageDigest messageDigest;
    try {
        messageDigest = MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        throw new IllegalStateException("SHA-256 message digest is unavailable", e);
    }

    messageDigest.update(concatenated);
    return messageDigest.digest();
}

```

.NET

```

/// <summary>
/// Takes two hashes, sorts them, concatenates them, and then returns the
/// hash of the concatenated array.
/// </summary>
/// <param name="h1">Byte array containing one of the hashes to compare.</param>
/// <param name="h2">Byte array containing one of the hashes to compare.</param>
/// <returns>The concatenated array of hashes.</returns>
private static byte[] Dot(byte[] h1, byte[] h2)
{
    if (h1.Length == 0)
    {
        return h2;
    }

    if (h2.Length == 0)
    {
        return h1;
    }

    HashAlgorithm hashAlgorithm = HashAlgorithm.Create("SHA256");
    HashComparer comparer = new HashComparer();
    if (comparer.Compare(h1, h2) < 0)
    {
        return hashAlgorithm.ComputeHash(h1.Concat(h2).ToArray());
    }
    else
    {

```

```

        return hashAlgorithm.ComputeHash(h2.Concat(h1).ToArray());
    }
}

private class HashComparer : IComparer<byte[]>
{
    private static readonly int HASH_LENGTH = 32;

    public int Compare(byte[] h1, byte[] h2)
    {
        if (h1.Length != HASH_LENGTH || h2.Length != HASH_LENGTH)
        {
            throw new ArgumentException("Invalid hash");
        }

        for (var i = h1.Length - 1; i >= 0; i--)
        {
            var byteEqual = (sbyte)h1[i] - (sbyte)h2[i];
            if (byteEqual != 0)
            {
                return byteEqual;
            }
        }

        return 0;
    }
}

```

Go

```

// Takes two hashes, sorts them, concatenates them, and then returns the hash of the
// concatenated array.
func dot(h1, h2 []byte) ([]byte, error) {
    compare, err := hashComparator(h1, h2)
    if err != nil {
        return nil, err
    }

    var concatenated []byte
    if compare < 0 {
        concatenated = append(h1, h2...)
    } else {
        concatenated = append(h2, h1...)
    }
}

```

```

    }

    newHash := sha256.Sum256(concatenated)
    return newHash[:], nil
}

func hashComparator(h1 []byte, h2 []byte) (int16, error) {
    if len(h1) != hashLength || len(h2) != hashLength {
        return 0, errors.New("invalid hash")
    }
    for i := range h1 {
        // Reverse index for little endianness
        index := hashLength - 1 - i

        // Handle byte being unsigned and overflow
        h1Int := int16(h1[index])
        h2Int := int16(h2[index])
        if h1Int > 127 {
            h1Int = 0 - (256 - h1Int)
        }
        if h2Int > 127 {
            h2Int = 0 - (256 - h2Int)
        }

        difference := h1Int - h2Int
        if difference != 0 {
            return difference, nil
        }
    }
    return 0, nil
}

```

Node.js

```

/**
 * Takes two hashes, sorts them, concatenates them, and calculates a digest based on
 * the concatenated hash.
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The digest calculated from the concatenated hash values.
 */
function dot(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {

```

```

        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }

    const newHashLib = createHash("sha256");

    let concatenated: Uint8Array;
    if (hashComparator(h1, h2) < 0) {
        concatenated = concatenate(h1, h2);
    } else {
        concatenated = concatenate(h2, h1);
    }
    newHashLib.update(concatenated);
    return newHashLib.digest();
}

/**
 * Compares two hashes by their signed byte values in little-endian order.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 * the first pair of non-matching
 *         bytes.
 * @throws RangeError When the hash is not the correct hash size.
 */
function hashComparator(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_SIZE || hash2.length !== HASH_SIZE) {
        throw new RangeError("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of arrays to concatenate, in the order provided.
 * @returns The concatenated array.

```

```

*/
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}

/**
 * Helper method that checks for equality between two Uint8Array.
 * @param expected Byte array containing one of the hashes to compare.
 * @param actual Byte array containing one of the hashes to compare.
 * @returns Boolean indicating equality between the two Uint8Array.
 */
function isEqual(expected: Uint8Array, actual: Uint8Array): boolean {
    if (expected === actual) return true;
    if (expected == null || actual == null) return false;
    if (expected.length !== actual.length) return false;

    for (let i = 0; i < expected.length; i++) {
        if (expected[i] !== actual[i]) {
            return false;
        }
    }
    return true;
}

```

Python

```

def dot(hash1, hash2):
    """
    Takes two hashes, sorts them, concatenates them, and then returns the
    hash of the concatenated array.

    :type hash1: bytes
    :param hash1: The hash value to compare.

```



```

:type hash2: bytes
:param hash2: The hash value to compare.

:rtype: bytes
:return: The new hash value generated from concatenated hash values.
"""
if len(hash1) != hash_length or len(hash2) != hash_length:
    raise ValueError('Illegal hash.')

hash_array1 = array('b', hash1)
hash_array2 = array('b', hash2)

difference = 0
for i in range(len(hash_array1) - 1, -1, -1):
    difference = hash_array1[i] - hash_array2[i]
    if difference != 0:
        break

if difference < 0:
    concatenated = hash1 + hash2
else:
    concatenated = hash2 + hash1

new_hash_lib = sha256()
new_hash_lib.update(concatenated)
new_digest = new_hash_lib.digest()
return new_digest

```

Run the full code example

Run the full code example as follows to perform all of the preceding steps from start to end.

Java

```

import com.amazon.ion.IonBlob;
import com.amazon.ion.IonDatagram;
import com.amazon.ion.IonList;
import com.amazon.ion.IonReader;
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;

```

```
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.GetBlockRequest;
import com.amazonaws.services.qldb.model.GetBlockResult;
import com.amazonaws.services.qldb.model.GetDigestRequest;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.ListIterator;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.qldb.QldbSessionClient;
import software.amazon.awssdk.services.qldb.QldbSessionClientBuilder;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;

public class BlockHashVerification {
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    private static final QldbDriver driver = createQldbDriver();
    private static final AmazonQLDB client =
AmazonQLDBClientBuilder.standard().build();
    private static final String region = "us-east-1";
    private static final String ledgerName = "vehicle-registration";
    private static final String tableName = "VehicleRegistration";
    private static final String vin = "KM8SRDHF6EU074761";
    private static final int HASH_LENGTH = 32;

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver() {
        QldbSessionClientBuilder sessionClientBuilder = QldbSessionClient.builder();
        sessionClientBuilder.region(Region.of(region));
    }
}
```

```

        return QldbDriver.builder()
            .ledger(ledgerName)
            .sessionClientBuilder(sessionClientBuilder)
            .build();
    }

    /**
     * Takes two hashes, sorts them, concatenates them, and then returns the
     * hash of the concatenated array.
     *
     * @param h1
     *         Byte array containing one of the hashes to compare.
     * @param h2
     *         Byte array containing one of the hashes to compare.
     * @return the concatenated array of hashes.
     */
    public static byte[] dot(final byte[] h1, final byte[] h2) {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }

        int byteEqual = 0;
        for (int i = h1.length - 1; i >= 0; i--) {
            byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                break;
            }
        }

        byte[] concatenated = new byte[h1.length + h2.length];
        if (byteEqual < 0) {
            System.arraycopy(h1, 0, concatenated, 0, h1.length);
            System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
        } else {
            System.arraycopy(h2, 0, concatenated, 0, h2.length);
            System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
        }

        MessageDigest messageDigest;
        try {
            messageDigest = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException("SHA-256 message digest is unavailable",
e);

```

```

    }

    messageDigest.update(concatenated);
    return messageDigest.digest();
}

public static void main(String[] args) {
    // Get a digest
    GetDigestRequest digestRequest = new
    GetDigestRequest().withName(ledgerName);
    GetDigestResult digestResult = client.getDigest(digestRequest);

    java.nio.ByteBuffer digest = digestResult.getDigest();

    // expectedDigest is the buffer we will use later to compare against our
    calculated digest
    byte[] expectedDigest = new byte[digest.remaining()];
    digest.get(expectedDigest);

    // Retrieve info for the given vin's document revisions
    Result result = driver.execute(txn -> {
        final String query = String.format("SELECT blockAddress, hash,
        metadata.id FROM _ql_committed_%s WHERE data.VIN = '%s'", tableName, vin);
        return txn.execute(query);
    });

    System.out.printf("Verifying document revisions for vin '%s' in table '%s'
    in ledger '%s'%n", vin, tableName, ledgerName);

    for (IonValue ionValue : result) {
        IonStruct ionStruct = (IonStruct)ionValue;

        // Get the requested fields
        IonValue blockAddress = ionStruct.get("blockAddress");
        IonBlob hash = (IonBlob)ionStruct.get("hash");
        String metadataId = ((IonString)ionStruct.get("id")).stringValue();

        System.out.printf("Verifying document revision for id '%s'%n",
        metadataId);

        String blockAddressText = blockAddress.toString();

        // Submit a request for the revision
        GetRevisionRequest revisionRequest = new GetRevisionRequest()

```

```
        .withName(ledgerName)
        .withBlockAddress(new
ValueHolder().withIonText(blockAddressText))
        .withDocumentId(metadataId)
        .withDigestTipAddress(digestResult.getDigestTipAddress());

// Get a result back
GetRevisionResult revisionResult = client.getRevision(revisionRequest);

String proofText = revisionResult.getProof().getIonText();

// Take the proof and convert it to a list of byte arrays
List<byte[]> internalHashes = new ArrayList<>();
IonReader reader = SYSTEM.newReader(proofText);
reader.next();
reader.stepIn();
while (reader.next() != null) {
    internalHashes.add(reader.newBytes());
}

// Calculate digest
byte[] calculatedDigest =
internalHashes.stream().reduce(hash.getBytes(), BlockHashVerification::dot);

boolean verified = Arrays.equals(expectedDigest, calculatedDigest);

if (verified) {
    System.out.printf("Successfully verified document revision for id
'%s'!\n", metadataId);
} else {
    System.out.printf("Document revision for id '%s' verification
failed!\n", metadataId);
    return;
}

// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest()
    .withName(ledgerName)
    .withBlockAddress(new
ValueHolder().withIonText(blockAddressText))
    .withDigestTipAddress(digestResult.getDigestTipAddress());

// Get a result back
GetBlockResult getBlockResult = client.getBlock(getBlockRequest);
```

```

        String blockText = getBlockResult.getBlock().getIonText();

        IonDatagram datagram = SYSTEM.getLoader().load(blockText);
        ionStruct = (IonStruct)datagram.get(0);

        final byte[] blockHash =
            ((IonBlob)ionStruct.get("blockHash")).getBytes();

        proofText = getBlockResult.getProof().getIonText();

        // Take the proof and create a list of hash binary data
        datagram = SYSTEM.getLoader().load(proofText);
        ListIterator<IonValue> listIter =
            ((IonList)datagram.iterator().next()).listIterator();

        internalHashes.clear();
        while (listIter.hasNext()) {
            internalHashes.add(((IonBlob)listIter.next()).getBytes());
        }

        // Calculate digest
        calculatedDigest = internalHashes.stream().reduce(blockHash,
BlockHashVerification::dot);

        verified = Arrays.equals(expectedDigest, calculatedDigest);

        if (verified) {
            System.out.printf("Block address '%s' successfully verified!\n",
blockAddressText);
        } else {
            System.out.printf("Block address '%s' verification failed!\n",
blockAddressText);
        }
    }
}
}

```

.NET

```

using Amazon.IonDotnet;
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;

```

```

using Amazon.QLDB;
using Amazon.QLDB.Driver;
using Amazon.QLDB.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;

namespace BlockHashVerification
{
    class BlockHashVerification
    {
        private static readonly string ledgerName = "vehicle-registration";
        private static readonly string tableName = "VehicleRegistration";
        private static readonly string vin = "KM8SRDHF6EU074761";
        private static readonly IQldbDriver driver =
            QldbDriver.Builder().WithLedger(ledgerName).Build();
        private static readonly IAmazonQLDB client = new AmazonQLDBClient();

        /// <summary>
        /// Takes two hashes, sorts them, concatenates them, and then returns the
        /// hash of the concatenated array.
        /// </summary>
        /// <param name="h1">Byte array containing one of the hashes to compare.</
param>
        /// <param name="h2">Byte array containing one of the hashes to compare.</
param>
        /// <returns>The concatenated array of hashes.</returns>
        private static byte[] Dot(byte[] h1, byte[] h2)
        {
            if (h1.Length == 0)
            {
                return h2;
            }

            if (h2.Length == 0)
            {
                return h1;
            }

            HashAlgorithm hashAlgorithm = HashAlgorithm.Create("SHA256");
            HashComparer comparer = new HashComparer();
            if (comparer.Compare(h1, h2) < 0)

```

```

        {
            return hashAlgorithm.ComputeHash(h1.Concat(h2).ToArray());
        }
        else
        {
            return hashAlgorithm.ComputeHash(h2.Concat(h1).ToArray());
        }
    }

private class HashComparer : IComparer<byte[]>
{
    private static readonly int HASH_LENGTH = 32;

    public int Compare(byte[] h1, byte[] h2)
    {
        if (h1.Length != HASH_LENGTH || h2.Length != HASH_LENGTH)
        {
            throw new ArgumentException("Invalid hash");
        }

        for (var i = h1.Length - 1; i >= 0; i--)
        {
            var byteEqual = (sbyte)h1[i] - (sbyte)h2[i];
            if (byteEqual != 0)
            {
                return byteEqual;
            }
        }

        return 0;
    }
}

static void Main()
{
    // Get a digest
    GetDigestRequest getDigestRequest = new GetDigestRequest
    {
        Name = ledgerName
    };
    GetDigestResponse getDigestResponse =
client.GetDigestAsync(getDigestRequest).Result;

```



```

        // expectedDigest is the buffer we will use later to compare against our
        calculated digest
        MemoryStream digest = getDigestResponse.Digest;
        byte[] expectedDigest = digest.ToArray();

        // Retrieve info for the given vin's document revisions
        var result = driver.Execute(txn => {
            string query = $"SELECT blockAddress, hash, metadata.id FROM
            _ql_committed_{tableName} WHERE data.VIN = '{vin}'";
            return txn.Execute(query);
        });

        Console.WriteLine($"Verifying document revisions for vin '{vin}' in
        table '{tableName}' in ledger '{ledgerName}'");

        foreach (IIonValue ionValue in result)
        {
            IIonStruct ionStruct = ionValue;

            // Get the requested fields
            IIonValue blockAddress = ionStruct.GetField("blockAddress");
            IIonBlob hash = ionStruct.GetField("hash");
            String metadataId = ionStruct.GetField("id").StringValue;

            Console.WriteLine($"Verifying document revision for id
            '{metadataId}'");

            // Use an Ion Reader to convert block address to text
            IIonReader reader = IonReaderBuilder.Build(blockAddress);
            StringWriter sw = new StringWriter();
            IIonWriter textWriter = IonTextWriterBuilder.Build(sw);
            textWriter.WriteValues(reader);
            string blockAddressText = sw.ToString();

            // Submit a request for the revision
            GetRevisionRequest revisionRequest = new GetRevisionRequest
            {
                Name = ledgerName,
                BlockAddress = new ValueHolder
                {
                    IonText = blockAddressText
                },
                DocumentId = metadataId,
                DigestTipAddress = getDigestResponse.DigestTipAddress
            }
        }
    }
}

```

```
};

// Get a response back
GetRevisionResponse revisionResponse =
client.GetRevisionAsync(revisionRequest).Result;

string proofText = revisionResponse.Proof.IonText;
IIonDatagram proofValue = IonLoader.Default.Load(proofText);

byte[] documentHash = hash.Bytes().ToArray();
foreach (IIonValue proofHash in proofValue.GetElementAt(0))
{
    // Calculate the digest
    documentHash = Dot(documentHash, proofHash.Bytes().ToArray());
}

bool verified = expectedDigest.SequenceEqual(documentHash);

if (verified)
{
    Console.WriteLine($"Successfully verified document revision for
id '{metadataId}'!");
}
else
{
    Console.WriteLine($"Document revision for id '{metadataId}'
verification failed!");
    return;
}

// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest
{
    Name = ledgerName,
    BlockAddress = new ValueHolder
    {
        IonText = blockAddressText
    },
    DigestTipAddress = getDigestResponse.DigestTipAddress
};

// Get a response back
GetBlockResponse getBlockResponse =
client.GetBlockAsync(getBlockRequest).Result;
```

```

        string blockText = getBlockResponse.Block.IonText;
        IIonDatagram blockValue = IonLoader.Default.Load(blockText);

        // blockValue is a IonDatagram, and the first value is an IonStruct
        containing the blockHash
        byte[] blockHash =
blockValue.GetElementAt(0).GetField("blockHash").Bytes().ToArray();

        proofText = getBlockResponse.Proof.IonText;
        proofValue = IonLoader.Default.Load(proofText);

        foreach (IIonValue proofHash in proofValue.GetElementAt(0))
        {
            // Calculate the digest
            blockHash = Dot(blockHash, proofHash.Bytes().ToArray());
        }

        verified = expectedDigest.SequenceEqual(blockHash);

        if (verified)
        {
            Console.WriteLine($"Block address '{blockAddressText}'
successfully verified!");
        }
        else
        {
            Console.WriteLine($"Block address '{blockAddressText}'
verification failed!");
        }
    }
}
}
}
}

```

Go

```

package main

import (
    "context"
    "crypto/sha256"
    "errors"

```

```

    "fmt"
    "reflect"

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    AWSSession "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldb"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/aws-labs/amazon-qlldb-driver-go/qlldbdriver"
)

const (
    hashLength = 32
    ledgerName = "vehicle-registration"
    tableName  = "VehicleRegistration"
    vin        = "KM8SRDHF6EU074761"
)

// Takes two hashes, sorts them, concatenates them, and then returns the hash of the
// concatenated array.
func dot(h1, h2 []byte) ([]byte, error) {
    compare, err := hashComparator(h1, h2)
    if err != nil {
        return nil, err
    }

    var concatenated []byte
    if compare < 0 {
        concatenated = append(h1, h2...)
    } else {
        concatenated = append(h2, h1...)
    }

    newHash := sha256.Sum256(concatenated)
    return newHash[:], nil
}

func hashComparator(h1 []byte, h2 []byte) (int16, error) {
    if len(h1) != hashLength || len(h2) != hashLength {
        return 0, errors.New("invalid hash")
    }
    for i := range h1 {
        // Reverse index for little endianness
        index := hashLength - 1 - i

```

```

        // Handle byte being unsigned and overflow
        h1Int := int16(h1[index])
        h2Int := int16(h2[index])
        if h1Int > 127 {
            h1Int = 0 - (256 - h1Int)
        }
        if h2Int > 127 {
            h2Int = 0 - (256 - h2Int)
        }

        difference := h1Int - h2Int
        if difference != 0 {
            return difference, nil
        }
    }
    return 0, nil
}

func main() {
    driverSession := AWSSession.Must(AWSSession.NewSession(aws.NewConfig()))
    qlldbSession := qlldbSession.New(driverSession)
    driver, err := qlldbdriver.New(ledgerName, qlldbSession, func(options
*qlldbdriver.DriverOptions) {})
    if err != nil {
        panic(err)
    }
    client := qlldb.New(driverSession)

    // Get a digest
    currentLedgerName := ledgerName
    input := qlldb.GetDigestInput{Name: &currentLedgerName}
    digestOutput, err := client.GetDigest(&input)
    if err != nil {
        panic(err)
    }

    // expectedDigest is the buffer we will later use to compare against our
    calculated digest
    expectedDigest := digestOutput.Digest

    // Retrieve info for the given vin's document revisions
    result, err := driver.Execute(context.Background(), func(txn
qlldbdriver.Transaction) (interface{}, error) {

```

```

        statement := fmt.Sprintf(
            "SELECT blockAddress, hash, metadata.id FROM _ql_committed_%s WHERE
data.VIN = '%s'",
            tableName,
            vin)
        result, err := txn.Execute(statement)
        if err != nil {
            return nil, err
        }

        results := make([]map[string]interface{}, 0)

        // Convert the result set into a map
        for result.Next(txn) {
            var doc map[string]interface{}
            err := ion.Unmarshal(result.GetCurrentData(), &doc)
            if err != nil {
                return nil, err
            }
            results = append(results, doc)
        }
        return results, nil
    })
    if err != nil {
        panic(err)
    }
    resultSlice := result.([]map[string]interface{})

    fmt.Printf("Verifying document revisions for vin '%s' in table '%s' in ledger
'%s'\n", vin, tableName, ledgerName)

    for _, value := range resultSlice {
        // Get the requested fields
        ionBlockAddress, err := ion.MarshalText(value["blockAddress"])
        if err != nil {
            panic(err)
        }
        blockAddress := string(ionBlockAddress)
        metadataId := value["id"].(string)
        documentHash := value["hash"].([]byte)

        fmt.Printf("Verifying document revision for id '%s'\n", metadataId)

        // Submit a request for the revision

```

```
revisionInput := qlldb.GetRevisionInput{
    BlockAddress:      &qlldb.ValueHolder{IonText: &blockAddress},
    DigestTipAddress:  digestOutput.DigestTipAddress,
    DocumentId:        &metadataId,
    Name:              &currentLedgerName,
}

// Get a result back
revisionOutput, err := client.GetRevision(&revisionInput)
if err != nil {
    panic(err)
}

proofText := revisionOutput.Proof.IonText

// Use ion.Reader to iterate over the proof's node hashes
reader := ion.NewReaderString(*proofText)
// Enter the struct containing node hashes
reader.Next()
if err := reader.StepIn(); err != nil {
    panic(err)
}

// Going through nodes and calculate digest
for reader.Next() {
    val, _ := reader.ByteValue()
    documentHash, err = dot(documentHash, val)
}

// Compare documentHash with the expected digest
verified := reflect.DeepEqual(documentHash, expectedDigest)

if verified {
    fmt.Printf("Successfully verified document revision for id '%s'!\n",
metadataId)
} else {
    fmt.Printf("Document revision for id '%s' verification failed!\n",
metadataId)
    return
}

// Submit a request for the block
blockInput := qlldb.GetBlockInput{
    Name:              &currentLedgerName,
```

```
        BlockAddress:      &qldb.ValueHolder{IonText: &blockAddress},
        DigestTipAddress: digestOutput.DigestTipAddress,
    }

    // Get a result back
    blockOutput, err := client.GetBlock(&blockInput)
    if err != nil {
        panic(err)
    }

    proofText = blockOutput.Proof.IonText

    block := new(map[string]interface{})
    err = ion.UnmarshalString(*blockOutput.Block.IonText, block)
    if err != nil {
        panic(err)
    }

    blockHash := (*block)["blockHash"].([]byte)

    // Use ion.Reader to iterate over the proof's node hashes
    reader = ion.NewReaderString(*proofText)
    // Enter the struct containing node hashes
    reader.Next()
    if err := reader.StepIn(); err != nil {
        panic(err)
    }

    // Going through nodes and calculate digest
    for reader.Next() {
        val, err := reader.ByteValue()
        if err != nil {
            panic(err)
        }
        blockHash, err = dot(blockHash, val)
    }

    // Compare blockHash with the expected digest
    verified = reflect.DeepEqual(blockHash, expectedDigest)

    if verified {
        fmt.Printf("Block address '%s' successfully verified!\n", blockAddress)
    } else {
        fmt.Printf("Block address '%s' verification failed!\n", blockAddress)
    }
}
```



```

        return
    }
}
}

```

Node.js

```

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { QLDB } from "aws-sdk"
import { GetBlockRequest, GetBlockResponse, GetDigestRequest, GetDigestResponse,
  GetRevisionRequest, GetRevisionResponse } from "aws-sdk/clients/qldb";
import { createHash } from "crypto";
import { dom, dumpText, load } from "ion-js"

const ledgerName: string = "vehicle-registration";
const tableName: string = "VehicleRegistration";
const vin: string = "KM8SRDHF6EU074761";
const driver: QldbDriver = new QldbDriver(ledgerName);
const qldbClient: QLDB = new QLDB();
const HASH_SIZE = 32;

/**
 * Takes two hashes, sorts them, concatenates them, and calculates a digest based on
 * the concatenated hash.
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The digest calculated from the concatenated hash values.
 */
function dot(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }

    const newHashLib = createHash("sha256");

    let concatenated: Uint8Array;
    if (hashComparator(h1, h2) < 0) {
        concatenated = concatenate(h1, h2);
    } else {
        concatenated = concatenate(h2, h1);
    }
}

```

```

    }
    newHashLib.update(concatenated);
    return newHashLib.digest();
}

/**
 * Compares two hashes by their signed byte values in little-endian order.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 * the first pair of non-matching
 * bytes.
 * @throws RangeError When the hash is not the correct hash size.
 */
function hashComparator(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_SIZE || hash2.length !== HASH_SIZE) {
        throw new RangeError("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of arrays to concatenate, in the order provided.
 * @returns The concatenated array.
 */
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}

```

```

}

/**
 * Helper method that checks for equality between two Uint8Array.
 * @param expected Byte array containing one of the hashes to compare.
 * @param actual Byte array containing one of the hashes to compare.
 * @returns Boolean indicating equality between the two Uint8Array.
 */
function isEqual(expected: Uint8Array, actual: Uint8Array): boolean {
    if (expected === actual) return true;
    if (expected == null || actual == null) return false;
    if (expected.length !== actual.length) return false;

    for (let i = 0; i < expected.length; i++) {
        if (expected[i] !== actual[i]) {
            return false;
        }
    }
    return true;
}

const main = async function (): Promise<void> {
    // Get a digest
    const getDigestRequest: GetDigestRequest = {
        Name: ledgerName
    };
    const getDigestResponse: GetDigestResponse = await
qldbClient.getDigest(getDigestRequest).promise();

    // expectedDigest is the buffer we will later use to compare against our
    calculated digest
    const expectedDigest: Uint8Array = <Uint8Array>getDigestResponse.Digest;

    const result: dom.Value[] = await driver.executeLambda(async (txn:
TransactionExecutor): Promise<dom.Value[]> => {
        const query: string = `SELECT blockAddress, hash, metadata.id FROM
_qldb_committed_${tableName} WHERE data.VIN = '${vin}'`;
        const queryResult: Result = await txn.execute(query);
        return queryResult.getResultList();
    });

    console.log(`Verifying document revisions for vin '${vin}' in table
'${tableName}' in ledger '${ledgerName}'`);

```

```
for (let value of result) {
  // Get the requested fields
  const blockAddress: dom.Value = value.get("blockAddress");
  const hash: dom.Value = value.get("hash");
  const metadataId: string = value.get("id").stringValue();

  console.log(`Verifying document revision for id '${metadataId}'`);

  // Submit a request for the revision
  const revisionRequest: GetRevisionRequest = {
    Name: ledgerName,
    BlockAddress: {
      IonText: dumpText(blockAddress)
    },
    DocumentId: metadataId,
    DigestTipAddress: getDigestResponse.DigestTipAddress
  };

  // Get a response back
  const revisionResponse: GetRevisionResponse = await
qlldbClient.getRevision(revisionRequest).promise();

  let proofValue: dom.Value = load(revisionResponse.Proof.IonText);

  let documentHash: Uint8Array = hash.uInt8ArrayValue();
  proofValue.elements().forEach((proofHash: dom.Value) => {
    // Calculate the digest
    documentHash = dot(documentHash, proofHash.uInt8ArrayValue());
  });

  let verified: boolean = isEqual(expectedDigest, documentHash);

  if (verified) {
    console.log(`Successfully verified document revision for id
'${metadataId}'!`);
  } else {
    console.log(`Document revision for id '${metadataId}' verification
failed!`);
    return;
  }

  // Submit a request for the block
  const getBlockRequest: GetBlockRequest = {
    Name: ledgerName,
```

```

        BlockAddress: {
            IonText: dumpText(blockAddress)
        },
        DigestTipAddress: getDigestResponse.DigestTipAddress
    };

    // Get a response back
    const getBlockResponse: GetBlockResponse = await
qlldbClient.getBlock(getBlockRequest).promise();

    const blockValue: dom.Value = load(getBlockResponse.Block.IonText)
    let blockHash: Uint8Array = blockValue.get("blockHash").uInt8ArrayValue();

    proofValue = load(getBlockResponse.Proof.IonText);

    proofValue.elements().forEach((proofHash: dom.Value) => {
        // Calculate the digest
        blockHash = dot(blockHash, proofHash.uInt8ArrayValue());
    });

    verified = isEqual(expectedDigest, blockHash);

    if (verified) {
        console.log(`Block address '${dumpText(blockAddress)}' successfully
verified!`);
    } else {
        console.log(`Block address '${dumpText(blockAddress)}' verification
failed!`);
    }
}
};

if (require.main === module) {
    main();
}

```

Python

```

from amazon.ion.simpleion import dumps, loads
from array import array
from boto3 import client
from functools import reduce
from hashlib import sha256

```

```

from pyqldb.driver.qldb_driver import QldbDriver

ledger_name = 'vehicle-registration'
table_name = 'VehicleRegistration'
vin = 'KM8SRDHF6EU074761'
qldb_client = client('qldb')
hash_length = 32

def query_doc_revision(txn):
    query = "SELECT blockAddress, hash, metadata.id FROM _ql_committed_{} WHERE
data.VIN = '{}'.format(table_name, vin)
    return txn.execute_statement(query)

def block_address_to_dictionary(ion_dict):
    """
    Convert a block address from IonPyDict into a dictionary.
    Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">,
sequenceNo: <sequenceNo>}"}

    :type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
    :param ion_dict: The block address value to convert.

    :rtype: dict
    :return: The converted dict.
    """
    block_address = {'IonText': {}}
    if not isinstance(ion_dict, str):
        py_dict = '{{strandId: "{}", sequenceNo:{}}}'.format(ion_dict['strandId'],
ion_dict['sequenceNo'])
        ion_dict = py_dict
    block_address['IonText'] = ion_dict
    return block_address

def dot(hash1, hash2):
    """
    Takes two hashes, sorts them, concatenates them, and then returns the
    hash of the concatenated array.

    :type hash1: bytes
    :param hash1: The hash value to compare.

```

```

:type hash2: bytes
:param hash2: The hash value to compare.

:rtype: bytes
:return: The new hash value generated from concatenated hash values.
"""
if len(hash1) != hash_length or len(hash2) != hash_length:
    raise ValueError('Illegal hash.')

hash_array1 = array('b', hash1)
hash_array2 = array('b', hash2)

difference = 0
for i in range(len(hash_array1) - 1, -1, -1):
    difference = hash_array1[i] - hash_array2[i]
    if difference != 0:
        break

if difference < 0:
    concatenated = hash1 + hash2
else:
    concatenated = hash2 + hash1

new_hash_lib = sha256()
new_hash_lib.update(concatenated)
new_digest = new_hash_lib.digest()
return new_digest

# Get a digest
get_digest_response = qlldb_client.get_digest(Name=ledger_name)

# expected_digest is the buffer we will later use to compare against our calculated
# digest
expected_digest = get_digest_response.get('Digest')
digest_tip_address = get_digest_response.get('DigestTipAddress')

qlldb_driver = QldbDriver(ledger_name=ledger_name)

# Retrieve info for the given vin's document revisions
result = qlldb_driver.execute_lambda(query_doc_revision)

print("Verifying document revisions for vin '{}' in table '{}' in ledger
'{}'.format(vin, table_name, ledger_name))

```

```
for value in result:
    # Get the requested fields
    block_address = value['blockAddress']
    document_hash = value['hash']
    metadata_id = value['id']

    print("Verifying document revision for id '{}'.format(metadata_id))

    # Submit a request for the revision and get a result back
    proof_response = qlldb_client.get_revision(Name=ledger_name,

BlockAddress=block_address_to_dictionary(block_address),
                                         DocumentId=metadata_id,
                                         DigestTipAddress=digest_tip_address)

    proof_text = proof_response.get('Proof').get('IonText')
    proof_hashes = loads(proof_text)

    # Calculate digest
    calculated_digest = reduce(dot, proof_hashes, document_hash)

    verified = calculated_digest == expected_digest
    if verified:
        print("Successfully verified document revision for id
'{}'!".format(metadata_id))
    else:
        print("Document revision for id '{}' verification
failed!".format(metadata_id))

    # Submit a request for the block and get a result back
    block_response = qlldb_client.get_block(Name=ledger_name,
BlockAddress=block_address_to_dictionary(block_address),
                                         DigestTipAddress=digest_tip_address)

    block_text = block_response.get('Block').get('IonText')
    block = loads(block_text)

    block_hash = block.get('blockHash')

    proof_text = block_response.get('Proof').get('IonText')
    proof_hashes = loads(proof_text)

    # Calculate digest
```



```
calculated_digest = reduce(dot, proof_hashes, block_hash)

verified = calculated_digest == expected_digest
if verified:
    print("Block address '{}' successfully
verified!".format(dumps(block_address,
                                                                    binary=False,
                                                                    omit_version_marker=True)))
else:
    print("Block address '{}' verification failed!".format(block_address))
```

Common errors for verification

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes runtime errors that are thrown by Amazon QLDB for verification requests.

The following is a list of common exceptions returned by the service. Each exception includes the specific error message, followed by the API operations that can throw it, a short description, and suggestions for possible solutions.

IllegalArgumentException

Message: The provided Ion value is not valid and cannot be parsed.

API operations: GetDigest, GetBlock, GetRevision

Make sure that you provide a valid [Amazon Ion](#) value before retrying your request.

IllegalArgumentException

Message: The provided block address is not valid.

API operations: GetDigest, GetBlock, GetRevision

Make sure that you provide a valid block address before retrying your request. A block address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"B1FTj1SXze9BIh1K0szcE3",sequenceNo:14}`

IllegalArgumentException

Message: The sequence number of the provided digest tip address is beyond the strand's latest committed record.

API operations: `GetDigest`, `GetBlock`, `GetRevision`

The digest tip address that you provide must have a sequence number less than or equal to the sequence number of the journal strand's latest committed record. Before retrying your request, make sure that you provide a digest tip address with a valid sequence number.

IllegalArgumentException

Message: The Strand ID of the provided block address is not valid.

API operations: `GetDigest`, `GetBlock`, `GetRevision`

The block address that you provide must have a strand ID that matches the journal's strand ID. Before retrying your request, make sure that you provide a block address with a valid strand ID.

IllegalArgumentException

Message: The sequence number of the provided block address is beyond the strand's latest committed record.

API operations: `GetBlock`, `GetRevision`

The block address that you provide must have a sequence number less than or equal to the sequence number of the strand's latest committed record. Before retrying your request, make sure that you provide a block address with a valid sequence number.

IllegalArgumentException

Message: The Strand ID of the provided block address must match the Strand ID of the provided digest tip address.

API operations: `GetBlock`, `GetRevision`

You can only verify a document revision or block if it exists in the same journal strand as the digest that you provide.

IllegalArgumentException

Message: The sequence number of the provided block address must not be greater than the sequence number of the provided digest tip address.

API operations: `GetBlock`, `GetRevision`

You can only verify a document revision or block if it's covered by the digest that you provide. This means that it was committed to the journal before the digest tip address.

IllegalArgumentException

Message: The provided Document ID was not found in the block at the specified block address.

API operation: `GetRevision`

The document ID that you provide must exist in the block address that you provide. Before retrying your request, make sure that these two parameters are consistent.

Exporting journal data from Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB uses an immutable transactional log, known as a *journal*, for data storage. The journal tracks every change to your committed data and maintains a complete and verifiable history of changes over time.

You can access the contents of the journal in your ledger for various purposes including analytics, auditing, data retention, verification, and exporting to other systems. The following topics describe how to export journal [blocks](#) from your ledger into an Amazon Simple Storage Service (Amazon S3) bucket in your AWS account. A journal export job writes your data in Amazon S3 as objects in either the text or binary representation of [Amazon Ion](#) format, or in *JSON Lines* text format.

In JSON Lines format, each block in an exported data object is a valid JSON object that is delimited by a newline. You can use this format to directly integrate JSON exports with analytics tools such as Amazon Athena and AWS Glue because these services can parse newline-delimited JSON automatically. For more information about the format, see [JSON Lines](#).

For information about Amazon S3, see the [Amazon Simple Storage Service User Guide](#).

Note

If you specify JSON as the output format of your export job, QLDB down-converts the Ion journal data to JSON in your exported data objects. For more information, see [Down-converting to JSON](#).

Topics

- [Requesting a journal export in QLDB](#)
- [Journal export output in QLDB](#)

- [Journal export permissions in QLDB](#)
- [Common errors for journal export](#)

Requesting a journal export in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB provides an API to request an export of your journal blocks for a specified date and time range and a specified Amazon S3 bucket destination. A journal export job can write the data objects in either the text or binary representation of [Amazon Ion](#) format, or in [JSON Lines](#) text format. You can use the AWS Management Console, an AWS SDK, or the AWS Command Line Interface (AWS CLI) to create an export job.

Topics

- [AWS Management Console](#)
- [QLDB API](#)
- [Export job expiration](#)

AWS Management Console

Follow these steps to submit a journal export request in QLDB using the QLDB console.

To request an export (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Export**.
3. Choose **Create export job**.
4. On the **Create export job** page, enter the following export settings:
 - **Ledger** – The ledger whose journal blocks you want to export.

- **Start date and time** – The inclusive start timestamp in Coordinated Universal Time (UTC) of the range of journal blocks to export. This timestamp must be earlier than the **End date and time**. If you provide a start timestamp that is earlier than the ledger's `CreationDateTime`, QLDB defaults it to the ledger's `CreationDateTime`.
- **End date and time** – The exclusive end timestamp (UTC) of the range of journal blocks to export. This date and time can't be in the future.
- **Destination for journal blocks** – The Amazon S3 bucket and prefix name in which your export job writes the data objects. Use the following Amazon S3 URI format.

```
s3://amzn-s3-demo-qldb-cloudtrail/prefix/
```

You must specify an S3 bucket name and an optional prefix name for the output objects. The following is an example.

```
s3://amzn-s3-demo-qldb-cloudtrail/journalExport/
```

The bucket name and prefix must both comply with the Amazon S3 naming rules and conventions. For more information about bucket naming, see [Bucket restrictions and limitations](#) in the *Amazon S3 Developer Guide*. For more information about key name prefixes, see [Object key and metadata](#).

 Note

Cross-region exports are not supported. The specified Amazon S3 bucket must be in the same AWS Region as your ledger.

- **S3 Encryption** – The encryption settings that are used by your export job to write data in an Amazon S3 bucket. To learn more about server-side encryption options in Amazon S3, see [Protecting data using server-side encryption](#) in the *Amazon S3 Developer Guide*.
 - **Bucket default encryption** – Use the default encryption settings of the specified Amazon S3 bucket.
 - **AES-256** – Use server-side encryption with Amazon S3 managed keys (SSE-S3).
 - **AWS-KMS** – Use server-side encryption with AWS KMS managed keys (SSE-KMS).

If you choose this type along with the **Choose a different AWS KMS key** option, you must also specify a symmetric encryption KMS key in the following Amazon Resource Name (ARN) format.

```
arn:aws:kms:aws-region:account-id:key/key-id
```

- **Service access** – The IAM role that grants QLDB write permissions in your Amazon S3 bucket. If applicable, the IAM role must also grant QLDB permissions to use your KMS key.

To pass a role to QLDB when requesting a journal export, you must have permissions to perform the `iam:PassRole` action on the IAM role resource.

- **Create and use a new service role** – Let the console create a new role for you with the required permissions for the specified Amazon S3 bucket.
- **Use an existing service role** – To learn how to manually create this role in IAM, see [Export permissions](#).
- **Output format** – The output format of your exported journal data
 - **Ion text** – (Default) Text representation of Amazon Ion
 - **Ion binary** – Binary representation of Amazon Ion
 - **JSON** – Newline-delimited JSON text format

If you choose JSON, QLDB down-converts the Ion journal data to JSON in your exported data objects. For more information, see [Down-converting to JSON](#).

5. When the settings are as you want them, choose **Create export job**.

The amount of time it takes for your export job to finish varies depending on the data size. If your request submission is successful, the console returns to the main **Export** page and lists your export jobs with their current status.

6. You can see your export objects on the Amazon S3 console.

Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

To learn more about the format of these output objects, see [Journal export output in QLDB](#).

Note

Export jobs expire seven days after they complete. For more information, see [Export job expiration](#).

QLDB API

You can also request a journal export by using the Amazon QLDB API with an AWS SDK or the AWS CLI. The QLDB API provides the following operations for use by application programs:

- `ExportJournalToS3` – Exports journal contents within a date and time range from a given ledger into a specified Amazon S3 bucket. An export job can write the data as objects in either the text or binary representation of Amazon Ion format, or in JSON Lines text format.
- `DescribeJournalS3Export` – Returns detailed information about a journal export job. The output includes its current status, creation time, and the parameters of your original export request.
- `ListJournalS3Exports` – Returns a list of journal export job descriptions for all ledgers that are associated with the current AWS account and Region. The output of each export job description includes the same details that are returned by `DescribeJournalS3Export`.
- `ListJournalS3ExportsForLedger` – Returns a list of journal export job descriptions for a given ledger. The output of each export job description includes the same details that are returned by `DescribeJournalS3Export`.

For complete descriptions of these API operations, see the [Amazon QLDB API reference](#).

For information about exporting journal data using the AWS CLI, see the [AWS CLI Command Reference](#).

Sample application (Java)

For Java code examples of basic export operations, see the GitHub repository [aws-samples/amazon-qldb-dmv-sample-java](#). For instructions on how to download and install this sample application, see [Installing the Amazon QLDB Java sample application](#). Before requesting an export, make sure that you follow Steps 1–3 in the [Java tutorial](#) to create a sample ledger and load it with sample data.

The tutorial code in the following classes provide examples of creating an export, checking the status of an export, and processing the output of an export.

Class	Description
ExportJournal	Exports journal blocks from the vehicle-registration sample ledger for a timestamp range of 10 minutes ago until now. Writes the output objects in a specified S3 bucket, or creates a unique bucket if one isn't provided.
DescribeJournalExport	Describes a journal export job for a specified exportId in the vehicle-registration sample ledger.
ListJournalExports	Returns a list of journal export job descriptions for the vehicle-registration sample ledger.
ValidateQldbHashChain	Validates the hash chain of the vehicle-registration sample ledger using a given exportId. If not provided, requests a new export to use for hash chain validation.

Export job expiration

Completed journal export jobs are subject to a 7-day retention period. They're automatically hard-deleted after this limit expires. This expiration period is a hard limit and can't be changed.

After a completed export job is deleted, you can no longer use the QLDB console or the following API operations to retrieve metadata about the job:

- `DescribeJournalS3Export`
- `ListJournalS3Exports`
- `ListJournalS3ExportsForLedger`

However, this expiration has no impact on the exported data itself. All of the metadata is preserved in the manifest files that are written by your exports. This expiration is designed to provide a smoother experience for the API operations that list journal export jobs. QLDB removes old export jobs to ensure that you only see recent exports without having to parse multiple pages of jobs.

Journal export output in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

An Amazon QLDB journal export job writes two manifest files in addition to the data objects that contain your journal blocks. These are all saved in the Amazon S3 bucket that you provided in your [export request](#). The following sections describe the format and contents of each output object.

Note

If you specify JSON as the output format of your export job, QLDB down-converts the Amazon Ion journal data to JSON in your exported data objects. For more information, proceed to [Down-converting to JSON](#).

Topics

- [Manifest files](#)
- [Data objects](#)
- [Down-converting to JSON](#)
- [Export processor library \(Java\)](#)

Manifest files

Amazon QLDB creates two manifest files in the provided S3 bucket for each export request. The *initial manifest* file is created as soon as you submit the export request. The *final manifest* file is

written after the export is complete. You can use these files to check the status of your export jobs in Amazon S3.

The format for the contents of the manifest files corresponds to the requested output format for the export.

Initial manifest

The initial manifest indicates that your export job has started. It contains the input parameters that you passed to the request. In addition to the Amazon S3 destination and the start and end time parameters for the export, this file also contains an `exportId`. The `exportId` is a unique ID that QLDB assigns to each export job.

The file-naming convention is as follows.

```
s3://amzn-s3-demo-qldb-cloudtrail/prefix/exportId.started.manifest
```

The following is an example of an initial manifest file and its contents in Ion text format.

```
s3://amzn-s3-demo-qldb-cloudtrail/journalExport/8UyXulxccYLAsbN1aon7e4.started.manifest
```

```
{
  ledgerName:"my-example-ledger",
  exportId:"8UyXulxccYLAsbN1aon7e4",
  inclusiveStartTime:2019-04-15T00:00:00.000Z,
  exclusiveEndTime:2019-04-15T22:00:00.000Z,
  bucket:"amzn-s3-demo-qldb-cloudtrail",
  prefix:"journalExport",
  objectEncryptionType:"NO_ENCRYPTION",
  outputFormat:"ION_TEXT"
}
```

The initial manifest includes the `outputFormat` only if it was specified in the export request. If you don't specify the output format, the exported data defaults to `ION_TEXT` format.

The [DescribeJournalS3Export](#) API operation and the content type of the exported Amazon S3 objects also indicate the output format.

Final manifest

The final manifest indicates that your export job for a particular journal *strand* has completed. The export job writes a separate final manifest file for each strand.

Note

In Amazon QLDB, a strand is a partition of your ledger's journal. QLDB currently supports journals with a single strand only.

The final manifest includes an ordered list of data object keys that were written during the export. The file naming convention is as follows.

```
s3://amzn-s3-demo-qldb-cloudtrail/prefix/exportId.strandId.completed.manifest
```

The `strandId` is a unique ID that QLDB assigns to the strand. The following is an example of a final manifest file and its contents in Ion text format.

```
s3://amzn-s3-demo-qldb-cloudtrail/  
journalExport/8UyXulxccYLAbsN1aon7e4.Jdxjkr9bSYB5jMHwCI464T.completed.manifest
```

```
{  
  keys:[  
    "2019/04/15/22/Jdxjkr9bSYB5jMHwCI464T.1-4.ion",  
    "2019/04/15/22/Jdxjkr9bSYB5jMHwCI464T.5-10.ion",  
    "2019/04/15/22/Jdxjkr9bSYB5jMHwCI464T.11-12.ion",  
    "2019/04/15/22/Jdxjkr9bSYB5jMHwCI464T.13-20.ion",  
    "2019/04/15/22/Jdxjkr9bSYB5jMHwCI464T.21-21.ion"  
  ]  
}
```

Data objects

Amazon QLDB writes journal data objects in the provided Amazon S3 bucket in either the text or binary representation of Amazon Ion format, or in *JSON Lines* text format.

In JSON Lines format, each block in an exported data object is a valid JSON object that is delimited by a newline. You can use this format to directly integrate JSON exports with analytics tools

such as Amazon Athena and AWS Glue because these services can parse newline-delimited JSON automatically. For more information about the format, see [JSON Lines](#).

Data object names

A journal export job writes these data objects with the following naming convention.

```
s3://amzn-s3-demo-qldb-cloudtrail/prefix/yyyy/mm/dd/hh/strandId.startSn-endSn.ion|.json
```

- The output data of each export job is broken up into chunks.
- yyyy/mm/dd/hh – The date and time when you submitted the export request. Objects that are exported within the same hour are grouped under the same Amazon S3 prefix.
- strandId – The unique ID of the particular strand that contains the journal block that is being exported.
- startSn-endSn – The sequence number range that is included in the object. A sequence number specifies the location of a block within a strand.

For example, suppose that you specify the following path.

```
s3://amzn-s3-demo-qldb-cloudtrail/journalExport/
```

Your export job creates an Amazon S3 data object that looks similar to the following. This example shows an object name in Ion format.

```
s3://amzn-s3-demo-qldb-cloudtrail/journalExport/2019/04/15/22/  
Jdxjkr9bSYB5jMHWcI464T.1-5.ion
```

Data object contents

Each data object contains journal block objects with the following format.

```
{  
  blockAddress: {  
    strandId: String,  
    sequenceNo: Int  
  },  
  transactionId: String,  
  blockTimestamp: Datetime,  
  blockHash: SHA256,
```

```

entriesHash: SHA256,
previousBlockHash: SHA256,
entriesHashList: [ SHA256 ],
transactionInfo: {
  statements: [
    {
      //PartiQL statement object
    }
  ],
  documents: {
    //document-table-statement mapping object
  }
},
revisions: [
  {
    //document revision object
  }
]
}

```

A *block* is an object that is committed to the journal during a transaction. A block contains transaction metadata along with entries that represent the document revisions that were committed in the transaction and the [PartiQL](#) statements that committed them.

The following is an example of a block with sample data in Ion text format. For information about the fields in a block object, see [Journal contents in Amazon QLDB](#).

Note

This block example is provided for informational purposes only. The hashes shown aren't real calculated hash values.

```

{
  blockAddress:{
    strandId:"JdxjkR9bSYB5jMHWcI464T",
    sequenceNo:1234
  },
  transactionId:"D35qctdJRU1L1N2VhxbwSn",
  blockTimestamp:2019-10-25T17:20:21.009Z,
  blockHash:{WYLOfZC1k0lYWT3lUsSr00NXh+Pw8MxxB+9zvTgSv1Q=}},
  entriesHash:{xN9X96atkMvhvF3nEy6jMSVQzKjHJfz1H3bsNeg8GMA=}},

```

```

previousBlockHash:{{IAfZ0h22ZjvcuHPSBCDy/6XNQTsQEmeY3GW0gBae8mg=}},
entriesHashList:[
  {{F7rQIKCnn0vXVWPexilGfJn5+MCrtsSQqqVdlQxXpS4=}},
  {{C+L8gRhkzVcxt3qRJpw8w6hVEqA5A6ImGne+E7iHizo=}}
],
transactionInfo:{
  statements:[
    {
      statement:"CREATE TABLE VehicleRegistration",
      startTime:2019-10-25T17:20:20.496Z,
      statementDigest:{{3jeSdej0gp6spJ8huZxDRUtp2fRXRqpOMtG43V0nXg8=}}
    },
    {
      statement:"CREATE INDEX ON VehicleRegistration (VIN)",
      startTime:2019-10-25T17:20:20.549Z,
      statementDigest:{{099D+5ZWDgA7r+aWeNUrWhc8ebBTXjgscq+mZ2dVibI=}}
    },
    {
      statement:"CREATE INDEX ON VehicleRegistration (LicensePlateNumber)",
      startTime:2019-10-25T17:20:20.560Z,
      statementDigest:{{B73tVJzVyVXicnH4n96NzU2L2JFY8e9Tjg895suWMew=}}
    },
    {
      statement:"INSERT INTO VehicleRegistration ?",
      startTime:2019-10-25T17:20:20.595Z,
      statementDigest:{{ggpon5qCXLo95K578YVhAD8ix0A0M5CcBx/W40Ey/Tk=}}
    }
  ],
  documents:{
    '8F0TPCmdNQ6JTRpiLj2TmW':{
      tableName:"VehicleRegistration",
      tableId:"BPxNiDQXCIB5l5F68KZo0z",
      statements:[3]
    }
  },
  revisions:[
    {
      hash:{{FR1IwcWew0yw1TnRk1o2YMF/qtwb7ohsu5FD8A4DSVg=}}
    },
    {
      blockAddress:{
        strandId:"JdxjkR9bSYB5jMHwcI464T",
        sequenceNo:1234
      }
    }
  ]
}

```

```

    },
    hash:{{t8Hj6/VC4SBitxnvBqJb0mrGytF2XAA/1c0AoSq2NQY=}},
    data:{
      VIN:"1N4AL11D75C109151",
      LicensePlateNumber:"LEWISR261LL",
      State:"WA",
      City:"Seattle",
      PendingPenaltyTicketAmount:90.25,
      ValidFromDate:2017-08-21,
      ValidToDate:2020-05-11,
      Owners:{
        PrimaryOwner:{
          PersonId:"GddsXfIYfDlKCEprOL0wYt"
        },
        SecondaryOwners:[]
      }
    },
    metadata:{
      id:"8F0TPCmdNQ6JTRpiLj2TmW",
      version:0,
      txTime:2019-10-25T17:20:20.618Z,
      txId:"D35qctdJRU1L1N2VhxbwSn"
    }
  }
]
}

```

In the `revisions` field, some revision objects might only contain a hash value and no other attributes. These are internal-only system revisions that don't contain user data. An export job includes these revisions in their respective blocks because the hashes of these revisions are part of the journal's full hash chain. The full hash chain is required for cryptographic verification.

Down-converting to JSON

If you specify JSON as the output format of your export job, QLDB down-converts the Amazon Ion journal data to JSON in your exported data objects. However, converting Ion to JSON is lossy in certain cases where your data uses the rich Ion types that don't exist in JSON.

For details about Ion to JSON conversion rules, see [Down-converting to JSON](#) in the *Amazon Ion Cookbook*.

Export processor library (Java)

QLDB provides an extensible framework for Java that streamlines the processing of exports in Amazon S3. This framework library handles the work of reading an export's output and iterating through the exported blocks in sequential order. To use this export processor, see the GitHub repository [awslabs/amazon-qldb-export-processor-java](https://github.com/aws-labs/amazon-qldb-export-processor-java).

Journal export permissions in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Before submitting a journal export request in Amazon QLDB, you must provide QLDB with write permissions in your specified Amazon S3 bucket. If you choose a customer managed AWS KMS key as the object encryption type for your Amazon S3 bucket, you must also provide QLDB with permissions to use your specified symmetric encryption key. Amazon S3 doesn't support [asymmetric KMS keys](#).

To provide your export job with the necessary permissions, you can make QLDB assume an IAM service role with the appropriate permissions policies. A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Note

To pass a role to QLDB when requesting a journal export, you must have permissions to perform the `iam:PassRole` action on the IAM role resource. This is in addition to the `qldb:ExportJournalToS3` permission on the QLDB ledger resource.

To learn how to control access to QLDB using IAM, see [How Amazon QLDB works with IAM](#). For a QLDB policy example, see [Identity-based policy examples for Amazon QLDB](#).

In this example, you create a role that allows QLDB to write objects into an Amazon S3 bucket on your behalf. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

If you're exporting a QLDB journal in your AWS account for the first time, you must first create an IAM role with the appropriate policies by doing the following. Or, you can [use the QLDB console](#) to automatically create the role for you. Otherwise, you can choose a role that you previously created.

Topics

- [Create a permissions policy](#)
- [Create an IAM role](#)

Create a permissions policy

Complete the following steps to create a permissions policy for a QLDB journal export job. This example shows an Amazon S3 bucket policy that grants QLDB permissions to write objects into your specified bucket. If applicable, the example also shows a key policy that allows QLDB to use your symmetric encryption KMS key.

For more information about Amazon S3 bucket policies, see [Using bucket policies and user policies](#) in the *Amazon Simple Storage Service User Guide*. To learn more about AWS KMS key policies, see [Using key policies in AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

Note

Your Amazon S3 bucket and KMS key must both be in the same AWS Region as your QLDB ledger.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation column on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. At the top of the page, choose **Create policy**.
4. Choose the **JSON** tab.
5. Enter a JSON policy document.
 - If you're using a customer managed KMS key for Amazon S3 object encryption, use the following example policy document. To use this policy, replace *amzn-s3-demo-qldb-cloudtrail*, *us-east-1*, *123456789012*, and *1234abcd-12ab-34cd-56ef-1234567890ab* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBJournalExportS3Permission",
      "Action": [
        "s3:PutObjectAcl",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::amzn-s3-demo-qldb-cloudtrail/*"
    },
    {
      "Sid": "QLDBJournalExportKMSPermission",
      "Action": [ "kms:GenerateDataKey" ],
      "Effect": "Allow",
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ]
}
```

- For other encryption types, use the following example policy document. To use this policy, replace *amzn-s3-demo-qldb-cloudtrail* in the example with your own Amazon S3 bucket name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBJournalExportS3Permission",
      "Action": [
```

```
        "s3:PutObjectAcl",
        "s3:PutObject"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:s3:::amzn-s3-demo-qldb-cloudtrail/*"
}
]
```

6. Choose **Review policy**.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring](#) in the *IAM User Guide*.

7. On the **Review policy** page, enter a **Name** and an optional **Description** for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

Create an IAM role

After creating a permissions policy for your QLDB journal export job, you can then create an IAM role and attach your policy to it.

To create the service role for QLDB (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **QLDB**, and then choose the **QLDB** use case.
5. Choose **Next**.
6. Select the box next to the policy that you created in the previous steps.
7. (Optional) Set a [permissions boundary](#). This is an advanced feature that is available for service roles, but not service-linked roles.

- a. Open the **Set permissions boundary** section, and then choose **Use a permissions boundary to control the maximum role permissions**.

IAM includes a list of the AWS managed and customer-managed policies in your account.

- b. Select the policy to use for the permissions boundary.
8. Choose **Next**.
9. Enter a role name or a role name suffix to help you identify the purpose of the role.

Important

When you name a role, note the following:

- Role names must be unique within your AWS account, and can't be made unique by case.

For example, don't create roles named both **PRODRROLE** and **prodrole**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.

- You can't edit the name of the role after it's created because other entities might reference the role.

10. (Optional) For **Description**, enter a description for the role.
11. (Optional) To edit the use cases and permissions for the role, in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.
12. (Optional) To help identify, organize, or search for the role, add tags as key-value pairs. For more information about using tags in IAM, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.
13. Review the role, and then choose **Create role**.

The following JSON document is an example of a trust policy that allows QLDB to assume an IAM role with specific permissions attached to it.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "qldb.amazonaws.com"
  },
  "Action": [ "sts:AssumeRole" ],
  "Condition": {
    "ArnEquals": {
      "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:*"
    },
    "StringEquals": {
      "aws:SourceAccount": "123456789012"
    }
  }
}
```

Note

This trust policy example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to prevent the *confused deputy* problem. With this trust policy, QLDB can assume the role for any QLDB resource in the account 123456789012 only.

For more information, see [Cross-service confused deputy prevention](#).

After creating your IAM role, return to the QLDB console and refresh the **Create export job** page so that it can find your new role.

Common errors for journal export

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes runtime errors that are thrown by Amazon QLDB for journal export requests.

The following is a list of common exceptions returned by the service. Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

AccessDeniedException

Message: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN*

You don't have permissions to pass an IAM role to the QLDB service. QLDB requires a role for all journal export requests, and you must have permissions to pass this role to QLDB. The role provides QLDB with write permissions in your specified Amazon S3 bucket.

Verify that you define an IAM policy that grants permission to perform the PassRole API operation on your specified IAM role resource for the QLDB service (`qldb.amazonaws.com`). For a policy example, see [Identity-based policy examples for Amazon QLDB](#).

IllegalArgumentException

Message: QLDB encountered an error validating S3 configuration: *errorCode errorMessage*

A possible cause for this error is that the provided Amazon S3 bucket doesn't exist in Amazon S3. Or, QLDB doesn't have enough permissions to write objects into your specified Amazon S3 bucket.

Verify that the S3 bucket name that you provide in your export job request is correct. For more information about bucket naming, see [Bucket restrictions and limitations](#) in the *Amazon Simple Storage Service User Guide*.

Also, verify that you define a policy for your specified bucket that grants PutObject and PutObjectAcl permissions to the QLDB service (`qldb.amazonaws.com`). To learn more, see [Export permissions](#).

IllegalArgumentException

Message: Unexpected response from Amazon S3 while validating the S3 configuration.

Response from S3: *errorCode errorMessage*

The attempt to write journal export data into the provided S3 bucket failed with the provided Amazon S3 error response. For more information about possible causes, see [Troubleshooting Amazon S3](#) in the *Amazon Simple Storage Service User Guide*.

IllegalArgumentException

Message: Amazon S3 bucket prefix must not exceed 128 characters

The prefix provided in the journal export request contains more than 128 characters.

IllegalArgumentException

Message: Start date must not be greater than end date

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in [ISO 8601](#) date and time format and in Coordinated Universal Time (UTC).

IllegalArgumentException

Message: End date cannot be in the future

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in ISO 8601 date and time format and in UTC.

IllegalArgumentException

Message: The supplied object encryption setting (`S3EncryptionConfiguration`) is not compatible with an AWS Key Management Service (AWS KMS) key

You provided a `KMSKeyArn` with an `ObjectEncryptionType` of either `NO_ENCRYPTION` or `SSE_S3`. You can only provide a customer managed AWS KMS key for an object encryption type of `SSE_KMS`. To learn more about server-side encryption options in Amazon S3, see [Protecting data using server-side encryption](#) in the *Amazon S3 Developer Guide*.

LimitExceededException

Message: Exceeded the limit of 2 concurrently running Journal export jobs

QLDB enforces a default limit of two concurrent journal export jobs.

Streaming journal data from Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB uses an immutable transactional log, known as a *journal*, for data storage. The journal tracks every change to your committed data and maintains a complete and verifiable history of changes over time.

You can create a *stream* in QLDB that captures every document revision that is committed to your journal and delivers this data to [Amazon Kinesis Data Streams](#) in near-real time. A QLDB stream is a continuous flow of data from your ledger's journal to a Kinesis data stream resource.

Then, you use the Kinesis streaming platform or the *Kinesis Client Library* to consume your stream, process the data records, and analyze the data contents. A QLDB stream writes your data to Kinesis Data Streams in three types of records: *control*, *block summary*, and *revision details*. For more information, see [QLDB stream records in Kinesis](#).

Topics

- [Common use cases](#)
- [Consuming your stream](#)
- [Delivery guarantee](#)
- [Delivery latency considerations](#)
- [Getting started with streams](#)
- [Creating and managing streams in QLDB](#)
- [Developing with streams in QLDB](#)
- [QLDB stream records in Kinesis](#)
- [Stream permissions in QLDB](#)
- [Common errors for journal streams in QLDB](#)

Common use cases

Streaming lets you use QLDB as a single, verifiable source of truth while integrating your journal data with other services. The following are some of the common use cases supported by QLDB journal streams:

- **Event-driven architecture** – Build applications in an event-driven architectural style with decoupled components. For example, a bank can use AWS Lambda functions to implement a notification system that alerts customers when their account balance drops below a threshold. In such a system, the account balances are maintained in a QLDB ledger, and any balance changes are recorded in the journal. The AWS Lambda function can trigger the notification logic upon consuming a balance update event that is committed to the journal and sent to a Kinesis data stream.
- **Real-time analytics** – Build Kinesis consumer applications that run real-time analytics on event data. With this capability, you can gain insights in near-real time and respond quickly to a changing business environment. For example, an ecommerce website can analyze product sales data and stop advertisements for a discounted product as soon as sales reach a limit.
- **Historical analytics** – Take advantage of the journal-oriented architecture of Amazon QLDB by replaying historical event data. You can choose to start a QLDB stream as of any point in time in the past, in which all revisions since that time are delivered to Kinesis Data Streams. Using this feature, you can build Kinesis consumer applications that run analytics jobs on historical data. For example, an ecommerce website can run analytics as needed to generate past sales metrics that were not previously captured.
- **Replication to purpose-built databases** – Connect QLDB ledgers to other purpose-built data stores using QLDB journal streams. For example, use the Kinesis streaming data platform to integrate with Amazon OpenSearch Service, which can provide full text search capabilities for QLDB documents. You can also build custom Kinesis consumer applications to replicate your journal data to other purpose-built databases that provide different materialized views. For example, replicate to Amazon Aurora for relational data or to Amazon Neptune for graph-based data.

Consuming your stream

Use Kinesis Data Streams to continuously consume, process, and analyze large streams of data records. In addition to Kinesis Data Streams, the Kinesis streaming data platform includes [Amazon Data Firehose](#) and [Amazon Managed Service for Apache Flink](#). You can use this platform to send

data records directly to services such as Amazon OpenSearch Service, Amazon Redshift, Amazon S3, or Splunk. For more information, see [Kinesis Data Streams consumers](#) in the *Amazon Kinesis Data Streams Developer Guide*.

You can also use the Kinesis Client Library (KCL) to build a stream consumer application to process data records in a custom way. The KCL simplifies coding by providing useful abstractions above the low-level Kinesis Data Streams API. To learn more about the KCL, see [Using the Kinesis Client Library](#) in the *Amazon Kinesis Data Streams Developer Guide*.

Delivery guarantee

QLDB streams provide an *at-least-once* delivery guarantee. Each [data record](#) that is produced by a QLDB stream is delivered to Kinesis Data Streams at least once. The same records can appear in a Kinesis data stream multiple times. So you must have deduplication logic in the consumer application layer if your use case requires it.

There are also no ordering guarantees. In some circumstances, QLDB blocks and revisions can be produced in a Kinesis data stream out of order. For more information, see [Handling duplicate and out-of-order records](#).

Delivery latency considerations

QLDB streams typically deliver updates to Kinesis Data Streams in near-real time. However, the following scenarios might create additional latency before newly committed QLDB data is emitted to a Kinesis data stream:

- Kinesis can throttle data that is streamed from QLDB, depending on your Kinesis Data Streams provisioning. For example, this might occur if you have multiple QLDB streams that write to a single Kinesis data stream, and the request rate of QLDB exceeds the capacity of the Kinesis stream resource. Throttling in Kinesis can also occur when using on-demand provisioning if the throughput grows to more than double the previous peak in less than 15 minutes.

You can measure this exceeded throughput by monitoring the Kinesis metric `WriteProvisionedThroughputExceeded`. For more information and possible solutions, see [How do I troubleshoot throttling errors in Kinesis Data Streams?](#).

- With QLDB streams, you can create an indefinite stream with a *start date and time* in the past and with no *end date and time*. By design, QLDB starts emitting newly committed data to Kinesis Data Streams only after all prior data from the specified start date and time is delivered

successfully. If you perceive additional latency in this scenario, you might need to wait for the prior data to be delivered, or you can start the stream from a later start date and time.

Getting started with streams

The following is a high-level overview of the steps that are required to get started with streaming journal data to Kinesis Data Streams:

1. Create a Kinesis Data Streams resource. For instructions, see [Creating and updating data streams](#) in the *Amazon Kinesis Data Streams Developer Guide*.
2. Create an IAM role that allows QLDB to assume write permissions for the Kinesis data stream. For instructions, see [Stream permissions in QLDB](#).
3. Create a QLDB journal stream. For instructions, see [Creating and managing streams in QLDB](#).
4. Consume the Kinesis data stream, as described in the previous section [Consuming your stream](#). For code examples that show how to use the Kinesis Client Library or AWS Lambda, see [Developing with streams in QLDB](#).

Creating and managing streams in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB provides API operations to create and manage a stream of journal data from your ledger to Amazon Kinesis Data Streams. The QLDB stream captures every document revision that is committed to your journal and sends it to a Kinesis data stream.

You can use the AWS Management Console, an AWS SDK, or the AWS Command Line Interface (AWS CLI) to create a journal stream. In addition, you can also use an [AWS CloudFormation](#) template to create streams. For more information, see the [AWS::QLDB::Stream](#) resource in the *AWS CloudFormation User Guide*.

Topics

- [Stream parameters](#)
- [Stream ARN](#)
- [AWS Management Console](#)
- [Stream states](#)
- [Handling impaired streams](#)

Stream parameters

To create a QLDB journal stream, you must provide the following configuration parameters:

Ledger name

The QLDB ledger whose journal data you want to stream to Kinesis Data Streams.

Stream name

The name that you want to assign to the QLDB journal stream. User-defined names can help identify and indicate the purpose of a stream.

Your stream name must be unique among other *active* streams for a given ledger. Stream names have the same naming constraints as ledger names, as defined in [Quotas and limits in Amazon QLDB](#).

In addition to the stream name, QLDB assigns a *stream ID* to each QLDB stream that you create. The stream ID is unique among all streams for a given ledger, regardless of their status.

Start date and time

The date and time from which to start streaming journal data. This value can be any date and time in the past but can't be in the future.

End date and time

(Optional) The date and time that specifies when the stream ends.

If you create an indefinite stream with no end time, you must manually cancel it to end the stream. You can also cancel an active, finite stream that has not yet reached its specified end date and time.

Destination Kinesis data stream

The Kinesis Data Streams target resource to which your stream writes the data records. To learn how to create a Kinesis data stream, see [Creating and updating data streams](#) in the *Amazon Kinesis Data Streams Developer Guide*.

Important

- Cross-Region and cross-account streams are not supported. The specified Kinesis data stream must be in the same AWS Region and account as your ledger.
- **Record aggregation in Kinesis Data Streams is enabled by default.** This option lets QLDB publish multiple data records in a single Kinesis Data Streams record, increasing the number of records sent per API call.

Record aggregation has important implications for processing records and **requires de-aggregation in your stream consumer**. To learn more, see [KPL key concepts](#) and [Consumer de-aggregation](#) in the *Amazon Kinesis Data Streams Developer Guide*.

IAM role

The IAM role that allows QLDB to assume write permissions to your Kinesis data stream. You can use the QLDB console to automatically create this role, or you can manually create it in IAM. To learn how to manually create it, see [Stream permissions](#).

To pass a role to QLDB when requesting a journal stream, you must have permissions to perform the `iam:PassRole` action on the IAM role resource.

Stream ARN

Every QLDB journal stream is a subresource of a ledger and is uniquely identified by an Amazon Resource Name (ARN). The following is an example ARN of a QLDB stream with a stream ID of `IiPT4brpZCqCq3f4MTHbYy` for a ledger named `exampleLedger`.

```
arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/IiPT4brpZCqCq3f4MTHbYy
```

The following section describes how to create and cancel a QLDB stream using the AWS Management Console.

AWS Management Console

Follow these steps to create or cancel a QLDB stream using the QLDB console.

To create a stream (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Streams**.
3. Choose **Create QLDB stream**.
4. On the **Create QLDB stream** page, enter the following settings:
 - **Stream name** – The name that you want to assign to the QLDB stream.
 - **Ledger** – The ledger whose journal data you want to stream.
 - **Start date and time** – The inclusive timestamp in Coordinated Universal Time (UTC) from which to start streaming journal data. This timestamp defaults to the current date and time. It can't be in the future and must be earlier than the **End date and time**.
 - **End date and time** – (Optional) The exclusive timestamp (UTC) that specifies when the stream ends. If you keep this parameter blank, the stream runs indefinitely until you cancel it.
 - **Destination stream** – The Kinesis Data Streams target resource to which your stream writes the data records. Use the following ARN format.

```
arn:aws:kinesis:aws-region:account-id:stream/kinesis-stream-name
```

The following is an example.

```
arn:aws:kinesis:us-east-1:123456789012:stream/stream-for-qldb
```

Cross-Region and cross-account streams are not supported. The specified Kinesis data stream must be in the same AWS Region and account as your ledger.

- **Enable record aggregation in Kinesis Data Streams** – (Enabled by default) Lets QLDB publish multiple data records in a single Kinesis Data Streams record, increasing the number of records sent per API call.
- **Service access** – The IAM role that grants QLDB write permissions to your Kinesis data stream.

To pass a role to QLDB when requesting a journal stream, you must have permissions to perform the `iam:PassRole` action on the IAM role resource.

- **Create and use a new service role** – Let the console create a new role for you with the required permissions for the specified Kinesis data stream.
- **Use an existing service role** – To learn how to manually create this role in IAM, see [Stream permissions](#).
- **Tags** – (Optional) Add metadata to the stream by attaching tags as key-value pairs. You can add tags to your stream to help organize and identify them. For more information, see [Tagging Amazon QLDB resources](#).

Choose **Add tag**, and then enter any key-value pairs as appropriate.

5. When the settings are as you want them, choose **Create QLDB stream**.

If your request submission is successful, the console returns to the main **Streams** page and lists your QLDB streams with their current status.

6. After your stream is active, use Kinesis to process your stream data with a [consumer application](#).

Open the Kinesis Data Streams console at <https://console.aws.amazon.com/kinesis/>.

For information about the format of the stream data records, see [QLDB stream records in Kinesis](#).

To learn how to handle streams that result in an error, see [Handling impaired streams](#).

To cancel a stream (console)

You can't restart a QLDB stream after you cancel it. To resume delivery of your data to Kinesis Data Streams, you can create a new QLDB stream.

1. Open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Streams**.
3. In the list of QLDB streams, select the active stream that you want to cancel.
4. Choose **Cancel stream**. Confirm this by entering **cancel stream** in the box provided.

For information about using the QLDB API with an AWS SDK or the AWS CLI to create and manage journal streams, see [Developing with streams in QLDB](#).

Stream states

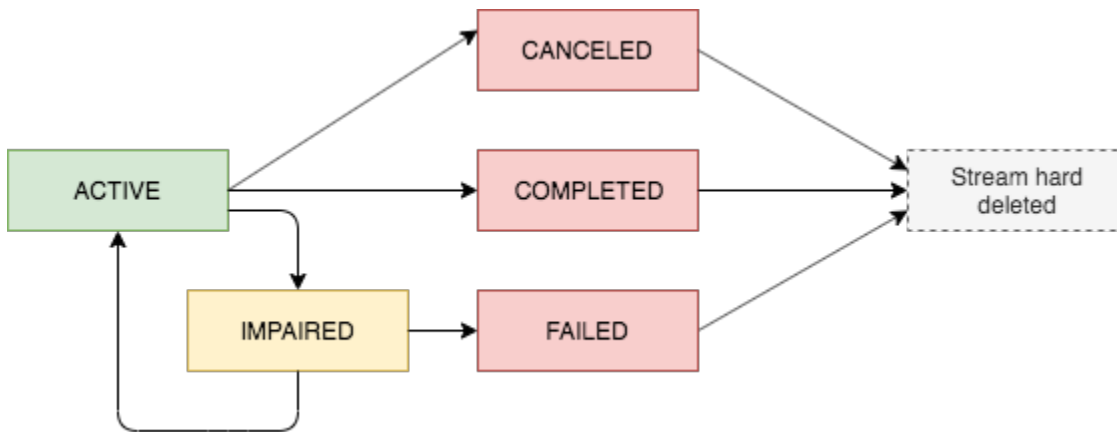
The status of a QLDB stream can be one of the following:

- **ACTIVE** – Is currently streaming or waiting to stream data (for an indefinite stream with no end time).
- **COMPLETED** – Has successfully finished streaming all journal blocks within the specified time range. This is a terminal state.
- **CANCELED** – Was ended by a user request before the specified end time and is no longer actively streaming data. This is a terminal state.
- **IMPAIRED** – Is unable to write records to Kinesis due to an error that requires your action. This is a recoverable, non-terminal state.

If you resolve the error within one hour, the stream automatically moves to **ACTIVE** state. If the error remains unresolved after one hour, the stream automatically moves to **FAILED** state.

- **FAILED** – Is unable to write records to Kinesis due to an error and is in an unrecoverable, terminal state.

The following diagram illustrates how a QLDB stream resource can transition between states.



Expiration for terminal streams

Stream resources that are in a terminal state (**CANCELED**, **COMPLETED**, and **FAILED**) are subject to a 7-day retention period. They're automatically hard-deleted after this limit expires.

After a terminal stream is deleted, you can no longer use the QLDB console or the QLDB API to describe or list the stream resource.

Handling impaired streams

If your stream encounters an error, it moves to `IMPAIRED` state first. QLDB continues to retry `IMPAIRED` streams for up to one hour.

If you resolve the error within one hour, the stream automatically moves to `ACTIVE` state. If the error remains unresolved after one hour, the stream automatically moves to `FAILED` state.

An impaired or failed stream can have one of the following error causes:

- `KINESIS_STREAM_NOT_FOUND` – The destination Kinesis Data Streams resource doesn't exist. Verify that the Kinesis data stream that you provided in your QLDB stream request is correct. Then, go to Kinesis and create the data stream that you specified.
- `IAM_PERMISSION_REVOKED` – QLDB doesn't have enough permissions to write data records to your specified Kinesis data stream. Verify that you define a policy for your specified Kinesis data stream that grants the QLDB service (`qldb.amazonaws.com`) permissions to the following actions:
 - `kinesis:PutRecord`
 - `kinesis:PutRecords`
 - `kinesis:DescribeStream`
 - `kinesis:ListShards`

Monitoring impaired streams

If a stream becomes impaired, the QLDB console displays a banner that shows details about the stream and the error that it encountered. You can also use the `DescribeJournalKinesisStream` API operation to get a stream's status and the underlying error cause.

In addition, you can use Amazon CloudWatch to create an alarm that monitors the `IsImpaired` metric of a stream. For information about monitoring QLDB metrics with CloudWatch, see [Amazon QLDB dimensions and metrics](#).

Developing with streams in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section summarizes the API operations that you can use with an AWS SDK or the AWS CLI to create and manage journal streams in Amazon QLDB. It also describes the sample applications that demonstrate these operations and use the Kinesis Client Library (KCL) or AWS Lambda to implement a stream consumer.

You can use the KCL to build consumer applications for Amazon Kinesis Data Streams. The KCL simplifies coding by providing useful abstractions above the low-level Kinesis Data Streams API. To learn more about the KCL, see [Using the Kinesis Client Library](#) in the *Amazon Kinesis Data Streams Developer Guide*.

Contents

- [QLDB journal stream APIs](#)
- [Sample applications](#)
 - [Basic operations \(Java\)](#)
 - [Integration with OpenSearch Service \(Python\)](#)
 - [Integration with Amazon SNS and Amazon SQS \(Python\)](#)

QLDB journal stream APIs

The QLDB API provides the following journal stream operations for use by application programs:

- `StreamJournalToKinesis` – Creates a journal stream for a given QLDB ledger. The stream captures every document revision that is committed to the ledger's journal and delivers the data to a specified Kinesis Data Streams resource.
- **Record aggregation in Kinesis Data Streams is enabled by default.** This option lets QLDB publish multiple data records in a single Kinesis Data Streams record, increasing the number of records sent per API call.

Record aggregation has important implications for processing records and **requires de-aggregation in your stream consumer**. To learn more, see [KPL key concepts](#) and [Consumer de-aggregation](#) in the *Amazon Kinesis Data Streams Developer Guide*.

- `DescribeJournalKinesisStream` – Returns detailed information about a given QLDB journal stream. The output includes the ARN, stream name, current status, creation time, and the parameters of your original stream creation request.
- `ListJournalKinesisStreamsForLedger` – Returns a list of all QLDB journal stream descriptors for a given ledger. The output of each stream descriptor includes the same details that are returned by `DescribeJournalKinesisStream`.
- `CancelJournalKinesisStream` – Ends a given QLDB journal stream. Before a stream can be canceled, its current status must be `ACTIVE`.

You can't restart a stream after you cancel it. To resume delivery of your data to Kinesis Data Streams, you can create a new QLDB stream.

For complete descriptions of these API operations, see the [Amazon QLDB API reference](#).

For information about creating and managing journal streams using the AWS CLI, see the [AWS CLI Command Reference](#).

Sample applications

QLDB provides sample applications that demonstrate various operations using journal streams. These applications are open source on the [AWS Samples GitHub site](#).

Topics

- [Basic operations \(Java\)](#)
- [Integration with OpenSearch Service \(Python\)](#)
- [Integration with Amazon SNS and Amazon SQS \(Python\)](#)

Basic operations (Java)

For a Java code example that demonstrates basic operations for QLDB journal streams, see the GitHub repository [aws-samples/amazon-qlldb-dmv-sample-java](#). For instructions on how to download and install this sample application, see [Installing the Amazon QLDB Java sample application](#).

Note

After you install the application, don't proceed to *Step 1* of the Java tutorial to create a ledger. This sample application for streaming creates the `vehicle-registration` ledger for you.

This sample application packages the complete source code from the [Java tutorial](#) and its dependencies, including the following modules:

- [AWS SDK for Java](#) – To create and delete both the QLDB and Kinesis Data Streams resources, including ledgers, QLDB journal streams, and Kinesis data streams.
- [Amazon QLDB driver for Java](#) – To run data transactions on a ledger using PartiQL statements, including creating tables and inserting documents.
- [Kinesis Client Library](#) – To consume and process data from a Kinesis data stream.

Running the code

The [StreamJournal](#) class contains tutorial code that demonstrates the following operations:

1. Create a ledger named `vehicle-registration`, create tables, and load them with sample data.

Note

Before running this code, make sure that you don't already have an active ledger named `vehicle-registration`.

2. Create a Kinesis data stream, an IAM role that allows QLDB to assume write permissions for the Kinesis data stream, and a QLDB journal stream.
3. Use the KCL to start a stream reader that processes the Kinesis data stream and logs each QLDB data record.
4. Use the stream data to validate the hash chain of the `vehicle-registration` sample ledger.
5. Clean up all resources by stopping the stream reader, canceling the QLDB journal stream, deleting the ledger, and deleting the Kinesis data stream.

To run the `StreamJournal` tutorial code, enter the following Gradle command from your project root directory.

```
./gradlew run -Dtutorial=streams.StreamJournal
```

Integration with OpenSearch Service (Python)

For a Python sample application that demonstrates how to integrate a QLDB stream with Amazon OpenSearch Service, see the GitHub repository [aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python](https://github.com/aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python). This application uses an AWS Lambda function to implement a Kinesis Data Streams consumer.

To clone the repository, enter the following `git` command.

```
git clone https://github.com/aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python.git
```

To run the sample application, see the [README](#) on GitHub for instructions.

Integration with Amazon SNS and Amazon SQS (Python)

For a Python sample application that demonstrates how to integrate a QLDB stream with Amazon Simple Notification Service (Amazon SNS), see the GitHub repository [aws-samples/amazon-qldb-streams-dmv-sample-lambda-python](https://github.com/aws-samples/amazon-qldb-streams-dmv-sample-lambda-python).

This application uses an AWS Lambda function to implement a Kinesis Data Streams consumer. It sends messages to an Amazon SNS topic, which has an Amazon Simple Queue Service (Amazon SQS) queue subscribed to it.

To clone the repository, enter the following `git` command.

```
git clone https://github.com/aws-samples/amazon-qldb-streams-dmv-sample-lambda-python.git
```

To run the sample application, see the [README](#) on GitHub for instructions.

QLDB stream records in Kinesis

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

An Amazon QLDB stream writes three types of data records to a given Amazon Kinesis Data Streams resource: *control*, *block summary*, and *revision details*. All three record types are written in the *binary representation* of the [Amazon Ion format](#).

Control records indicate the start and completion of your QLDB streams. Whenever a revision is committed to your journal, a QLDB stream writes all of the associated journal block data in block summary and revision details records.

The three record types are polymorphic. They all consist of a common top-level record that contains the QLDB stream ARN, the record type, and the record payload. This top-level record has the following format.

```
{
  qlldbStreamArn: string,
  recordType: string,
  payload: {
    //control | block summary | revision details record
  }
}
```

The `recordType` field can have one of three values:

- CONTROL
- BLOCK_SUMMARY
- REVISION_DETAILS

The following sections describe the format and contents of each individual payload record.

Note

QLDB writes all stream records to Kinesis Data Streams in the binary representation of Amazon Ion. The following examples are provided in the text representation of Ion to illustrate the record contents in a readable format.

Topics

- [Control records](#)
- [Block summary records](#)
- [Revision details records](#)
- [Handling duplicate and out-of-order records](#)

Control records

A QLDB stream writes *control* records to indicate its start and completion events. The following are examples of control records with sample data for each `controlRecordType`:

- **CREATED** – The first record that a QLDB stream writes to Kinesis to indicate that your newly created stream is active.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
  IiPT4brpZCqCq3f4MTHbYy",
  recordType:"CONTROL",
  payload:{
    controlRecordType:"CREATED"
  }
}
```

- **COMPLETED** – The last record that a QLDB stream writes to Kinesis to indicate that your stream has reached the specified end date and time. This record is not written if you cancel the stream.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
  IiPT4brpZCqCq3f4MTHbYy",
  recordType:"CONTROL",
  payload:{
    controlRecordType:"COMPLETED"
  }
}
```



```
}
}
```

Block summary records

A *block summary* record represents a journal block in which your document revisions are committed. A [block](#) is an object that is committed to your QLDB journal during a transaction.

The payload of a block summary record contains the block address, timestamp, and other metadata of the transaction that committed the block. It also includes summary attributes of the revisions in the block and the PartiQL statements that committed them. The following is an example of a block summary record with sample data.

Note

This block summary example is provided for informational purposes only. The hashes shown aren't real calculated hash values.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
  IiPT4brpZCqCq3f4MTHbYy",
  recordType:"BLOCK_SUMMARY",
  payload:{
    blockAddress:{
      strandId:"E1YL30RGoqrFCbbaQn3K6m",
      sequenceNo:60807
    },
    transactionId:"9RWohCo7My4GGkxRETAJ6M",
    blockTimestamp:2019-09-18T17:00:14.601000001Z,
    blockHash:{{6Pk9KDYJd38ci09oaHxx0D2grtgh4QBBqbDS6i9quX8=}},
    entriesHash:{{r5YoH6+NXDXxgoRzPREGAWJfn73K1ZE0eTfbTxZWUDU=}},
    previousBlockHash:{{K3ti0Agk7DEponywKcQCPRYVHb5RuyxdmQFTfrloptA=}},
    entriesHashList:[
      {{pbzvz6ofJC7mD2jvgfyrY/VtR01zIZHoWy8T1Vcx1Go=}},
      {{k2brC23DLMercmi0WHiURaGwHu0mQtLzdNPuviE2rcs=}},
      {{hvw1EV8k4o0kI036kb10/+UUSFUQqCanKuDGr0aP9nQ=}},
      {{ZrLbkyzDcpJ9KWsZMZqRuKUKG/czLIJ4US+K5E31b+Q=}}
    ],
    transactionInfo:{
```

```

statements:[
  {
    statement:"SELECT * FROM Person WHERE GovId = ?",
    startTime:2019-09-18T17:00:14.587Z,
    statementDigest:{p4Dn0DiuYD3Xm9UQQ75YLwmoMbSfJmop0mTfMnXs26M=}
  },
  {
    statement:"INSERT INTO Person ?",
    startTime:2019-09-18T17:00:14.594Z,
    statementDigest:{klMLkLf5VJqk6JUPtHkQp0sDdG4HmuUaq/VaApQf1U=}
  },
  {
    statement:"INSERT INTO VehicleRegistration ?",
    startTime:2019-09-18T17:00:14.598Z,
    statementDigest:{B0g09BwVNrzRYFoe7t+GVLpJ6uZcLKf5t/chkfRhspI=}
  }
],
documents:{
  '7z20pEBgVCvCtwvx4a2JGn':{
    tableName:"Person",
    tableId:"LSkFkQvkIOjCmpTZpkfnp9",
    statements:[1]
  },
  'K0FpsSLpydLDr7hi6KUzqk':{
    tableName:"VehicleRegistration",
    tableId:"Ad3A07z0Zffc7Gpso7BXy0",
    statements:[2]
  }
},
revisionSummaries:[
  {
    hash:{uDthuiqSy4FwjZssyCiyFd90XoPSlIwomHBdF/0rmkE=}},
    documentId:"7z20pEBgVCvCtwvx4a2JGn"
  },
  {
    hash:{qJID/amu0gN3dpG5Tg0FfIFTh/U5yFkfT+g/06k5sPM=}},
    documentId:"K0FpsSLpydLDr7hi6KUzqk"
  }
]
}

```

In the `revisionSummaries` field, some revisions might not have a `documentId`. These are internal-only system revisions that don't contain user data. A QLDB stream includes these revisions in their respective block summary records because the hashes of these revisions are part of the journal's full hash chain. The full hash chain is required for cryptographic verification.

Only the revisions that do have a document ID are published in separate revision details records, as described in the following section.

Revision details records

A *revision details* record represents a document revision that is committed to your journal. The payload contains all of the attributes from the [committed view](#) of the revision, along with the associated table name and table ID. The following is an example of a revision record with sample data.

```
{
  qlldbStreamArn:"arn:aws:qlldb:us-east-1:123456789012:stream/exampleLedger/
  IiPT4brpZCqCq3f4MTHbYy",
  recordType:"REVISION_DETAILS",
  payload:{
    tableInfo:{
      tableName:"VehicleRegistration",
      tableId:"Ad3A07z0Zffc7Gpso7BXy0"
    },
    revision:{
      blockAddress:{
        strandId:"E1YL30RGoqrFCbbaQn3K6m",
        sequenceNo:60807
      },
      hash:{[qJID/amu0gN3dpG5Tg0FfIFTh/U5yFkfT+g/06k5sPM=]},
      data:{
        VIN:"1N4AL11D75C109151",
        LicensePlateNumber:"LEWISR261LL",
        State:"WA",
        City:"Seattle",
        PendingPenaltyTicketAmount:90.25,
        ValidFromDate:2017-08-21,
        ValidToDate:2020-05-11,
        Owners:{
          PrimaryOwner:{PersonId:"7z20pEBgVCvCtwvx4a2JGn"},
          SecondaryOwners:[]
        }
      }
    }
  }
}
```

```
    },  
    metadata:{  
      id:"K0FpsSLpydLD17hi6KUzqk",  
      version:0,  
      txTime:2019-09-18T17:00:14.602Z,  
      txId:"9RWohCo7My4GGkxRETAJ6M"  
    }  
  }  
}  
}
```

Handling duplicate and out-of-order records

QLDB streams can publish duplicate and out-of-order records to Kinesis Data Streams. So, a consumer application might need to implement its own logic to identify and handle such scenarios. The block summary and revision details records include fields that you can use for this purpose. Combined with the features of downstream services, these fields can indicate both a unique identity and a strict order for the records.

For example, consider a stream that integrates QLDB with an OpenSearch index to provide full text search capabilities over documents. In this use case, you need to avoid indexing stale (out-of-order) revisions of a document. To enforce ordering and deduplication, you can use the document ID and external version fields in OpenSearch, along with the document ID and version fields in a revision details record.

For an example of deduplication logic in a sample application that integrates QLDB with Amazon OpenSearch Service, see the GitHub repository [aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python](https://github.com/aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python).

Stream permissions in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Before creating an Amazon QLDB stream, you must provide QLDB with write permissions to your specified Amazon Kinesis Data Streams resource. If you're using a customer managed AWS KMS key

for server-side encryption of your Kinesis stream, you must also provide QLDB with permissions to use your specified symmetric encryption key. Kinesis Data Streams doesn't support [asymmetric KMS keys](#).

To provide your QLDB stream with the necessary permissions, you can make QLDB assume an IAM service role with the appropriate permissions policies. A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Note

To pass a role to QLDB when requesting a journal stream, you must have permissions to perform the `iam:PassRole` action on the IAM role resource. This is in addition to the `qldb:StreamJournalToKinesis` permission on the QLDB stream subresource. To learn how to control access to QLDB using IAM, see [How Amazon QLDB works with IAM](#). For a QLDB policy example, see [Identity-based policy examples for Amazon QLDB](#).

In this example, you create a role that allows QLDB to write data records to a Kinesis data stream on your behalf. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

If you're streaming a QLDB journal in your AWS account for the first time, you must first create an IAM role with the appropriate policies by doing the following. Or, you can [use the QLDB console](#) to automatically create the role for you. Otherwise, you can choose a role that you previously created.

Topics

- [Create a permissions policy](#)
- [Create an IAM role](#)

Create a permissions policy

Complete the following steps to create a permissions policy for a QLDB stream. This example shows a Kinesis Data Streams policy that grants QLDB permissions to write data records to your specified Kinesis data stream. If applicable, the example also shows a key policy that allows QLDB to use your symmetric encryption KMS key.

For more information about Kinesis Data Streams policies, see [Controlling access to Amazon Kinesis Data Streams resources using IAM](#) and [Permissions to use user-generated KMS keys](#) in the *Amazon Kinesis Data Streams Developer Guide*. To learn more about AWS KMS key policies, see [Using key policies in AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

Note

Your Kinesis data stream and KMS key must both be in the same AWS Region and account as your QLDB ledger.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation column on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. At the top of the page, choose **Create policy**.
4. Choose the **JSON** tab.
5. Enter a JSON policy document.
 - If you're using a customer managed KMS key for server-side encryption of your Kinesis stream, use the following example policy document. To use this policy, replace *us-east-1*, *123456789012*, *kinesis-stream-name*, and *1234abcd-12ab-34cd-56ef-1234567890ab* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBStreamKinesisPermissions",
      "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
        "kinesis:ListShards" ],
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/kinesis-
        stream-name"
```

```

    },
    {
        "Sid": "QLDBStreamKMSPermission",
        "Action": [ "kms:GenerateDataKey" ],
        "Effect": "Allow",
        "Resource": "arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
]
}

```

- Otherwise, use the following example policy document. To use this policy, replace *us-east-1*, *123456789012*, and *kinesis-stream-name* in the example with your own information.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBStreamKinesisPermissions",
            "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
"kinesis:ListShards" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/kinesis-
stream-name"
        }
    ]
}

```

6. Choose **Review policy**.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring](#) in the *IAM User Guide*.

7. On the **Review policy** page, enter a **Name** and an optional **Description** for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

Create an IAM role

After creating a permissions policy for your QLDB stream, you can then create an IAM role and attach your policy to it.

To create the service role for QLDB (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **QLDB**, and then choose the **QLDB** use case.
5. Choose **Next**.
6. Select the box next to the policy that you created in the previous steps.
7. (Optional) Set a [permissions boundary](#). This is an advanced feature that is available for service roles, but not service-linked roles.
 - a. Open the **Set permissions boundary** section, and then choose **Use a permissions boundary to control the maximum role permissions**.

IAM includes a list of the AWS managed and customer-managed policies in your account.

 - b. Select the policy to use for the permissions boundary.
8. Choose **Next**.
9. Enter a role name or a role name suffix to help you identify the purpose of the role.

Important

When you name a role, note the following:

- Role names must be unique within your AWS account, and can't be made unique by case.

For example, don't create roles named both **PRODRROLE** and **prodrole**. When a role name is used in a policy or as part of an ARN, the role name is case sensitive, however when a role name appears to customers in the console, such as during the sign-in process, the role name is case insensitive.

- You can't edit the name of the role after it's created because other entities might reference the role.

10. (Optional) For **Description**, enter a description for the role.
11. (Optional) To edit the use cases and permissions for the role, in the **Step 1: Select trusted entities** or **Step 2: Add permissions** sections, choose **Edit**.
12. (Optional) To help identify, organize, or search for the role, add tags as key-value pairs. For more information about using tags in IAM, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.
13. Review the role, and then choose **Create role**.

The following JSON document is an example of a trust policy that allows QLDB to assume an IAM role with specific permissions attached to it.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "qldb.amazonaws.com"
      },
      "Action": [ "sts:AssumeRole" ],
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Note

This trust policy example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to prevent the *confused deputy* problem. With this trust policy, QLDB can assume the role for any QLDB stream in the account 123456789012 for the ledger `myExampleLedger` only. For more information, see [Cross-service confused deputy prevention](#).

After creating your IAM role, return to the QLDB console and refresh the **Create QLDB stream** page so that it can find your new role.

Common errors for journal streams in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes runtime errors that are thrown by Amazon QLDB for journal stream requests.

The following is a list of common exceptions returned by the service. Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

AccessDeniedException

Message: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN*

You don't have permissions to pass an IAM role to the QLDB service. QLDB requires a role for all journal stream requests, and you must have permissions to pass this role to QLDB. The role provides QLDB with write permissions in your specified Amazon Kinesis Data Streams resource.

Verify that you define an IAM policy that grants permission to perform the `PassRole` API operation on your specified IAM role resource for the QLDB service (`qldb.amazonaws.com`). For a policy example, see [Identity-based policy examples for Amazon QLDB](#).

IllegalArgumentException

Message: QLDB encountered an error validating Kinesis Data Streams: Response from Kinesis: *errorCode errorMessage*

A possible cause for this error is that the provided Kinesis Data Streams resource doesn't exist. Or, QLDB doesn't have enough permissions to write data records to your specified Kinesis data stream.

Verify that the Kinesis data stream that you provide in your stream request is correct. For more information, see [Creating and updating data streams](#) in the *Amazon Kinesis Data Streams Developer Guide*.

Also, verify that you define a policy for your specified Kinesis data stream that grants the QLDB service (qldb.amazonaws.com) permissions to the following actions. For more information, see [Stream permissions](#).

- kinesis:PutRecord
- kinesis:PutRecords
- kinesis:DescribeStream
- kinesis:ListShards

IllegalArgumentException

Message: Unexpected response from Kinesis Data Streams while validating the Kinesis configuration. Response from Kinesis: *errorCode errorMessage*

The attempt to write data records to the provided Kinesis data stream failed with the provided Kinesis error response. For more information about possible causes, see [Troubleshooting Amazon Kinesis Data Streams producers](#) in the *Amazon Kinesis Data Streams Developer Guide*.

IllegalArgumentException

Message: Start date must not be greater than end date.

Both InclusiveStartTime and ExclusiveEndTime must be in [ISO 8601](#) date and time format and in Coordinated Universal Time (UTC).

IllegalArgumentException

Message: Start date cannot be in the future.

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in ISO 8601 date and time format and in UTC.

LimitExceededException

Message: Exceeded the limit of 5 concurrently running Journal streams to Kinesis Data Streams

QLDB enforces a default limit of five concurrent journal streams.

Ledger management in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This chapter describes how to use the QLDB API, the AWS Command Line Interface (AWS CLI), and AWS CloudFormation to do ledger management operations in Amazon QLDB.

You can also perform these same tasks using the AWS Management Console. For more information, see [Accessing Amazon QLDB using the console](#).

Topics

- [Basic operations for Amazon QLDB ledgers](#)
- [Creating Amazon QLDB resources with AWS CloudFormation](#)
- [Tagging Amazon QLDB resources](#)

Basic operations for Amazon QLDB ledgers

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can use the QLDB API or the AWS Command Line Interface (AWS CLI) to create, update, and delete ledgers in Amazon QLDB. You can also list all the ledgers in your account, or get information about a specific ledger.

The following topics provide short code examples that show common steps for ledger operations using the AWS SDK for Java and the AWS CLI.

Topics

- [Creating a ledger](#)
- [Describing a ledger](#)
- [Updating a ledger](#)
- [Updating a ledger permissions mode](#)
- [Deleting a ledger](#)
- [Listing ledgers](#)

For code examples that demonstrate these operations in a complete sample application, see the following [Getting started with the driver](#) tutorials and GitHub repositories:

- Java: [Tutorial](#) | [GitHub repository](#)
- Node.js: [Tutorial](#) | [GitHub repository](#)
- Python: [Tutorial](#) | [GitHub repository](#)

Creating a ledger

Use the `CreateLedger` operation to create a ledger in your AWS account. You must provide the following information:

- **Ledger name** – The name of the ledger that you want to create in your account. The name must be unique among all of your ledgers in the current AWS Region.

Naming constraints for ledger names are defined in [Quotas and limits in Amazon QLDB](#).

- **Permissions mode** – The permissions mode to assign to the ledger. Choose one of the following options:
 - **Allow all** – A legacy permissions mode that enables access control with API-level granularity for ledgers.

This mode allows users who have the `SendCommand` API permission for this ledger to run all PartiQL commands (hence, `ALLOW_ALL`) on any tables in the specified ledger. This mode disregards any table-level or command-level IAM permissions policies that you create for the ledger.

- **Standard** – (*Recommended*) A permissions mode that enables access control with finer granularity for ledgers, tables, and PartiQL commands. We strongly recommend using this permissions mode to maximize the security of your ledger data.

By default, this mode denies all requests to run any PartiQL commands on any tables in this ledger. To allow PartiQL commands, you must create IAM permissions policies for specific table resources and PartiQL actions, in addition to the `SendCommand` API permission for the ledger. For information, see [Getting started with the standard permissions mode in Amazon QLDB](#).

- **Deletion protection** – (Optional) The flag that prevents a ledger from being deleted by any user. If you don't specify it during ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger. You can disable it by using the `UpdateLedger` operation to set the flag to `false`.

- **AWS KMS key** – (Optional) The key in AWS Key Management Service (AWS KMS) to use for encryption of data at rest. Choose one of the following types of AWS KMS keys:
 - **AWS owned KMS key** – Use a KMS key that is owned and managed by AWS on your behalf.

If you don't define this parameter during ledger creation, the ledger uses this type of key by default. You can also use the string `AWS_OWNED_KMS_KEY` to specify this key type. This option requires no additional setup.

- **Customer managed KMS key** – Use a symmetric encryption KMS key in your account that you create, own, and manage. QLDB doesn't support [asymmetric keys](#).

This option requires you to create a KMS key or use an existing key in your account. For instructions on creating a customer managed key, see [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*.

You can specify a customer managed KMS key by using an ID, alias, or Amazon Resource Name (ARN). To learn more, see [Key identifiers \(KeyId\)](#) in the *AWS Key Management Service Developer Guide*.

 Note

Cross-Region keys are not supported. The specified KMS key must be in the same AWS Region as your ledger.

For more information, see [Encryption at rest in Amazon QLDB](#).

- **Tags** – (Optional) Add metadata to the ledger by attaching tags as key-value pairs. You can add tags to your ledger to help organize and identify them. For more information, see [Tagging Amazon QLDB resources](#).

The ledger isn't ready for use until QLDB creates it and sets its status to ACTIVE.

Creating a ledger (Java)

To create a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `CreateLedgerRequest` class to provide the request information.

You must provide the ledger name and a permissions mode.

3. Run the `createLedger` method by providing the request object as a parameter.

The `createLedger` request returns a `CreateLedgerResult` object that has information about the ledger. See the next section for an example of using the `DescribeLedger` operation to check your ledger's status after you create it.

The following examples demonstrate the preceding steps.

Example – Use default configuration settings

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
CreateLedgerRequest request = new CreateLedgerRequest()
    .WithName(ledgerName)
    .withPermissionsMode(PermissionsMode.STANDARD);
CreateLedgerResult result = client.createLedger(request);
```

Note

The ledger uses the following default settings if you don't specify them:

- Deletion protection – Enabled (`true`).
- KMS key – AWS owned KMS key.

Example – Disable deletion protection, use a customer managed KMS key, and attach tags

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();

Map<String, String> tags = new HashMap<>();
```



```
tags.put("IsTest", "true");
tags.put("Domain", "Test");

CreateLedgerRequest request = new CreateLedgerRequest()
    .withName(ledgerName)
    .withPermissionsMode(PermissionsMode.STANDARD)
    .withDeletionProtection(false)
    .withKmsKey("arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
    .withTags(tags);
CreateLedgerResult result = client.createLedger(request);
```

Creating a ledger (AWS CLI)

Create a new ledger named `vehicle-registration` using default configuration settings.

Example

```
aws qlldb create-ledger --name vehicle-registration --permissions-mode STANDARD
```

Note

The ledger uses the following default settings if you don't specify them:

- Deletion protection – Enabled (`true`).
- KMS key – AWS owned KMS key.

Or, create a new ledger named `vehicle-registration` with deletion protection disabled, with a specified customer managed KMS key, and with specified tags.

Example

```
aws qlldb create-ledger \
  --name vehicle-registration \
  --no-deletion-protection \
  --permissions-mode STANDARD \
  --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
  --tags IsTest=true,Domain=Test
```

Creating a ledger (AWS CloudFormation)

You can also use an [AWS CloudFormation](#) template to create ledgers. For more information, see the [AWS::QLDB::Ledger](#) resource in the *AWS CloudFormation User Guide*.

Describing a ledger

Use the `DescribeLedger` operation to view details about a ledger. You must provide the ledger name. The output from `DescribeLedger` is in the same format as that from `CreateLedger`. It includes the following information:

- **Ledger name** – The name of the ledger that you want to describe.
- **ARN** – The Amazon Resource Name (ARN) for the ledger in the following format.

```
arn:aws:qldb:aws-region:account-id:ledger/ledger-name
```

- **Deletion protection** – The flag that indicates whether the deletion protection feature is enabled.
- **Creation date and time** – The date and time, in epoch time format, when the ledger was created.
- **State** – The current status of the ledger. This can be one of the following values:
 - CREATING
 - ACTIVE
 - DELETING
 - DELETED
- **Permissions mode** – The permissions mode that is assigned to the ledger. This can be one of the following values:
 - **ALLOW_ALL** – A legacy permissions mode that enables access control with API-level granularity for ledgers.
 - **STANDARD** – A permissions mode that enables access control with finer granularity for ledgers, tables, and PartiQL commands.
- **Encryption description** – Information about the encryption of data at rest in the ledger. This includes the following items:
 - **AWS KMS key ARN** – The ARN of the customer managed KMS key that the ledger uses for encryption at rest. If this is undefined, the ledger uses an AWS owned KMS key for encryption.
 - **Encryption status** – The current status of encryption at rest for the ledger. This can be one of the following values:

- **ENABLED** – Encryption is fully enabled using the specified key.
- **UPDATING** – The specified key change is actively being processed.

Key changes in QLDB are asynchronous. The ledger is fully accessible without any performance impact while the key change is being processed. The amount of time it takes to update a key varies depending on the ledger size.

- **KMS_KEY_INACCESSIBLE** – The specified customer managed KMS key is not accessible, and the ledger is impaired. Either the key was disabled or deleted, or the grants on the key were revoked. When a ledger is impaired, it's not accessible and doesn't accept any read or write requests.

An impaired ledger automatically returns to an active state after you restore the grants on the key, or after you reenables the key that was disabled. However, deleting a customer managed KMS key is irreversible. After a key is deleted, you can no longer access the ledgers that are protected with that key, and the data becomes unrecoverable permanently.

- **Inaccessible AWS KMS key** – The date and time, in epoch time format, when the KMS key first became inaccessible, in the case of an error.

This is undefined if the KMS key is accessible.

For more information, see [Encryption at rest in Amazon QLDB](#).

Note

After you create a QLDB ledger, it becomes ready for use when its status changes from **CREATING** to **ACTIVE**.

Describing a ledger (Java)

To describe a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class. Or, you can use the same instance of the `AmazonQLDB` client that you instantiated for the `CreateLedger` request.
2. Create an instance of the `DescribeLedgerRequest` class and provide the ledger name that you want to describe.
3. Run the `describeLedger` method by providing the request object as a parameter.

4. The `describeLedger` request returns a `DescribeLedgerResult` object that has current information about the ledger.

The following code example demonstrates the preceding steps. You can call the `describeLedger` method of the client to get ledger information at any time.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
DescribeLedgerRequest request = new DescribeLedgerRequest().withName(ledgerName);
DescribeLedgerResult result = client.describeLedger(request);
System.out.printf("%s: ARN: %s \t State: %s \t CreationDateTime: %s \t
    DeletionProtection: %s
                \t PermissionsMode: %s \t EncryptionDescription: %s",
    result.getName(),
    result.getArn(),
    result.getState(),
    result.getCreationDateTime(),
    result.getDeletionProtection(),
    result.getPermissionsMode(),
    result.getEncryptionDescription());
```

Describing a ledger (AWS CLI)

Describe the vehicle-registration ledger that you just created.

Example

```
aws qlldb describe-ledger --name vehicle-registration
```

Updating a ledger

The `UpdateLedger` operation currently lets you change the following configuration settings for an existing ledger:

- **Deletion protection** – The flag that prevents a ledger from being deleted by any user. If this feature is enabled, you must first disable it by setting the flag to `false` before you can delete the ledger.

If you don't define this parameter, no changes are made to the deletion protection setting of the ledger.

- **AWS KMS key** – The key in AWS Key Management Service (AWS KMS) to use for encryption of data at rest. If you don't define this parameter, no changes are made to the KMS key of the ledger.

 Note

Amazon QLDB launched support for customer managed AWS KMS keys on July 22, 2021. Any ledgers that were created before the launch are protected by AWS owned keys by default, but are currently not eligible for encryption at rest using customer managed keys.

You can view the creation time of your ledger on the QLDB console.

Use one of the following options:

- **AWS owned KMS key** – Use a KMS key that is owned and managed by AWS on your behalf. To use this type of key, specify the string `AWS_OWNED_KMS_KEY` for this parameter. This option requires no additional setup.
- **Customer managed KMS key** – Use a symmetric encryption KMS key in your account that you create, own, and manage. QLDB doesn't support [asymmetric keys](#).

This option requires you to create a KMS key or use an existing key in your account. For instructions on creating a customer managed key, see [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*.

You can specify a customer managed KMS key by using an ID, alias, or Amazon Resource Name (ARN). To learn more, see [Key identifiers \(KeyId\)](#) in the *AWS Key Management Service Developer Guide*.

 Note

Cross-Region keys are not supported. The specified KMS key must be in the same AWS Region as your ledger.

Key changes in QLDB are asynchronous. The ledger is fully accessible without any performance impact while the key change is being processed.

You can switch keys as often as necessary, but the amount of time it takes to update a key varies depending on the ledger size. You can use the `DescribeLedger` operation to check the encryption at rest status.

For more information, see [Encryption at rest in Amazon QLDB](#).

The output from `UpdateLedger` is in the same format as that from `CreateLedger`.

Updating a ledger (Java)

To update a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `UpdateLedgerRequest` class to provide the request information.

You must provide the ledger name along with a new Boolean value for deletion protection or a new string value for the KMS key.

3. Run the `updateLedger` method by providing the request object as a parameter.

The following code examples demonstrate the preceding steps. The `updateLedger` request returns an `UpdateLedgerResult` object that has updated information about the ledger.

Example – Disable deletion protection

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
    .withName(ledgerName)
    .withDeletionProtection(false);
UpdateLedgerResult result = client.updateLedger(request);
```

Example – Use a customer managed KMS key

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
    .withName(ledgerName)
    .withKmsKey("arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
UpdateLedgerResult result = client.updateLedger(request);
```

Example – Use an AWS owned KMS key

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
    .withName(ledgerName)
    .withKmsKey("AWS_OWNED_KMS_KEY")
UpdateLedgerResult result = client.updateLedger(request);
```

Updating a ledger (AWS CLI)

If your vehicle-registration ledger has deletion protection enabled, you must first disable it before you can delete it.

Example

```
aws qlldb update-ledger --name vehicle-registration --no-deletion-protection
```

You can also change the ledger's encryption at rest settings to use a customer managed KMS key.

Example

```
aws qlldb update-ledger --name vehicle-registration --kms-key arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

Or, you can change the encryption at rest settings to use an AWS owned KMS key.

Example

```
aws qlldb update-ledger --name vehicle-registration --kms-key AWS_OWNED_KMS_KEY
```

Updating a ledger permissions mode

The `UpdateLedgerPermissionsMode` operation lets you change the permissions mode of an existing ledger. Choose one of the following options:

- **Allow all** – A legacy permissions mode that enables access control with API-level granularity for ledgers.

This mode allows users who have the `SendCommand` API permission for this ledger to run all PartiQL commands (hence, `ALLOW_ALL`) on any tables in the specified ledger. This mode

disregards any table-level or command-level IAM permissions policies that you create for the ledger.

- **Standard** – (*Recommended*) A permissions mode that enables access control with finer granularity for ledgers, tables, and PartiQL commands. We strongly recommend using this permissions mode to maximize the security of your ledger data.

By default, this mode denies all requests to run any PartiQL commands on any tables in this ledger. To allow PartiQL commands, you must create IAM permissions policies for specific table resources and PartiQL actions, in addition to the `SendCommand` API permission for the ledger. For information, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Important

Before switching to the STANDARD permissions mode, you must first create all required IAM policies and table tags to avoid disruption to your users. To learn more, proceed to [Migrating to the standard permissions mode](#).

Updating a ledger permissions mode (Java)

To update a ledger permissions mode using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `UpdateLedgerPermissionsModeRequest` class to provide the request information.

You must provide the ledger name along with a new string value for the permissions mode.

3. Run the `updateLedgerPermissionsMode` method by providing the request object as a parameter.

The following code examples demonstrate the preceding steps. The `updateLedgerPermissionsMode` request returns an `UpdateLedgerPermissionsModeResult` object that has updated information about the ledger.

Example – Assign the standard permissions mode

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
```



```
UpdateLedgerPermissionsModeRequest request = new UpdateLedgerPermissionsModeRequest()
    .withName(ledgerName)
    .withPermissionsMode(PermissionsMode.STANDARD);
UpdateLedgerPermissionsModeResult result = client.updateLedgerPermissionsMode(request);
```

Updating a ledger permissions mode (AWS CLI)

Assign the STANDARD permissions mode to your vehicle-registration ledger.

Example

```
aws qldb update-ledger-permissions-mode --name vehicle-registration --permissions-mode
STANDARD
```

Migrating to the standard permissions mode

To migrate to the STANDARD permissions mode, we recommend analyzing your QLDB access patterns and adding IAM policies that grant your users the appropriate permissions to access their resources.

Before switching to the STANDARD permissions mode, you must first create all required IAM policies and table tags. Otherwise, switching the permissions mode can disrupt users until you either create the correct IAM policies, or revert the permissions mode back to ALLOW_ALL. For information about creating these policies, see [Getting started with the standard permissions mode in Amazon QLDB](#).

You can also use an AWS managed policy to grant full access to all QLDB resources. The AmazonQLDBFullAccess and AmazonQLDBConsoleFullAccess managed policies include all QLDB actions, including all PartiQL actions. Attaching one of these policies to a principal is equivalent to the ALLOW_ALL permissions mode for that principal. For more information, see [AWS managed policies for Amazon QLDB](#).

Deleting a ledger

Use the DeleteLedger operation to delete a ledger and all of its contents. Deleting a ledger is an unrecoverable operation.

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger.

When you issue a `DeleteLedger` request, the ledger's state changes from `ACTIVE` to `DELETING`. It might take a while to delete the ledger, depending on the amount of storage it uses. When the `DeleteLedger` operation concludes, the ledger no longer exists in QLDB.

Deleting a ledger (Java)

To delete a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `DeleteLedgerRequest` class and provide the ledger name that you want to delete.
3. Run the `deleteLedger` method by providing the request object as a parameter.

The following code example demonstrates the preceding steps.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
DeleteLedgerRequest request = new DeleteLedgerRequest().withName(ledgerName);
DeleteLedgerResult result = client.deleteLedger(request);
```

Deleting a ledger (AWS CLI)

Delete your vehicle-registration ledger.

Example

```
aws qlldb delete-ledger --name vehicle-registration
```

Listing ledgers

The `ListLedgers` operation returns summary information of all QLDB ledgers for the current AWS account and Region.

Listing ledgers (Java)

To list ledgers in your account using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.

2. Create an instance of the `ListLedgersRequest` class.

If you received a value for `NextToken` in the response from a previous `ListLedgers` call, you must provide that value in this request to get the next page of results.

3. Run the `listLedgers` method by providing the request object as a parameter.
4. The `listLedgers` request returns a `ListLedgersResult` object. This object has a list of `LedgerSummary` objects and a pagination token that indicates whether there are more results available:
 - If `NextToken` is empty, the last page of results has been processed and there are no more results.
 - If `NextToken` is not empty, there are more results available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListLedgers` call.

The following code example demonstrates the preceding steps.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
List<LedgerSummary> ledgerSummaries = new ArrayList<>();
String nextToken = null;
do {
    ListLedgersRequest request = new ListLedgersRequest().withNextToken(nextToken);
    ListLedgersResult result = client.listLedgers(request);
    ledgerSummaries.addAll(result.getLedgers());
    nextToken = result.getNextToken();
} while (nextToken != null);
```

Listing ledgers (AWS CLI)

List all ledgers in the current AWS account and Region.

Example

```
aws qlldb list-ledgers
```

Creating Amazon QLDB resources with AWS CloudFormation

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all of the AWS resources that you want (such as QLDB ledgers), and AWS CloudFormation provisions and configures those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your QLDB resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

QLDB and AWS CloudFormation templates

To provision and configure resources for QLDB and related services, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the *AWS CloudFormation User Guide*.

QLDB supports creating ledgers and journal streams in AWS CloudFormation. For more information, including examples of JSON and YAML templates for ledgers and streams, see the following resource type references in the *AWS CloudFormation User Guide*:

- [AWS::QLDB::Ledger reference](#)
- [AWS::QLDB::Stream reference](#)

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation API Reference](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

Tagging Amazon QLDB resources

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

A *tag* is a custom attribute label that you assign or that AWS assigns to an AWS resource. Each tag has two parts:

- A *tag key* (for example, CostCenter, Environment, or Project). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, 111122223333 or Production). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an Amazon QLDB ledger that you assign to an Amazon S3 bucket.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see [Using cost allocation tags](#) in the [AWS Billing User Guide](#).
- Control access to your AWS resources with AWS Identity and Access Management (IAM). For information, see [Attribute-based access control \(ABAC\) with QLDB](#) in this developer guide and [Control access using IAM tags](#) in the *IAM User Guide*.

For tips on using tags, see the [AWS Tagging Strategies](#) post on the *AWS Answers* blog.

The following sections provide more information about tags for Amazon QLDB.

Topics

- [Supported resources in Amazon QLDB](#)
- [Tag naming and usage conventions](#)
- [Managing tags](#)
- [Tagging resources on creation](#)

Supported resources in Amazon QLDB

The following resources in Amazon QLDB support tagging:

- ledger
- table
- journal stream

For information about adding and managing tags, see [Managing tags](#).

Tag naming and usage conventions

The following basic naming and usage conventions apply to using tags with Amazon QLDB resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: . : + = @ _ / - (hyphen).
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags, and consistently implement that strategy across all resource types. For example, decide whether to use Costcenter, costcenter, or CostCenter, and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.

- The `aws :` prefix is reserved for AWS use. You can't edit or delete a tag's key or value when the tag has a tag key with the `aws :` prefix. Tags with this prefix do not count against your tags per resource limit.

Managing tags

Tags are made up of the `Key` and `Value` properties on a resource. You can use the Amazon QLDB console, the AWS CLI, or the QLDB API to add, edit, or delete the values for these properties. You can also use the AWS Resource Groups [Tag Editor](#) to manage tags.

For information about working with tags, see the following API operations:

- [ListTagsForResource](#) in the *Amazon QLDB API Reference*
- [TagResource](#) in the *Amazon QLDB API Reference*
- [UntagResource](#) in the *Amazon QLDB API Reference*

To use the QLDB tagging panel (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. In the list of **Ledgers**, choose the ledger name whose tags you want to manage.
4. On the ledger details page, locate the **Tags** card and choose **Manage tags**.
5. On the **Manage tags** page, you can add, edit, or remove any tags as appropriate for your ledger. When the tag keys and values are as you want them, choose **Save**.

Tagging resources on creation

For QLDB resources that support tagging, you can define tags while you're creating the resource by using the AWS Management Console, the AWS CLI, or the QLDB API. By tagging resources while they're being created, you can eliminate the need to run custom tagging scripts after resource creation.

After a resource is tagged, you can control access to the resource based on those tags. For example, you can grant full access only to table resources that have a specific tag. For a JSON policy example, see [Full access to all actions based on table tags](#).

Note

Table and stream resources don't inherit the tags of their root ledger resource.

You can also define table tags by specifying them in a `CREATE TABLE` PartiQL statement. To learn more, see [Tagging tables](#).

Security in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon QLDB, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using QLDB. The following topics show you how to configure QLDB to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your QLDB resources.

Topics

- [Data protection in Amazon QLDB](#)
- [Identity and Access Management for Amazon QLDB](#)
- [Logging and monitoring in Amazon QLDB](#)
- [Compliance validation for Amazon QLDB](#)

- [Resilience in Amazon QLDB](#)
- [Infrastructure security in Amazon QLDB](#)

Data protection in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The AWS [shared responsibility model](#) applies to data protection in Amazon QLDB. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with QLDB or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

 Note

This guidance about avoiding tags or free-form fields for sensitive information refers to the *metadata* of a QLDB ledger resource, rather than data stored within the ledger. Your data stored within a QLDB ledger resource is not used for billing or diagnostic logs.

Topics

- [Encryption at rest in Amazon QLDB](#)
- [Encryption in transit in Amazon QLDB](#)

Encryption at rest in Amazon QLDB

 Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

All data stored in Amazon QLDB is fully encrypted at rest by default. QLDB *encryption at rest* provides enhanced security by encrypting all ledger data at rest using encryption keys in AWS Key Management Service (AWS KMS). This functionality helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive ledger applications that meet strict encryption compliance and regulatory requirements.

Encryption at rest integrates with AWS KMS for managing the encryption key that is used to protect your QLDB ledgers. For more information about AWS KMS, see [AWS Key Management Service concepts](#) in the *AWS Key Management Service Developer Guide*.

In QLDB, you can specify the type of AWS KMS key for each ledger resource. When you create a new ledger or update an existing ledger, you can choose one of the following types of KMS keys to protect your ledger data:

- *AWS owned key* – The default encryption type. The key is owned by QLDB (no additional charge).
- *Customer managed key* – The key is stored in your AWS account and is created, owned, and managed by you. You have full control over the key (AWS KMS charges apply).

Note

Amazon QLDB launched support for customer managed AWS KMS keys on July 22, 2021. Any ledgers that were created before the launch are protected by AWS owned keys by default, but are currently not eligible for encryption at rest using customer managed keys. You can view the creation time of your ledger on the QLDB console.

When you access a ledger, QLDB decrypts the data transparently. You can switch between the AWS owned key and the customer managed key at any given time. You don't have to change any code or applications to use or manage encrypted data.

You can specify an encryption key when you create a new ledger or change the encryption key on an existing ledger by using the AWS Management Console, the QLDB API, or the AWS Command Line Interface (AWS CLI). For more information, see [Using customer managed keys in Amazon QLDB](#).

Note

By default, Amazon QLDB automatically enables encryption at rest using AWS owned keys at no additional charge. However, AWS KMS charges apply for using a customer managed key. For information about pricing, see [AWS Key Management Service pricing](#). QLDB encryption at rest is available in all AWS Regions where QLDB is available.

Topics

- [Encryption at rest: How it works in Amazon QLDB](#)
- [Using customer managed keys in Amazon QLDB](#)

Encryption at rest: How it works in Amazon QLDB

QLDB *encryption at rest* encrypts your data using 256-bit Advanced Encryption Standard (AES-256). This helps secure your data from unauthorized access to the underlying storage. All data stored in QLDB ledgers is encrypted at rest by default. Server-side encryption is transparent, which means that changes to applications aren't required.

Encryption at rest integrates with AWS Key Management Service (AWS KMS) for managing the encryption key that is used to protect your QLDB ledgers. When creating a new ledger or updating an existing ledger, you can choose one of the following types of AWS KMS keys:

- *AWS owned key* – The default encryption type. The key is owned by QLDB (no additional charge).
- *Customer managed key* – The key is stored in your AWS account and is created, owned, and managed by you. You have full control over the key (AWS KMS charges apply).

Topics

- [AWS owned key](#)
- [Customer managed key](#)
- [How Amazon QLDB uses grants in AWS KMS](#)
- [Restoring grants in AWS KMS](#)
- [Encryption at rest considerations](#)

AWS owned key

AWS owned keys aren't stored in your AWS account. They're part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned keys to protect your data.

You don't need to create or manage AWS owned keys. However, you can't view or track AWS owned keys, or audit their use. You aren't charged a monthly fee or a usage fee for AWS owned keys, and they don't count against the AWS KMS quotas for your account.

For more information, see [AWS owned keys](#) in the *AWS Key Management Service Developer Guide*.

Customer managed key

Customer managed keys are KMS keys in your AWS account that you create, own, and manage. You have full control over these KMS keys. QLDB supports symmetric encryption KMS keys only.

Use a customer managed key to get the following features:

- Setting and maintaining key policies, IAM policies, and grants to control access to the key
- Enabling and disabling the key
- Rotating cryptographic material for the key
- Creating key tags and aliases
- Scheduling the key for deletion
- Importing your own key material or using a custom key store that you own and manage
- Using AWS CloudTrail and Amazon CloudWatch Logs to track the requests that QLDB sends to AWS KMS on your behalf

For more information, see [Customer managed keys](#) in the *AWS Key Management Service Developer Guide*.

Customer managed keys [incur a charge](#) for each API call, and AWS KMS quotas apply to these KMS keys. For more information, see [AWS KMS resource or request quotas](#).

When you specify a customer managed key as the KMS key for a ledger, all ledger data in both journal storage and indexed storage is protected with the same customer managed key.

Inaccessible customer managed keys

If you disable your customer managed key, schedule the key for deletion, or revoke the grants on the key, the status of your ledger encryption becomes `KMS_KEY_INACCESSIBLE`. In this state, the ledger is impaired and doesn't accept any read or write requests. An inaccessible key prevents all users and the QLDB service from encrypting or decrypting data—and from performing read and write operations in the ledger. QLDB must have access to your KMS key to ensure that you can continue to access your ledger and to prevent data loss.

Important

An impaired ledger automatically returns to an active state after you restore the grants on the key, or after you reenable the key that was disabled.

However, deleting a customer managed key is *irreversible*. After a key is deleted, you can no longer access the ledgers that are protected with that key, and **the data becomes unrecoverable permanently**.

To check the encryption status of a ledger, use the AWS Management Console or the [DescribeLedger](#) API operation.

How Amazon QLDB uses grants in AWS KMS

QLDB requires *grants* to use your customer managed key. When you create a ledger that is protected with a customer managed key, QLDB creates grants on your behalf by sending [CreateGrant](#) requests to AWS KMS. Grants in AWS KMS are used to give QLDB access to a KMS key in a customer AWS account. For more information, see [Using Grants](#) in the *AWS Key Management Service Developer Guide*.

QLDB requires the grants to use your customer managed key for the following AWS KMS operations:

- [DescribeKey](#) – Verify that the specified symmetric encryption KMS key is valid.
- [GenerateDataKey](#) – Generate a unique symmetric data key that QLDB uses to encrypt data at rest in your ledger.
- [Decrypt](#) – Decrypt the data key that was encrypted by your customer managed key.
- [Encrypt](#) – Encrypt plaintext into ciphertext using your customer managed key.

You can revoke a grant to remove the service's access to the customer managed key at any time. If you do, the key becomes inaccessible, and QLDB loses access to any of the ledger data protected by the customer managed key. In this state, the ledger is impaired and doesn't accept any read or write requests until you restore the grants on the key.

Restoring grants in AWS KMS

To restore grants on a customer managed key and recover access to a ledger in QLDB, you can update the ledger and specify the same KMS key. For instructions, see [Updating the AWS KMS key of an existing ledger](#).

Encryption at rest considerations

Consider the following when you're using encryption at rest in QLDB:

- Server-side encryption at rest is enabled by default on all QLDB ledger data and can't be disabled. You can't encrypt only a subset of data in a ledger.

- Encryption at rest only encrypts data while it is static (at rest) on a persistent storage media. If data security is a concern for data in transit or data in use, you might need to take additional measures as follows:
 - *Data in transit*: All your data in QLDB is encrypted in transit. By default, communications to and from QLDB use the HTTPS protocol, which protects network traffic by using Secure Sockets Layer (SSL)/Transport Layer Security (TLS) encryption.
 - *Data in use*: Protect your data before sending it to QLDB by using client-side encryption.

To learn how to implement customer managed keys for ledgers, proceed to [Using customer managed keys in Amazon QLDB](#).

Using customer managed keys in Amazon QLDB

You can use the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the QLDB API to specify the AWS KMS key for new ledgers and existing ledgers in Amazon QLDB. The following topics describe how to manage and monitor the usage of your customer managed keys in QLDB.

Topics

- [Prerequisites](#)
- [Specifying the AWS KMS key for a new ledger](#)
- [Updating the AWS KMS key of an existing ledger](#)
- [Monitoring your AWS KMS keys](#)

Prerequisites

Before you can protect a QLDB ledger with a customer managed key, you must first create the key in AWS Key Management Service (AWS KMS). You must also specify a key policy that allows QLDB to create grants on that AWS KMS key on your behalf.

Creating a customer managed key

To create a customer managed key, follow the steps in [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*. QLDB doesn't support [asymmetric keys](#).

Setting a key policy

Key policies are the primary way to control access to customer managed keys in AWS KMS. Every customer managed key must have exactly one key policy. The statements in the key policy document determine who has permission to use the KMS key and how they can use it. For more information, see [Using key policies in AWS KMS](#).

You can specify a key policy when you create your customer managed key. To change a key policy for an existing customer managed key, see [Changing a key policy](#).

To allow QLDB to use your customer managed key, the key policy must include permissions for the following AWS KMS actions:

- [kms:CreateGrant](#) – Adds a [grant](#) to a customer managed key. Grants control access to a specified KMS key.

When you create or update a ledger with a specified customer managed key, QLDB creates grants that allow access to the [grant operations](#) that it requires. The grant operations include the following:

- [GenerateDataKey](#)
- [Decrypt](#)
- [Encrypt](#)
- [kms:DescribeKey](#) – Returns detailed information about a customer managed key. QLDB uses this information to validate the key.

Key policy example

The following is a key policy example that you can use for QLDB. This policy allows principals that are authorized to use QLDB from the account 111122223333 to call the DescribeKey and CreateGrant operations on the resource `arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`.

To use this policy, replace *us-east-1*, *111122223333*, and *1234abcd-12ab-34cd-56ef-1234567890ab* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access to principals authorized to use Amazon QLDB",
```

```

    "Effect" : "Allow",
    "Principal" : {
        "AWS" : "*"
    },
    "Action" : [
        "kms:DescribeKey",
        "kms:CreateGrant"
    ],
    "Resource" : "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "Condition" : {
        "StringEquals" : {
            "kms:ViaService" : "qldb.us-east-1.amazonaws.com",
            "kms:CallerAccount" : "111122223333"
        }
    }
}

```

Specifying the AWS KMS key for a new ledger

Follow these steps to specify a KMS key when you create new ledger using the QLDB console or the AWS CLI.

You can specify a customer managed key by using an ID, alias, or Amazon Resource Name (ARN). To learn more, see [Key identifiers \(KeyId\)](#) in the *AWS Key Management Service Developer Guide*.

Note

Cross-Region keys are not supported. The specified KMS key must be in the same AWS Region as your ledger.

Creating a ledger (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. Choose **Create Ledger**.
3. On the **Create Ledger** page, do the following:

- **Ledger information** – Enter a **Ledger name** that is unique among all ledgers in the current AWS account and Region.
- **Permissions mode** – Choose a permissions mode to assign to the ledger:
 - **Allow all**
 - **Standard** (*Recommended*)
- **Encrypt data at rest** – Choose the type of KMS key to use for encryption at rest:
 - **Use AWS owned KMS key** – Use a KMS key that is owned and managed by AWS on your behalf. This is the default option and requires no additional setup.
 - **Choose a different AWS KMS key** – Use a symmetric encryption KMS key in your account that you create, own, and manage.

To create a new key by using the AWS KMS console, choose **Create an AWS KMS key**. For more information, see [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*.

To use an existing KMS key, choose one from the dropdown list or specify a KMS key ARN.

4. When the settings are as you want them, choose **Create ledger**.

You can access your QLDB ledger when its status becomes **Active**. This can take several minutes.

Creating a ledger (AWS CLI)

Use the AWS CLI to create a ledger in QLDB with the default AWS owned key or a customer managed key.

Example – To create a ledger with the default AWS owned key

```
aws qlldb create-ledger --name my-example-ledger --permissions-mode STANDARD
```

Example – To create a ledger with a customer managed key

```
aws qlldb create-ledger \  
  --name my-example-ledger \  
  --permissions-mode STANDARD \  
  --kms-key arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

Updating the AWS KMS key of an existing ledger

You can also use the QLDB console or the AWS CLI to update the KMS key of an existing ledger to an AWS owned key or a customer managed key at any time.

Note

Amazon QLDB launched support for customer managed AWS KMS keys on July 22, 2021. Any ledgers that were created before the launch are protected by AWS owned keys by default, but are currently not eligible for encryption at rest using customer managed keys. You can view the creation time of your ledger on the QLDB console.

Key changes in QLDB are asynchronous. The ledger is fully accessible without any performance impact while the key change is being processed. The amount of time it takes to update a key varies depending on the ledger size.

You can specify a customer managed key by using an ID, alias, or Amazon Resource Name (ARN). To learn more, see [Key identifiers \(KeyId\)](#) in the *AWS Key Management Service Developer Guide*.

Note

Cross-Region keys are not supported. The specified KMS key must be in the same AWS Region as your ledger.

Updating a ledger (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select the ledger that you want to update, and then choose **Edit ledger**.
4. On the **Edit ledger** page, choose the type of KMS key to use for encryption at rest:
 - **Use AWS owned KMS key** – Use a KMS key that is owned and managed by AWS on your behalf. This is the default option and requires no additional setup.
 - **Choose a different AWS KMS key** – Use a symmetric encryption KMS key in your account that you create, own, and manage.

To create a new key by using the AWS KMS console, choose **Create an AWS KMS key**. For more information, see [Creating symmetric encryption KMS keys](#) in the *AWS Key Management Service Developer Guide*.

To use an existing KMS key, choose one from the dropdown list or specify a KMS key ARN.

5. Choose **Confirm changes**.

Updating a ledger (AWS CLI)

Use the AWS CLI to update an existing ledger in QLDB with the default AWS owned key or a customer managed key.

Example – To update a ledger with the default AWS owned key

```
aws qlldb update-ledger --name my-example-ledger --kms-key AWS_OWNED_KMS_KEY
```

Example – To update a ledger with a customer managed key

```
aws qlldb update-ledger \  
  --name my-example-ledger \  
  --kms-key arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

Monitoring your AWS KMS keys

If you use a customer managed key to protect your Amazon QLDB ledgers, you can use [AWS CloudTrail](#) or [Amazon CloudWatch Logs](#) to track the requests that QLDB sends to AWS KMS on your behalf. For more information, see [Monitoring AWS KMS keys](#) in the *AWS Key Management Service Developer Guide*.

The following examples are CloudTrail log entries for the operations `CreateGrant`, `GenerateDataKey`, `Decrypt`, `Encrypt`, and `DescribeKey`.

CreateGrant

When you specify a customer managed key to protect your ledger, QLDB sends `CreateGrant` requests to AWS KMS on your behalf to allow access to your KMS key. In addition, QLDB uses the `RetireGrant` operation to remove grants when you delete a ledger.

The grants that QLDB creates are specific to a ledger. The principal in the CreateGrant request is the user who created the table.

The event that records the CreateGrant operation is similar to the following example event. The parameters include the Amazon Resource Name (ARN) of the customer managed key, the grantee principal and retiring principal (the QLDB service), and the operations that the grant covers.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAIOSFODNN7EXAMPLE:sample-user",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/sample-user",
    "accountId": "111122223333",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2021-06-04T21:37:11Z"
      }
    },
    "invokedBy": "qldb.amazonaws.com"
  },
  "eventTime": "2021-06-04T21:40:00Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "CreateGrant",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "qldb.amazonaws.com",
  "userAgent": "qldb.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "granteePrincipal": "qldb.us-west-2.amazonaws.com",
    "operations": [
```

```

        "DescribeKey",
        "GenerateDataKey",
        "Decrypt",
        "Encrypt"
    ],
    "retiringPrincipal": "qldb.us-west-2.amazonaws.com"
},
"responseElements": {
    "grantId":
    "b3c83f999187ccc0979ef2ff86a1572237b6bba309c0ebce098c34761f86038a"
},
"requestID": "e99188d7-3b82-424e-b63e-e086d848ed60",
"eventID": "88dc7ba5-4952-4d36-9ca8-9ab5d9598bab",
"readOnly": false,
"resources": [
    {
        "accountId": "111122223333",
        "type": "AWS::KMS::Key",
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
],
"eventType": "AwsApiCall",
"managementEvent": true,
"eventCategory": "Management",
"recipientAccountId": "111122223333"
}

```

GenerateDataKey

When you specify a customer managed key to protect your ledger, QLDB creates a unique data key. It sends a `GenerateDataKey` request to AWS KMS that specifies the customer managed key for the ledger.

The event that records the `GenerateDataKey` operation is similar to the following example event. The user is the QLDB service account. The parameters include the ARN of the customer managed key, a data key specifier that requires a 32-byte length, and the encryption context that identifies the internal key hierarchy node.

```

{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",

```

```

    "invokedBy": "qldb.amazonaws.com"
  },
  "eventTime": "2021-06-04T21:40:01Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "qldb.amazonaws.com",
  "userAgent": "qldb.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "numberOfBytes": 32,
    "encryptionContext": {
      "key-hierarchy-node-id": "LY4HWMnkeZWKYi6MlitVJC",
      "key-hierarchy-node-version": "1"
    }
  },
  "responseElements": null,
  "requestID": "786977c9-e77c-467a-bff5-9ad5124a4462",
  "eventID": "b3f082cb-3e75-454e-bf0a-64be13075436",
  "readOnly": true,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "111122223333",
  "sharedEventID": "26688de5-0b1c-43d3-bc4f-a18029b08446"
}

```

Decrypt

When you access a ledger, QLDB calls the Decrypt operation to decrypt the ledger's stored data key so that it can access the encrypted data in the ledger.

The event that records the Decrypt operation is similar to the following example event. The user is the QLDB service account. The parameters include the ARN of the customer managed key and the encryption context that identifies the internal key hierarchy node.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "qldb.amazonaws.com"
  },
  "eventTime": "2021-06-04T21:40:56Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "qldb.amazonaws.com",
  "userAgent": "qldb.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
    "encryptionContext": {
      "key-hierarchy-node-id": "LY4HWMnkeZWKYi6MlitVJC",
      "key-hierarchy-node-version": "1"
    }
  },
  "responseElements": null,
  "requestID": "28f2dd18-3cc1-4fe2-82f7-5154f4933ebf",
  "eventID": "603ad5d4-4744-4505-9c21-bd4a6cbd4b20",
  "readOnly": true,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "111122223333",
  "sharedEventID": "7b6ce3e3-a764-42ec-8f90-5418c97ec411"
}
```

Encrypt

QLDB calls the Encrypt operation to encrypt plaintext into ciphertext using your customer managed key.

The event that records the Encrypt operation is similar to the following example event. The user is the QLDB service account. The parameters include the ARN of the customer managed key and the encryption context that specifies the internal unique ID of the ledger.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "qldb.amazonaws.com"
  },
  "eventTime": "2021-06-04T21:40:01Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Encrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "qldb.amazonaws.com",
  "userAgent": "qldb.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "encryptionContext": {
      "LedgerId": "F6qRNziJLUXA4Vy2ZUv8YY"
    },
    "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
  },
  "responseElements": null,
  "requestID": "b2daca7d-4606-4302-a2d7-5b3c8d30c64d",
  "eventID": "b8aace05-2e37-4fed-ae6f-a45a1c6098df",
  "readOnly": true,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
```

```

    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "ce420ab0-288e-4b4f-ae8-541e18a28aa5"
  }

```

DescribeKey

QLDB calls the `DescribeKey` operation to determine whether the KMS key that you specified exists in the AWS account and Region.

The event that records the `DescribeKey` operation is similar to the following example event. The principal is the user in your AWS account who specified the KMS key. The parameters include the ARN of the customer managed key.

```

{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAIOSFODNN7EXAMPLE:sample-user",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/sample-user",
    "accountId": "111122223333",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2021-06-04T21:37:11Z"
      }
    },
    "invokedBy": "qldb.amazonaws.com"
  },
  "eventTime": "2021-06-04T21:40:00Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "DescribeKey",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "qldb.amazonaws.com",

```

```

    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
      "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "a30586af-c783-4d25-8fda-33152c816c36",
    "eventID": "7a9caf07-2b27-44ab-afe4-b259533ebb88",
    "readOnly": true,
    "resources": [
      {
        "accountId": "111122223333",
        "type": "AWS::KMS::Key",
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
      }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
  }
}

```

Encryption in transit in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB only accepts secure connections that use the HTTPS protocol, which protects network traffic by using Secure Sockets Layer (SSL)/Transport Layer Security (TLS). *Encryption in transit* provides an additional layer of data protection by encrypting your data as it travels to and from QLDB. Organizational policies, industry or government regulations, and compliance requirements often require the use of encryption in transit to increase the data security of your applications when they transmit data over the network.

QLDB also offers FIPS endpoints in selected Regions. Unlike standard AWS endpoints, FIPS endpoints use a TLS software library that complies with Federal Information Processing Standard (FIPS) 140-2. These endpoints might be required by enterprises that interact with the United States government. For more information, see [FIPS endpoints](#) in the *AWS General Reference*. For a complete list of Regions and endpoints that are available for QLDB, see [Amazon QLDB endpoints and quotas](#).

Identity and Access Management for Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use QLDB resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How Amazon QLDB works with IAM](#)
- [Getting started with the standard permissions mode in Amazon QLDB](#)
- [Identity-based policy examples for Amazon QLDB](#)
- [Cross-service confused deputy prevention](#)
- [AWS managed policies for Amazon QLDB](#)
- [Troubleshooting Amazon QLDB identity and access](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in QLDB.

Service user – If you use the QLDB service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more QLDB features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in QLDB, see [Troubleshooting Amazon QLDB identity and access](#).

Service administrator – If you're in charge of QLDB resources at your company, you probably have full access to QLDB. It's your job to determine which QLDB features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with QLDB, see [How Amazon QLDB works with IAM](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to QLDB. To view example QLDB identity-based policies that you can use in IAM, see [Identity-based policy examples for Amazon QLDB](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If

you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [AWS Multi-factor authentication in IAM](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating

IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [Use cases for IAM users](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can [switch from a user to an IAM role \(console\)](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource

(instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Use an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their

permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choose between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [Service control policies](#) in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – RCPs are JSON policies that you can use to set the maximum available permissions for resources in your accounts without updating the IAM policies attached to each resource that you own. The RCP limits permissions for resources in member accounts and can impact the effective permissions for identities, including the AWS account root user, regardless of whether they belong to your organization. For more information about

- Organizations and RCPs, including a list of AWS services that support RCPs, see [Resource control policies \(RCPs\)](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How Amazon QLDB works with IAM

Before you use IAM to manage access to QLDB, learn what IAM features are available to use with QLDB.

 **Important**

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

IAM features you can use with Amazon QLDB

IAM feature	QLDB support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes

IAM feature	QLDB support
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	No
Service roles	Yes
Service-linked roles	No

To get a high-level view of how QLDB and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for QLDB

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for QLDB

To view examples of QLDB identity-based policies, see [Identity-based policy examples for Amazon QLDB](#).

Resource-based policies within QLDB

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for QLDB

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of QLDB actions, see [Actions defined by Amazon QLDB](#) in the *Service Authorization Reference*.

Policy actions in QLDB use the following prefix before the action:

```
qldb
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
    "qldb:action1",  
    "qldb:action2"  
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "qldb:Describe*"
```

To interact with the QLDB transactional data API (*QLDB Session*) by running [PartiQL](#) statements on a ledger, you must grant permission to the `SendCommand` action as follows.

```
"Action": "qldb:SendCommand"
```

For ledgers in the `STANDARD` permissions mode, see the [PartiQL permissions reference](#) for additional required permissions for each PartiQL command.

To view examples of QLDB identity-based policies, see [Identity-based policy examples for Amazon QLDB](#).

Policy resources for QLDB

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of QLDB resource types and their ARNs, see [Resources defined by Amazon QLDB](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions defined by Amazon QLDB](#).

In QLDB, the primary resources are *ledgers*. QLDB also supports additional resource types: *tables* and *streams*. However, you can create tables and streams only in the context of an existing ledger.

A QLDB table is a materialized view of an unordered collection of document revisions from the ledger's journal. In the STANDARD permissions mode of a ledger, you must create IAM policies that grant permissions to run PartiQL statements on this table resource. With permissions on a table resource, you can run statements that access the current state of the table. You can also query the revision history of the table by using the built-in `history()` function. To learn more, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Note

The `CREATE TABLE` statement creates a table with a unique ID and the provided table name. The provided table name must be unique among all active tables. However, QLDB lets you deactivate tables, so there might be multiple inactive tables that share the same table name. Therefore, table resource ARNs refer to the system-assigned unique ID rather than the user-defined table name.

Each ledger also provides a system-defined catalog resource that you can query to list all of the tables and indexes in a ledger. For more information about the QLDB data object model, see [Core concepts and terminology in Amazon QLDB](#).

These resources have unique ARNs associated with them, as shown in the following table.

Resource Type	ARN
ledger	<code>arn:\${Partition}:qldb:\${Region}:\${Account}:ledger/\${LedgerName}</code>

Resource Type	ARN
table	arn:\${Partition}:qldb:\${Region}:\${Account}:ledger/\${LedgerName}/table/\${TableId}
catalog	arn:\${Partition}:qldb:\${Region}:\${Account}:ledger/\${LedgerName}/information_schema/user_tables
stream	arn:\${Partition}:qldb:\${Region}:\${Account}:stream/\${LedgerName}/\${StreamId}

For example, to specify the myExampleLedger resource in your statement, use the following ARN.

```
"Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
```

To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
    "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger1",
    "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger2"
]
```

To view examples of QLDB identity-based policies, see [Identity-based policy examples for Amazon QLDB](#).

Policy condition keys for QLDB

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition *block*) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of QLDB condition keys, see [Condition keys for Amazon QLDB](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon QLDB](#).

The `PartiQLDropIndex` and `PartiQLDropTable` actions support the `qldb:Purge` condition key. This condition key filters access by the value of `purge` that is specified in a PartiQL `DROP` statement. However, QLDB currently supports only `purge = true` for `DROP INDEX` statements, and `purge = false` for `DROP TABLE` statements. Other QLDB actions support some global condition keys.

To view examples of QLDB identity-based policies, see [Identity-based policy examples for Amazon QLDB](#).

Access control lists (ACLs) in QLDB

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with QLDB

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then

you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [Define permissions with ABAC authorization](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

For more information about tagging QLDB resources, see [Tagging Amazon QLDB resources](#).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see [Updating QLDB ledgers based on tags](#).

Using Temporary credentials with QLDB

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switch from a user to an IAM role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate

temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

Cross-service principal permissions for QLDB

Supports forward access sessions (FAS): No

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for QLDB

Supports service roles: Yes

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break QLDB functionality. Edit service roles only when QLDB provides guidance to do so.

QLDB supports service roles for the `ExportJournalToS3` and `StreamJournalToKinesis` API operations, as described in the following section.

Choosing an IAM role in QLDB

When you export or stream journal blocks in QLDB, you must choose a role to allow QLDB to write objects to the given destination on your behalf. If you have previously created a service role, then QLDB provides you with a list of roles to choose from. It's important to choose a role that allows access to write into your specified Amazon S3 bucket for an export, or to your specified Amazon Kinesis Data Streams resource for a stream. For more information, see [Journal export permissions in QLDB](#) or [Stream permissions in QLDB](#).

Note

To pass a role to QLDB when requesting a journal export or stream, you must have permissions to perform the `iam:PassRole` action on the IAM role resource. This is in addition to permissions to perform `qldb:ExportJournalToS3` on the QLDB ledger resource, or `qldb:StreamJournalToKinesis` on the QLDB stream subresource.

Service-linked roles for QLDB

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](#). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Getting started with the standard permissions mode in Amazon QLDB

Use this section to get started with the standard permissions mode in Amazon QLDB. This section provides a reference table to help you when writing an identity-based policy in AWS Identity and Access Management (IAM) for PartiQL actions and table resources in QLDB. It also includes a step-by-step tutorial for creating permissions policies in IAM, and instructions for finding a table ARN and creating table tags in QLDB.

Topics

- [The STANDARD permissions mode](#)
- [PartiQL permissions reference](#)
- [Finding a table ID and ARN](#)
- [Tagging tables](#)
- [Quick start tutorial: Creating permissions policies](#)

The STANDARD permissions mode

QLDB now supports a STANDARD permissions mode for ledger resources. The standard permissions mode enables access control with finer granularity for ledgers, tables, and PartiQL commands. By default, this mode denies all requests to run any PartiQL commands on any tables in a ledger.

Note

Previously, the only available permissions mode for a ledger was ALLOW_ALL. The ALLOW_ALL mode enables access control with API-level granularity for ledgers, and continues to be supported—but is not recommended—for QLDB ledgers. This mode allows users who have the SendCommand API permission to run all PartiQL commands on any tables in the ledger that is specified by the permissions policy (hence, "allow all" PartiQL commands).

You can change the permissions mode of existing ledgers from ALLOW_ALL to STANDARD. For information, see [Migrating to the standard permissions mode](#).

To allow commands in the standard mode, you must create a permissions policy in IAM for specific table resources and PartiQL actions. This is in addition to the SendCommand API permission for the ledger. To facilitate policies in this mode, QLDB introduced a [set of IAM actions](#) for PartiQL commands, and Amazon Resource Names (ARNs) for QLDB tables. For more information about the QLDB data object model, see [Core concepts and terminology in Amazon QLDB](#).

PartiQL permissions reference

The following table lists each QLDB PartiQL command, the corresponding IAM actions that you must grant permissions for to perform the command, and the AWS resources that you can grant the permissions for. You specify the actions in the policy's Action field, and you specify the resource value in the policy's Resource field.

Important

- IAM policies that grant permissions to these PartiQL commands only apply to your ledger if the STANDARD permissions mode is assigned to the ledger. Such policies are not applicable to ledgers in the ALLOW_ALL permissions mode.

To learn how to specify the permissions mode when you're creating or updating a ledger, see [Basic operations for Amazon QLDB ledgers](#), or [Step 1: Create a new ledger](#) in *Getting started with the console*.

- To run any PartiQL commands on a ledger, you must also grant permission to the SendCommand API action for the ledger resource. This is in addition to the PartiQL actions and table resources that are listed in the following table. For more information, see [Running data transactions](#).

Amazon QLDB PartiQL commands and required permissions

Command	Required permissions (IAM actions)	Resources	Dependent actions
CREATE TABLE	qldb:PartiQLCreateTable	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/*	qldb:TagResource (for tagging on creation)
DROP TABLE	qldb:PartiQLDropTable	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	
UNDROP TABLE	qldb:PartiQLUndropTable	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	
CREATE INDEX	qldb:PartiQLCreateIndex	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	

Command	Required permissions (IAM actions)	Resources	Dependent actions
DROP INDEX	qldb:Part iQLDropIndex	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	
DELETE FROM-REMOVE (for entire documents)	qldb:Part iQLDelete	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	qldb:Part iQLSelect
INSERT	qldb:Part iQLInsert	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	
UPDATE FROM (INSERT, REMOVE, or SET)	qldb:Part iQLUpdate	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	qldb:Part iQLSelect
REDACT_REVISION (stored procedure)	qldb:Part iQLRedact	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	

Command	Required permissions (IAM actions)	Resources	Dependent actions
SELECT FROM table_name	qldb:PartiQLSelect	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	
SELECT FROM information_schema.user_tables	qldb:PartiQLSelect	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /information_schema/user_tables	
SELECT FROM history(table_name)	qldb:PartiQLHistoryFunction	arn:aws:qldb: <i>region</i> : <i>account-id</i> :ledger/ <i>ledger-name</i> /table/ <i>table-id</i>	

For examples of IAM policy documents that grant permissions to these PartiQL commands, proceed to [Quick start tutorial: Creating permissions policies](#) or see [Identity-based policy examples for Amazon QLDB](#).

Finding a table ID and ARN

You can find a table ID by using the AWS Management Console or by querying the table [information_schema.user_tables](#). To view table details on the console, or to query this system catalog table, you must have SELECT permission on the system catalog resource. For example, to find the table ID of the `Vehicle` table, you can run the following statement.

```
SELECT * FROM information_schema.user_tables
WHERE name = 'Vehicle'
```

This query returns results in a format similar to the following example.

```
{
  tableId: "Au1EiThbt8s0z9wM26REZN",
  name: "Vehicle",
  indexes: [
    { indexId: "Djg2nt0yIs2GY0T29Kud1z", expr: "[VIN]", status: "ONLINE" },
    { indexId: "4tPW3fUhaVhDinRgKRLhGU", expr: "[LicensePlateNumber]", status:
"BUILDING" }
  ],
  status: "ACTIVE"
}
```

To grant permissions to run PartiQL statements on a table, you specify a table resource in the following ARN format.

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/table/${table-id}
```

The following is an example of a table ARN for table ID Au1EiThbt8s0z9wM26REZN.

```
arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/Au1EiThbt8s0z9wM26REZN
```

Using the console

You can also use the QLDB console to find a table ARN.

To find the ARN of a table (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. In the list of **Ledgers**, choose the ledger name whose table ARN you want to find.
4. On the ledger details page, under the **Tables** tab, locate the table name whose ARN you want to find. To copy the ARN, choose the copy icon



next to it.

Tagging tables

You can tag your table resources. To manage tags for existing tables, use the AWS Management Console or the API operations `TagResource`, `UntagResource`, and `ListTagsForResource`. For more information, see [Tagging Amazon QLDB resources](#).

Note

Table resources don't inherit the tags of their root ledger resource. Tagging tables on creation is currently supported for ledgers in the STANDARD permissions mode only.

You can also define table tags while you're creating the table by using the QLDB console or by specifying them in a `CREATE TABLE PartiQL` statement. The following example creates a table named `Vehicle` with the tag `environment=production`.

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

Tagging tables on creation requires access to both the `qldb:PartiQLCreateTable` and `qldb:TagResource` actions.

By tagging resources while they're being created, you can eliminate the need to run custom tagging scripts after resource creation. After a table is tagged, you can control access to the table based on those tags. For example, you can grant full access only to tables that have a specific tag. For a JSON policy example, see [Full access to all actions based on table tags](#).

Using the console

You can also use the QLDB console to define table tags while you're creating the table.

To tag a table on creation (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at <https://console.aws.amazon.com/qldb>.
2. In the navigation pane, choose **Ledgers**.
3. In the list of **Ledgers**, choose the ledger name that you want to create the table in.
4. On the ledger details page, under the **Tables** tab, choose **Create table**.

5. On the **Create table** page, do the following:

- **Table name** – Enter a table name.
- **Tags** – Add metadata to the table by attaching tags as key-value pairs. You can add tags to your table to help organize and identify them.

Choose **Add tag**, and then enter any key-value pairs as appropriate.

6. When the settings are as you want them, choose **Create table**.

Quick start tutorial: Creating permissions policies

This tutorial guides you through steps to create permissions policies in IAM for an Amazon QLDB ledger in the STANDARD permissions mode. You can then assign the permissions to your users, groups, or roles.

For more examples of IAM policy documents that grant permissions to PartiQL commands and table resources, see [Identity-based policy examples for Amazon QLDB](#).

Topics

- [Prerequisites](#)
- [Creating a read-only policy](#)
- [Creating a full access policy](#)
- [Creating a read-only policy for a specific table](#)
- [Assigning permissions](#)

Prerequisites

Before you get started, make sure that you do the following:

1. Follow the AWS setup instructions in [Accessing Amazon QLDB](#), if you haven't already done so. These steps include signing up for AWS and creating an administrative user.
2. Create a new ledger and choose the STANDARD permissions mode for the ledger. To learn how, see [Step 1: Create a new ledger](#) in *Getting started with the console*, or [Basic operations for Amazon QLDB ledgers](#).

Creating a read-only policy

To use the JSON policy editor to create a read-only policy for *all tables* in a ledger in the standard permissions mode, do the following:

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation column on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. At the top of the page, choose **Create policy**.
4. Choose the **JSON** tab.
5. Copy and paste the following JSON policy document. This example policy grants *read-only* access to all tables in a ledger.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/information_schema/user_tables"
      ]
    }
  ]
}
```

```
]
}
```

6. Choose **Review policy**.

Note

You can switch between the **Visual editor** and **JSON** tabs any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring](#) in the *IAM User Guide*.

- On the **Review policy** page, enter a **Name** and an optional **Description** for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

Creating a full access policy

To create a full access policy for all tables in a QLDB ledger in the standard permissions mode, do the following:

- Repeat the [previous steps](#) using the following policy document. This example policy grants access to *all PartiQL commands* for *all tables* in a ledger, by using wildcards (*) to cover all PartiQL actions and all resources under a ledger.

Warning

This is an example of using a wildcard character (*) to allow all PartiQL actions, including administrative and read/write operations on all tables in a QLDB ledger. Instead, it's a best practice to explicitly specify each action to be granted, and only what that user, role, or group needs.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

{
  "Sid": "QLDBSendCommandPermission",
  "Effect": "Allow",
  "Action": "qldb:SendCommand",
  "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
},
{
  "Sid": "QLDBPartiQLFullPermissions",
  "Effect": "Allow",
  "Action": [
    "qldb:PartiQL*"
  ],
  "Resource": [
    "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
    "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
  ]
}
]
}

```

Creating a read-only policy for a specific table

To create a read-only access policy for a specific table in a QLDB ledger in the standard permissions mode, do the following:

1. Find the ARN for the table by using the AWS Management Console or by querying the system catalog table `information_schema.user_tables`. For instructions, see [Finding a table ID and ARN](#).
2. Use the table ARN to create a policy that allows read-only access to the table. To do this, repeat the [previous steps](#) using the following policy document.

This example policy grants *read-only* access to the *specified table only*. In this example, the table ID is `Au1EiThbt8s0z9wM26REZN`. To use this policy, replace `us-east-1`, `123456789012`, `myExampleLedger`, and `Au1EiThbt8s0z9wM26REZN` in the example with your own information.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {

```

```

    "Sid": "QLDBSendCommandPermission",
    "Effect": "Allow",
    "Action": "qldb:SendCommand",
    "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
  },
  {
    "Sid": "QLDBPartiQLReadOnlyPermissions",
    "Effect": "Allow",
    "Action": [
      "qldb:PartiQLSelect",
      "qldb:PartiQLHistoryFunction"
    ],
    "Resource": [
      "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
table/Au1EiThbt8s0z9wM26REZN"
    ]
  }
]
}

```

Assigning permissions

After creating a QLDB permissions policy, you then assign the permissions as follows.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:

- Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
- (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

Identity-based policy examples for Amazon QLDB

By default, users and roles don't have permission to create or modify QLDB resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Create IAM policies \(console\)](#) in the *IAM User Guide*.

For details about actions and resource types defined by QLDB, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for Amazon QLDB](#) in the *Service Authorization Reference*.

Contents

- [Policy best practices](#)
- [Using the QLDB console](#)
 - [Query history permissions](#)
 - [Full access console permissions without query history](#)
- [Allow users to view their own permissions](#)
- [Running data transactions](#)
 - [Standard permissions for PartiQL actions and table resources](#)
 - [Full access to all actions](#)
 - [Full access to all actions based on table tags](#)
 - [Read/write access](#)
 - [Read-only access](#)
 - [Read-only access to a specific table](#)
 - [Allow access to create tables](#)
 - [Allow access to create tables based on request tags](#)
- [Exporting a journal to an Amazon S3 bucket](#)
- [Streaming a journal to Kinesis Data Streams](#)
- [Updating QLDB ledgers based on tags](#)

⚠ Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Policy best practices

Identity-based policies determine whether someone can create, access, or delete QLDB resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the QLDB console

To access the Amazon QLDB console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the QLDB resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles have full access to the QLDB console and all of its features, attach the following AWS managed policy to the entities. For more information, see [AWS managed policies for Amazon QLDB](#), and [Adding permissions to a user](#) in the *IAM User Guide*.

```
AmazonQLDBConsoleFullAccess
```

Query history permissions

In addition to QLDB permissions, some console features require permissions for the *Database Query Metadata Service* (service prefix: dbqms). This is an internal-only service that manages your recent and saved queries on the console query editor for QLDB and other AWS services. For a full list of DBQMS API actions, see [Database Query Metadata Service](#) in the *Service Authorization Reference*.

To allow query history permissions, you can use the AWS managed policy [AmazonQLDBConsoleFullAccess](#). This policy uses a wildcard (dbqms : *) to allow all DBQMS actions for all resources.

Or, you can create a custom IAM policy and include the following DBQMS actions. The PartiQL query editor on the QLDB console requires permissions to use these actions for query history features.

```
dbqms:CreateFavoriteQuery
dbqms:CreateQueryHistory
dbqms>DeleteFavoriteQueries
dbqms>DeleteQueryHistory
dbqms:DescribeFavoriteQueries
dbqms:DescribeQueryHistory
dbqms:UpdateFavoriteQuery
```

Full access console permissions without query history

To allow full access to the QLDB console *without* any query history permissions, you can create a custom IAM policy that excludes [all DBQMS actions](#). For example, the following policy document allows the same permissions that are granted by the AWS managed policy [AmazonQLDBConsoleFullAccess](#), *except* actions that begin with the service prefix dbqms.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "qldb:CreateLedger",
        "qldb:UpdateLedger",
        "qldb:UpdateLedgerPermissionsMode",
        "qldb>DeleteLedger",
        "qldb:ListLedgers",
        "qldb:DescribeLedger",
        "qldb:ExportJournalToS3",
        "qldb:ListJournalS3Exports",
        "qldb:ListJournalS3ExportsForLedger",
        "qldb:DescribeJournalS3Export",
        "qldb:CancelJournalKinesisStream",
        "qldb:DescribeJournalKinesisStream",
        "qldb:ListJournalKinesisStreamsForLedger",
        "qldb:StreamJournalToKinesis",
        "qldb:GetBlock",
        "qldb:GetDigest",
        "qldb:GetRevision",
        "qldb:TagResource",
```

```

        "qldb:UntagResource",
        "qldb:ListTagsForResource",
        "qldb:SendCommand",
        "qldb:ExecuteStatement",
        "qldb:ShowCatalog",
        "qldb:InsertSampleData",
        "qldb:PartiQLCreateIndex",
        "qldb:PartiQLDropIndex",
        "qldb:PartiQLCreateTable",
        "qldb:PartiQLDropTable",
        "qldb:PartiQLUndropTable",
        "qldb:PartiQLDelete",
        "qldb:PartiQLInsert",
        "qldb:PartiQLUpdate",
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
    ],
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": [
        "kinesis:ListStreams",
        "kinesis:DescribeStream"
    ],
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "qldb.amazonaws.com"
        }
    }
}
]
}

```

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Running data transactions

To interact with the QLDB transactional data API (*QLDB Session*) by running [PartiQL](#) statements on a ledger, you must grant permission to the `SendCommand` API action. The following JSON document is an example of a policy that grants permission to only the `SendCommand` API action on the ledger `myExampleLedger`.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    }
  ]
}
```

If `myExampleLedger` uses the `ALLOW_ALL` permissions mode, this policy grants permissions to run all PartiQL commands on any table in the ledger.

You can also use an AWS managed policy to grant full access to all QLDB resources. For more information, see [AWS managed policies for Amazon QLDB](#).

Standard permissions for PartiQL actions and table resources

For ledgers in the `STANDARD` permissions mode, you can refer to the following IAM policy documents as examples of granting the appropriate PartiQL permissions. For a list of the required permissions for each PartiQL command, see the [PartiQL permissions reference](#).

Topics

- [Full access to all actions](#)
- [Full access to all actions based on table tags](#)
- [Read/write access](#)
- [Read-only access](#)

- [Read-only access to a specific table](#)
- [Allow access to create tables](#)
- [Allow access to create tables based on request tags](#)

Full access to all actions

The following JSON policy document grants full access to use *all* PartiQL commands on *all* tables in *myExampleLedger*. This policy produces the same effect as using the `ALLOW_ALL` permissions mode for the ledger.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLFullPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLCreateIndex",
        "qldb:PartiQLDropIndex",
        "qldb:PartiQLCreateTable",
        "qldb:PartiQLDropTable",
        "qldb:PartiQLUndropTable",
        "qldb:PartiQLDelete",
        "qldb:PartiQLInsert",
        "qldb:PartiQLUpdate",
        "qldb:PartiQLRedact",
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/information_schema/user_tables"
      ]
    }
  ]
}
```



```

    ]
  }
]
}

```

Full access to all actions based on table tags

The following JSON policy document uses a condition that is based on table resource tags to grant full access to use *all* PartiQL commands on *all* tables in *myExampleLedger*. Permissions are granted only if the table tag `environment` has the value `development`.

Warning

This is an example of using a wildcard character (*) to allow all PartiQL actions, including administrative and read/write operations on all tables in a QLDB ledger. Instead, it's a best practice to explicitly specify each action to be granted, and only what that user, role, or group needs.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLFullPermissionsBasedOnTags",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQL*"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/information_schema/user_tables"
      ]
    }
  ]
}

```

```

    ],
    "Condition": {
      "StringEquals": { "aws:ResourceTag/environment": "development" }
    }
  }
]
}

```

Read/write access

The following JSON policy document grants permissions to select, insert, update, and delete data on all tables in `myExampleLedger`. This policy doesn't grant permissions to redact data or alter the schema—for example, creating and dropping tables and indexes.

Note

An UPDATE statement requires permissions to both the `qldb:PartiQLUpdate` and `qldb:PartiQLSelect` actions on the table that is being modified. When you run an UPDATE statement, it performs a read operation in addition to the update operation. Requiring both actions ensures that only users who are allowed to read the contents of a table are granted UPDATE permissions.

Similarly, a DELETE statement requires permissions to both the `qldb:PartiQLDelete` and `qldb:PartiQLSelect` actions.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadWritePermissions",
      "Effect": "Allow",
      "Action": [

```

```

        "qldb:PartiQLDelete",
        "qldb:PartiQLInsert",
        "qldb:PartiQLUpdate",
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
    ],
    "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
    ]
}
]
}

```

Read-only access

The following JSON policy document grants read-only permissions on all tables in myExampleLedger. To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
      ]
    }
  ]
}

```

```
]
}
```

Read-only access to a specific table

The following JSON policy document grants read-only permissions on a specific table in myExampleLedger. In this example, the table ID is Au1EiThbt8s0z9wM26REZN.

To use this policy, replace *us-east-1*, *123456789012*, *myExampleLedger*, and *Au1EiThbt8s0z9wM26REZN* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadOnlyPermissionsOnTable",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
table/Au1EiThbt8s0z9wM26REZN"
      ]
    }
  ]
}
```

Allow access to create tables

The following JSON policy document grants permission to create tables in myExampleLedger. The qldb:PartiQLCreateTable action requires permissions to the table resource type. However, the table ID of a new table isn't known at the time when you run a CREATE TABLE statement. So, a policy that grants the qldb:PartiQLCreateTable permission must use a wildcard (*) in the table ARN to specify the resource.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLCreateTablePermission",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLCreateTable"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*"
      ]
    }
  ]
}
```

Allow access to create tables based on request tags

The following JSON policy document uses a condition based on the `aws:RequestTag` context key to grant permission to create tables in `myExampleLedger`. Permissions are granted only if the request tag environment has the value `development`. Tagging tables on creation requires access to both the `qldb:PartiQLCreateTable` and `qldb:TagResource` actions. To learn how to tag tables on creation, see [Tagging tables](#).

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
```

```

        "Action": "qldb:SendCommand",
        "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
        "Sid": "QLDBPartiQLCreateTablePermission",
        "Effect": "Allow",
        "Action": [
            "qldb:PartiQLCreateTable",
            "qldb:TagResource"
        ],
        "Resource": [
            "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*"
        ],
        "Condition": {
            "StringEquals": { "aws:RequestTag/environment": "development" }
        }
    }
]
}

```

Exporting a journal to an Amazon S3 bucket

Step 1: QLDB journal export permissions

In the following example, you grant a user in your AWS account permissions to perform the `qldb:ExportJournalToS3` action on a QLDB ledger resource. You also grant permissions to perform the `iam:PassRole` action on the IAM role resource that you want to pass to the QLDB service. This is required for all journal export requests.

To use this policy, replace *us-east-1*, *123456789012*, *myExampleLedger*, and *qldb-s3-export* in the example with your own information.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBJournalExportPermission",
      "Effect": "Allow",
      "Action": "qldb:ExportJournalToS3",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "IAMPassRolePermission",

```

```

    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::123456789012:role/qldb-s3-export",
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": "qldb.amazonaws.com"
      }
    }
  }
]
}

```

Step 2: Amazon S3 bucket permissions

In the following example, you use an IAM role to grant QLDB access to write into one of your Amazon S3 buckets, `amzn-s3-demo-bucket`. This is also required for all QLDB journal exports.

In addition to granting the `s3:PutObject` permission, the policy also grants the `s3:PutObjectAcl` permission for the ability to set the access control list (ACL) permissions for an object.

To use this policy, replace `amzn-s3-demo-bucket` in the example with your Amazon S3 bucket name.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBJournalExportS3Permissions",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}

```

Then, you attach this permissions policy to an IAM role that QLDB can assume to access your Amazon S3 bucket. The following JSON document is an example of a trust policy that allows QLDB to assume the IAM role for any QLDB resource in the account `123456789012` only.

To use this policy, replace *us-east-1* and *123456789012* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "qldb.amazonaws.com"
      },
      "Action": [ "sts:AssumeRole" ],
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Streaming a journal to Kinesis Data Streams

Step 1: QLDB journal stream permissions

In the following example, you grant a user in your AWS account permissions to perform the `qldb:StreamJournalToKinesis` action on all QLDB stream subresources in a ledger. You also grant permissions to perform the `iam:PassRole` action on the IAM role resource that you want to pass to the QLDB service. This is required for all journal stream requests.

To use this policy, replace *us-east-1*, *123456789012*, *myExampleLedger*, and *qldb-kinesis-stream* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBJournalStreamPermission",
```



```

    "Effect": "Allow",
    "Action": "qldb:StreamJournalToKinesis",
    "Resource": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
  },
  {
    "Sid": "IAMPassRolePermission",
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::123456789012:role/qldb-kinesis-stream",
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": "qldb.amazonaws.com"
      }
    }
  }
]
}

```

Step 2: Kinesis Data Streams permissions

In the following example, you use an IAM role to grant QLDB access to write data records to your Amazon Kinesis data stream, *stream-for-qldb*. This is also required for all QLDB journal streams.

To use this policy, replace *us-east-1*, *123456789012*, and *stream-for-qldb* in the example with your own information.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBStreamKinesisPermissions",
      "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
        "kinesis:ListShards" ],
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/stream-for-qldb"
    }
  ]
}

```

Then, you attach this permissions policy to an IAM role that QLDB can assume to access your Kinesis data stream. The following JSON document is an example of a trust policy that allows

QLDB to assume an IAM role for any QLDB stream in the account 123456789012 for the ledger `myExampleLedger` only.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "qldb.amazonaws.com"
      },
      "Action": [ "sts:AssumeRole" ],
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Updating QLDB ledgers based on tags

You can use conditions in your identity-based policy to control access to QLDB resources based on tags. This example shows how you might create a policy that allows updating a ledger. However, permission is granted only if the ledger tag `Owner` has the value of the user name of that user. This policy also grants the permissions necessary to complete this action on the console.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListLedgersInConsole",
      "Effect": "Allow",
      "Action": "qldb:ListLedgers",
```

```
        "Resource": "*"
    },
    {
        "Sid": "UpdateLedgerIfOwner",
        "Effect": "Allow",
        "Action": "qldb:UpdateLedger",
        "Resource": "arn:aws:qldb:*:*:ledger/*",
        "Condition": {
            "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
        }
    }
]
```

You can attach this policy to the users in your account. If a user named `richard-roe` tries to update a QLDB ledger, the ledger must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise, he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case sensitive. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

Cross-service confused deputy prevention

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem.

Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent the confused deputy problem, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the [aws:SourceArn](#) and [aws:SourceAccount](#) global condition context keys in resource policies to limit the permissions that Amazon QLDB gives another service to the resource. If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when they're used in the same policy statement.

The following table lists possible values of `aws:SourceArn` for the [ExportJournalToS3](#) and [StreamsJournalToKinesis](#) QLDB API operations. These operations are in scope for this security issue because they call AWS Security Token Service (AWS STS) to assume an IAM role that you specify.

API operation	Called service	aws:SourceArn
ExportJournalToS3	AWS STS (AssumeRole)	Allows QLDB to assume the role for any QLDB resources in the account: <pre>arn:aws:qldb: <i>us-east-1</i> :<i>123456789012</i> :*</pre> Currently, QLDB only supports this wildcard ARN for journal exports.
StreamsJournalToKinesis	AWS STS (AssumeRole)	Allows QLDB to assume the role for a specific QLDB stream: <pre>arn:aws:qldb: <i>us-east-1</i> :<i>123456789012</i> :stream/<i>myExampleLedger /IiPT4brpZCqCq3f4MTHbYy</i></pre> Note: You can only specify a stream ID in the ARN after the stream resource is created. Using this ARN, you can allow the role to only be used for a single QLDB stream. Allows QLDB to assume the role for any QLDB streams of a ledger: <pre>arn:aws:qldb: <i>us-east-1</i> :<i>123456789012</i> :stream/<i>myExampleLedger</i> /*</pre> Allows QLDB to assume the role for any QLDB streams in the account:

API operation	Called service	aws:SourceArn
		<pre>arn:aws:qldb: <i>us-east-1</i> :<i>123456789012</i> :stream/*</pre> Allows QLDB to assume the role for any QLDB resources in the account: <pre>arn:aws:qldb: <i>us-east-1</i> :<i>123456789012</i> :*</pre>

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you're specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcard characters (*) for the unknown portions of the ARN—for example, `arn:aws:qldb:us-east-1:123456789012:*`.

The following trust policy example for an IAM role shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys to prevent the confused deputy problem. With this trust policy, QLDB can assume the role for any QLDB stream in the account `123456789012` for the ledger `myExampleLedger` only.

To use this policy, replace *us-east-1*, *123456789012*, and *myExampleLedger* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ConfusedDeputyPreventionExamplePolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "qldb.amazonaws.com"
      },
      "Action": [ "sts:AssumeRole" ],
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

```
}  
}  
}  
}
```

AWS managed policies for Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see [AWS managed policies](#) in the *IAM User Guide*.

For more information about the QLDB API operations in these AWS managed policies, see the [Amazon QLDB API reference](#).

Topics

- [AWS managed policy: AmazonQLDBReadOnly](#)
- [AWS managed policy: AmazonQLDBFullAccess](#)

- [AWS managed policy: AmazonQLDBConsoleFullAccess](#)
- [QLDB updates to AWS managed policies](#)

AWS managed policy: AmazonQLDBReadOnly

Use the [AmazonQLDBReadOnly](#) policy to grant *read-only* permissions to all QLDB resources. You can attach this policy to your IAM identities.

Permissions details

This policy includes the following permissions for the `qldb` service.

- Allows principals to describe and list all QLDB resources and their tags. These resources include ledgers, Amazon S3 export jobs, and streams to Kinesis Data Streams.
- Allows principals to get a block, digest, or revision from the journal in any ledger to verify the data cryptographically.
- Doesn't allow principals to run any PartiQL commands on any tables in any ledgers.

AWS managed policy: AmazonQLDBFullAccess

Use the [AmazonQLDBFullAccess](#) policy to grant full *administrative* permissions to all QLDB resources through the QLDB API or the AWS CLI. You can attach this policy to your IAM identities.

Permissions details

This policy includes the following permissions.

- `qldb`
 - Allows principals to create, describe, list, and manage all QLDB resources and their tags. These resources include ledgers, Amazon S3 export jobs, and streams to Kinesis Data Streams.
 - Allows principals to run all PartiQL commands on all tables in any ledger by using the [QLDB driver](#) or the [QLDB shell](#).
 - Allows principals to get a block, digest, or revision from the journal in any ledger to verify the data cryptographically.

- `iam` – Allows principals to pass any IAM role resource in your account to the QLDB service. This is required for all journal export and stream requests.

AWS managed policy: `AmazonQLDBConsoleFullAccess`

Use the [AmazonQLDBConsoleFullAccess](#) policy to grant full *administrative* permissions to all QLDB resources through the AWS Management Console, the QLDB API, or the AWS CLI. You can attach this policy to your IAM identities.

Permissions details

This policy includes the following permissions.

- `qldb`
 - Allows principals to create, describe, list, and manage all QLDB resources and their tags. These resources include ledgers, Amazon S3 export jobs, and streams to Kinesis Data Streams.
 - Allows principals to run all PartiQL commands on all tables in any ledger by using the QLDB console, the [QLDB driver](#), or the [QLDB shell](#).
 - Allows principals to insert sample application data in any ledger by using the QLDB console.
 - Allows principals to get a block, digest, or revision from the journal in any ledger to verify the data cryptographically.
- `dbqms` – Allows principals to use all actions in the [Database Query Metadata Service](#). This is an internal-only service that the QLDB console requires to create, describe, and manage recent and saved queries for the PartiQL query editor.
- `kinesis` – Allows principals to describe and list Amazon Kinesis Data Streams resources. These resources are the target destinations that QLDB stream resources can write data to.
- `iam` – Allows principals to pass any IAM role resource in your account to the QLDB service. This is required for all journal export and stream requests.

QLDB updates to AWS managed policies

View details about updates to AWS managed policies for QLDB since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the QLDB [Release history](#) page.

Change	Description	Date
AmazonQLDBFullAccess , AmazonQLDBConsoleFullAccess – Update to existing policies	QLDB added a new permission to allow principals to redact document revisions in all ledgers in the STANDARD permissions mode.	November 4, 2022
AmazonQLDBFullAccess , AmazonQLDBConsoleFullAccess – Update to existing policies	QLDB added new permissions to allow principals to pass any IAM role resource in your account to the QLDB service. This is required for all journal export and stream requests.	September 2, 2021
AmazonQLDBReadOnly – Update to an existing policy	QLDB removed a duplicate <code>qldb:GetBlock</code> action that was previously listed twice, and reordered the "Effect" field so that it appears before the "Action" field.	July 1, 2021
AmazonQLDBFullAccess , AmazonQLDBConsoleFullAccess – Update to existing policies	QLDB added new permissions to allow principals to update the permissions mode in all ledgers, and to run all PartiQL commands in all ledgers in the new STANDARD permissions mode. The STANDARD permissions mode supports table-level access control and granularity for PartiQL commands. To facilitate the new permissions mode, QLDB introduced a set of IAM actions for PartiQL	May 27, 2021

Change	Description	Date
	command types, and Amazon Resource Names (ARNs) for QLDB table resources. These two policies are updated to include the new PartiQL actions to grant full access to STANDARD ledgers.	
QLDB started tracking changes	QLDB started tracking changes for its AWS managed policies.	March 1, 2021

Troubleshooting Amazon QLDB identity and access

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Use the following information to help you diagnose and fix common issues that you might encounter when working with QLDB and IAM.

Topics

- [I am not authorized to perform an action in QLDB](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my QLDB resources](#)

I am not authorized to perform an action in QLDB

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a fictional *myExampleLedger* resource but does not have the fictional `qldb:DescribeLedger` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
qldb:DescribeLedger on resource: myExampleLedger
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *myExampleLedger* resource using the `qldb:DescribeLedger` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to QLDB.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in QLDB. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

For troubleshooting guidance about this error specific to the journal export or stream operations, see [Troubleshooting Amazon QLDB](#).

I want to allow people outside of my AWS account to access my QLDB resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether QLDB supports these features, see [How Amazon QLDB works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Logging and monitoring in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon QLDB and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. However, before you start monitoring QLDB, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

The next step is to establish a baseline for normal QLDB performance in your environment, by measuring performance at various times and under different load conditions. As you monitor QLDB, store historical monitoring data so that you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

To establish a baseline you should, at a minimum, monitor the following items:

- Read and write I/Os and storage, so that you can track your ledger's consumption patterns for billing purposes.
- Command latency, so that you can track your ledger's performance when running data operations.
- Exceptions, so that you can determine whether any requests resulted in an error.

Topics

- [Monitoring tools](#)
- [Monitoring with Amazon CloudWatch](#)
- [Automating Amazon QLDB with CloudWatch Events](#)
- [Logging Amazon QLDB API calls with AWS CloudTrail](#)

Monitoring tools

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

AWS provides various tools that you can use to monitor Amazon QLDB. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Topics

- [Automated monitoring tools](#)
- [Manual monitoring tools](#)

Automated monitoring tools

You can use the following automated monitoring tools to watch QLDB and report when something is wrong:

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see [Monitoring with Amazon CloudWatch](#).
- **Amazon CloudWatch Logs** – Monitor, store, and access your log files from AWS CloudTrail or other sources. For more information, see [Monitoring log files](#) in the *Amazon CloudWatch User Guide*.
- **Amazon CloudWatch Events** – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see [What is Amazon CloudWatch Events](#) in the *Amazon CloudWatch User Guide*.
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see [Working with CloudTrail log files](#) in the *AWS CloudTrail User Guide*.

Manual monitoring tools

Another important part of monitoring QLDB involves manually monitoring those items that the CloudWatch alarms don't cover. The QLDB, CloudWatch, Trusted Advisor, and other AWS Management Console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on Amazon QLDB.

- The QLDB dashboard shows the following:
 - Read and write I/Os
 - Journal and indexed storage
 - Command latency
 - Exceptions
- The CloudWatch home page shows the following:

- Current alarms and status
- Graphs of alarms and resources
- Service health status

In addition, you can use CloudWatch to do the following:

- Create [customized dashboards](#) to monitor the services you care about
- Graph metric data to troubleshoot issues and discover trends
- Search and browse all your AWS resource metrics
- Create and edit alarms to be notified of problems

Monitoring with Amazon CloudWatch

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can monitor Amazon QLDB using CloudWatch, which collects and processes raw data from Amazon QLDB into readable, near-real-time metrics. It records these statistics for two weeks so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, QLDB metric data is automatically sent to CloudWatch in 1 or 15-minute periods. For more information, see [What are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs?](#) in the *Amazon CloudWatch User Guide*.

Topics

- [How do I use QLDB metrics?](#)
- [Amazon QLDB metrics and dimensions](#)
- [Creating CloudWatch alarms to monitor Amazon QLDB](#)

How do I use QLDB metrics?

The metrics reported by QLDB provide information that you can analyze in different ways. The following list shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list.

- You can monitor `JournalStorage` and `IndexedStorage` over a specified time period, to track how much disk space your ledger is consuming.
- You can monitor `ReadIOs` and `WriteIOs` over a specified time period, to track how many requests your ledger is processing.
- You can monitor `CommandLatency` to track your ledger's performance for data operations and analyze the types of commands that result in the most latency.

Amazon QLDB metrics and dimensions

When you interact with Amazon QLDB, it sends the following metrics and dimensions to CloudWatch. Storage metrics are reported every 15 minutes, and all other metrics are aggregated and reported every minute. You can use the following procedures to view the metrics for QLDB.

To view metrics using the CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region. On the navigation bar, choose the Region where your AWS resources reside. For more information, see [Regions and endpoints](#).
3. In the navigation pane, choose **Metrics**.
4. Under the **All metrics** tab, choose **QLDB**.

To view metrics using the AWS CLI

- At a command prompt, use the following command.

```
aws cloudwatch list-metrics --namespace "AWS/QLDB"
```

CloudWatch displays the following metrics for QLDB.

Amazon QLDB dimensions and metrics

The metrics and dimensions that Amazon QLDB sends to Amazon CloudWatch are listed here.

QLDB metrics

Metric	Description
JournalStorage	The total amount of disk space used by the ledger's journal, reported in 15-minute intervals. The journal contains the complete, immutable, and verifiable history of all the changes to your data. Units: Bytes Dimensions: LedgerName
IndexedStorage	The total amount of disk space used by the ledger's tables, indexes, and indexed history, reported in 15-minute intervals. Indexed storage consists of ledger data that is optimized for high-performance queries. The maximum is 16 TiB. Units: Bytes Dimensions: LedgerName
ReadIOs	The number of read I/O requests, reported in one-minute intervals. This captures all types of read operations, including data transactions, verification requests, journal exports, and journal streams. Units: Count Dimensions: LedgerName
WriteIOs	The number of write I/O requests, reported in one-minute intervals. Units: Count

Metric	Description
	Dimensions: LedgerName
CommandLatency	The amount of time taken for data operations, reported in one-minute intervals. Units: Milliseconds Dimensions: CommandType, LedgerName
IsImpaired	The flag that indicates if a journal stream to Kinesis Data Streams is impaired, reported in one-minute intervals. A value of 1 indicates that the stream is in impaired state, and 0 indicates otherwise. Units: Boolean (0 or 1) Dimensions: LedgerName, StreamId
OccConflictExceptions	The number of requests to QLDB that generate an OccConflictException . For information about optimistic concurrency control (OCC), see Amazon QLDB concurrency model. Units: Count
Session4xxExceptions	The number of requests to QLDB that generate an HTTP 4xx error. Units: Count
Session5xxExceptions	The number of requests to QLDB that generate an HTTP 5xx error. Units: Count
SessionRateExceededExceptions	The number of requests to QLDB that generate a SessionRateExceededException . Units: Count

Dimensions for QLDB metrics

The metrics for QLDB are qualified by the values for the account, ledger name, stream ID, or command type. You can use the CloudWatch console to retrieve QLDB data along any of the dimensions in the following table.

Dimension	Description
LedgerName	This dimension limits the data to a specific ledger. This value can be any ledger name in the current AWS Region and the current AWS account.
StreamId	This dimension limits the data to a specific journal stream. This value can be any stream ID for a ledger in the current AWS Region and the current AWS account.
CommandType	This dimension limits the data to one of the following QLDB data API commands: - AbortTransaction - CommitTransaction - EndSession - ExecuteStatement - FetchPage - StartSession - StartTransaction To learn how QLDB uses these commands to manage data operations, see Session management with the driver.

Creating CloudWatch alarms to monitor Amazon QLDB

You can create an Amazon CloudWatch alarm that sends an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period that you specify. It performs one or more actions based on the value of the metric

relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods.

For more information about creating CloudWatch alarms, see [Using Amazon CloudWatch alarms](#) in the *Amazon CloudWatch User Guide*.

Automating Amazon QLDB with CloudWatch Events

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon CloudWatch Events enables you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to CloudWatch Events in near-real time. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SQS queue

Amazon QLDB reports an event to CloudWatch Events whenever the state of a *ledger* resource in your AWS account changes. Events are currently emitted on a guaranteed, at-least-once basis for QLDB ledger resources only.

The following is an example of an event that QLDB reported, in which a ledger's state changed to DELETING.

```
{
  "version" : "0",
  "id" : "2f6557eb-e361-54ef-0f9f-99dd9f171c62",
  "detail-type" : "QLDB Ledger State Change",
  "source" : "aws.qldb",
  "account" : "123456789012",
  "time" : "2019-07-24T21:59:17Z",
  "region" : "us-east-1",
  "resources" : ["arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"],
  "detail" : {
    "ledgerName" : "exampleLedger",
    "state" : "DELETING"
  }
}
```

Some examples of using CloudWatch Events with QLDB can include but aren't limited to the following:

- Activating a Lambda function whenever a new ledger is initially created in CREATING state and eventually becomes ACTIVE.
- Notifying an Amazon SNS topic when the state of your ledger changes to DELETING and then to DELETED.

For more information, see the [Amazon CloudWatch Events User Guide](#).

Logging Amazon QLDB API calls with AWS CloudTrail

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in QLDB. CloudTrail captures all resource management API calls for QLDB as events. The calls that are captured include calls from the QLDB console and code calls to the QLDB API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket, including events for QLDB. If

you don't configure a trail, you can still view the most recent events on the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to QLDB, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

QLDB information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported activity occurs in QLDB, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail event history](#).

For an ongoing record of events in your AWS account, including events for QLDB, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs.

For more information, see the following topics in the *AWS CloudTrail User Guide*:

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple Regions](#)
- [Receiving CloudTrail log files from multiple accounts](#)

All QLDB resource management and non-transactional data API actions are logged by CloudTrail and are documented in the [Amazon QLDB API reference](#). For example, calls to the `CreateLedger`, `DescribeLedger`, and `DeleteLedger` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or user credentials

- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the [CloudTrail `userIdentity` element](#).

Understanding QLDB log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates these actions:

- `CreateLedger`
- `DescribeLedger`
- `ListTagsForResource`
- `TagResource`
- `UntagResource`
- `ListLedgers`
- `GetDigest`
- `GetBlock`
- `GetRevision`
- `ExportJournalToS3`
- `DescribeJournalS3Export`
- `ListJournalS3ExportsForLedger`
- `ListJournalS3Exports`
- `DeleteLedger`

```
{
  "endTime": 1561497717208,
  "startTime": 1561497687254,
  "calls": [
    {
```

```

"cloudtrailEvent": {
  "userIdentity": {
    "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
  },
  "eventTime": "2019-06-25T21:21:27Z",
  "eventSource": "qldb.amazonaws.com",
  "eventName": "CreateLedger",
  "awsRegion": "us-east-2",
  "errorCode": null,
  "requestParameters": {
    "Name": "CloudtrailTest",
    "PermissionsMode": "ALLOW_ALL"
  },
  "responseElements": {
    "CreationDateTime": 1.561497687403E9,
    "Arn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
    "State": "CREATING",
    "Name": "CloudtrailTest"
  },
  "requestID": "3135aec7-978f-11e9-b313-1dd92a14919e",
  "eventID": "bf703ff9-676f-41dd-be6f-5f666c9f7852",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
},
"rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:27Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"CreateLedger\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"Name\":\"CloudtrailTest\",\"PermissionsMode\":\"ALLOW_ALL\"},\"responseElements\":{\"CreationDateTime\":1.561497687403E9,\"Arn\":\"arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest\",\"State\":\"CREATING\",\"Name\":\"CloudtrailTest\"},\"requestID\":\"3135aec7-978f-11e9-b313-1dd92a14919e\",\"eventID\":\"bf703ff9-676f-41dd-be6f-5f666c9f7852\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}\",
"name": "CreateLedger",

```



```

    "request": [
      "com.amazonaws.services.qldb.model.CreateLedgerRequest",
      {
        "customRequestHeaders": null,
        "customQueryParameters": null,
        "name": "CloudtrailTest",
        "tags": null,
        "permissionsMode": "ALLOW_ALL"
      }
    ],
    "requestId": "3135aec7-978f-11e9-b313-1dd92a14919e"
  },
  {
    "cloudtrailEvent": {
      "userIdentity": {
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
      },
      "eventTime": "2019-06-25T21:21:43Z",
      "eventSource": "qldb.amazonaws.com",
      "eventName": "DescribeLedger",
      "awsRegion": "us-east-2",
      "errorCode": null,
      "requestParameters": {
        "name": "CloudtrailTest"
      },
      "responseElements": null,
      "requestID": "3af51ba0-978f-11e9-8ae6-837dd17a19f8",
      "eventID": "be128e61-3e38-4503-83de-49fdc7fc0afb",
      "readOnly": true,
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":" +
    "{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\", \"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\", \"accountId\":\"123456789012\", \"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\", \"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\", \"creationDate\":\"2019-06-25T21:21:25Z\"}, \"sessionIssuer\":{\"type\":\"Role\", \"principalId\":\"AKIAIOSFODNN7EXAMPLE\", \"arn\":\"arn:aws:iam::123456789012:role/Admin\", \"accountId\":\"123456789012\", \"userName\":\"Admin\"}}}, \"eventTime\":\"2019-06-25T21:21:43Z\", \"eventSource\":\"qldb.amazonaws.com\", \"eventName\":\"DescribeLedger\", \"awsRegion\":\"us-east-2\", \"sourceIPAddress\":\"192.0.2.01\", \"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation"
  }

```

```

\","requestParameters":{"name":"CloudtrailTest"},"responseElements
":null,"requestID":"3af51ba0-978f-11e9-8ae6-837dd17a19f8"},"eventID":
"be128e61-3e38-4503-83de-49fdc7fc0afb"},"readOnly":true,"eventType":"AwsApiCall
"},"recipientAccountId":"123456789012"},"
  "name": "DescribeLedger",
  "request": [
    "com.amazonaws.services.qldb.model.DescribeLedgerRequest",
    {
      "customRequestHeaders": null,
      "customQueryParameters": null,
      "name": "CloudtrailTest"
    }
  ],
  "requestId": "3af51ba0-978f-11e9-8ae6-837dd17a19f8"
},
{
  "cloudtrailEvent": {
    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },
    "eventTime": "2019-06-25T21:21:44Z",
    "eventSource": "qldb.amazonaws.com",
    "eventName": "TagResource",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": {
      "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest",
      "Tags": {
        "TagKey": "TagValue"
      }
    },
    "responseElements": null,
    "requestID": "3b1d6371-978f-11e9-916c-b7d64ec76521",
    "eventID": "6101c94a-7683-4431-812b-9a91afb8c849",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  },
  "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z

```

```

\"},"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"TagResource\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"resourceArn\":\"arn
%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest\",\"Tags\":{\"TagKey
\":\"TagValue\"}},\"responseElements\":null,\"requestID\":\"3b1d6371-978f-11e9-916c-
b7d64ec76521\",\"eventID\":\"6101c94a-7683-4431-812b-9a91afb8c849\",\"readOnly\":false,
\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}},
  "name": "TagResource",
  "request": [
    "com.amazonaws.services.qldb.model.TagResourceRequest",
    {
      "customRequestHeaders": null,
      "customQueryParameters": null,
      "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
      "tags": {
        "TagKey": "TagValue"
      }
    }
  ],
  "requestId": "3b1d6371-978f-11e9-916c-b7d64ec76521"
},
{
  "cloudtrailEvent": {
    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },
    "eventTime": "2019-06-25T21:21:44Z",
    "eventSource": "qldb.amazonaws.com",
    "eventName": "ListTagsForResource",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": {
      "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest"
    },
    "responseElements": null,
    "requestID": "3b56c321-978f-11e9-8527-2517d5bfa8fd",
    "eventID": "375e57d7-cf94-495a-9a48-ac2192181c02",
    "readOnly": true,
    "eventType": "AwsApiCall",

```

```

    "recipientAccountId": "123456789012"
  },
  "rawCloudtrailEvent": "{ \"eventVersion\": \"1.05\", \"userIdentity\": { \"type\": \"AssumedRole\", \"principalId\": \"AKIAIOSFODNN7EXAMPLE:test-user\", \"arn\": \"arn:aws:sts::123456789012:assumed-role/Admin/test-user\", \"accountId\": \"123456789012\", \"accessKeyId\": \"AKIAI44QH8DHBEXAMPLE\", \"sessionContext\": { \"attributes\": { \"mfaAuthenticated\": \"false\", \"creationDate\": \"2019-06-25T21:21:25Z\" }, \"sessionIssuer\": { \"type\": \"Role\", \"principalId\": \"AKIAIOSFODNN7EXAMPLE\", \"arn\": \"arn:aws:iam::123456789012:role/Admin\", \"accountId\": \"123456789012\", \"userName\": \"Admin\" } } }, \"eventTime\": \"2019-06-25T21:21:44Z\", \"eventSource\": \"qldb.amazonaws.com\", \"eventName\": \"ListTagsForResource\", \"awsRegion\": \"us-east-2\", \"sourceIPAddress\": \"192.0.2.01\", \"userAgent\": \"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\", \"requestParameters\": { \"resourceArn\": \"arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest\" }, \"responseElements\": null, \"requestID\": \"3b56c321-978f-11e9-8527-2517d5bfa8fd\", \"eventID\": \"375e57d7-cf94-495a-9a48-ac2192181c02\", \"readOnly\": true, \"eventType\": \"AwsApiCall\", \"recipientAccountId\": \"123456789012\" }",
  "name": "ListTagsForResource",
  "request": [
    "com.amazonaws.services.qldb.model.ListTagsForResourceRequest",
    {
      "customRequestHeaders": null,
      "customQueryParameters": null,
      "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest"
    }
  ],
  "requestId": "3b56c321-978f-11e9-8527-2517d5bfa8fd"
},
{
  "cloudtrailEvent": {
    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },
    "eventTime": "2019-06-25T21:21:44Z",
    "eventSource": "qldb.amazonaws.com",
    "eventName": "UntagResource",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": {
      "tagKeys": "TagKey",
      "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest"
    }
  }
}

```

```

    },
    "responseElements": null,
    "requestID": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9",
    "eventID": "bcdcdca3-699f-4363-b092-88242780406f",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  },
  "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"UntagResource\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"tagKeys\":\"TagKey\",\"resourceArn\":\"arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest\"},\"responseElements\":null,\"requestID\":\"3b87e59b-978f-11e9-8b9a-bb6dc3a800a9\",\"eventID\":\"bcdcdca3-699f-4363-b092-88242780406f\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}\",
    "name": "UntagResource",
    "request": [
      "com.amazonaws.services.qldb.model.UntagResourceRequest",
      {
        "customRequestHeaders": null,
        "customQueryParameters": null,
        "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
        "tagKeys": [
          "TagKey"
        ]
      }
    ],
    "requestId": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9"
  },
  {
    "cloudtrailEvent": {
      "userIdentity": {
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
      },
      "eventTime": "2019-06-25T21:21:44Z",

```

```

    "eventSource": "qldb.amazonaws.com",
    "eventName": "ListLedgers",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": null,
    "responseElements": null,
    "requestID": "3bafb877-978f-11e9-a6de-dbe6464b9dec",
    "eventID": "6ebe7d49-af59-4f29-aaa2-beffe536e20c",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  },
  "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"ListLedgers\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":null,\"responseElements\":null,\"requestID\":\"3bafb877-978f-11e9-a6de-dbe6464b9dec\",\"eventID\":\"6ebe7d49-af59-4f29-aaa2-beffe536e20c\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}\",
  "name": "ListLedgers",
  "request": [
    "com.amazonaws.services.qldb.model.ListLedgersRequest",
    {
      "customRequestHeaders": null,
      "customQueryParameters": null,
      "maxResults": null,
      "nextToken": null
    }
  ],
  "requestId": "3bafb877-978f-11e9-a6de-dbe6464b9dec"
},
{
  "cloudtrailEvent": {
    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    }
  },

```

```

    "eventTime": "2019-06-25T21:21:49Z",
    "eventSource": "qldb.amazonaws.com",
    "eventName": "GetDigest",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": {
      "name": "CloudtrailTest"
    },
    "responseElements": null,
    "requestID": "3cddd8a1-978f-11e9-a6de-dbe6464b9dec",
    "eventID": "a5cb60db-e6c5-4f5e-a5fc-0712249622b3",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  },
  "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:49Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"GetDigest\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"name\":\"CloudtrailTest\"},\"responseElements\":null,\"requestID\":\"3cddd8a1-978f-11e9-a6de-dbe6464b9dec\",\"eventID\":\"a5cb60db-e6c5-4f5e-a5fc-0712249622b3\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}\",
    "name": "GetDigest",
    "request": [
      "com.amazonaws.services.qldb.model.GetDigestRequest",
      {
        "customRequestHeaders": null,
        "customQueryParameters": null,
        "name": "CloudtrailTest",
        "digestTipAddress": null
      }
    ],
    "requestId": "3cddd8a1-978f-11e9-a6de-dbe6464b9dec"
  },
  {
    "cloudtrailEvent": {

```

```

    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },
    "eventTime": "2019-06-25T21:21:50Z",
    "eventSource": "qldb.amazonaws.com",
    "eventName": "GetBlock",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": {
      "BlockAddress": {
        "IonText": "{strandId:\\\"2P2nsG3K2RwHQccUbnAMAJ\\\",sequenceNo:0}"
      },
      "name": "CloudtrailTest",
      "DigestTipAddress": {
        "IonText": "{strandId:\\\"2P2nsG3K2RwHQccUbnAMAJ\\\",sequenceNo:0}"
      }
    },
    "responseElements": null,
    "requestID": "3eaea09f-978f-11e9-bdc2-c1e55368155e",
    "eventID": "1f7da83f-d829-4e35-953d-30b925ceee66",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  },
  "rawCloudtrailEvent": "{\\\"eventVersion\\\":\\\"1.05\\\",\\\"userIdentity\\\":{\\\"type\\\":\\\"AssumedRole\\\",\\\"principalId\\\":\\\"AKIAIOSFODNN7EXAMPLE:test-user\\\",\\\"arn\\\":\\\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\\\",\\\"accountId\\\":\\\"123456789012\\\",\\\"accessKeyId\\\":\\\"AKIAI44QH8DHBEXAMPLE\\\",\\\"sessionContext\\\":{\\\"attributes\\\":{\\\"mfaAuthenticated\\\":\\\"false\\\",\\\"creationDate\\\":\\\"2019-06-25T21:21:25Z\\\"},\\\"sessionIssuer\\\":{\\\"type\\\":\\\"Role\\\",\\\"principalId\\\":\\\"AKIAIOSFODNN7EXAMPLE\\\",\\\"arn\\\":\\\"arn:aws:iam::123456789012:role/Admin\\\",\\\"accountId\\\":\\\"123456789012\\\",\\\"userName\\\":\\\"Admin\\\"}}},\\\"eventTime\\\":\\\"2019-06-25T21:21:50Z\\\",\\\"eventSource\\\":\\\"qldb.amazonaws.com\\\",\\\"eventName\\\":\\\"GetBlock\\\",\\\"awsRegion\\\":\\\"us-east-2\\\",\\\"sourceIPAddress\\\":\\\"192.0.2.01\\\",\\\"userAgent\\\":\\\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\\\",\\\"requestParameters\\\":{\\\"BlockAddress\\\":{\\\"IonText\\\":\\\"{strandId:\\\"\\\\\\\\\\\"2P2nsG3K2RwHQccUbnAMAJ\\\\\\\\\\\",sequenceNo:0}\\\"}},\\\"name\\\":\\\"CloudtrailTest\\\",\\\"DigestTipAddress\\\":{\\\"IonText\\\":\\\"{strandId:\\\"\\\\\\\\\\\"2P2nsG3K2RwHQccUbnAMAJ\\\\\\\\\\\",sequenceNo:0}\\\"}},\\\"responseElements\\\":null,\\\"requestID\\\":\\\"3eaea09f-978f-11e9-bdc2-c1e55368155e\\\",\\\"eventID\\\":\\\"1f7da83f-d829-4e35-953d-30b925ceee66\\\",\\\"readOnly\\\":true,\\\"eventType\\\":\\\"AwsApiCall\\\",\\\"recipientAccountId\\\":\\\"123456789012\\\"}}",
    "name": "GetBlock",
    "request": [

```



```

    "com.amazonaws.services.qldb.model.GetBlockRequest",
    {
      "customRequestHeaders": null,
      "customQueryParameters": null,
      "name": "CloudtrailTest",
      "blockAddress": {
        "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAJ\",sequenceNo:0}"
      },
      "digestTipAddress": {
        "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAJ\",sequenceNo:0}"
      }
    },
    ],
    "requestId": "3eaea09f-978f-11e9-bdc2-c1e55368155e"
  },
  {
    "cloudtrailEvent": {
      "userIdentity": {
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
      },
      "eventTime": "2019-06-25T21:21:55Z",
      "eventSource": "qldb.amazonaws.com",
      "eventName": "GetRevision",
      "awsRegion": "us-east-2",
      "errorCode": null,
      "requestParameters": {
        "BlockAddress": {
          "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAJ\",sequenceNo:1}"
        },
        "name": "CloudtrailTest",
        "DocumentId": "8UyXvDw6ApoFfVOA2HPfUE",
        "DigestTipAddress": {
          "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAJ\",sequenceNo:1}"
        }
      },
      "responseElements": null,
      "requestID": "41e19139-978f-11e9-aaed-dfe1dafe37ab",
      "eventID": "43bf2661-5046-41ec-a1d3-87706954aa10",
      "readOnly": true,
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\", \"userIdentity\": {\"type\": \"AssumedRole\", \"principalId\": \"AKIAIOSFODNN7EXAMPLE:test-user\", \"arn"

```

```

\":"arn:aws:sts::123456789012:assumed-role/Admin/test-user\","\\"accountId\":"
\\"123456789012\","\\"accessKeyId\":"\\"AKIAI44QH8DHBEXAMPLE\","\\"sessionContext\":"
{\\"attributes\":"{\\"mfaAuthenticated\":"\\"false\","\\"creationDate\":"\\"2019-06-25T21:21:25Z
\\"},\\"sessionIssuer\":"{\\"type\":"\\"Role\","\\"principalId\":"\\"AKIAIOSFODNN7EXAMPLE\","
\\"arn\":"\\"arn:aws:iam::123456789012:role/Admin\","\\"accountId\":"\\"123456789012\","
\\"userName\":"\\"Admin\"]]}\","eventTime\":"\\"2019-06-25T21:21:55Z\","\\"eventSource\":"
\\"qldb.amazonaws.com\","\\"eventName\":"\\"GetRevision\","\\"awsRegion\":"\\"us-east-2\","
\\"sourceIPAddress\":"\\"192.0.2.01\","\\"userAgent\":"\\"aws-internal/3 aws-sdk-java/1.11.575
Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
kotlin/1.3.21 vendor/Oracle_Corporation\","\\"requestParameters\":"{\\"BlockAddress
\":"{\\"IonText\":"\\"{strandId:\\\\"2P2nsG3K2RwHQccUbnAMAJ\\\\"",sequenceNo:1}\\"},\\"name
\":"\\"CloudtrailTest\","\\"DocumentId\":"\\"8UyXvDw6ApoFfVOA2HPfUE\","\\"DigestTipAddress
\":"{\\"IonText\":"\\"{strandId:\\\\"2P2nsG3K2RwHQccUbnAMAJ\\\\"",sequenceNo:1}\\"}},
\\"responseElements\":"null,\\"requestID\":"\\"41e19139-978f-11e9-aaed-dfe1dafe37ab\","
\\"eventID\":"\\"43bf2661-5046-41ec-a1d3-87706954aa10\","\\"readOnly\":"true,\\"eventType\":"
\\"AwsApiCall\","\\"recipientAccountId\":"\\"123456789012\\"}]",
  "name": "GetRevision",
  "request": [
    "com.amazonaws.services.qldb.model.GetRevisionRequest",
    {
      "customRequestHeaders": null,
      "customQueryParameters": null,
      "name": "CloudtrailTest",
      "blockAddress": {
        "ionText": "{strandId:\\\\"2P2nsG3K2RwHQccUbnAMAJ\\\\"",sequenceNo:1}"
      },
      "documentId": "8UyXvDw6ApoFfVOA2HPfUE",
      "digestTipAddress": {
        "ionText": "{strandId:\\\\"2P2nsG3K2RwHQccUbnAMAJ\\\\"",sequenceNo:1}"
      }
    }
  ],
  "requestId": "41e19139-978f-11e9-aaed-dfe1dafe37ab"
},
{
  "cloudtrailEvent": {
    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },
    "eventTime": "2019-06-25T21:21:56Z",
    "eventSource": "qldb.amazonaws.com",
    "eventName": "ExportJournalToS3",
    "awsRegion": "us-east-2",
    "errorCode": null,

```

```

    "requestParameters": {
      "InclusiveStartTime": 1.561497687254E9,
      "name": "CloudtrailTest",
      "S3ExportConfiguration": {
        "Bucket": "amzn-s3-demo-qldb-cloudtrail",
        "Prefix": "CloudtrailTestsJournalExport",
        "EncryptionConfiguration": {
          "ObjectEncryptionType": "SSE_S3"
        }
      },
      "ExclusiveEndTime": 1.561497715795E9
    },
    "responseElements": {
      "ExportId": "BabQhsmJRYDCGMnA2xYBDG"
    },
    "requestID": "423815f8-978f-11e9-afcf-55f7d0f3583d",
    "eventID": "1b5abdc4-52fa-435f-857e-8995ef7a19b7",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  },
  "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"ExportJournalToS3\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"InclusiveStartTime\":1.561497687254E9,\"name\":\"CloudtrailTest\",\"S3ExportConfiguration\":{\"Bucket\":\"amzn-s3-demo-qldb-cloudtrail\",\"Prefix\":\"CloudtrailTestsJournalExport\",\"EncryptionConfiguration\":{\"ObjectEncryptionType\":\"SSE_S3\"}},\"ExclusiveEndTime\":1.561497715795E9},\"responseElements\":{\"ExportId\":\"BabQhsmJRYDCGMnA2xYBDG\"},\"requestID\":\"423815f8-978f-11e9-afcf-55f7d0f3583d\",\"eventID\":\"1b5abdc4-52fa-435f-857e-8995ef7a19b7\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}\",
    "name": "ExportJournalToS3",
    "request": [
      "com.amazonaws.services.qldb.model.ExportJournalToS3Request",
      {

```

```

        "customRequestHeaders": null,
        "customQueryParameters": null,
        "name": "CloudtrailTest",
        "inclusiveStartTime": 1561497687254,
        "exclusiveEndTime": 1561497715795,
        "s3ExportConfiguration": {
            "bucket": "amzn-s3-demo-qldb-cloudtrail",
            "prefix": "CloudtrailTestsJournalExport",
            "encryptionConfiguration": {
                "objectEncryptionType": "SSE_S3",
                "kmsKeyArn": null
            }
        }
    },
    "requestId": "423815f8-978f-11e9-afcf-55f7d0f3583d"
},
{
    "cloudtrailEvent": {
        "userIdentity": {
            "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "DescribeJournalS3Export",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
            "name": "CloudtrailTest",
            "exportId": "BabQhsmJRYDCGMnA2xYBDG"
        },
        "responseElements": null,
        "requestId": "427ebbbc-978f-11e9-8888-e9894c9c4bb9",
        "eventId": "ca8ffc88-16ff-45f5-9042-d94fadb389c3",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    },
    "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z\"}},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",

```

```

\"arn\": \"arn:aws:iam::123456789012:role/Admin\", \"accountId\": \"123456789012\",
\"userName\": \"Admin\"}}}, \"eventTime\": \"2019-06-25T21:21:56Z\", \"eventSource\":
\"qldb.amazonaws.com\", \"eventName\": \"DescribeJournalS3Export\", \"awsRegion\":
\"us-east-2\", \"sourceIPAddress\": \"192.0.2.01\", \"userAgent\": \"aws-internal/3
aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\", \"requestParameters\": {\"name
\": \"CloudtrailTest\", \"exportId\": \"BabQhsmJRYDCGMnA2xYBDG\"}, \"responseElements
\": null, \"requestID\": \"427ebbbc-978f-11e9-8888-e9894c9c4bb9\", \"eventID\":
\"ca8ffc88-16ff-45f5-9042-d94fadb389c3\", \"readOnly\": true, \"eventType\": \"AwsApiCall
\", \"recipientAccountId\": \"123456789012\"}},
  \"name\": \"DescribeJournalS3Export\",
  \"request\": [
    \"com.amazonaws.services.qldb.model.DescribeJournalS3ExportRequest\",
    {
      \"customRequestHeaders\": null,
      \"customQueryParameters\": null,
      \"name\": \"CloudtrailTest\",
      \"exportId\": \"BabQhsmJRYDCGMnA2xYBDG\"
    }
  ],
  \"requestId\": \"427ebbbc-978f-11e9-8888-e9894c9c4bb9\"
},
{
  \"cloudtrailEvent\": {
    \"userIdentity\": {
      \"arn\": \"arn:aws:sts::123456789012:assumed-role/Admin/test-user\"
    },
    \"eventTime\": \"2019-06-25T21:21:56Z\",
    \"eventSource\": \"qldb.amazonaws.com\",
    \"eventName\": \"ListJournalS3ExportsForLedger\",
    \"awsRegion\": \"us-east-2\",
    \"errorCode\": null,
    \"requestParameters\": {
      \"name\": \"CloudtrailTest\"
    },
    \"responseElements\": null,
    \"requestID\": \"429ca40c-978f-11e9-8c4b-d13a8018a286\",
    \"eventID\": \"34f0e76b-58a5-45be-881c-786d22e34e96\",
    \"readOnly\": true,
    \"eventType\": \"AwsApiCall\",
    \"recipientAccountId\": \"123456789012\"
  },
  \"rawCloudtrailEvent\": \"{\\\"eventVersion\\\":\\\"1.05\\\",\\\"userIdentity\\\":{\\\"type
\\\":\\\"AssumedRole\\\",\\\"principalId\\\":\\\"AKIAIOSFODNN7EXAMPLE:test-user\\\",\\\"arn

```

```

\":"arn:aws:sts::123456789012:assumed-role/Admin/test-user\","accountId\":"
\":"123456789012\","accessKeyId\":"AKIAI44QH8DHBEXAMPLE\","sessionContext\":"
{\\"attributes\\":{\\"mfaAuthenticated\\":\\"false\\","creationDate\\":\\"2019-06-25T21:21:25Z
\\"},\\"sessionIssuer\\":{\\"type\\":\\"Role\\","principalId\\":\\"AKIAIOSFODNN7EXAMPLE\\",
\\"arn\\":\\"arn:aws:iam::123456789012:role/Admin\\","accountId\\":\\"123456789012\\",
\\"userName\\":\\"Admin\\"}}},\\"eventTime\\":\\"2019-06-25T21:21:56Z\\","eventSource
\\":\\"qldb.amazonaws.com\\","eventName\\":\\"ListJournalS3ExportsForLedger\\",
\\"awsRegion\\":\\"us-east-2\\","sourceIPAddress\\":\\"192.0.2.01\\","userAgent\\":
\\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-
Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation
\\","requestParameters\\":{\\"name\\":\\"CloudtrailTest\\"},\\"responseElements
\\":null,\\"requestID\\":\\"429ca40c-978f-11e9-8c4b-d13a8018a286\\","eventID\\":
\\"34f0e76b-58a5-45be-881c-786d22e34e96\\","readOnly\\":true,\\"eventType\\":\\"AwsApiCall
\\","recipientAccountId\\":\\"123456789012\\"}",
  "name": "ListJournalS3ExportsForLedger",
  "request": [
    "com.amazonaws.services.qldb.model.ListJournalS3ExportsForLedgerRequest",
    {
      "customRequestHeaders": null,
      "customQueryParameters": null,
      "name": "CloudtrailTest",
      "maxResults": null,
      "nextToken": null
    }
  ],
  "requestId": "429ca40c-978f-11e9-8c4b-d13a8018a286"
},
{
  "cloudtrailEvent": {
    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },
    "eventTime": "2019-06-25T21:21:56Z",
    "eventSource": "qldb.amazonaws.com",
    "eventName": "ListJournalS3Exports",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": null,
    "responseElements": null,
    "requestID": "42cc1814-978f-11e9-befb-f5dbaa142118",
    "eventID": "4c24d7d6-810c-4cf4-884e-00482278b6ce",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  }
}

```

```

    },
    "rawCloudtrailEvent": "{ \"eventVersion\": \"1.05\", \"userIdentity\": { \"type\": \"AssumedRole\", \"principalId\": \"AKIAIOSFODNN7EXAMPLE:test-user\", \"arn\": \"arn:aws:sts::123456789012:assumed-role/Admin/test-user\", \"accountId\": \"123456789012\", \"accessKeyId\": \"AKIAI44QH8DHBEXAMPLE\", \"sessionContext\": { \"attributes\": { \"mfaAuthenticated\": \"false\", \"creationDate\": \"2019-06-25T21:21:25Z\" }, \"sessionIssuer\": { \"type\": \"Role\", \"principalId\": \"AKIAIOSFODNN7EXAMPLE\", \"arn\": \"arn:aws:iam::123456789012:role/Admin\", \"accountId\": \"123456789012\", \"userName\": \"Admin\" } } }, \"eventTime\": \"2019-06-25T21:21:56Z\", \"eventSource\": \"qldb.amazonaws.com\", \"eventName\": \"ListJournalS3Exports\", \"awsRegion\": \"us-east-2\", \"sourceIPAddress\": \"192.0.2.01\", \"userAgent\": \"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\", \"requestParameters\": null, \"responseElements\": null, \"requestID\": \"42cc1814-978f-11e9-befb-f5dbaa142118\", \"eventID\": \"4c24d7d6-810c-4cf4-884e-00482278b6ce\", \"readOnly\": true, \"eventType\": \"AwsApiCall\", \"recipientAccountId\": \"123456789012\" },
    \"name\": \"ListJournalS3Exports\",
    \"request\": [
        \"com.amazonaws.services.qldb.model.ListJournalS3ExportsRequest\",
        {
            \"customRequestHeaders\": null,
            \"customQueryParameters\": null,
            \"maxResults\": null,
            \"nextToken\": null
        }
    ],
    \"requestId\": \"42cc1814-978f-11e9-befb-f5dbaa142118\"
},
{
    \"cloudtrailEvent\": {
        \"userIdentity\": {
            \"arn\": \"arn:aws:sts::123456789012:assumed-role/Admin/test-user\"
        },
        \"eventTime\": \"2019-06-25T21:21:57Z\",
        \"eventSource\": \"qldb.amazonaws.com\",
        \"eventName\": \"DeleteLedger\",
        \"awsRegion\": \"us-east-2\",
        \"errorCode\": null,
        \"requestParameters\": {
            \"name\": \"CloudtrailTest\"
        },
        \"responseElements\": null,
        \"requestID\": \"42f439b9-978f-11e9-8b2c-69ef598d66e9\",
        \"eventID\": \"429f5163-cba5-4d86-bd7e-f606e057c6cf\",

```

```

        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    },
    "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:57Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"DeleteLedger\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"name\":\"CloudtrailTest\"},\"responseElements\":null,\"requestID\":\"42f439b9-978f-11e9-8b2c-69ef598d66e9\",\"eventID\":\"429f5163-cba5-4d86-bd7e-f606e057c6cf\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}\",
    "name": "DeleteLedger",
    "request": [
        "com.amazonaws.services.qldb.model.DeleteLedgerRequest",
        {
            "customRequestHeaders": null,
            "customQueryParameters": null,
            "name": "CloudtrailTest"
        }
    ],
    "requestId": "42f439b9-978f-11e9-8b2c-69ef598d66e9"
}
]
}

```

Compliance validation for Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Third-party auditors assess the security and compliance of Amazon QLDB as part of multiple AWS compliance programs, including but not limited to the following:

- System and Organization Controls (SOC)
- Payment Card Industry (PCI)
- International Organization for Standardization (ISO)
- Information System Security Management and Assessment Program (ISMAP)
- Health Insurance Portability and Accountability Act (HIPAA)

 Note

This is not a comprehensive list of Amazon QLDB certifications.

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security Compliance & Governance](#) – These solution implementation guides discuss architectural considerations and provide steps for deploying security and compliance features.
- [HIPAA Eligible Services Reference](#) – Lists HIPAA eligible services. Not all AWS services are HIPAA eligible.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [AWS Customer Compliance Guides](#) – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see [Security Hub controls reference](#).
- [Amazon GuardDuty](#) – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Storage durability

QLDB *journal storage* features synchronous replication to multiple Availability Zones on transaction commits. This ensures that even a full Availability Zone failure of journal storage would not

compromise data integrity or the ability to maintain an active service. Additionally, the QLDB journal features asynchronous archives to fault-tolerant storage. This feature supports disaster recovery in the highly unlikely event of simultaneous storage failure for multiple Availability Zones.

QLDB *indexed storage* is backed by replication to multiple Availability Zones. This ensures that even a full Availability Zone failure of indexed storage would not compromise data integrity or the ability to maintain an active service.

Data durability features

In addition to the AWS global infrastructure, QLDB offers the following features to help support your data resiliency and backup needs.

QLDB service features

On-demand journal export

QLDB provides an on-demand journal export feature. Access the contents of your journal by exporting journal blocks from your ledger into an Amazon S3 bucket. You can use this data for various purposes such as data retention, analytics, and auditing. For more information, see [Exporting journal data from Amazon QLDB](#).

Backup and restore

Automated restore for exports is not supported at this time. Export provides a basic ability to create additional data redundancy at your defined frequency. However, a restore scenario is application dependent, where the exported records are presumably written back to a new ledger by leveraging the same or similar ingestion method.

QLDB doesn't provide a dedicated backup and related restore feature at this time.

Journal streams

QLDB also provides a continuous journal stream capability. You can integrate QLDB journal streams with the Amazon Kinesis streaming platform to process real-time journal data. For more information, see [Streaming journal data from Amazon QLDB](#).

QLDB design feature

QLDB is designed to be resilient against logical corruption. The QLDB journal is immutable, ensuring that all committed transactions are persisted to the journal. In addition, every committed

document change is recorded, as this enables *point-in-time* visibility for any unintended changes to ledger data.

QLDB doesn't provide an automated recovery feature for logical corruption scenarios at this time.

Infrastructure security in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

As a managed service, Amazon QLDB is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access QLDB through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using programmatic credentials that are associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

You can also use a *virtual private cloud* (VPC) endpoint for QLDB. Interface VPC endpoints enable your Amazon VPC resources to use their private IP addresses to access QLDB with no exposure to the public internet. For more information, see [Access Amazon QLDB using an interface endpoint \(AWS PrivateLink\)](#).

Access Amazon QLDB using an interface endpoint (AWS PrivateLink)

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

You can use AWS PrivateLink to create a private connection between your VPC and Amazon QLDB. You can access QLDB as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to access QLDB.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for QLDB.

For more information, see [Access AWS services through AWS PrivateLink](#) in the *AWS PrivateLink Guide*.

Topics

- [Considerations for QLDB](#)
- [Create an interface endpoint for QLDB](#)
- [Create an endpoint policy for your interface endpoint](#)
- [Availability of interface endpoints for QLDB](#)

Considerations for QLDB

Before you set up an interface endpoint for QLDB, review [Considerations](#) in the *AWS PrivateLink Guide*.

Note

QLDB only supports making calls to the *QLDB Session* transactional data API through the interface endpoint. This API includes only the [SendCommand](#) operation. In the STANDARD permissions mode of a ledger, you can control permissions to specific PartiQL actions in this API.

Create an interface endpoint for QLDB

You can create an interface endpoint for QLDB using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see [Create an interface endpoint](#) in the *AWS PrivateLink Guide*.

Create an interface endpoint for QLDB using the following service name:

```
com.amazonaws.region.qldb.session
```

If you enable private DNS for the interface endpoint, you can make API requests to QLDB using its default Regional DNS name. For example, `session.qldb.us-east-1.amazonaws.com`.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to QLDB through the interface endpoint. To control the access allowed to QLDB from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, users, and roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see [Control access to services using endpoint policies](#) in the *AWS PrivateLink Guide*.

You can also use the `Condition` field in a policy that is attached to a user, group, or role to allow access only from a specified interface endpoint. When used together, endpoint policies and IAM policies can restrict access to specific QLDB actions on specified ledgers to a specified interface endpoint.

Endpoint policy example: Restrict access to a specific QLDB ledger

The following is an example of a custom endpoint policy for QLDB. When you attach this policy to your interface endpoint, it grants access to the `SendCommand` action and the `PartiQL` read-only actions for all principals on the specified ledger resource. In this example, the ledger must be in the `STANDARD` permissions mode.

To use this policy, replace `us-east-1`, `123456789012`, and `myExampleLedger` in the example with your own information.

```
{  
  "Statement": [  
    {  
      "Action": "qldb:SendCommand",  
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger",  
      "Effect": "Allow",  
      "Principal": "*"   
    },  
    {  
      "Action": "qldb:PartiQLRead",  
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger",  
      "Effect": "Allow",  
      "Principal": "*"   
    }  
  ]  
}
```

```

{
  "Sid": "QLDBSendCommandPermission",
  "Principal": "*",
  "Effect": "Allow",
  "Action": "qldb:SendCommand",
  "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
},
{
  "Sid": "QLDBPartiQLReadOnlyPermissions",
  "Principal": "*",
  "Effect": "Allow",
  "Action": [
    "qldb:PartiQLSelect",
    "qldb:PartiQLHistoryFunction"
  ],
  "Resource": [
    "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
    "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
  ]
}
]
}

```

IAM policy example: Restrict access to a QLDB ledger from a specific interface endpoint only

The following is an example of an IAM identity-based policy for QLDB. When you attach this policy to a user, role, or group, it allows SendCommand access to a ledger resource only from the specified interface endpoint.

To use this policy, replace *us-east-1*, *123456789012*, *myExampleLedger*, and *vpce-1a2b3c4d* in the example with your own information.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessFromSpecificInterfaceEndpoint",
      "Effect": "Deny",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger",
      "Condition": {
        "StringNotEquals": {

```

```
    "aws:sourceVpce": "vpce-1a2b3c4d"  
  }  
}  
}  
]  
}
```

Availability of interface endpoints for QLDB

Amazon QLDB supports interface endpoints with policies in all of the AWS Regions where QLDB is available. For a complete list of available Regions, see [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*.

Troubleshooting Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following sections provide an aggregate list of common errors that you might encounter when using Amazon QLDB, and guidance for how to troubleshoot them.

For troubleshooting guidance specific to IAM access, see [Troubleshooting Amazon QLDB identity and access](#).

For best practices for tuning your PartiQL statements, see [Optimizing query performance](#).

Topics

- [Running transactions using the QLDB driver](#)
- [Exporting journal data](#)
- [Streaming journal data](#)
- [Verifying journal data](#)

Running transactions using the QLDB driver

This section lists common exceptions that the Amazon QLDB driver can return when you use it to run PartiQL transactions on a ledger. For more information about this feature, see [Getting started with the driver](#). For best practices for configuring and using the driver, see [Driver recommendations](#).

Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

CapacityExceededException

Message: Capacity exceeded

Amazon QLDB rejected the request because it exceeded the processing capacity of the ledger. QLDB enforces an internal scaling limit per ledger to maintain the health and performance of the service. This limit varies depending on the workload size of each individual request. For example, a request can have an increased workload if it performs inefficient data transactions, such as table scans that result from a non-index qualified query.

We recommend that you wait before retrying the request. If your application consistently encounters this exception, optimize your statements and decrease the rate and volume of the requests that you send to the ledger. Examples of statement optimization include running fewer statements per transaction and tuning your table indexes. To learn how to optimize statements and avoid table scans, see [Optimizing query performance](#).

We also recommend using the latest version of the QLDB driver. The driver has a default retry policy that uses [Exponential Backoff and Jitter](#) to automatically retry on exceptions such as this. The concept of exponential backoff is to use progressively longer wait times between retries for consecutive error responses.

InvalidSessionException

Message: Transaction *transactionId* has expired

A transaction exceeded its maximum lifetime. A transaction can run for up to 30 seconds before being committed. After this timeout limit, any work done on the transaction is rejected, and QLDB discards the session. This limit protects the client from leaking sessions by starting transactions and not committing or canceling them.

If this is a common exception in your application, it's likely that transactions are simply taking too long to run. If transaction runtime is taking longer than 30 seconds, optimize your statements to speed up the transactions. Examples of statement optimization include running fewer statements per transaction and tuning your table indexes. For more information, see [Optimizing query performance](#).

InvalidSessionException

Message: Session *sessionId* has expired

QLDB discarded the session because it exceeded its maximum total lifetime. QLDB discards sessions after 13–17 minutes, regardless of an active transaction. Sessions can be lost or impaired for a number of reasons, such as hardware failure, network failure, or application restarts. So, QLDB enforces a maximum lifetime on sessions to ensure that client software is resilient to session failure.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the latest version of the QLDB driver, which manages the session pool and its health on the application's behalf.

InvalidSessionException

Message: No such session

The client tried to transact with QLDB using a session that doesn't exist. Assuming that the client is using a session that previously existed, the session might no longer exist because of one of the following:

- If a session is involved in an internal server failure (that is, an error with HTTP response code 500), QLDB might choose to discard the session completely, rather than allow the customer to transact with a session of uncertain state. Then, any retry attempts on that session fail with this error.
- Expired sessions are eventually forgotten by QLDB. Then, any attempts to continue using the session result in this error, rather than the initial `InvalidSessionException`.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the latest version of the QLDB driver, which manages the session pool and its health on the application's behalf.

RateExceededException

Message: The rate was exceeded

QLDB throttled a client based on the caller's identity. QLDB enforces throttling on a per-Region, per-account basis using a [token bucket](#) throttling algorithm. QLDB does this to help the performance of the service, and to ensure fair usage for all QLDB customers. For example, trying to acquire a large number of concurrent sessions using the `StartSessionRequest` operation might lead to throttling.

To maintain your application health and mitigate further throttling, you can retry on this exception using [Exponential Backoff and Jitter](#). The concept of exponential backoff is to use progressively longer wait times between retries for consecutive error responses. We recommend using the latest version of the QLDB driver. The driver has a default retry policy that uses exponential backoff and jitter to automatically retry on exceptions such as this.

The latest version of the QLDB driver can also help if your application is consistently getting throttled by QLDB for `StartSessionRequest` calls. The driver maintains a pool of sessions that are reused across transactions, which can help to reduce the number of

`StartSessionRequest` calls that your application makes. To request an increase in API throttling limits, contact the [AWS Support Center](#).

LimitExceededException

Message: Exceeded the session limit

A ledger exceeded its quota (also known as a *limit*) on the number of active sessions. This quota is defined in [Quotas and limits in Amazon QLDB](#). A ledger's active session count is eventually consistent, and ledgers consistently running near the quota might periodically see this exception.

To maintain your application's health, we recommend retrying on this exception. To avoid this exception, ensure that you have not configured more than 1,500 concurrent sessions to be used for a single ledger across all clients. For example, you can use the [maxConcurrentTransactions](#) method of the [Amazon QLDB driver for Java](#) to configure the maximum number of available sessions in a driver instance.

QldbClientException

Message: A streamed result is only valid when the parent transaction is open

The transaction is closed, and it can't be used to retrieve the results from QLDB. A transaction closes when it's either committed or canceled.

This exception occurs when the client is working directly with the `Transaction` object, and it's trying to retrieve results from QLDB after committing or canceling a transaction. To mitigate this issue, the client must read the data before closing the transaction.

Exporting journal data

This section lists common exceptions that QLDB can return when you export journal data from a ledger into an Amazon S3 bucket. For more information about this feature, see [Exporting journal data from Amazon QLDB](#).

Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

AccessDeniedException

Message: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN*

You don't have permissions to pass an IAM role to the QLDB service. QLDB requires a role for all journal export requests, and you must have permissions to pass this role to QLDB. The role provides QLDB with write permissions in your specified Amazon S3 bucket.

Verify that you define an IAM policy that grants permission to perform the `PassRole` API operation on your specified IAM role resource for the QLDB service (`qldb.amazonaws.com`). For a policy example, see [Identity-based policy examples for Amazon QLDB](#).

IllegalArgumentException

Message: QLDB encountered an error validating S3 configuration: *errorCode errorMessage*

A possible cause for this error is that the provided Amazon S3 bucket doesn't exist in Amazon S3. Or, QLDB doesn't have enough permissions to write objects into your specified Amazon S3 bucket.

Verify that the S3 bucket name that you provide in your export job request is correct. For more information about bucket naming, see [Bucket restrictions and limitations](#) in the *Amazon Simple Storage Service User Guide*.

Also, verify that you define a policy for your specified bucket that grants `PutObject` and `PutObjectAcl` permissions to the QLDB service (`qldb.amazonaws.com`). To learn more, see [Export permissions](#).

IllegalArgumentException

Message: Unexpected response from Amazon S3 while validating the S3 configuration.

Response from S3: *errorCode errorMessage*

The attempt to write journal export data into the provided S3 bucket failed with the provided Amazon S3 error response. For more information about possible causes, see [Troubleshooting Amazon S3](#) in the *Amazon Simple Storage Service User Guide*.

IllegalArgumentException

Message: Amazon S3 bucket prefix must not exceed 128 characters

The prefix provided in the journal export request contains more than 128 characters.

IllegalArgumentException

Message: Start date must not be greater than end date

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in [ISO 8601](#) date and time format and in Coordinated Universal Time (UTC).

IllegalArgumentException

Message: End date cannot be in the future

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in ISO 8601 date and time format and in UTC.

IllegalArgumentException

Message: The supplied object encryption setting (`S3EncryptionConfiguration`) is not compatible with an AWS Key Management Service (AWS KMS) key

You provided a `KMSKeyArn` with an `ObjectEncryptionType` of either `NO_ENCRYPTION` or `SSE_S3`. You can only provide a customer managed AWS KMS key for an object encryption type of `SSE_KMS`. To learn more about server-side encryption options in Amazon S3, see [Protecting data using server-side encryption](#) in the *Amazon S3 Developer Guide*.

LimitExceededException

Message: Exceeded the limit of 2 concurrently running Journal export jobs

QLDB enforces a default limit of two concurrent journal export jobs.

Streaming journal data

This section lists common exceptions that QLDB can return when you stream journal data from a ledger to Amazon Kinesis Data Streams. For more information about this feature, see [Streaming journal data from Amazon QLDB](#).

Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

AccessDeniedException

Message: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN*

You don't have permissions to pass an IAM role to the QLDB service. QLDB requires a role for all journal stream requests, and you must have permissions to pass this role to QLDB. The role provides QLDB with write permissions in your specified Amazon Kinesis Data Streams resource.

Verify that you define an IAM policy that grants permission to perform the `PassRole` API operation on your specified IAM role resource for the QLDB service (`qldb.amazonaws.com`). For a policy example, see [Identity-based policy examples for Amazon QLDB](#).

IllegalArgumentException

Message: QLDB encountered an error validating Kinesis Data Streams: Response from Kinesis: *errorCode errorMessage*

A possible cause for this error is that the provided Kinesis Data Streams resource doesn't exist. Or, QLDB doesn't have enough permissions to write data records to your specified Kinesis data stream.

Verify that the Kinesis data stream that you provide in your stream request is correct. For more information, see [Creating and updating data streams](#) in the *Amazon Kinesis Data Streams Developer Guide*.

Also, verify that you define a policy for your specified Kinesis data stream that grants the QLDB service (`qldb.amazonaws.com`) permissions to the following actions. For more information, see [Stream permissions](#).

- `kinesis:PutRecord`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`
- `kinesis:ListShards`

IllegalArgumentException

Message: Unexpected response from Kinesis Data Streams while validating the Kinesis configuration. Response from Kinesis: *errorCode errorMessage*

The attempt to write data records to the provided Kinesis data stream failed with the provided Kinesis error response. For more information about possible causes, see [Troubleshooting Amazon Kinesis Data Streams producers](#) in the *Amazon Kinesis Data Streams Developer Guide*.

IllegalArgumentException

Message: Start date must not be greater than end date.

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in [ISO 8601](#) date and time format and in Coordinated Universal Time (UTC).

IllegalArgumentException

Message: Start date cannot be in the future.

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in ISO 8601 date and time format and in UTC.

LimitExceededException

Message: Exceeded the limit of 5 concurrently running Journal streams to Kinesis Data Streams

QLDB enforces a default limit of five concurrent journal streams.

Verifying journal data

This section lists common exceptions that QLDB can return when you verify journal data in a ledger. For more information about this feature, see [Data verification in Amazon QLDB](#).

Each exception includes the specific error message, followed by the API operations that can throw it, a short description, and suggestions for possible solutions.

IllegalArgumentException

Message: The provided Ion value is not valid and cannot be parsed.

API operations: `GetDigest`, `GetBlock`, `GetRevision`

Make sure that you provide a valid [Amazon Ion](#) value before retrying your request.

IllegalArgumentException

Message: The provided block address is not valid.

API operations: `GetDigest`, `GetBlock`, `GetRevision`

Make sure that you provide a valid block address before retrying your request. A block address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"BlFTjlSXze9BIh1K0szcE3",sequenceNo:14}`

IllegalArgumentException

Message: The sequence number of the provided digest tip address is beyond the strand's latest committed record.

API operations: `GetDigest`, `GetBlock`, `GetRevision`

The digest tip address that you provide must have a sequence number less than or equal to the sequence number of the journal strand's latest committed record. Before retrying your request, make sure that you provide a digest tip address with a valid sequence number.

`IllegalArgumentException`

Message: The Strand ID of the provided block address is not valid.

API operations: `GetDigest`, `GetBlock`, `GetRevision`

The block address that you provide must have a strand ID that matches the journal's strand ID. Before retrying your request, make sure that you provide a block address with a valid strand ID.

`IllegalArgumentException`

Message: The sequence number of the provided block address is beyond the strand's latest committed record.

API operations: `GetBlock`, `GetRevision`

The block address that you provide must have a sequence number less than or equal to the sequence number of the strand's latest committed record. Before retrying your request, make sure that you provide a block address with a valid sequence number.

`IllegalArgumentException`

Message: The Strand ID of the provided block address must match the Strand ID of the provided digest tip address.

API operations: `GetBlock`, `GetRevision`

You can only verify a document revision or block if it exists in the same journal strand as the digest that you provide.

`IllegalArgumentException`

Message: The sequence number of the provided block address must not be greater than the sequence number of the provided digest tip address.

API operations: `GetBlock`, `GetRevision`

You can only verify a document revision or block if it's covered by the digest that you provide. This means that it was committed to the journal before the digest tip address.

IllegalArgumentException

Message: The provided Document ID was not found in the block at the specified block address.

API operation: `GetRevision`

The document ID that you provide must exist in the block address that you provide. Before retrying your request, make sure that these two parameters are consistent.

Amazon QLDB PartiQL reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB supports a *subset* of the [PartiQL](#) query language. The following topics describe the QLDB implementation of PartiQL.

Note

- QLDB does not support all PartiQL operations.
- All PartiQL statements in QLDB are subject to transaction limits, as defined in [Quotas and limits in Amazon QLDB](#).
- This reference provides basic syntax and usage examples of PartiQL statements that you manually run on the QLDB console or the QLDB shell. For code examples that show how to programmatically run similar statements using the QLDB driver, see the tutorials in [Getting started with the driver](#).

Topics

- [What is PartiQL?](#)
- [PartiQL in Amazon QLDB](#)
- [PartiQL quick tips in QLDB](#)
- [Amazon QLDB PartiQL reference conventions](#)
- [Data types in Amazon QLDB](#)
- [Amazon QLDB documents](#)
- [Querying Ion with PartiQL in Amazon QLDB](#)
- [PartiQL commands in Amazon QLDB](#)
- [PartiQL functions in Amazon QLDB](#)

- [PartiQL stored procedures in Amazon QLDB](#)
- [PartiQL operators in Amazon QLDB](#)
- [Reserved keywords in Amazon QLDB](#)
- [Amazon Ion data format reference in Amazon QLDB](#)

What is PartiQL?

PartiQL provides SQL-compatible query access across multiple data stores containing structured data, semistructured data, and nested data. It's widely used within Amazon and is now available as part of many AWS services, including QLDB.

For the PartiQL specification and a tutorial on the core query language, see the [PartiQL documentation](#).

PartiQL extends [SQL-92](#) to support documents in the Amazon Ion data format. For information about Amazon Ion, see the [Amazon Ion data format reference in Amazon QLDB](#).

PartiQL in Amazon QLDB

To run PartiQL queries in QLDB, you can use one of the following:

- The *PartiQL editor* on the AWS Management Console for QLDB
- The command line QLDB shell
- An AWS provided QLDB driver to run queries programmatically

For information about using these methods to access QLDB, see [Accessing Amazon QLDB](#).

To learn how to control access to run each PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

PartiQL quick tips in QLDB

The following is a short summary of tips and best practices for working with PartiQL in QLDB:

- **Understand concurrency and transaction limits** – All statements, including SELECT queries, are subject to [optimistic concurrency control \(OCC\)](#) conflicts and [transaction limits](#), including a 30-second transaction timeout.

- **Use indexes** – Use high-cardinality indexes and run targeted queries to optimize your statements and avoid full table scans. To learn more, see [Optimizing query performance](#).
- **Use equality predicates** – Indexed lookups require an *equality* operator (= or IN). Inequality operators (<, >, LIKE, BETWEEN) don't qualify for indexed lookups and result in full table scans.
- **Use inner joins only** – QLDB supports inner joins only. As a best practice, join on fields that are indexed for each table that you're joining. Choose high-cardinality indexes for both the join criteria and the equality predicates.

Amazon QLDB PartiQL reference conventions

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section explains the conventions that are used to write the syntax for the PartiQL commands, functions, and expressions described in the *Amazon QLDB PartiQL reference*. These conventions are not to be confused with the [syntax and semantics](#) of the PartiQL query language itself.

Character	Description
CAPS	Words in capital letters are keywords.
[]	Brackets denote optional arguments or clauses. Multiple arguments in brackets indicate that you can choose any number of the arguments. In addition, arguments in brackets on separate lines indicate that the QLDB parser expects the arguments to be in the order that they're listed in the syntax.
	Pipes indicate that you can choose between the arguments.
<i>red italics</i>	Words in red italics indicate placeholders. Insert the appropriate value in place of the word in red italics.
...	An ellipsis indicates that you can repeat the preceding element.

Character	Description
'	Values in single quotation marks indicate that you must enter the single quotes. In PartiQL, single quotes denote string values, or field names in Amazon Ion structures.
"	Values in double quotation marks indicate that you must enter the double quotes. In PartiQL, double quotes denote quoted identifiers.
`	Values in backticks indicate that you must enter the backticks. In PartiQL, backticks denote Ion literal values.

Data types in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB stores documents in [Amazon Ion](#) format. Amazon Ion is a data serialization format (both in text form and in binary-encoded form) that is a superset of JSON. The following table lists the Ion data types that you can use in QLDB documents.

Data type	Description
null	A generic null value
bool	Boolean values
int	Signed integers of arbitrary size
decimal	Decimal-encoded real numbers of arbitrary precision

Data type	Description
float	Binary-encoded floating point numbers (64-bit IEEE)
timestamp	Date/time/timezone moments of arbitrary precision
string	Unicode text literals
symbol	Unicode symbolic atoms (identifiers)
blob	Binary data of user-defined encoding
clob	Text data of user-defined encoding
struct	Unordered collections of name-value pairs
list	Ordered heterogeneous collections of values

See the [Ion specification document](#) on the Amazon GitHub site for a full list of Ion core data types with complete descriptions and value formatting details.

Amazon QLDB documents

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB stores data records as documents, which are just [Amazon Ion](#) struct objects that are inserted into a table. For the Ion specification, see the [Amazon Ion GitHub](#) site.

Topics

- [Ion document structure](#)
- [PartiQL-Ion type mapping](#)

- [Document ID](#)

Ion document structure

Like JSON, QLDB documents are composed of name-value pairs in the following structure.

```
{
  name1: value1,
  name2: value2,
  name3: value3,
  ...
  nameN: valueN
}
```

The names are symbol tokens, and the values are unrestricted. Each name-value pair is called a *field*. The value of a field can be any of the Ion [Data types](#), including container types: nested structures, lists, and lists of structures.

Also like JSON, a `struct` is denoted by curly braces (`{ ... }`), and a `list` is denoted by square brackets (`[ ... ]`). The following example is a document from the sample data in [Getting started with the Amazon QLDB console](#) that contains values of various types.

```
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  PendingPenaltyTicketAmount: 90.25,
  ValidFrom: 2017-08-21T,
  ValidTo: 2020-05-11T,
  Owners: {
    PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSab0s" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWc0Iu64s" }]
  }
}
```

Important

In Ion, double quotation marks denote string values, and unquoted symbols represent field names. But in PartiQL, single quotation marks denote both strings and field names.

This difference in syntax allows the PartiQL query language to maintain SQL compatibility, and the Amazon Ion data format to maintain JSON compatibility. For details on the syntax and semantics of PartiQL in QLDB, see [Querying Ion with PartiQL](#).

PartiQL-Ion type mapping

In QLDB, PartiQL extends SQL's type system to cover the Ion data model. This mapping is described as follows:

- SQL scalar types are covered by their Ion counterparts. For example:
 - CHAR and VARCHAR are Unicode sequences that map to the Ion `string` type.
 - NUMBER maps to the Ion `decimal` type.
- Ion's `struct` type is equivalent to a SQL tuple, which traditionally represents a table *row*.
 - However, with open content and without schema, queries that rely on the ordered nature of a SQL tuple are not supported (such as the output order of `SELECT *`).
- In addition to NULL, PartiQL has a MISSING type. This is a specialization of NULL and indicates the lack of a field. This type is necessary because Ion `struct` fields might be sparse.

Document ID

QLDB assigns a *document ID* to each document that you insert into a table. All system-assigned IDs are universally unique identifiers (UUID) that are each represented in a Base62-encoded string (for example, `3Qv67yjXEwB9SjmvkuG6Cp`). For more information, see [Unique IDs in Amazon QLDB](#).

Each document *revision* is uniquely identified by a combination of the document ID and a zero-based version number.

The document ID and version fields are included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table). For more information about views in QLDB, see [Core concepts](#). To learn more about metadata, see [Querying document metadata](#).

Querying Ion with PartiQL in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

When you query data in Amazon QLDB, you write statements in PartiQL format, but QLDB returns results in Amazon Ion format. PartiQL is intended to be SQL-compatible, whereas Ion is an extension of JSON. This leads to syntactic differences between how you notate data in your query statements compared to how your query results are displayed.

This section describes basic syntax and semantics for running PartiQL statements manually by using the [QLDB console](#) or the [QLDB shell](#).

Tip

When you run PartiQL queries programmatically, the best practice is to use parameterized statements. You can use a question mark (?) as a bind variable placeholder in your statements to avoid these syntax rules. This is also more secure and efficient. To learn more, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

Topics

- [Syntax and semantics](#)
- [Backtick notation](#)
- [Path navigation](#)
- [Aliasing](#)

- [PartiQL specification](#)

Syntax and semantics

When using the QLDB console or the QLDB shell to query Ion data, the following are the fundamental syntax and semantics of PartiQL:

Case sensitivity

All QLDB system object names—including field names, table names, and ledger names—are case sensitive.

String values

In Ion, double quotation marks (" . . . ") denote a [string](#).

In PartiQL, single quotation marks (' . . . ') denote a string.

Symbols and identifiers

In Ion, single quotation marks (' . . . ') denote a [symbol](#). A subset of symbols in Ion called *identifiers* are represented by unquoted text.

In PartiQL, double quotation marks (" . . . ") denote a quoted PartiQL identifier, such as a [reserved word](#) that is used as a table name. Unquoted text represents a regular PartiQL identifier, such as a table name that isn't a reserved word.

Ion literals

Any Ion literals can be denoted with backticks (` . . . `) in a PartiQL statement.

Field names

Ion field names are case-sensitive symbols. PartiQL lets you denote field names with single quotation marks in a DML statement. This is a shorthand alternative to using PartiQL's `cast` function to define a symbol. It's also more intuitive than using backticks to denote a literal Ion symbol.

Literals

Literals of the PartiQL query language correspond to the Ion data types, as follows:

Scalars

Follow the SQL syntax when applicable, as described in [PartiQL-Ion type mapping](#) section. For example:

- 5
- 'foo'
- null

Structs

Also known as tuples or objects in many formats and other data models.

Denoted by curly braces ({ . . . }) with struct elements separated by commas.

- { 'id' : 3, 'arr': [1, 2] }

Lists

Also known as arrays.

Denoted by square brackets ([. . .]) with list elements separated by commas.

- [1, 'foo']

Bags

Unordered collections in PartiQL.

Denoted by double angle brackets (<< . . . >>) with bag elements separated by commas. In QLDB, a table can be thought of as a bag. However, a bag can't be nested within documents in a table.

- << 1, 'foo' >>

Example

The following is an example of the syntax for an INSERT statement with various Ion types.

```
INSERT INTO VehicleRegistration VALUE
{
  'VIN' : 'KM8SRDHF6EU074761', --string
  'RegNum' : 1722, --integer
  'State' : 'WA',
```

```

    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75, --decimal
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSab0s' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': 'IN7MvYtUjKp1GMZu0F6CG9' }
        ]
    },
    'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate' : `2020-06-25T`
}

```

Backtick notation

PartiQL fully covers all Ion data types, so you can write any statement without using backticks. But there are cases where this Ion literal syntax can make your statements clearer and more concise.

For example, to insert a document with Ion timestamp and symbol values, you can write the following statement using purely PartiQL syntax only.

```

INSERT INTO myTable VALUE
{
    'myTimestamp': to_timestamp('2019-09-04T'),
    'mySymbol': cast('foo' as symbol)
}

```

This is fairly verbose, so instead, you can use backticks to simplify your statement.

```

INSERT INTO myTable VALUE
{
    'myTimestamp': `2019-09-04T`,
    'mySymbol': `foo`
}

```

You can also enclose the entire structure in backticks to save a few more keystrokes.

```

INSERT INTO myTable VALUE
`${
    myTimestamp: 2019-09-04T,
    mySymbol: foo
}`

```

```
}`
```

Important

Strings and symbols are different classes in PartiQL. This means that even if they have the same text, they aren't equal. For example, the following PartiQL expressions evaluate to different Ion values.

```
'foo'
```

```
`foo`
```

Path navigation

When writing data manipulation language (DML) or query statements, you can access fields within nested structures using path steps. PartiQL supports dot notation for accessing field names of a parent structure. The following example accesses the `Model` field of a parent `Vehicle`.

```
Vehicle.Model
```

To access a specific element of a list, you can use the square brackets operator to denote a zero-based ordinal number. The following example accesses the element of `SecondaryOwners` with an ordinal number of 2. In other words, this is the third element of the list.

```
SecondaryOwners[2]
```

Aliasing

QLDB supports open content and schema. So, when you're accessing particular fields in a statement, the best way to ensure that you get the results that you expect is to use aliases. For example, if you don't specify an explicit alias, the system generates an implicit one for your FROM sources.

```
SELECT VIN FROM Vehicle  
--is rewritten to
```

```
SELECT Vehicle.VIN FROM Vehicle AS Vehicle
```

But the results are unpredictable for field name conflicts. If another field named VIN exists in a nested structure within the documents, the VIN values returned by this query might surprise you. As a best practice, write the following statement instead. This query declares `v` as an alias that ranges over the `Vehicle` table. The `AS` keyword is optional.

```
SELECT v.VIN FROM Vehicle [ AS ] v
```

Aliasing is particularly useful when pathing into nested collections within a document. For example, the following statement declares `o` as an alias that ranges over the collection `VehicleRegistration.Owners`.

```
SELECT o.SecondaryOwners  
FROM VehicleRegistration AS r, @r.Owners AS o
```

The `@` character is technically optional here. But it explicitly indicates that you want the `Owners` structure within `VehicleRegistration`, not a different collection named `Owners` (if one existed).

PartiQL specification

For more information about the PartiQL query language, see the [PartiQL Specification](#).

PartiQL commands in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

PartiQL extends SQL-92 to support documents in the Amazon Ion data format. Amazon QLDB supports the following PartiQL commands.

To learn how to control access to run each PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Note

- QLDB does not support all PartiQL commands.
- All PartiQL statements in QLDB are subject to transaction limits, as defined in [Quotas and limits in Amazon QLDB](#).
- This reference provides basic syntax and usage examples of PartiQL statements that you manually run on the QLDB console or the QLDB shell. For code examples that show how to run similar statements using a supported programming language, see the tutorials in [Getting started with the driver](#).

DDL statements (data definition language)

Data definition language (DDL) is the set of PartiQL statements that you use to manage database objects, such as tables and indexes. You use DDL to create and drop these objects.

- [CREATE INDEX](#)
- [CREATE TABLE](#)
- [DROP INDEX](#)
- [DROP TABLE](#)
- [UNDROP TABLE](#)

DML statements (data manipulation language)

Data manipulation language (DML) is the set of PartiQL statements that you use to manage data in QLDB tables. You use DML statements to add, modify, or delete data in a table.

The following DML and query language statements are supported:

- [DELETE](#)
- [FROM \(INSERT, REMOVE, or SET\)](#)
- [INSERT](#)
- [SELECT](#)
- [UPDATE](#)

CREATE INDEX command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the `CREATE INDEX` command to create an index for a document field on a table.

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Important

QLDB requires an index to efficiently look up a document. Without an index, QLDB needs to do a full table scan when reading documents. This can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a `WHERE` predicate clause using an *equality* operator (`=` or `IN`) on an indexed field or a document ID. For more information, see [Optimizing query performance](#).

Note the following constraints when creating indexes:

- An index can only be created on a single top-level field. Composite, nested, unique, and function-based indexes are not supported.
- You can create an index on any [Ion data types](#), including `list` and `struct`. However, you can only do the indexed lookup by equality of the whole Ion value regardless of the Ion type. For example, when using a `list` type as an index, you can't do an indexed lookup by one item inside the list.
- Query performance is improved only when you use an equality predicate; for example, `WHERE indexedField = 123` or `WHERE indexedField IN (456, 789)`.

QLDB doesn't honor inequalities in query predicates. As a result, range filtered scans are not implemented.

- Names of indexed fields are case sensitive and can have a maximum of 128 characters.
- Index creation in QLDB is asynchronous. The amount of time it takes to finish building an index on a non-empty table varies depending on the table size. For more information, see [Managing indexes](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Examples](#)
- [Running programmatically using the driver](#)

Syntax

```
CREATE INDEX ON table_name (field)
```

Parameters

table_name

The name of the table where you want to create the index. The table must already exist.

The table name is case sensitive.

field

The document field name for which to create the index. The field must be a top-level attribute.

Names of indexed fields are case sensitive and can have a maximum of 128 characters.

You can create an index on any [Amazon Ion data types](#), including `list` and `struct`. However, you can only do the indexed lookup by equality of the whole Ion value regardless of the Ion type. For example, when using a `list` type as an index, you can't do an indexed lookup by one item inside the list.

Return value

`tableId` – The unique ID of the table that you created the index on.

Examples

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

CREATE TABLE command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the `CREATE TABLE` command to create a new table.

Tables have simple names with no namespaces. QLDB supports open content and doesn't enforce schema, so you don't define attributes or data types when creating tables.

Note

To learn how to control access to run this PartiQL command in a ledger, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Tagging tables on creation](#)
- [Examples](#)
- [Running programmatically using the driver](#)

Syntax

```
CREATE TABLE table_name [ WITH (aws_tags = `{'key': 'value'}`) ]
```

Parameters

table_name

The unique name of the table to create. An active table with the same name must not already exist. The following are the naming constraints:

- Must only contain 1–128 alphanumeric characters or underscores.
- Must have a letter or an underscore for the first character.
- Can have any combination of alphanumeric characters and underscores for the remaining characters.
- Is case sensitive.
- Must not be a QLDB PartiQL [reserved word](#).

'key': 'value'

(Optional) The tags to attach to the table resource during creation. Each tag is defined as a key-value pair, where the key and value are each denoted by single quotation marks. Each key-value pair is defined inside an Amazon Ion structure that is denoted by backticks.

Tagging tables on creation is currently supported for ledgers in the STANDARD permissions mode only.

Return value

`tableId` – The unique ID of the table that you created.

Tagging tables on creation

Note

Tagging tables on creation is currently supported for ledgers in the STANDARD permissions mode only.

Optionally, you can tag your table resources by specifying tags in a `CREATE TABLE` statement. For more information about tags, see [Tagging Amazon QLDB resources](#). The following example creates a table named `Vehicle` with the tag `environment=production`.

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

Tagging tables on creation requires access to both the `qldb: PartiQLCreateTable` and `qldb: TagResource` actions. To learn more about permissions for QLDB resources, see [How Amazon QLDB works with IAM](#).

By tagging resources while they're being created, you can eliminate the need to run custom tagging scripts after resource creation. After a table is tagged, you can control access to the table based on those tags. For example, you can grant full access only to tables that have a specific tag. For a JSON policy example, see [Full access to all actions based on table tags](#).

Examples

```
CREATE TABLE VehicleRegistration
```

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'development'}`)
```

```
CREATE TABLE Vehicle WITH (aws_tags = `{'key1': 'value1', 'key2': 'value2'}`)
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

DELETE command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the DELETE command to mark an active document as deleted in a table by creating a new, but final revision of the document. This final revision indicates that the document is deleted. This operation ends the lifecycle of a document, which means that no further document revisions with the same document ID can be created.

This operation is irreversible. You can still query the revision history of a deleted document by using the [History function](#).

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Examples](#)
- [Running programmatically using the driver](#)

Syntax

```
DELETE FROM table_name [ AS table_alias ] [ BY id_alias ]  
[ WHERE condition ]
```

Parameters

table_name

The name of the user table containing the data to be deleted. DML statements are only supported in the default [user view](#). Each statement can only run on a single table.

AS *table_alias*

(Optional) A user-defined alias that ranges over a table to be deleted from. The AS keyword is optional.

BY *id_alias*

(Optional) A user-defined alias that binds to the `id` metadata field of each document in the result set. The alias must be declared in the FROM clause using the BY keyword. This is useful when you want to filter on the [document ID](#) while querying the default user view. For more information, see [Using the BY clause to query document ID](#).

WHERE *condition*

The selection criteria for the documents to be deleted.

Note

If you omit the WHERE clause, then all of the documents in the table are deleted.

Return value

`documentId` – The unique ID of each document that you deleted.

Examples

```
DELETE FROM VehicleRegistration AS r
```

```
WHERE r.VIN = '1HVBBAANXWH544237'
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

DROP INDEX command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the `DROP INDEX` command to delete an index on a table.

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Examples](#)

Syntax

```
DROP INDEX "indexId" ON table_name WITH (purge = true)
```

Note

The clause `WITH (purge = true)` is required for all `DROP INDEX` statements, and `true` is currently the only supported value.

The keyword `purge` is case sensitive and must be all lowercase.

Parameters

"*indexId*"

The unique ID of the index to drop, denoted by double quotation marks.

ON *table_name*

The name of the table whose index you want to drop.

Return value

`tableId` – The unique ID of the table whose index you dropped.

Examples

```
DROP INDEX "4tPW3fUhaVhDinRgKRLhGU" ON VehicleRegistration WITH (purge = true)
```

DROP TABLE command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the `DROP TABLE` command to deactivate an existing table. You can use the [UNDROP TABLE](#) statement to reactivate it. Deactivating or reactivating a table has no effect on its documents or indexes.

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Examples](#)

Syntax

```
DROP TABLE table_name
```

Parameters

table_name

The name of the table to deactivate. The table must already exist and have a status of ACTIVE.

Return value

tableId – The unique ID of the table that you deactivated.

Examples

```
DROP TABLE VehicleRegistration
```

FROM (INSERT, REMOVE, or SET) command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, a statement that starts with FROM is a PartiQL extension that lets you to insert and remove specific elements within a document. You can also use this statement to update existing elements in a document, similar to the [UPDATE](#) command.

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Nested collections](#)
- [Return value](#)
- [Examples](#)
- [Running programmatically using the driver](#)

Syntax

FROM-INSERT

Insert a new element within an existing document. To insert a new top-level document into a table, you must use [INSERT](#).

```
FROM table_name [ AS table_alias ] [ BY id_alias ]  
[ WHERE condition ]
```

```
INSERT INTO element VALUE data [ AT key_name ]
```

FROM-REMOVE

Remove an existing element within a document, or remove an entire top-level document. The latter is semantically the same as the traditional [DELETE](#) syntax.

```
FROM table_name [ AS table_alias ] [ BY id_alias ]  
[ WHERE condition ]  
REMOVE element
```

FROM-SET

Update one or more elements within a document. If an element doesn't exist, it's inserted. This is semantically the same as the traditional [UPDATE](#) syntax.

```
FROM table_name [ AS table_alias ] [ BY id_alias ]  
[ WHERE condition ]  
SET element = data [, element = data, ... ]
```

Parameters

table_name

The name of the user table containing the data to be modified. DML statements are only supported in the default [user view](#). Each statement can only run on a single table.

In this clause, you can also include one or more collections that are nested within the specified table. For more details, see [Nested collections](#).

AS *table_alias*

(Optional) A user-defined alias that ranges over a table to be modified. All table aliases that are used in the SET, REMOVE, INSERT INTO, or WHERE clause must be declared in the FROM clause. The AS keyword is optional.

BY *id_alias*

(Optional) A user-defined alias that binds to the `id` metadata field of each document in the result set. The alias must be declared in the FROM clause using the BY keyword. This is useful when you want to filter on the [document ID](#) while querying the default user view. For more information, see [Using the BY clause to query document ID](#).

WHERE *condition*

The selection criteria for the documents to be modified.

Note

If you omit the WHERE clause, then all of the documents in the table are modified.

element

A document element to be created or modified.

data

A new value for the element.

AT *key_name*

A key name to be added within the documents to be modified. You must specify the corresponding VALUE along with the key name. This is required for inserting a new value AT a specific position within a document.

Nested collections

While you can run a DML statement on a single table only, you can specify nested collections within documents in that table as additional sources. Each alias that you declare for a nested collection can be used in the WHERE clause and the SET, INSERT INTO, or REMOVE clause.

For example, the FROM sources of the following statement include both the `VehicleRegistration` table and the nested `Owners.SecondaryOwners` structure.

```
FROM VehicleRegistration r, @r.Owners.SecondaryOwners o
WHERE r.VIN = '1N4AL11D75C109151' AND o.PersonId = 'abc123'
SET o.PersonId = 'def456'
```

This example updates the specific element of the `SecondaryOwners` list that has a `PersonId` of `'abc123'` within the `VehicleRegistration` document that has a VIN of `'1N4AL11D75C109151'`. This expression lets you specify an element of a list by its value rather than its index.

Return value

`documentId` – The unique ID of each document that you updated or deleted.

Examples

Modify an element within a document. If the element doesn't exist, it's inserted.

```
FROM Vehicle AS v
WHERE v.VIN = '1N4AL11D75C109151' AND v.Color = 'Silver'
SET v.Color = 'Shiny Gray'
```

Modify or insert an element and filter on the system-assigned document `id` metadata field.

```
FROM Vehicle AS v BY v_id
WHERE v_id = 'documentId'
SET v.Color = 'Shiny Gray'
```

Modify the `PersonId` field of the *first* element in the `Owners.SecondaryOwners` list within a document.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
```

Remove an existing element within a document.

```
FROM Person AS p
WHERE p.GovId = '111-22-3333'
REMOVE p.Address
```

Remove a whole document from a table.

```
FROM Person AS p
WHERE p.GovId = '111-22-3333'
REMOVE p
```

Remove the *first* element of the `Owners.SecondaryOwners` list within a document in the `VehicleRegistration` table.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
REMOVE r.Owners.SecondaryOwners[0]
```

Insert `{ 'Mileage' : 26500 }` as a top-level name-value pair within a document in the `Vehicle` table.

```
FROM Vehicle AS v
WHERE v.VIN = '1N4AL11D75C109151'
INSERT INTO v VALUE 26500 AT 'Mileage'
```

Append `{ 'PersonId' : 'abc123' }` as a name-value pair in the `Owners.SecondaryOwners` field of a document in the `VehicleRegistration` table. Note that `Owners.SecondaryOwners` must already exist and must be a list data type for this statement to be valid. Otherwise, the keyword `AT` is required in the `INSERT INTO` clause.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' }
```

Insert `{ 'PersonId' : 'abc123' }` as the *first* element in the existing `Owners.SecondaryOwners` list within a document.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' } AT 0
```

Append multiple name-value pairs to the existing `Owners.SecondaryOwners` list within a document.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners << { 'PersonId' : 'abc123' }, { 'PersonId' :
  'def456' } >>
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

INSERT command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the INSERT command to add one or more Amazon Ion documents to a table.

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Examples](#)
- [Running programmatically using the driver](#)

Syntax

Insert a single document.

```
INSERT INTO table_name VALUE document
```


Insert multiple documents.

```
INSERT INTO table_name << document, document, ... >>
```

Parameters

table_name

The name of the user table where you want to insert the data. The table must already exist. DML statements are only supported in the default [user view](#).

document

A valid [QLDB document](#). You must specify at least one document. Multiple documents must be separated by commas.

The document must be denoted by curly braces ({ . . . }).

Each field name in the document is a case-sensitive Ion symbol that can be denoted by *single* quotation marks (' . . . ') in PartiQL.

String values are also denoted by *single* quotation marks (' . . . ') in PartiQL.

Any Ion literals can be denoted with backticks (` . . . `).

Note

Double angle brackets (<< . . . >>) denote an unordered collection (known as a *bag* in PartiQL) and are required only if you want to insert multiple documents.

Return value

documentId – The unique ID of each document that you inserted.

Examples

Insert a single document.

```
INSERT INTO VehicleRegistration VALUE  
{
```

```

'VIN' : 'KM8SRDHF6EU074761', --string
'RegNum' : 1722, --integer
'State' : 'WA',
'City' : 'Kent',
'PendingPenaltyTicketAmount' : 130.75, --decimal
'Owners' : { --nested struct
    'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSab0s' },
    'SecondaryOwners' : [ --list of structs
        { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
        { 'PersonId': 'IN7MvYtUj1GZu0F6CG9' }
    ]
},
'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
'ValidToDate' : `2020-06-25T`
}

```

This statement returns the unique ID of the document that you inserted, as follows.

```

{
    documentId: "2kKu0PNB07D2iTPBrUTWGl"
}

```

Insert multiple documents.

```

INSERT INTO Person <<
{
    'FirstName' : 'Raul',
    'LastName' : 'Lewis',
    'DOB' : `1963-08-19T`,
    'GovId' : 'LEWISR261LL',
    'GovIdType' : 'Driver License',
    'Address' : '1719 University Street, Seattle, WA, 98109'
},
{
    'FirstName' : 'Brent',
    'LastName' : 'Logan',
    'DOB' : `1967-07-03T`,
    'GovId' : 'LOGANB486CG',
    'GovIdType' : 'Driver License',
    'Address' : '43 Stockert Hollow Road, Everett, WA, 98203'
},
{
    'FirstName' : 'Alexis',

```

```
'LastName' : 'Pena',  
'DOB' : `1974-02-10T`,  
'GovId' : '744 849 301',  
'GovIdType' : 'SSN',  
'Address' : '4058 Melrose Street, Spokane Valley, WA, 99206'  
}  
>>
```

This statement returns the unique ID of each document that you inserted, as follows.

```
{  
  documentId: "6WXzLscsJ3bDWW97Dy8nyp"  
},  
{  
  documentId: "35e0ToZyTGJ7LGvcwrkX65"  
},  
{  
  documentId: "BVHPcH612o7JR0Q4yP8jiH"  
}
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

SELECT command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the SELECT command to retrieve data from one or more tables. Every SELECT query in QLDB is processed in a transaction and is subject to a [transaction timeout limit](#).

The order of the results is not specific and can vary for each SELECT query. You shouldn't rely on the results order for any query in QLDB.

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Warning

When you run a query in QLDB without an indexed lookup, it invokes a full table scan. PartiQL supports such queries because it's SQL compatible. However, *don't* run table scans for production use cases in QLDB. Table scans can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator on an indexed field or a document ID; for example, WHERE indexedField = 123 or WHERE indexedField IN (456, 789). For more information, see [Optimizing query performance](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Joins](#)
- [Nested query limitations](#)
- [Examples](#)
- [Running programmatically using the driver](#)

Syntax

```
SELECT [ VALUE ] expression [ AS field_alias ] [, expression, ... ]  
FROM source [ AS source_alias ] [ AT idx_alias ] [ BY id_alias ] [, source, ... ]  
[ WHERE condition ]
```

Parameters

VALUE

A qualifier for your expression that makes the query return the raw data type value, rather than the value being wrapped in a tuple structure.

expression

A projection formed from the * wildcard or a projection list of one or more document fields from the result set. An expression can consist of calls to [PartiQL functions](#) or fields that are modified by [PartiQL operators](#).

AS *field_alias*

(Optional) A temporary, user-defined alias for the field that is used in the final result set. The AS keyword is optional.

If you don't specify an alias for an expression that isn't a simple field name, the result set applies a default name to that field.

FROM *source*

A source to be queried. The only sources currently supported are table names, [inner joins](#) between tables, nested SELECT queries (subject to [Nested query limitations](#)), and [history function](#) calls for a table.

You must specify at least one source. Multiple sources must be separated by commas.

AS *source_alias*

(Optional) A user-defined alias that ranges over a source to be queried. All source aliases that are used in the SELECT OR WHERE clause must be declared in the FROM clause. The AS keyword is optional.

AT *idx_alias*

(Optional) A user-defined alias that binds to the index (ordinal) number of each element within a list from the source. The alias must be declared in the FROM clause using the AT keyword.

BY *id_alias*

(Optional) A user-defined alias that binds to the id metadata field of each document in the result set. The alias must be declared in the FROM clause using the BY keyword. This is useful

when you want to project or filter on the [document ID](#) while querying the default user view. For more information, see [Using the BY clause to query document ID](#).

WHERE *condition*

The selection criteria and join criteria (if applicable) for the query.

Note

If you omit the WHERE clause, then all of the documents in the table are retrieved.

Joins

Only inner joins are currently supported. You can write inner join queries using the explicit INNER JOIN clause, as follows. In this syntax, JOIN must be paired with ON, and the INNER keyword is optional.

```
SELECT expression
FROM table1 AS t1 [ INNER ] JOIN table2 AS t2
ON t1.element = t2.element
```

Or, you can write inner joins using the implicit syntax, as follows.

```
SELECT expression
FROM table1 AS t1, table2 AS t2
WHERE t1.element = t2.element
```

Nested query limitations

You can write nested queries (subqueries) within SELECT expressions and within FROM sources. The main restriction is that only the outermost query can access the global database environment. For example, suppose that you have a ledger with tables `VehicleRegistration` and `Person`. The following nested query is not valid because the inner SELECT tries to access `Person`.

```
SELECT r.VIN,
       (SELECT p.PersonId FROM Person AS p WHERE p.PersonId =
        r.Owners.PrimaryOwner.PersonId) AS PrimaryOwner
```

```
FROM VehicleRegistration AS r
```

Whereas the following nested query is valid.

```
SELECT r.VIN, (SELECT o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM VehicleRegistration AS r
```

Examples

The following query shows a basic SELECT all wildcard with a standard WHERE predicate clause that uses the IN operator.

```
SELECT * FROM Vehicle
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

The following shows SELECT projections with a string filter.

```
SELECT FirstName, LastName, Address
FROM Person
WHERE Address LIKE '%Seattle%'
AND GovId = 'LEWISR261LL'
```

The following shows a correlated subquery that flattens nested data. Note that the @ character is technically optional here. But it explicitly indicates that you want the Owners structure that is nested within VehicleRegistration, not a different collection named Owners (if one existed). For more context, see [Nested data](#) in the chapter *Working with data and history*.

```
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

The following shows a subquery in the SELECT list that projects nested data, and an implicit inner join.

```
SELECT
```

```
v.Make,  
v.Model,  
(SELECT VALUE o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner  
FROM  
  VehicleRegistration AS r, Vehicle AS v  
WHERE  
  r.VIN = v.VIN AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

The following shows an explicit inner join.

```
SELECT  
  v.Make,  
  v.Model,  
  r.Owners  
FROM  
  VehicleRegistration AS r JOIN Vehicle AS v  
ON  
  r.VIN = v.VIN  
WHERE  
  r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

The following shows a projection of the document id metadata field, using the BY clause.

```
SELECT  
  r_id,  
  r.VIN  
FROM  
  VehicleRegistration AS r BY r_id  
WHERE  
  r_id = 'documentId'
```

The following uses the BY clause to join the DriversLicense and Person tables on their PersonId and document id fields respectively.

```
SELECT * FROM DriversLicense AS d INNER JOIN Person AS p BY pid  
ON d.PersonId = pid  
WHERE pid = 'documentId'
```

The following uses the [Committed view](#) to join the DriversLicense and Person tables on their PersonId and document id fields respectively.


```
SELECT * FROM DriversLicense AS d INNER JOIN _ql_committed_Person AS cp
ON d.PersonId = cp.metadata.id
WHERE cp.metadata.id = 'documentId'
```

The following returns the PersonId and index (ordinal) number of each person in the Owners.SecondaryOwners list for a document in table VehicleRegistration.

```
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = 'KM8SRDHF6EU074761'
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

UPDATE command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the UPDATE command to modify the value of one or more elements within a document. If an element doesn't exist, it's inserted.

You can also use this command to explicitly insert and remove specific elements within a document, similar to [FROM \(INSERT, REMOVE, or SET\)](#) statements.

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Examples](#)
- [Running programmatically using the driver](#)

Syntax**UPDATE-SET**

Update one or more elements within a document. If an element doesn't exist, it's inserted. This is semantically the same as the [FROM-SET](#) statement.

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]  
SET element = data [, element = data, ... ]  
[ WHERE condition ]
```

UPDATE-INSERT

Insert a new element within an existing document. To insert a new top-level document into a table, you must use [INSERT](#).

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]  
INSERT INTO element VALUE data [ AT key_name ]  
[ WHERE condition ]
```

UPDATE-REMOVE

Remove an existing element within a document, or remove an entire top-level document. The latter is semantically the same as the traditional [DELETE](#) syntax.

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]  
REMOVE element  
[ WHERE condition ]
```

Parameters

table_name

The name of the user table containing the data to be modified. DML statements are only supported in the default [user view](#). Each statement can only run on a single table.

AS *table_alias*

(Optional) A user-defined alias that ranges over a table to be updated. The AS keyword is optional.

BY *id_alias*

(Optional) A user-defined alias that binds to the `id` metadata field of each document in the result set. The alias must be declared in the UPDATE clause using the BY keyword. This is useful when you want to filter on the [document ID](#) while querying the default user view. For more information, see [Using the BY clause to query document ID](#).

element

A document element to be created or modified.

data

A new value for the element.

AT *key_name*

A key name to be added within the documents to be modified. You must specify the corresponding VALUE along with the key name. This is required for inserting a new value AT a specific position within a document.

WHERE *condition*

The selection criteria for the documents to be modified.

Note

If you omit the WHERE clause, then all of the documents in the table are modified.

Return value

`documentId` – The unique ID of each document that you updated.

Examples

Update a field in a document. If the field doesn't exist, it's inserted.

```
UPDATE Person AS p
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE p.GovId = '111-22-3333'
```

Filter on the system-assigned document id metadata field.

```
UPDATE Person AS p BY pid
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE pid = 'documentId'
```

Overwrite an entire document.

```
UPDATE Person AS p
SET p = {
  'FirstName' : 'Rosemarie',
  'LastName' : 'Holloway',
  'DOB' : `1977-06-18T`,
  'GovId' : '111-22-3333',
  'GovIdType' : 'Driver License',
  'Address' : '4637 Melrose Street, Ellensburg, WA, 98926'
}
WHERE p.GovId = '111-22-3333'
```

Modify the `PersonId` field of the *first* element in the `Owners.SecondaryOwners` list within a document.

```
UPDATE VehicleRegistration AS r
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
WHERE r.VIN = '1N4AL11D75C109151'
```

Insert `{ 'Mileage' : 26500 }` as a top-level name-value pair within a document in the `Vehicle` table.

```
UPDATE Vehicle AS v
INSERT INTO v VALUE 26500 AT 'Mileage'
WHERE v.VIN = '1N4AL11D75C109151'
```

Append `{ 'PersonId' : 'abc123' }` as a name-value pair in the `Owners.SecondaryOwners` field of a document in the `VehicleRegistration` table. Note that `Owners.SecondaryOwners` must already exist and must be a list data type for this statement to be valid. Otherwise, the keyword `AT` is required in the `INSERT INTO` clause.

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' }
WHERE r.VIN = '1N4AL11D75C109151'
```

Insert `{ 'PersonId' : 'abc123' }` as the *first* element in the existing `Owners.SecondaryOwners` list within a document.

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' } AT 0
WHERE r.VIN = '1N4AL11D75C109151'
```

Append multiple name-value pairs to the existing `Owners.SecondaryOwners` list within a document.

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners << { 'PersonId' : 'abc123' }, { 'PersonId' :
  'def456' } >>
WHERE r.VIN = '1N4AL11D75C109151'
```

Remove an existing element within a document.

```
UPDATE Person AS p
REMOVE p.Address
WHERE p.GovId = '111-22-3333'
```

Remove a whole document from a table.

```
UPDATE Person AS p
REMOVE p
```

```
WHERE p.GovId = '111-22-3333'
```

Remove the *first* element of the `Owners.SecondaryOwners` list within a document in the `VehicleRegistration` table.

```
UPDATE VehicleRegistration AS r
REMOVE r.Owners.SecondaryOwners[0]
WHERE r.VIN = '1N4AL11D75C109151'
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver, see the following tutorials in *Getting started with the driver*:

- Java: [Quick start tutorial](#) | [Cookbook reference](#)
- .NET: [Quick start tutorial](#) | [Cookbook reference](#)
- Go: [Quick start tutorial](#) | [Cookbook reference](#)
- Node.js: [Quick start tutorial](#) | [Cookbook reference](#)
- Python: [Quick start tutorial](#) | [Cookbook reference](#)

UNDROP TABLE command in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the `UNDROP TABLE` command to reactivate a table that you previously dropped (that is, deactivated). Deactivating or reactivating a table has no effect on its documents or indexes.

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Examples](#)

Syntax

```
UNDROP TABLE "tableId"
```

Parameters

"*tableId*"

The unique ID of the table to reactivate, denoted by double quotation marks.

The table must have been previously dropped, meaning that it exists in the [system catalog table](#) `information_schema.user_tables` and has a status of `INACTIVE`. There must also be no active, existing table with the same name.

Return value

`tableId` – The unique ID of the table that you reactivated.

Examples

```
UNDROP TABLE "5PLf9SXwndd631PaSIa006"
```

PartiQL functions in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

PartiQL in Amazon QLDB supports the following built-in variants of SQL standard functions.

 Note

Any SQL functions that aren't included in this list are not currently supported in QLDB. This function reference is based on the PartiQL documentation [Built-in Functions](#).

Unknown type (null and missing) propagation

Unless otherwise stated, all of these functions propagate null and missing argument values. *Propagation* of NULL or MISSING is defined as returning NULL if any function argument is either NULL or MISSING. The following are examples of this propagation.

```
CHAR_LENGTH(null)      -- null
CHAR_LENGTH(missing) -- null (also returns null)
```

Aggregate functions

- [AVG](#)
- [COUNT](#)
- [MAX](#)
- [MIN](#)
- [SIZE](#)
- [SUM](#)

Conditional functions

- [COALESCE](#)
- [EXISTS](#)
- [NULLIF](#)

Date and time functions

- [DATE_ADD](#)

- [DATE_DIFF](#)
- [EXTRACT](#)
- [UTCNOW](#)

Scalar functions

- [TXID](#)

String functions

- [CHAR_LENGTH](#)
- [CHARACTER_LENGTH](#)
- [LOWER](#)
- [SUBSTRING](#)
- [TRIM](#)
- [UPPER](#)

Data type formatting functions

- [CAST](#)
- [TO_STRING](#)
- [TO_TIMESTAMP](#)

AVG function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the AVG function to return the average (arithmetic mean) of the input expression values. This function works with numeric values and ignores null or missing values.

Syntax

```
AVG ( expression )
```

Arguments

expression

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- int
- decimal
- float

Return type: decimal

Examples

```
SELECT AVG(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r -- 147.19
SELECT AVG(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >> -- 2.
```

Related functions

- [COUNT](#)
- [MAX](#)
- [MIN](#)
- [SIZE](#)
- [SUM](#)

CAST function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the CAST function to evaluate a given expression to a value and convert the value to a specified target data type. If the conversion can't be made, the function returns an error.

Syntax

```
CAST ( expression AS type )
```

Arguments

expression

The field name or expression that evaluates to a value that the function converts. Converting null values returns nulls. This parameter can be any of the supported [Data types](#).

type

The name of the target data type for conversion. This parameter can be one of the supported [Data types](#).

Return type

The data type that is specified by the *type* argument.

Examples

The following examples show the propagation of unknown types (NULL or MISSING).

```
CAST(null    AS null)    -- null
CAST(missing AS null)    -- null
CAST(missing AS missing) -- missing
CAST(null    AS missing) -- missing
```

```
CAST(null AS boolean) -- null (null AS any data type name results in null)
CAST(missing AS boolean) -- missing (missing AS any data type name results in missing)
```

Any value that is not an unknown type can't be cast to NULL or MISSING.

```
CAST(true AS null) -- error
CAST(true AS missing) -- error
CAST(1 AS null) -- error
CAST(1 AS missing) -- error
```

The following examples show casting AS boolean.

```
CAST(true AS boolean) -- true no-op
CAST(0 AS boolean) -- false
CAST(1 AS boolean) -- true
CAST('1e0' AS boolean) -- true (float)
CAST('1d0' AS boolean) -- true (decimal)
CAST('a' AS boolean) -- false
CAST('true' AS boolean) -- true (SqlName string 'true')
CAST('true' AS boolean) -- true (Ion symbol 'true')
CAST('false' AS boolean) -- false (Ion symbol 'false')
```

The following examples show casting AS integer.

```
CAST(true AS integer) -- 1
CAST(false AS integer) -- 0
CAST(1 AS integer) -- 1
CAST('1d0' AS integer) -- 1
CAST('1d3' AS integer) -- 1000
CAST(1.00 AS integer) -- 1
CAST(1.45 AS integer) -- 1
CAST(1.75 AS integer) -- 1
CAST('12' AS integer) -- 12
CAST('aa' AS integer) -- error
CAST('22' AS integer) -- 22
CAST('x' AS integer) -- error
```

The following examples show casting AS float.

```
CAST(true AS float) -- 1e0
CAST(false AS float) -- 0e0
```

```

CAST(1      AS float) -- 1e0
CAST(`1d0`  AS float) -- 1e0
CAST(`1d3`  AS float) -- 1000e0
CAST(1.00   AS float) -- 1e0
CAST('12'   AS float) -- 12e0
CAST('aa'   AS float) -- error
CAST(`'22'` AS float) -- 22e0
CAST(`'x'`  AS float) -- error

```

The following examples show casting AS decimal.

```

CAST(true   AS decimal) -- 1.
CAST(false  AS decimal) -- 0.
CAST(1      AS decimal) -- 1.
CAST(`1d0`  AS decimal) -- 1. (REPL printer serialized to 1.)
CAST(`1d3`  AS decimal) -- 1d3
CAST(1.00   AS decimal) -- 1.00
CAST('12'   AS decimal) -- 12.
CAST('aa'   AS decimal) -- error
CAST(`'22'` AS decimal) -- 22.
CAST(`'x'`  AS decimal) -- error

```

The following examples show casting AS timestamp.

```

CAST(`2001T`           AS timestamp) -- 2001T
CAST('2001-01-01T'     AS timestamp) -- 2001-01-01T
CAST(`'2010-01-01T00:00:00.000Z'` AS timestamp) -- 2010-01-01T00:00:00.000Z
CAST(true              AS timestamp) -- error
CAST(2001              AS timestamp) -- error

```

The following examples show casting AS symbol.

```

CAST(`'xx'`           AS symbol) -- xx (`'xx'` is an Ion symbol)
CAST('xx'             AS symbol) -- xx ('xx' is a string)
CAST(42               AS symbol) -- '42'
CAST(`1e0`            AS symbol) -- '1.0'
CAST(`1d0`            AS symbol) -- '1'
CAST(true             AS symbol) -- 'true'
CAST(false            AS symbol) -- 'false'
CAST(`2001T`          AS symbol) -- '2001T'
CAST(`2001-01-01T00:00:00.000Z` AS symbol) -- '2001-01-01T00:00:00.000Z'

```

The following examples show casting AS string.

```
CAST(`'xx'` AS string) -- "xx" (`'xx'` is an Ion symbol)
CAST('xx' AS string) -- "xx" ('xx' is a string)
CAST(42 AS string) -- "42"
CAST(`1e0` AS string) -- "1.0"
CAST(`1d0` AS string) -- "1"
CAST(true AS string) -- "true"
CAST(false AS string) -- "false"
CAST(`2001T` AS string) -- "2001T"
CAST(`2001-01-01T00:00:00.000Z` AS string) -- "2001-01-01T00:00:00.000Z"
```

The following examples show casting AS struct.

```
CAST(`{ a: 1 }` AS struct) -- {a:1}
CAST(true AS struct) -- err
```

The following examples show casting AS list.

```
CAST(`[1, 2, 3]` AS list) -- [1,2,3]
CAST(<<'a', { 'b':2 }>> AS list) -- ["a",{ 'b':2}]
CAST({ 'b':2 } AS list) -- error
```

The following examples are runnable statements that include some of the previous examples.

```
SELECT CAST(true AS integer) FROM << 0 >> -- 1
SELECT CAST('2001-01-01T' AS timestamp) FROM << 0 >> -- 2001-01-01T
SELECT CAST('xx' AS symbol) FROM << 0 >> -- xx
SELECT CAST(42 AS string) FROM << 0 >> -- "42"
```

Related functions

- [TO_STRING](#)
- [TO_TIMESTAMP](#)

CHAR_LENGTH function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the CHAR_LENGTH function to return the number of characters in the specified string, where *character* is defined as a single unicode code point.

Syntax

```
CHAR_LENGTH ( string )
```

CHAR_LENGTH is a synonym of [CHARACTER_LENGTH function in Amazon QLDB](#).

Arguments

string

The field name or expression of data type string that the function evaluates.

Return type

int

Examples

```
SELECT CHAR_LENGTH('') FROM << 0 >>          -- 0
SELECT CHAR_LENGTH('abcdefg') FROM << 0 >>    -- 7
SELECT CHAR_LENGTH('e#') FROM << 0 >>         -- 2 (because 'e#' is two code points: the
letter 'e' and combining character U+032B)
```

Related functions

- [LOWER](#)

- [SUBSTRING](#)
- [TRIM](#)
- [UPPER](#)

CHARACTER_LENGTH function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Synonym of the CHAR_LENGTH function.

See [CHAR_LENGTH function in Amazon QLDB](#).

COALESCE function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, given a list of one or more arguments, use the COALESCE function to evaluate the arguments in order from left to right and return the first value that is not an unknown type (NULL or MISSING). If all argument types are unknown, the result is NULL.

The COALESCE function doesn't propagate NULL and MISSING.

Syntax

```
COALESCE ( expression [, expression, ... ] )
```


Arguments

expression

The list of one or more field names or expressions that the function evaluates. Each argument can be any of the supported [Data types](#).

Return type

Any supported data type. The return type is either NULL or the same as the type of the first expression that evaluates to a non-null and non-missing value.

Examples

```
SELECT COALESCE(1, null) FROM << 0 >>      -- 1
SELECT COALESCE(null, null, 1) FROM << 0 >>  -- 1
SELECT COALESCE(null, 'string') FROM << 0 >> -- "string"
```

Related functions

- [EXISTS](#)
- [NULLIF](#)

COUNT function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the COUNT function to return the number of documents that are defined by the given expression. This function has two variations:

- COUNT(*) – Counts all of the documents in the target table whether or not they include null or missing values.

- **COUNT(expression)** – Computes the number of documents with non-null values in a specific, existing field or expression.

Warning

The COUNT function is not optimized, so we don't recommend using it without an indexed lookup. When you run a query in QLDB without an indexed lookup, it invokes a full table scan. This can cause performance problems on large tables, including concurrency conflicts and transaction timeouts.

To avoid table scans, you must run statements with a WHERE predicate clause using an *equality* operator (= or IN) on an indexed field or a document ID. For more information, see [Optimizing query performance](#).

Syntax

```
COUNT ( * | expression )
```

Arguments

expression

The field name or expression that the function operates on. This parameter can be any of the supported [Data types](#).

Return type

int

Examples

```
SELECT COUNT(*) FROM VehicleRegistration r WHERE r.LicensePlateNumber = 'CA762X' -- 1
SELECT COUNT(r.VIN) FROM Vehicle r WHERE r.VIN = '1N4AL11D75C109151' -- 1
SELECT COUNT(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >> -- 3
```

Related functions

- [AVG](#)

- [MAX](#)
- [MIN](#)
- [SIZE](#)
- [SUM](#)

DATE_ADD function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the DATE_ADD function to increment a given timestamp value by a specified interval.

Syntax

```
DATE_ADD( datetimepart, interval, timestamp )
```

Arguments

datetimepart

The date or time part that the function operates on. This parameter can be one of the following:

- year
- month
- day
- hour
- minute
- second

interval

The integer that specifies the interval to add to the given *timestamp*. A negative integer subtracts the interval.

timestamp

The field name or expression of data type `timestamp` that the function increments.

An Ion timestamp literal value can be denoted with backticks (``...``). For formatting details and examples of timestamp values, see [Timestamps](#) in the Amazon Ion specification document.

Return type

`timestamp`

Examples

```
DATE_ADD(year, 5, `2010-01-01T`)           -- 2015-01-01T
DATE_ADD(month, 1, `2010T`)                 -- 2010-02T (result adds precision as
necessary)
DATE_ADD(month, 13, `2010T`)                -- 2011-02T (2010T is equivalent to
2010-01-01T00:00:00.000Z)
DATE_ADD(day, -1, `2017-01-10T`)            -- 2017-01-09T
DATE_ADD(hour, 1, `2017T`)                  -- 2017-01-01T01:00Z
DATE_ADD(hour, 1, `2017-01-02T03:04Z`)      -- 2017-01-02T04:04Z
DATE_ADD(minute, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:05:05.006Z
DATE_ADD(second, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:04:06.006Z

-- Runnable statements
SELECT DATE_ADD(year, 5, `2010-01-01T`) FROM << 0 >> -- 2015-01-01T
SELECT DATE_ADD(day, -1, `2017-01-10T`) FROM << 0 >> -- 2017-01-09T
```

Related functions

- [DATE_DIFF](#)
- [EXTRACT](#)
- [TO_STRING](#)
- [TO_TIMESTAMP](#)
- [UTCNOW](#)

DATE_DIFF function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the DATE_DIFF function to return the difference between the specified date parts of two given timestamps.

Syntax

```
DATE_DIFF( datetimepart, timestamp1, timestamp2 )
```

Arguments

datetimepart

The date or time part that the function operates on. This parameter can be one of the following:

- year
- month
- day
- hour
- minute
- second

timestamp1, *timestamp2*

The two field names or expressions of data type timestamp that the function compares. If *timestamp2* is later than *timestamp1*, the result is positive. If *timestamp2* is earlier than *timestamp1*, the result is negative.

An Ion timestamp literal value can be denoted with backticks (` . . . `). For formatting details and examples of timestamp values, see [Timestamps](#) in the Amazon Ion specification document.

Return type

int

Examples

```
DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`)           -- 1
DATE_DIFF(year, `2010-12T`, `2011-01T`)                 -- 0 (must be at least 12
  months apart to evaluate as a 1 year difference)
DATE_DIFF(month, `2010T`, `2010-05T`)                   -- 4 (2010T is equivalent to
  2010-01-01T00:00:00.000Z)
DATE_DIFF(month, `2010T`, `2011T`)                     -- 12
DATE_DIFF(month, `2011T`, `2010T`)                     -- -12
DATE_DIFF(month, `2010-12-31T`, `2011-01-01T`)         -- 0 (must be at least a full
  month apart to evaluate as a 1 month difference)
DATE_DIFF(day, `2010-01-01T23:00Z`, `2010-01-02T01:00Z`) -- 0 (must be at least 24
  hours apart to evaluate as a 1 day difference)

-- Runnable statements
SELECT DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`) FROM << 0 >> -- 1
SELECT DATE_DIFF(month, `2010T`, `2010-05T`) FROM << 0 >>      -- 4
```

Related functions

- [DATE_ADD](#)
- [EXTRACT](#)
- [TO_STRING](#)
- [TO_TIMESTAMP](#)
- [UTCNOW](#)

EXISTS function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, given a PartiQL value, use the EXISTS function to return TRUE if the value is a non-empty collection. Otherwise, this function returns FALSE. If the input to EXISTS is not a container, the result is FALSE.

The EXISTS function doesn't propagate NULL and MISSING.

Syntax

```
EXISTS ( value )
```

Arguments

value

The field name or expression that the function evaluates. This parameter can be any of the supported [Data types](#).

Return type

bool

Examples

```
EXISTS(`[]`)           -- false (empty list)
EXISTS(`[1, 2, 3]`)    -- true (non-empty list)
EXISTS(`[missing]`)    -- true (non-empty list)
EXISTS(`{}`)           -- false (empty struct)
EXISTS(`{ a: 1 }`)     -- true (non-empty struct)
EXISTS(`()`)           -- false (empty s-expression)
EXISTS(`(+ 1 2)`)      -- true (non-empty s-expression)
EXISTS(1)              -- false
EXISTS(`2017T`)        -- false
EXISTS(null)           -- false
EXISTS(missing)        -- error

-- Runnable statements
SELECT EXISTS(`[]`) FROM << 0 >>      -- false
SELECT EXISTS(`[1, 2, 3]`) FROM << 0 >> -- true
```

Related functions

- [COALESCE](#)
- [NULLIF](#)

EXTRACT function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the EXTRACT function to return the integer value of a specified date or time part from a given timestamp.

Syntax

```
EXTRACT ( datetimepart FROM timestamp )
```

Arguments

datetimepart

The date or time part that the function extracts. This parameter can be one of the following:

- year
- month
- day
- hour
- minute
- second
- timezone_hour
- timezone_minute

timestamp

The field name or expression of data type `timestamp` that the function extracts from. If this parameter is an unknown type (NULL or MISSING), the function returns NULL.

An Ion timestamp literal value can be denoted with backticks (``...``). For formatting details and examples of timestamp values, see [Timestamps](#) in the Amazon Ion specification document.

Return type

`int`

Examples

```
EXTRACT(YEAR FROM `2010-01-01T`)           -- 2010
EXTRACT(MONTH FROM `2010T`)                 -- 1 (equivalent to
2010-01-01T00:00:00.000Z)
EXTRACT(MONTH FROM `2010-10T`)             -- 10
EXTRACT(HOUR FROM `2017-01-02T03:04:05+07:08`) -- 3
EXTRACT(MINUTE FROM `2017-01-02T03:04:05+07:08`) -- 4
EXTRACT(TIMEZONE_HOUR FROM `2017-01-02T03:04:05+07:08`) -- 7
EXTRACT(TIMEZONE_MINUTE FROM `2017-01-02T03:04:05+07:08`) -- 8

-- Runnable statements
SELECT EXTRACT(YEAR FROM `2010-01-01T`) FROM << 0 >> -- 2010
SELECT EXTRACT(MONTH FROM `2010T`) FROM << 0 >>      -- 1
```

Related functions

- [DATE_ADD](#)
- [DATE_DIFF](#)
- [TO_STRING](#)
- [TO_TIMESTAMP](#)
- [UTCNOW](#)

LOWER function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the LOWER function to convert all uppercase characters to lowercase characters in a given string.

Syntax

```
LOWER ( string )
```

Arguments

string

The field name or expression of data type string that the function converts.

Return type

string

Examples

```
SELECT LOWER('AbCdEfG!@#') FROM << 0 >> -- 'abcdefg!@#'
```

Related functions

- [CHAR_LENGTH](#)
- [SUBSTRING](#)
- [TRIM](#)
- [UPPER](#)

MAX function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the MAX function to return the maximum value in a set of numeric values.

Syntax

```
MAX ( expression )
```

Arguments

expression

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- int
- decimal
- float

Supported return types:

- int
- decimal
- float

Examples

```
SELECT MAX(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r -- 442.30
SELECT MAX(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >> -- 3
```

Related functions

- [AVG](#)
- [COUNT](#)
- [MIN](#)
- [SIZE](#)
- [SUM](#)

MIN function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the MIN function to return the minimum value in a set of numeric values.

Syntax

```
MIN ( expression )
```

Arguments

expression

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- `int`
- `decimal`
- `float`

Supported return types:

- `int`
- `decimal`
- `float`

Examples

```
SELECT MIN(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r -- 30.45
SELECT MIN(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>          -- 1
```

Related functions

- [AVG](#)
- [COUNT](#)
- [MAX](#)
- [SIZE](#)
- [SUM](#)

NULLIF function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, given two expressions, use the `NULLIF` function to return `NULL` if the two expressions evaluate to the same value. Otherwise, this function returns the result of evaluating the first expression.

The NULLIF function doesn't propagate NULL and MISSING.

Syntax

```
NULLIF ( expression1, expression2 )
```

Arguments

expression1, *expression2*

The two field names or expressions that the function compares. These parameters can be any of the supported [Data types](#).

Return type

Any supported data type. The return type is either NULL or the same as the type of the first expression.

Examples

```
NULLIF(1, 1)           -- null
NULLIF(1, 2)           -- 1
NULLIF(1.0, 1)         -- null
NULLIF(1, '1')         -- 1
NULLIF([1], [1])       -- null
NULLIF(1, NULL)        -- 1
NULLIF(NULL, 1)        -- null
NULLIF(null, null)     -- null
NULLIF(missing, null)  -- null
NULLIF(missing, missing) -- null

-- Runnable statements
SELECT NULLIF(1, 1) FROM << 0 >>  -- null
SELECT NULLIF(1, '1') FROM << 0 >> -- 1
```

Related functions

- [COALESCE](#)
- [EXISTS](#)

SIZE function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the SIZE function to return the number of elements in a given container data type (list, structure, or bag).

Syntax

```
SIZE ( container )
```

Arguments

container

The container field name or expression that the function operates on.

Data types

Supported argument types:

- list
- structure
- bag

Return type: int

If the input to SIZE is not a container, the function throws an error.

Examples

```
SIZE(`[]`)          -- 0
```

```

SIZE(`[null]`)           -- 1
SIZE(`[1,2,3]`)          -- 3
SIZE(<<'foo', 'bar'>>)    -- 2
SIZE(`{foo: bar}`)       -- 1 (number of key-value pairs)
SIZE(`[{foo: 1}, {foo: 2}]`) -- 2
SIZE(12)                  -- error

-- Runnable statements
SELECT SIZE(`[]`) FROM << 0 >>      -- 0
SELECT SIZE(`[1,2,3]`) FROM << 0 >> -- 3

```

Related functions

- [AVG](#)
- [COUNT](#)
- [MAX](#)
- [MIN](#)
- [SUM](#)

SUBSTRING function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the SUBSTRING function to return a substring from a given string. The substring starts from the specified start index and ends at the last character of the string, or at the specified length.

Syntax

```
SUBSTRING ( string, start-index [, length ] )
```


Arguments

string

The field name or expression of data type `string` from which to extract a substring.

start-index

The start position within the *string* from which to begin the extraction. This number can be negative.

The first character of *string* has an index of 1.

length

(Optional) The number of characters (code points) to extract from the *string*, starting at *start-index* and ending at $(\text{start-index} + \text{length}) - 1$. In other words, the length of the substring. This number can't be negative.

If this parameter is not provided, the function proceeds until the end of the *string*.

Return type

`string`

Examples

```
SUBSTRING('123456789', 0)      -- '123456789'
SUBSTRING('123456789', 1)      -- '123456789'
SUBSTRING('123456789', 2)      -- '23456789'
SUBSTRING('123456789', -4)     -- '123456789'
SUBSTRING('123456789', 0, 999) -- '123456789'
SUBSTRING('123456789', 0, 2)   -- '1'
SUBSTRING('123456789', 1, 999) -- '123456789'
SUBSTRING('123456789', 1, 2)   -- '12'
SUBSTRING('1', 1, 0)           -- ''
SUBSTRING('1', 1, 0)           -- ''
SUBSTRING('1', -4, 0)          -- ''
SUBSTRING('1234', 10, 10)      -- ''

-- Runnable statements
SELECT SUBSTRING('123456789', 1) FROM << 0 >>    -- "123456789"
```

```
SELECT SUBSTRING('123456789', 1, 2) FROM << 0 >> -- "12"
```

Related functions

- [CHAR_LENGTH](#)
- [LOWER](#)
- [TRIM](#)
- [UPPER](#)

SUM function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the SUM function to return the sum of the input field or expression values. This function works with numeric values and ignores null or missing values.

Syntax

```
SUM ( expression )
```

Arguments

expression

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- int
- decimal

- `float`

Supported return types:

- `int` – For integer arguments
- `decimal` – For decimal or floating point arguments

Examples

```
SELECT SUM(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r -- 735.95
SELECT SUM(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >> -- 6
```

Related functions

- [AVG](#)
- [COUNT](#)
- [MAX](#)
- [MIN](#)
- [SIZE](#)

TO_STRING function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the `TO_STRING` function to return a string representation of a given timestamp in the specified format pattern.

Syntax

```
TO_STRING ( timestamp, 'format' )
```

Arguments

timestamp

The field name or expression of data type `timestamp` that the function converts to a string.

An Ion timestamp literal value can be denoted with backticks (``...``). For formatting details and examples of timestamp values, see [Timestamps](#) in the Amazon Ion specification document.

format

The string literal that specifies the format pattern of the result, in terms of its date parts. For valid formats, see [Timestamp format strings](#).

Return type

string

Examples

```
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y')           -- "July 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMM d, yyyy')        -- "Jul 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'M-d-yy')             -- "7-20-69"
TO_STRING(`1969-07-20T20:18Z`, 'MM-d-y')             -- "07-20-1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y h:m a')    -- "July 20, 1969 8:18
PM"
TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd'T'H:m:ssX')  --
"1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00Z`, 'y-MM-dd'T'H:m:ssX')  --
"1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd'T'H:m:ssXXXX') --
"1969-07-20T20:18:00+0800"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd'T'H:m:ssXXXXX') --
"1969-07-20T20:18:00+08:00"

-- Runnable statements
SELECT TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y') FROM << 0 >>      -- "July 20,
1969"
SELECT TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd'T'H:m:ssX') FROM << 0 >> --
"1969-07-20T20:18:00Z"
```

Related functions

- [CAST](#)
- [DATE_ADD](#)
- [DATE_DIFF](#)
- [EXTRACT](#)
- [TO_TIMESTAMP](#)
- [UTCNOW](#)

TO_TIMESTAMP function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, given a string that represents a timestamp, use the TO_TIMESTAMP function to convert the string to a timestamp data type. This is the inverse operation of TO_STRING.

Syntax

```
TO_TIMESTAMP ( string [, 'format' ] )
```

Arguments

string

The field name or expression of data type `string` that the function converts to a timestamp.

format

(Optional) The string literal that defines the format pattern of the input *string*, in terms of its date parts. For valid formats, see [Timestamp format strings](#).

If this argument is omitted, the function assumes that the *string* is in the format of a [standard lon timestamp](#). This is the recommended way to parse an lon timestamp using this function.

Zero padding is optional when using a single-character format symbol (such as y, M, d, H, h, m, s) but is required for their zero-padded variants (such as yyyy, MM, dd, HH, hh, mm, ss).

Special treatment is given to two-digit years (format symbol yy). 1900 is added to values greater than or equal to 70, and 2000 is added to values less than 70.

Month names and AM or PM indicators are case insensitive.

Return type

timestamp

Examples

```
TO_TIMESTAMP('2007T') -- `2007T`
TO_TIMESTAMP('2007-02-23T12:14:33.079-08:00') -- `2007-02-23T12:14:33.079-08:00`
TO_TIMESTAMP('2016', 'y') -- `2016T`
TO_TIMESTAMP('2016', 'yyyy') -- `2016T`
TO_TIMESTAMP('02-2016', 'MM-yyyy') -- `2016-02T`
TO_TIMESTAMP('Feb 2016', 'MMM yyyy') -- `2016-02T`
TO_TIMESTAMP('February 2016', 'MMMM yyyy') -- `2016-02T`

-- Runnable statements
SELECT TO_TIMESTAMP('2007T') FROM << 0 >> -- 2007T
SELECT TO_TIMESTAMP('02-2016', 'MM-yyyy') FROM << 0 >> -- 2016-02T
```

Related functions

- [CAST](#)
- [DATE_ADD](#)
- [DATE_DIFF](#)
- [EXTRACT](#)
- [TO_STRING](#)
- [UTCNOW](#)

TRIM function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the TRIM function to trim a given string by removing the leading and trailing blank spaces or a specified *set* of characters.

Syntax

```
TRIM ( [ LEADING | TRAILING | BOTH [ characters ] FROM ] string )
```

Arguments

LEADING

(Optional) Indicates that the blank spaces or specified characters are to be removed from the beginning of *string*. If not specified, the default behavior is BOTH.

TRAILING

(Optional) Indicates that the blank spaces or specified characters are to be removed from the end of *string*. If not specified, the default behavior is BOTH.

BOTH

(Optional) Indicates that both the leading and trailing blank spaces or specified characters are to be removed from the beginning and end of *string*.

characters

(Optional) The *set* of characters to remove, specified as a string.

If this parameter is not provided, blank spaces are removed.

string

The field name or expression of data type string that the functions trims.

Return type

string

Examples

```
TRIM('      foobar      ')          -- 'foobar'
TRIM('      \tfoobar\t      ')      -- '\tfoobar\t'
TRIM(LEADING FROM '      foobar      ') -- 'foobar'
TRIM(TRAILING FROM '      foobar      ') -- '      foobar'
TRIM(BOTH FROM '      foobar      ')  -- 'foobar'
TRIM(BOTH '1' FROM '11foobar11')      -- 'foobar'
TRIM(BOTH '12' FROM '1112211foobar22211122') -- 'foobar'

-- Runnable statements
SELECT TRIM('      foobar      ') FROM << 0 >>          -- "foobar"
SELECT TRIM(LEADING FROM '      foobar      ') FROM << 0 >> -- "foobar"
```

Related functions

- [CHAR_LENGTH](#)
- [LOWER](#)
- [SUBSTRING](#)
- [UPPER](#)

TXID function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the TXID function to return the unique transaction ID of the current statement that you're running. This is the value that is assigned to a document's txId metadata field when the current transaction is committed to the journal.

Syntax

```
TXID()
```

Arguments

None

Return type

string

Examples

```
SELECT TXID() FROM << 0 >> -- "L7S9iJqcn9W2M4q0En27ay"  
SELECT TXID() FROM Person WHERE GovId = 'LEWISR261LL' -- "BKeMb48PNyvHWJGZHkaodG"
```

UPPER function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the UPPER function to convert all lowercase characters to uppercase characters in a given string.

Syntax

```
UPPER ( string )
```

Arguments

string

The field name or expression of data type string that the function converts.

Return type

string

Examples

```
SELECT UPPER('AbCdEfG!@#') FROM << 0 >> -- 'ABCDEFG!@#'
```

Related functions

- [CHAR_LENGTH](#)
- [LOWER](#)
- [SUBSTRING](#)
- [TRIM](#)

UTCNOW function in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, use the UTCNOW function to return the current time in Coordinated Universal Time (UTC) as a timestamp.

Syntax

```
UTCNOW()
```

This function requires that you specify a FROM clause in a SELECT query.

Arguments

None

Return type

timestamp

Examples

```
SELECT UTCNOW() FROM << 0 >>  -- 2019-12-27T20:12:16.999Z
SELECT UTCNOW() FROM Person WHERE GovId = 'LEWISR261LL'  -- 2019-12-27T20:12:26.999Z

INSERT INTO Person VALUE { 'firstName': 'Jane', 'createdAt': UTCNOW() }
UPDATE Person p SET p.updatedAt = UTCNOW() WHERE p.firstName = 'John'
```

Related functions

- [DATE_ADD](#)
- [DATE_DIFF](#)
- [EXTRACT](#)
- [TO_STRING](#)
- [TO_TIMESTAMP](#)

Timestamp format strings

 **Important**

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section provides reference information for timestamp format strings.

Timestamp format strings apply to the TO_STRING and TO_TIMESTAMP functions. These strings can contain date part separators (such as '-', '/', or ':') and the following format symbols.

Format	Example	Description
yy	70	Two-digit year

Format	Example	Description
y	1970	Four-digit year
yyyy	1970	Zero-padded four-digit year
M	1	Month integer
MM	01	Zero-padded month integer
MMM	Jan	Abbreviated month name
MMMM	January	Full month name
d	2	Day of the month (1–31)
dd	02	Zero-padded day of the month (01–31)
a	AM or PM	Meridian indicator (for 12-hour clock)
h	3	Hour (12-hour clock, 1–12)
hh	03	Zero-padded hour (12-hour clock, 01–12)
H	3	Hour (24-hour clock, 0–23)
HH	03	Zero-padded hour (24-hour clock, 00–23)
m	4	Minutes (0–59)
mm	04	Zero-padded minutes (00–59)
s	5	Seconds (0–59)
ss	05	Zero-padded seconds (00–59)

Format	Example	Description
S	0	Fraction of a second (precision: 0.1, range: 0.0–0.9)
SS	06	Fraction of a second (precision: 0.01, range: 0.0–0.99)
SSS	060	Fraction of a second (precision: 0.001, range: 0.0–0.999)
X	+07 or Z	Offset from UTC in hours, or "Z" if the offset is 0
XX	+0700 or Z	Offset from UTC in hours and minutes, or "Z" if the offset is 0
XXX	+07:00 or Z	Offset from UTC in hours and minutes, or "Z" if the offset is 0
x	+07	Offset from UTC in hours
xx	+0700	Offset from UTC in hours and minutes
xxx	+07:00	Offset from UTC in hours and minutes

PartiQL stored procedures in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

In Amazon QLDB, you can use the EXEC command to run PartiQL stored procedures in the following syntax.

```
EXEC stored_procedure_name argument [, ... ]
```

QLDB supports the following system stored procedures only:

Topics

- [REDACT_REVISION stored procedure in Amazon QLDB](#)

REDACT_REVISION stored procedure in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Note

Any ledgers that were created before July 22, 2021 are currently not eligible for redaction. You can view the creation time of your ledger on the Amazon QLDB console.

In Amazon QLDB, use the REDACT_REVISION stored procedure to permanently delete an individual, inactive document revision in both indexed storage and journal storage. This stored procedure deletes all of the user data in the specified revision. However, it leaves the journal sequence and the document metadata, including the document ID and hash, unchanged. *This operation is irreversible.*

The specified document revision must be an inactive revision in history. The latest active revision of a document is not eligible for redaction.

After you submit a redaction request by running this stored procedure, QLDB processes the redaction of data asynchronously. After a redaction is complete, the user data of the specified revision (represented by the data structure) is replaced by a new dataHash field. The value of this field is the [Amazon Ion](#) hash of the removed data structure. As a result, the ledger maintains its

overall data integrity and remains cryptographically verifiable through the existing verification API operations.

For an example of a redaction operation with sample data, see [Redaction example](#) in [Redacting document revisions](#).

Note

To learn how to control access to run this PartiQL command on specific tables, see [Getting started with the standard permissions mode in Amazon QLDB](#).

Topics

- [Redaction considerations and limitations](#)
- [Syntax](#)
- [Arguments](#)
- [Return value](#)
- [Examples](#)

Redaction considerations and limitations

Before you get started with data redaction in Amazon QLDB, make sure that you review the following considerations and limitations:

- The REDACT_REVISION stored procedure targets your user data in an individual, inactive document revision. To redact multiple revisions, you must run the stored procedure once for each revision. You can redact one revision per transaction.
- To redact particular fields within a document revision, you must use a separate data manipulation language (DML) statement to modify the revision first. For more information, see [Redacting a particular field within a revision](#).
- After QLDB receives a redaction request, you can't cancel or alter the request. To confirm whether a redaction is complete, you can check if the data structure of a revision has been replaced by a dataHash field. To learn more, see [Checking whether a redaction is complete](#).
- Redaction has no impact on any QLDB data that is replicated outside of the QLDB service. This includes any exports to Amazon S3 and streams to Amazon Kinesis Data Streams. You must use other data retention methods to manage any data stored outside of QLDB.

- Redaction has no impact on literal values in PartiQL statements that are recorded in the journal. As a best practice, you should run parameterized statements programmatically by using variable placeholders instead of literal values. A placeholder is written in the journal as a question mark (?) instead of any sensitive information that might require redaction.

To learn how to programmatically run PartiQL statements using the QLDB driver, see the tutorials for each supported programming language in [Getting started with the driver](#).

Syntax

```
EXEC REDACT_REVISION `block-address`, 'table-id', 'document-id'
```

Arguments

block-address

The journal block location of the document revision to be redacted. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

This is an Ion literal value that is denoted by backticks. For example:

```
`{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`
```

To learn how to find the block address, see [Querying document metadata](#).

'table-id'

The unique ID of the table whose document revision you want to redact, denoted by single quotation marks.

To learn how to find the table ID, see [Querying the system catalog](#).

'document-id'

The unique document ID of the revision to be redacted, denoted by single quotation marks.

To learn how to find the document ID, see [Querying document metadata](#).

Return value

An Amazon Ion structure that represents the document revision to be redacted, in the following format.

```
{
  blockAddress: {
    strandId: String,
    sequenceNo: Int
  },
  tableId: String,
  documentId: String,
  version: Int
}
```

Return structure fields

- **blockAddress** – The journal block location of the revision to be redacted. An address has the following two fields.
 - **strandId** – The unique ID of the journal strand that contains the block.
 - **sequenceNo** – An index number that specifies the location of the block within the strand.
- **tableId** – The unique ID of the table whose revision you are redacting.
- **documentId** – The unique document ID of the revision to be redacted.
- **version** – The version number of the document revision to be redacted.

The following is an example of the return structure with sample data.

```
{
  blockAddress: {
    strandId: "CsRnx0RDoNK6ANEEePa1ov",
    sequenceNo: 134
  },
  tableId: "6GZumdHggk1LdMGyQq9DNX",
  documentId: "IXlQPSbfyKMIIsygePeKrZ",
  version: 0
}
```

Examples

```
EXEC REDACT_REVISION `{strandId:"7z2P0AyQKWD8oFYmGNhi8D", sequenceNo:7}` ,
'8F0TPCmdNQ6JTRpiLj2TmW', '05K8zpGYWynDlE0K5afDRc'
```

PartiQL operators in Amazon QLDB

 **Important**

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

PartiQL in Amazon QLDB supports the following [SQL standard operators](#).

 **Note**

Any SQL operators that aren't included in this list are not currently supported in QLDB.

Arithmetic operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Comparison operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Logical operators

Operator	Description
AND	TRUE if all the conditions separated by AND are TRUE
BETWEEN	TRUE if the operand is within the range of comparisons
IN	TRUE if the operand is equal to one of a list of expressions
IS	TRUE if the operand is a given data type, including NULL or MISSING
LIKE	TRUE if the operand matches a pattern
NOT	Reverses the value of a given Boolean expression

Operator	Description
OR	TRUE if any of the conditions separated by OR is TRUE

String operators

Operator	Description
	Concatenates two strings on either side of the operator and returns the concatenated string. If one or both strings is NULL, the result of the concatenation is null.

Reserved keywords in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following is a list of PartiQL reserved keywords in Amazon QLDB. You can use a reserved keyword as a quoted identifier with double quotation marks (for example, "user"). For information about PartiQL quoting conventions in QLDB, see [Querying Ion with PartiQL](#).

Important

The keywords in this list are all considered reserved because PartiQL is backwards compatible with [SQL-92](#). However, QLDB only supports a subset of these reserved words. For the list of SQL keywords that QLDB currently supports, see the following topics:

- [PartiQL functions](#)
- [PartiQL operators](#)

- [PartiQL commands](#)

ABSOLUTE
ACTION
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BAG
BEGIN
BETWEEN
BIT
BIT_LENGTH
BLOB
BOOL
BOOLEAN
BOTH
BY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHARACTER_LENGTH
CHAR_LENGTH
CHECK
CLOB
CLOSE
COALESCE
COLLATE
COLLATION

COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DATE
DATE_ADD
DATE_DIFF
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DISCONNECT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
END
END-EXEC
ESCAPE
EXCEPT

EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST

LEADING
LEFT
LEVEL
LIKE
LIMIT
LIST
LOCAL
LOWER
MATCH
MAX
MIN
MINUTE
MISSING
MODULE
MONTH
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NO
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
ON
ONLY
OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PAD
PARTIAL
PIVOT
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY

PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
RELATIVE
REMOVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SESSION
SESSION_USER
SET
SEXP
SIZE
SMALLINT
SOME
SPACE
SQL
SQLCODE
SQLERROR
SQLSTATE
STRING
STRUCT
SUBSTRING
SUM
SYMBOL
SYSTEM_USER
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE

TO
TO_STRING
TO_TIMESTAMP
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
TRIM
TRUE
TUPLE
TXID
UNDROP
UNION
UNIQUE
UNKNOWN
UNPIVOT
UPDATE
UPPER
USAGE
USER
USING
UTCNOW
VALUE
VALUES
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE

Amazon Ion data format reference in Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

Amazon QLDB uses a data notation model that unifies [Amazon Ion](#) with a subset of [PartiQL](#) types. This section provides a reference overview of the Ion document data format, separate from its integration with PartiQL.

Querying Ion with PartiQL in Amazon QLDB

For the syntax and semantics of querying Ion data with PartiQL in QLDB, see [Querying Ion with PartiQL](#) in the *Amazon QLDB PartiQL reference*.

For code examples that query and process Ion data in a QLDB ledger, see [Amazon Ion code examples](#) and [Working with Amazon Ion](#).

Topics

- [What is Amazon Ion?](#)
- [Ion specification](#)
- [JSON compatible](#)
- [Extensions from JSON](#)
- [Ion text example](#)
- [API references](#)
- [Amazon Ion code examples in QLDB](#)

What is Amazon Ion?

Ion is an open-source, richly typed, self-describing, hierarchical data serialization format that was originally developed internally at Amazon. It's based on an abstract data model that lets you store both structured and unstructured data. It's a superset of JSON, meaning that any valid JSON

document is also a valid Ion document. This guide assumes a baseline working knowledge of JSON. If you aren't already familiar with JSON, see [Introducing JSON](#) for more information.

You can notate Ion documents interchangeably in either human-readable text form or binary-encoded form. Like JSON, the text form is easy to read and write, supporting rapid prototyping. The binary encoding is more compact and efficient to persist, transmit, and parse. An Ion processor can transcode between both formats to represent exactly the same set of data structures without any loss of data. This feature lets applications optimize how they process data for different use cases.

Note

The Ion data model is strictly value-based and doesn't support references. Thus, the data model can represent data hierarchies that can be nested to arbitrary depth, but not directed graphs.

Ion specification

For a full list of Ion core data types with complete descriptions and value formatting details, see the [Ion specification document](#) on the Amazon GitHub site.

To streamline application development, Amazon Ion provides client libraries that process Ion data for you. For code examples of common use cases for processing Ion data, see the [Amazon Ion Cookbook](#) on GitHub.

JSON compatible

Similar to JSON, you compose Amazon Ion documents with a set of primitive data types and a set of recursively defined container types. Ion includes the following traditional JSON data types:

- `null`: A generic, untyped null (empty) value. Additionally, as described in the following section, Ion supports a distinct null type for each primitive type.
- `bool`: Boolean values.
- `string`: Unicode text literals.
- `list`: Ordered heterogeneous collections of values.
- `struct`: Unordered collections of name-value pairs. Like JSON, `struct` allows multiple values per name, but this is generally discouraged.

Extensions from JSON

Number types

Instead of the ambiguous JSON number type, Amazon Ion strictly defines numbers as one of the following types:

- `int`: Signed integers of arbitrary size.
- `decimal`: Decimal-encoded real numbers of arbitrary precision.
- `float`: Binary-encoded floating point numbers (64-bit IEEE).

When parsing documents, an Ion processor assigns number types as follows:

- `int`: Numbers with no exponent or decimal point (for example, `100200`).
- `decimal`: Numbers with a decimal point and no exponent (for example, `0.00001`, `200.0`).
- `float`: Numbers with an exponent, such as scientific notation or E-notation (for example, `2e0`, `3.1e-4`).

New data types

Amazon Ion adds the following data types:

- `timestamp`: Date/time/timezone moments of arbitrary precision.
- `symbol`: Unicode symbolic atoms (such as identifiers).
- `blob`: Binary data of user-defined encoding.
- `clob`: Text data of user-defined encoding.
- `sexp`: Ordered collections of values with application-defined semantics.

Null types

In addition to the generic null type defined by JSON, Amazon Ion supports a distinct null type for each primitive type. This indicates a lack of value while maintaining a strict data type.

```
null
null.null      // Identical to untyped null
```

```
null.bool
null.int
null.float
null.decimal
null.timestamp
null.string
null.symbol
null.blob
null.clob
null.struct
null.list
null.sexp
```

Ion text example

```
// Here is a struct, which is similar to a JSON object.
{
    // Field names don't always have to be quoted.
    name: "fido",

    // This is an integer.
    age: 7,

    // This is a timestamp with day precision.
    birthday: 2012-03-01T,

    // Here is a list, which is like a JSON array.
    toys: [
        // These are symbol values, which are like strings,
        // but get encoded as integers in binary.
        ball,
        rope
    ],
}
```

API references

- [ion-go](#)
- [ion-java](#)
- [ion-js](#)
- [ion-python](#)

Amazon Ion code examples in QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section provides code examples that process Amazon Ion data by reading and writing document values in an Amazon QLDB ledger. The code examples use the QLDB driver to run PartiQL statements on the ledger. These examples are part of the sample application in [Getting started with Amazon QLDB using a sample application tutorial](#), and are open source on the [AWS Samples GitHub site](#).

For general code examples that show common use cases of processing Ion data, see the [Amazon Ion Cookbook](#) on GitHub.

Running the code

The tutorial code for each programming language does the following steps:

1. Connect to the vehicle-registration sample ledger.
2. Create a table named IonTypes.
3. Insert a document into the table with a single Name field.
4. For each supported [Ion data type](#):
 - a. Update the document's Name field with a literal value of the data type.
 - b. Query the table to get the latest revision of the document.
 - c. Validate that the value of Name retained its original data type properties by checking that it matches the expected type.
5. Drop the IonTypes table.

Note

Before you run this tutorial code, you must create a ledger named vehicle-registration.

Java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonBlob;
import com.amazon.ion.IonBool;
import com.amazon.ion.IonClob;
import com.amazon.ion.IonDecimal;
import com.amazon.ion.IonFloat;
import com.amazon.ion.IonInt;
import com.amazon.ion.IonList;
import com.amazon.ion.IonNull;
import com.amazon.ion.IonSexp;
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSymbol;
import com.amazon.ion.IonTimestamp;
import com.amazon.ion.IonValue;
import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.annotation.JsonCreator;
```



```
import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.Collections;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;

/**
 * Insert all the supported Ion types into a ledger and verify that they are stored
 * and can be retrieved properly, retaining
 * their original properties.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public class InsertIonTypes {
    public static final Logger log = LoggerFactory.getLogger(InsertIonTypes.class);
    public static final String TABLE_NAME = "IonTypes";

    private InsertIonTypes() {}

    /**
     * Update a document's Name value in the database. Then, query the value of the
     * Name key and verify the expected Ion type was
     * saved.
     *
     * @param txn
     *           The {@link TransactionExecutor} for statement execution.
     * @param ionValue
     *           The {@link IonValue} to set the document's Name value to.
     *
     * @throws AssertionError when no value is returned for the Name key or if the
     * value does not match the expected type.
     */
    public static void updateRecordAndVerifyType(final TransactionExecutor txn,
        final IonValue ionValue) {
        final String updateStatement = String.format("UPDATE %s SET Name = ?",
            TABLE_NAME);
        final List<IonValue> parameters = Collections.singletonList(ionValue);
        txn.execute(updateStatement, parameters);
        log.info("Updated document.");
    }
}
```

```

        final String searchQuery = String.format("SELECT VALUE Name FROM %s",
TABLE_NAME);
        final Result result = txn.execute(searchQuery);

        if (result.isEmpty()) {
            throw new AssertionError("Did not find any values for the Name key.");
        }
        for (IonValue value : result) {
            if (!ionValue.getClass().isInstance(value)) {
                throw new AssertionError(String.format("The queried value, %s, is
not an instance of %s.",
                    value.getClass().toString(),
ionValue.getClass().toString()));
            }
            if (!value.getType().equals(ionValue.getType())) {
                throw new AssertionError(String.format("The queried value type, %s,
does not match %s.",
                    value.getType().toString(), ionValue.getType().toString()));
            }
        }

        log.info("Successfully verified value is instance of {} with type {}.",
ionValue.getClass().toString(),
            ionValue.getType().toString());
    }

    /**
     * Delete a table.
     *
     * @param txn
     *         The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *         The name of the table to delete.
     */
    public static void deleteTable(final TransactionExecutor txn, final String
tableName) {
        log.info("Deleting {} table...", tableName);
        final String statement = String.format("DROP TABLE %s", tableName);
        txn.execute(statement);
        log.info("{} table successfully deleted.", tableName);
    }

    public static void main(final String... args) {
        final IonBlob ionBlob = Constants.SYSTEM.newBlob("hello".getBytes());

```

```

        final IonBool ionBool = Constants.SYSTEM.newBool(true);
        final IonClob ionClob = Constants.SYSTEM.newClob("{}{'This is a CLOB of
text.'}}".getBytes());
        final IonDecimal ionDecimal = Constants.SYSTEM.newDecimal(0.1);
        final IonFloat ionFloat = Constants.SYSTEM.newFloat(0.2);
        final IonInt ionInt = Constants.SYSTEM.newInt(1);
        final IonList ionList = Constants.SYSTEM.newList(new int[]{1, 2});
        final IonNull ionNull = Constants.SYSTEM.newNull();
        final IonSexp ionSexp = Constants.SYSTEM.newSexp(new int[]{2, 3});
        final IonString ionString = Constants.SYSTEM.newString("string");
        final IonStruct ionStruct = Constants.SYSTEM.newEmptyStruct();
        ionStruct.put("brand", Constants.SYSTEM.newString("ford"));
        final IonSymbol ionSymbol = Constants.SYSTEM.newSymbol("abc");
        final IonTimestamp ionTimestamp =
Constants.SYSTEM.newTimestamp(Timestamp.now());

        final IonBlob ionNullBlob = Constants.SYSTEM.newNullBlob();
        final IonBool ionNullBool = Constants.SYSTEM.newNullBool();
        final IonClob ionNullClob = Constants.SYSTEM.newNullClob();
        final IonDecimal ionNullDecimal = Constants.SYSTEM.newNullDecimal();
        final IonFloat ionNullFloat = Constants.SYSTEM.newNullFloat();
        final IonInt ionNullInt = Constants.SYSTEM.newNullInt();
        final IonList ionNullList = Constants.SYSTEM.newNullList();
        final IonSexp ionNullSexp = Constants.SYSTEM.newNullSexp();
        final IonString ionNullString = Constants.SYSTEM.newNullString();
        final IonStruct ionNullStruct = Constants.SYSTEM.newNullStruct();
        final IonSymbol ionNullSymbol = Constants.SYSTEM.newNullSymbol();
        final IonTimestamp ionNullTimestamp = Constants.SYSTEM.newNullTimestamp();

        ConnectToLedger.getDriver().execute(txn -> {
            CreateTable.createTable(txn, TABLE_NAME);
            final Document document = new
Document(Constants.SYSTEM.newString("val"));
            InsertDocument.insertDocuments(txn, TABLE_NAME,
Collections.singletonList(document));

            updateRecordAndVerifyType(txn, ionBlob);
            updateRecordAndVerifyType(txn, ionBool);
            updateRecordAndVerifyType(txn, ionClob);
            updateRecordAndVerifyType(txn, ionDecimal);
            updateRecordAndVerifyType(txn, ionFloat);
            updateRecordAndVerifyType(txn, ionInt);
            updateRecordAndVerifyType(txn, ionList);

```

```

        updateRecordAndVerifyType(txn, ionNull);
        updateRecordAndVerifyType(txn, ionSexp);
        updateRecordAndVerifyType(txn, ionString);
        updateRecordAndVerifyType(txn, ionStruct);
        updateRecordAndVerifyType(txn, ionSymbol);
        updateRecordAndVerifyType(txn, ionTimestamp);

        updateRecordAndVerifyType(txn, ionNullBlob);
        updateRecordAndVerifyType(txn, ionNullBool);
        updateRecordAndVerifyType(txn, ionNullClob);
        updateRecordAndVerifyType(txn, ionNullDecimal);
        updateRecordAndVerifyType(txn, ionNullFloat);
        updateRecordAndVerifyType(txn, ionNullInt);
        updateRecordAndVerifyType(txn, ionNullList);
        updateRecordAndVerifyType(txn, ionNullSexp);
        updateRecordAndVerifyType(txn, ionNullString);
        updateRecordAndVerifyType(txn, ionNullStruct);
        updateRecordAndVerifyType(txn, ionNullSymbol);
        updateRecordAndVerifyType(txn, ionNullTimestamp);

        deleteTable(txn, TABLE_NAME);
    });
}

/**
 * This class represents a simple document with a single key, Name, to use for
the IonTypes table.
 */
private static class Document {
    private final IonValue name;

    @JsonCreator
    private Document(@JsonProperty("Name") final IonValue name) {
        this.name = name;
    }

    @JsonProperty("Name")
    private IonValue getName() {
        return name;
    }
}
}

```

Node.js

```

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this
 * software and associated documentation files (the "Software"), to deal in the
 * Software
 * without restriction, including without limitation the rights to use, copy,
 * modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 * to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 * COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qlldb-driver-nodejs";
import { AssertionError } from "assert";
import { dom, IonType, IonTypes } from "ion-js";

import { insertDocument } from "./InsertDocument";
import { getQldbDriver } from "./ConnectToLedger";
import { createTable } from "./CreateTable";
import { error, log } from "./qlldb/LogUtil";

const TABLE_NAME: string = "IonTypes";

/**
 * Delete a table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to delete.
 * @returns Promise which fulfills with void.
 */

```

```

export async function deleteTable(txn: TransactionExecutor, tableName: string):
  Promise<void> {
    log(`Deleting ${tableName} table...`);
    const statement: string = `DROP TABLE ${tableName}`;
    await txn.execute(statement);
    log(`${tableName} table successfully deleted.`);
  }

/**
 * Update a document's Name value in QLDB. Then, query the value of the Name key and
 * verify the expected Ion type was
 * saved.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param parameter The IonValue to set the document's Name value to.
 * @param ionType The Ion type that the Name value should be.
 * @returns Promise which fulfills with void.
 */
async function updateRecordAndVerifyType(
  txn: TransactionExecutor,
  parameter: any,
  ionType: IonType
): Promise<void> {
  const updateStatement: string = `UPDATE ${TABLE_NAME} SET Name = ?`;
  await txn.execute(updateStatement, parameter);
  log("Updated record.");

  const searchStatement: string = `SELECT VALUE Name FROM ${TABLE_NAME}`;
  const result: Result = await txn.execute(searchStatement);

  const results: dom.Value[] = result.getResultList();

  if (0 === results.length) {
    throw new AssertionError({
      message: "Did not find any values for the Name key."
    });
  }

  results.forEach((value: dom.Value) => {
    if (value.getType().binaryTypeId !== ionType.binaryTypeId) {
      throw new AssertionError({
        message: `The queried value type, ${value.getType().name}, does not
match expected type, ${ionType.name}.`
      });
    }
  })
}

```

```

    });

    log(`Successfully verified value is of type ${ionType.name}.`);
}

/**
 * Insert all the supported Ion types into a table and verify that they are stored
 * and can be retrieved properly,
 * retaining their original properties.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qlldbDriver: QldbDriver = getQldbDriver();
        await qlldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await createTable(txn, TABLE_NAME);
            await insertDocument(txn, TABLE_NAME, [{ "Name": "val" }]);
            await updateRecordAndVerifyType(txn, dom.load("null"), IonTypes.NULL);
            await updateRecordAndVerifyType(txn, true, IonTypes.BOOL);
            await updateRecordAndVerifyType(txn, 1, IonTypes.INT);
            await updateRecordAndVerifyType(txn, 3.2, IonTypes.FLOAT);
            await updateRecordAndVerifyType(txn, dom.load("5.5"), IonTypes.DECIMAL);
            await updateRecordAndVerifyType(txn, dom.load("2020-02-02"),
IonTypes.TIMESTAMP);
            await updateRecordAndVerifyType(txn, dom.load("abc123"),
IonTypes.SYMBOL);
            await updateRecordAndVerifyType(txn, dom.load("\"string\""),
IonTypes.STRING);
            await updateRecordAndVerifyType(txn, dom.load("{ \"clob\" }"),
IonTypes.CLOB);
            await updateRecordAndVerifyType(txn, dom.load("{ blob }"),
IonTypes.BLOB);
            await updateRecordAndVerifyType(txn, dom.load("(1 2 3)"),
IonTypes.SEXP);
            await updateRecordAndVerifyType(txn, dom.load("[1, 2, 3]"),
IonTypes.LIST);
            await updateRecordAndVerifyType(txn, dom.load("{brand: ford}"),
IonTypes.STRUCT);
            await deleteTable(txn, TABLE_NAME);
        });
    } catch (e) {
        error(`Error updating and validating Ion types: ${e}`);
    }
}

```

```

if (require.main === module) {
    main();
}

```

Python

```

# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
# this
# software and associated documentation files (the "Software"), to deal in the
# Software
# without restriction, including without limitation the rights to use, copy, modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime
from decimal import Decimal
from logging import basicConfig, getLogger, INFO

from amazon.ion.simple_types import IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict,
    IonPyFloat, IonPyInt, IonPyList, \
        IonPyNull, IonPySymbol, IonPyText, IonPyTimestamp
from amazon.ion.simpleion import loads
from amazon.ion.symbols import SymbolToken
from amazon.ion.core import IonType

from pyqldb.samples.create_table import create_table
from pyqldb.samples.constants import Constants
from pyqldb.samples.insert_document import insert_documents
from pyqldb.samples.model.sample_data import convert_object_to_ion

```



```

from pyqldb.samples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

TABLE_NAME = 'IonTypes'

def update_record_and_verify_type(driver, parameter, ion_object, ion_type):
    """
    Update a record in the database table. Then query the value of the record and
    verify correct ion type saved.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to Ion
    for filling in parameters of the
        statement.

    :type
    ion_object: :py:obj:`IonPyBool`/:py:obj:`IonPyBytes`/:py:obj:`IonPyDecimal`/:py:obj:`IonPyD
    /:py:obj:`IonPyFloat`/:py:obj:`IonPyInt`/:py:obj:`IonPyList`/:py:obj:`IonPyNull`
    /:py:obj:`IonPySymbol`/:py:obj:`IonPyText`/:py:obj:`IonPyTimestamp`
    :param ion_object: The Ion object to verify against.

    :type ion_type: :py:class:`amazon.ion.core.IonType`
    :param ion_type: The Ion type to verify against.

    :raises TypeError: When queried value is not an instance of Ion type.
    """
    update_query = 'UPDATE {} SET Name = ?'.format(TABLE_NAME)
    driver.execute_lambda(lambda executor: executor.execute_statement(update_query,
parameter))
    logger.info('Updated record.')

    search_query = 'SELECT VALUE Name FROM {}'.format(TABLE_NAME)
    cursor = driver.execute_lambda(lambda executor:
executor.execute_statement(search_query))

    for c in cursor:

```

```

        if not isinstance(c, ion_object):
            raise TypeError('The queried value is not an instance of
{}'.format(ion_object.__name__))

        if c.ion_type is not ion_type:
            raise TypeError('The queried value type does not match
{}'.format(ion_type))

        logger.info("Successfully verified value is instance of '{}' with type
'{}'.format(ion_object.__name__, ion_type))
        return cursor

def delete_table(driver, table_name):
    """
    Delete a table.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to delete.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Deleting '{}' table...".format(table_name))
    cursor = driver.execute_lambda(lambda executor: executor.execute_statement('DROP
TABLE {}'.format(table_name)))
    logger.info("'{}' table successfully deleted.".format(table_name))
    return len(list(cursor))

def insert_and_verify_ion_types(driver):
    """
    Insert all the supported Ion types and Python values that are convertible to Ion
into a ledger and verify that they
are stored and can be retrieved properly, retaining their original properties.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: A QLDB Driver object.
    """
    python_bytes = str.encode('hello')
    python_bool = True

```

```

python_float = float('0.2')
python_decimal = Decimal('0.1')
python_string = "string"
python_int = 1
python_null = None
python_datetime = datetime(2016, 12, 20, 5, 23, 43)
python_list = [1, 2]
python_dict = {"brand": "Ford"}

ion_clob = convert_object_to_ion(loads('{{"This is a CLOB of text."}}'))
ion_blob = convert_object_to_ion(python_bytes)
ion_bool = convert_object_to_ion(python_bool)
ion_decimal = convert_object_to_ion(python_decimal)
ion_float = convert_object_to_ion(python_float)
ion_int = convert_object_to_ion(python_int)
ion_list = convert_object_to_ion(python_list)
ion_null = convert_object_to_ion(python_null)
ion_sexp = convert_object_to_ion(loads('(cons 1 2)'))
ion_string = convert_object_to_ion(python_string)
ion_struct = convert_object_to_ion(python_dict)
ion_symbol = convert_object_to_ion(SymbolToken(text='abc', sid=123))
ion_timestamp = convert_object_to_ion(python_datetime)

ion_null_clob = convert_object_to_ion(loads('null.clob'))
ion_null_blob = convert_object_to_ion(loads('null.blob'))
ion_null_bool = convert_object_to_ion(loads('null.bool'))
ion_null_decimal = convert_object_to_ion(loads('null.decimal'))
ion_null_float = convert_object_to_ion(loads('null.float'))
ion_null_int = convert_object_to_ion(loads('null.int'))
ion_null_list = convert_object_to_ion(loads('null.list'))
ion_null_sexp = convert_object_to_ion(loads('null.sexp'))
ion_null_string = convert_object_to_ion(loads('null.string'))
ion_null_struct = convert_object_to_ion(loads('null.struct'))
ion_null_symbol = convert_object_to_ion(loads('null.symbol'))
ion_null_timestamp = convert_object_to_ion(loads('null.timestamp'))

create_table(driver, TABLE_NAME)
insert_documents(driver, TABLE_NAME, [{'Name': 'val'}])
update_record_and_verify_type(driver, python_bytes, IonPyBytes, IonType.BLOB)
update_record_and_verify_type(driver, python_bool, IonPyBool, IonType.BOOL)
update_record_and_verify_type(driver, python_float, IonPyFloat, IonType.FLOAT)
update_record_and_verify_type(driver, python_decimal, IonPyDecimal,
IonType.DECIMAL)
update_record_and_verify_type(driver, python_string, IonPyText, IonType.STRING)

```

```

    update_record_and_verify_type(driver, python_int, IonPyInt, IonType.INT)
    update_record_and_verify_type(driver, python_null, IonPyNull, IonType.NULL)
    update_record_and_verify_type(driver, python_datetime, IonPyTimestamp,
IonType.TIMESTAMP)
    update_record_and_verify_type(driver, python_list, IonPyList, IonType.LIST)
    update_record_and_verify_type(driver, python_dict, IonPyDict, IonType.STRUCT)
    update_record_and_verify_type(driver, ion_clob, IonPyBytes, IonType.CLOB)
    update_record_and_verify_type(driver, ion_blob, IonPyBytes, IonType.BLOB)
    update_record_and_verify_type(driver, ion_bool, IonPyBool, IonType.BOOL)
    update_record_and_verify_type(driver, ion_decimal, IonPyDecimal,
IonType.DECIMAL)
    update_record_and_verify_type(driver, ion_float, IonPyFloat, IonType.FLOAT)
    update_record_and_verify_type(driver, ion_int, IonPyInt, IonType.INT)
    update_record_and_verify_type(driver, ion_list, IonPyList, IonType.LIST)
    update_record_and_verify_type(driver, ion_null, IonPyNull, IonType.NULL)
    update_record_and_verify_type(driver, ion_sexp, IonPyList, IonType.SEXP)
    update_record_and_verify_type(driver, ion_string, IonPyText, IonType.STRING)
    update_record_and_verify_type(driver, ion_struct, IonPyDict, IonType.STRUCT)
    update_record_and_verify_type(driver, ion_symbol, IonPySymbol, IonType.SYMBOL)
    update_record_and_verify_type(driver, ion_timestamp, IonPyTimestamp,
IonType.TIMESTAMP)
    update_record_and_verify_type(driver, ion_null_clob, IonPyNull, IonType.CLOB)
    update_record_and_verify_type(driver, ion_null_blob, IonPyNull, IonType.BLOB)
    update_record_and_verify_type(driver, ion_null_bool, IonPyNull, IonType.BOOL)
    update_record_and_verify_type(driver, ion_null_decimal, IonPyNull,
IonType.DECIMAL)
    update_record_and_verify_type(driver, ion_null_float, IonPyNull, IonType.FLOAT)
    update_record_and_verify_type(driver, ion_null_int, IonPyNull, IonType.INT)
    update_record_and_verify_type(driver, ion_null_list, IonPyNull, IonType.LIST)
    update_record_and_verify_type(driver, ion_null_sexp, IonPyNull, IonType.SEXP)
    update_record_and_verify_type(driver, ion_null_string, IonPyNull,
IonType.STRING)
    update_record_and_verify_type(driver, ion_null_struct, IonPyNull,
IonType.STRUCT)
    update_record_and_verify_type(driver, ion_null_symbol, IonPyNull,
IonType.SYMBOL)
    update_record_and_verify_type(driver, ion_null_timestamp, IonPyNull,
IonType.TIMESTAMP)
    delete_table(driver, TABLE_NAME)

def main(ledger_name=Constants.LEDGER_NAME):
    """

```

```
    Insert all the supported Ion types and Python values that are convertible to Ion
    into a ledger and verify that they
    are stored and can be retrieved properly, retaining their original properties.
    """
```

```
    try:
```

```
        with create_qldb_driver(ledger_name) as driver:
```

```
            insert_and_verify_ion_types(driver)
```

```
    except Exception as e:
```

```
        logger.exception('Error updating and validating Ion types.')
```

```
        raise e
```

```
if __name__ == '__main__':
```

```
    main()
```

Amazon QLDB API reference

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This chapter describes the low-level API operations for Amazon QLDB that are accessible via HTTP, the AWS Command Line Interface (AWS CLI), or an AWS SDK:

- *Amazon QLDB* – The QLDB resource management API (also known as the *control plane*). This API is used only for managing ledger resources and for non-transactional data operations. You can use these operations to create, delete, describe, list, and update ledgers. You can also verify journal data cryptographically, and export or stream journal blocks.
- *Amazon QLDB Session* – The QLDB transactional data API. You can use this API to run data transactions on a ledger with [PartiQL](#) statements.

Important

Instead of interacting directly with the *QLDB Session* API, we recommend using the QLDB driver or the QLDB shell to run data transactions on a ledger.

- If you're working with an AWS SDK, use the QLDB driver. The driver provides a high-level abstraction layer above the *QLDB Session* data API and manages the `SendCommand` operation for you. For information and a list of supported programming languages, see [Getting started with the driver](#).
- If you're working with the AWS CLI, use the QLDB shell. The shell is a command line interface that uses the QLDB driver to interact with a ledger. For information, see [Using the Amazon QLDB shell \(data API only\)](#).

Topics

- [Actions](#)
- [Data Types](#)

- [Common Errors](#)
- [Common Parameters](#)

Actions

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following actions are supported by Amazon QLDB:

- [CancelJournalKinesisStream](#)
- [CreateLedger](#)
- [DeleteLedger](#)
- [DescribeJournalKinesisStream](#)
- [DescribeJournalS3Export](#)
- [DescribeLedger](#)
- [ExportJournalToS3](#)
- [GetBlock](#)
- [GetDigest](#)
- [GetRevision](#)
- [ListJournalKinesisStreamsForLedger](#)
- [ListJournalS3Exports](#)
- [ListJournalS3ExportsForLedger](#)
- [ListLedgers](#)
- [ListTagsForResource](#)
- [StreamJournalToKinesis](#)
- [TagResource](#)
- [UntagResource](#)
- [UpdateLedger](#)

- [UpdateLedgerPermissionsMode](#)

The following actions are supported by Amazon QLDB Session:

- [SendCommand](#)

Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following actions are supported by Amazon QLDB:

- [CancelJournalKinesisStream](#)
- [CreateLedger](#)
- [DeleteLedger](#)
- [DescribeJournalKinesisStream](#)
- [DescribeJournalS3Export](#)
- [DescribeLedger](#)
- [ExportJournalToS3](#)
- [GetBlock](#)
- [GetDigest](#)
- [GetRevision](#)
- [ListJournalKinesisStreamsForLedger](#)
- [ListJournalS3Exports](#)
- [ListJournalS3ExportsForLedger](#)
- [ListLedgers](#)
- [ListTagsForResource](#)
- [StreamJournalToKinesis](#)

- [TagResource](#)
- [UntagResource](#)
- [UpdateLedger](#)
- [UpdateLedgerPermissionsMode](#)

CancelJournalKinesisStream

Service: Amazon QLDB

Ends a given Amazon QLDB journal stream. Before a stream can be canceled, its current status must be ACTIVE.

You can't restart a stream after you cancel it. Canceled QLDB stream resources are subject to a 7-day retention period, so they are automatically deleted after this limit expires.

Request Syntax

```
DELETE /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

streamId

The UUID (represented in Base62-encoded text) of the QLDB journal stream to be canceled.

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

```
Content-type: application/json

{
  "StreamId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

StreamId

The UUID (Base62-encoded text) of the canceled QLDB journal stream.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateLedger

Service: Amazon QLDB

Creates a new ledger in your AWS account in the current Region.

Request Syntax

```
POST /ledgers HTTP/1.1
Content-type: application/json

{
  "DeletionProtection": boolean,
  "KmsKey": "string",
  "Name": "string",
  "PermissionsMode": "string",
  "Tags": {
    "string" : "string"
  }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

DeletionProtection

Specifies whether the ledger is protected from being deleted by any user. If not defined during ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger. You can disable it by calling the `UpdateLedger` operation to set this parameter to `false`.

Type: Boolean

Required: No

KmsKey

The key in AWS Key Management Service (AWS KMS) to use for encryption of data at rest in the ledger. For more information, see [Encryption at rest](#) in the *Amazon QLDB Developer Guide*.

Use one of the following options to specify this parameter:

- **AWS_OWNED_KMS_KEY:** Use an AWS KMS key that is owned and managed by AWS on your behalf.
- **Undefined:** By default, use an AWS owned KMS key.
- **A valid symmetric customer managed KMS key:** Use the specified symmetric encryption KMS key in your account that you create, own, and manage.

Amazon QLDB does not support asymmetric keys. For more information, see [Using symmetric and asymmetric keys](#) in the *AWS Key Management Service Developer Guide*.

To specify a customer managed KMS key, you can use its key ID, Amazon Resource Name (ARN), alias name, or alias ARN. When using an alias name, prefix it with "alias/". To specify a key in a different AWS account, you must use the key ARN or alias ARN.

For example:

- Key ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- Key ARN: arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- Alias name: alias/ExampleAlias
- Alias ARN: arn:aws:kms:us-east-2:111122223333:alias/ExampleAlias

For more information, see [Key identifiers \(KeyId\)](#) in the *AWS Key Management Service Developer Guide*.

Type: String

Length Constraints: Maximum length of 1600.

Required: No

Name

The name of the ledger that you want to create. The name must be unique among all of the ledgers in your AWS account in the current Region.

Naming constraints for ledger names are defined in [Quotas in Amazon QLDB](#) in the *Amazon QLDB Developer Guide*.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^--)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

PermissionsMode

The permissions mode to assign to the ledger that you want to create. This parameter can have one of the following values:

- **ALLOW_ALL**: A legacy permissions mode that enables access control with API-level granularity for ledgers.

This mode allows users who have the `SendCommand` API permission for this ledger to run all PartiQL commands (hence, **ALLOW_ALL**) on any tables in the specified ledger. This mode disregards any table-level or command-level IAM permissions policies that you create for the ledger.

- **STANDARD**: (*Recommended*) A permissions mode that enables access control with finer granularity for ledgers, tables, and PartiQL commands.

By default, this mode denies all user requests to run any PartiQL commands on any tables in this ledger. To allow PartiQL commands to run, you must create IAM permissions policies for specific table resources and PartiQL actions, in addition to the `SendCommand` API permission for the ledger. For information, see [Getting started with the standard permissions mode](#) in the *Amazon QLDB Developer Guide*.

Note

We strongly recommend using the **STANDARD** permissions mode to maximize the security of your ledger data.

Type: String

Valid Values: `ALLOW_ALL` | `STANDARD`

Required: Yes

Tags

The key-value pairs to add as tags to the ledger that you want to create. Tag keys are case sensitive. Tag values are case sensitive and can be null.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Arn": "string",
  "CreationDateTime": number,
  "DeletionProtection": boolean,
  "KmsKeyArn": "string",
  "Name": "string",
  "PermissionsMode": "string",
  "State": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Arn](#)

The Amazon Resource Name (ARN) for the ledger.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

[CreationDateTime](#)

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

DeletionProtection

Specifies whether the ledger is protected from being deleted by any user. If not defined during ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger. You can disable it by calling the `UpdateLedger` operation to set this parameter to `false`.

Type: Boolean

KmsKeyArn

The ARN of the customer managed KMS key that the ledger uses for encryption at rest. If this parameter is undefined, the ledger uses an AWS owned KMS key for encryption.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Name

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

The permissions mode of the ledger that you created.

Type: String

Valid Values: `ALLOW_ALL` | `STANDARD`

State

The current status of the ledger.

Type: String

Valid Values: `CREATING` | `ACTIVE` | `DELETING` | `DELETED`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

LimitExceededException

You have reached the limit on the maximum number of resources allowed.

HTTP Status Code: 400

ResourceAlreadyExistsException

The specified resource already exists.

HTTP Status Code: 409

ResourceInUseException

The specified resource can't be modified at this time.

HTTP Status Code: 409

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)

- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteLedger

Service: Amazon QLDB

Deletes a ledger and all of its contents. This action is irreversible.

If deletion protection is enabled, you must first disable it before you can delete the ledger. You can disable it by calling the `UpdateLedger` operation to set this parameter to `false`.

Request Syntax

```
DELETE /ledgers/name HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger that you want to delete.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceInUseException

The specified resource can't be modified at this time.

HTTP Status Code: 409

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DescribeJournalKinesisStream

Service: Amazon QLDB

Returns detailed information about a given Amazon QLDB journal stream. The output includes the Amazon Resource Name (ARN), stream name, current status, creation time, and the parameters of the original stream creation request.

This action does not return any expired journal streams. For more information, see [Expiration for terminal streams](#) in the *Amazon QLDB Developer Guide*.

Request Syntax

```
GET /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

streamId

The UUID (represented in Base62-encoded text) of the QLDB journal stream to describe.

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Stream": {
    "Arn": "string",
    "CreationTime": number,
    "ErrorCause": "string",
    "ExclusiveEndTime": number,
    "InclusiveStartTime": number,
    "KinesisConfiguration": {
      "AggregationEnabled": boolean,
      "StreamArn": "string"
    },
    "LedgerName": "string",
    "RoleArn": "string",
    "Status": "string",
    "StreamId": "string",
    "StreamName": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Stream

Information about the QLDB journal stream returned by a `DescribeJournalS3Export` request.

Type: [JournalKinesisStreamDescription](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DescribeJournalS3Export

Service: Amazon QLDB

Returns information about a journal export job, including the ledger name, export ID, creation time, current status, and the parameters of the original export creation request.

This action does not return any expired export jobs. For more information, see [Export job expiration](#) in the *Amazon QLDB Developer Guide*.

If the export job with the given `ExportId` doesn't exist, then throws `ResourceNotFoundException`.

If the ledger with the given `Name` doesn't exist, then throws `ResourceNotFoundException`.

Request Syntax

```
GET /ledgers/name/journal-s3-exports/exportId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

exportId

The UUID (represented in Base62-encoded text) of the journal export job to describe.

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z0-9]+$`

Required: Yes

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "ExportDescription": {
    "ExclusiveEndTime": number,
    "ExportCreationTime": number,
    "ExportId": "string",
    "InclusiveStartTime": number,
    "LedgerName": "string",
    "OutputFormat": "string",
    "RoleArn": "string",
    "S3ExportConfiguration": {
      "Bucket": "string",
      "EncryptionConfiguration": {
        "KmsKeyArn": "string",
        "ObjectEncryptionType": "string"
      },
      "Prefix": "string"
    },
    "Status": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ExportDescription

Information about the journal export job returned by a DescribeJournalS3Export request.

Type: [JournalS3ExportDescription](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DescribeLedger

Service: Amazon QLDB

Returns information about a ledger, including its state, permissions mode, encryption at rest settings, and when it was created.

Request Syntax

```
GET /ledgers/name HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger that you want to describe.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Arn": "string",
  "CreationDateTime": number,
  "DeletionProtection": boolean,
  "EncryptionDescription": {
    "EncryptionStatus": "string",
    "InaccessibleKmsKeyDateTime": number,
    "KmsKeyArn": "string"
  },
  "Name": "string",
```

```
"PermissionsMode": "string",  
"State": "string"  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Arn

The Amazon Resource Name (ARN) for the ledger.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

CreationDateTime

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

DeletionProtection

Specifies whether the ledger is protected from being deleted by any user. If not defined during ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger. You can disable it by calling the `UpdateLedger` operation to set this parameter to `false`.

Type: Boolean

EncryptionDescription

Information about the encryption of data at rest in the ledger. This includes the current status, the AWS KMS key, and when the key became inaccessible (in the case of an error). If this parameter is undefined, the ledger uses an AWS owned KMS key for encryption.

Type: [LedgerEncryptionDescription](#) object

Name

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^~)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

The permissions mode of the ledger.

Type: String

Valid Values: `ALLOW_ALL` | `STANDARD`

State

The current status of the ledger.

Type: String

Valid Values: `CREATING` | `ACTIVE` | `DELETING` | `DELETED`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)

- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ExportJournalToS3

Service: Amazon QLDB

Exports journal contents within a date and time range from a ledger into a specified Amazon Simple Storage Service (Amazon S3) bucket. A journal export job can write the data objects in either the text or binary representation of Amazon Ion format, or in *JSON Lines* text format.

If the ledger with the given Name doesn't exist, then throws `ResourceNotFoundException`.

If the ledger with the given Name is in `CREATING` status, then throws `ResourcePreconditionNotMetException`.

You can initiate up to two concurrent journal export requests for each ledger. Beyond this limit, journal export requests throw `LimitExceededException`.

Request Syntax

```
POST /ledgers/name/journal-s3-exports HTTP/1.1
Content-type: application/json
```

```
{
  "ExclusiveEndTime": number,
  "InclusiveStartTime": number,
  "OutputFormat": "string",
  "RoleArn": "string",
  "S3ExportConfiguration": {
    "Bucket": "string",
    "EncryptionConfiguration": {
      "KmsKeyArn": "string",
      "ObjectEncryptionType": "string"
    },
    "Prefix": "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

ExclusiveEndTime

The exclusive end date and time for the range of journal contents to export.

The `ExclusiveEndTime` must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC). For example: 2019-06-13T21:36:34Z.

The `ExclusiveEndTime` must be less than or equal to the current UTC date and time.

Type: Timestamp

Required: Yes

InclusiveStartTime

The inclusive start date and time for the range of journal contents to export.

The `InclusiveStartTime` must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC). For example: 2019-06-13T21:36:34Z.

The `InclusiveStartTime` must be before `ExclusiveEndTime`.

If you provide an `InclusiveStartTime` that is before the ledger's `CreationDateTime`, Amazon QLDB defaults it to the ledger's `CreationDateTime`.

Type: Timestamp

Required: Yes

OutputFormat

The output format of your exported journal data. A journal export job can write the data objects in either the text or binary representation of [Amazon Ion](#) format, or in [JSON Lines](#) text format.

Default: ION_TEXT

In JSON Lines format, each journal block in an exported data object is a valid JSON object that is delimited by a newline. You can use this format to directly integrate JSON exports with analytics tools such as Amazon Athena and AWS Glue because these services can parse newline-delimited JSON automatically.

Type: String

Valid Values: ION_BINARY | ION_TEXT | JSON

Required: No

RoleArn

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal export job to do the following:

- Write objects into your Amazon S3 bucket.
- (Optional) Use your customer managed key in AWS Key Management Service (AWS KMS) for server-side encryption of your exported data.

To pass a role to QLDB when requesting a journal export, you must have permissions to perform the `iam:PassRole` action on the IAM role resource. This is required for all journal export requests.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

S3ExportConfiguration

The configuration settings of the Amazon S3 bucket destination for your export request.

Type: [S3ExportConfiguration](#) object

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json
```

```
{  
  "ExportId": "string"  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[ExportId](#)

The UUID (represented in Base62-encoded text) that QLDB assigns to each journal export job.

To describe your export request and check the status of the job, you can use `ExportId` to call `DescribeJournalS3Export`.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetBlock

Service: Amazon QLDB

Returns a block object at a specified address in a journal. Also returns a proof of the specified block for verification if `DigestTipAddress` is provided.

For information about the data contents in a block, see [Journal contents](#) in the *Amazon QLDB Developer Guide*.

If the specified ledger doesn't exist or is in `DELETING` status, then throws `ResourceNotFoundException`.

If the specified ledger is in `CREATING` status, then throws `ResourcePreconditionNotMetException`.

If no block exists with the specified address, then throws `InvalidParameterException`.

Request Syntax

```
POST /ledgers/name/block HTTP/1.1
Content-type: application/json
```

```
{
  "BlockAddress": {
    "IonText": "string"
  },
  "DigestTipAddress": {
    "IonText": "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

[name](#)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

BlockAddress

The location of the block that you want to request. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"B1FTj1SXze9BIh1K0szcE3",sequenceNo:14}`.

Type: [ValueHolder](#) object

Required: Yes

DigestTipAddress

The latest block location covered by the digest for which to request a proof. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"B1FTj1SXze9BIh1K0szcE3",sequenceNo:49}`.

Type: [ValueHolder](#) object

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Block": {
    "IonText": "string"
  },
  "Proof": {
    "IonText": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Block

The block data object in Amazon Ion format.

Type: [ValueHolder](#) object

Proof

The proof object in Amazon Ion format returned by a GetBlock request. A proof contains the list of hash values required to recalculate the specified digest using a Merkle tree, starting with the specified block.

Type: [ValueHolder](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)

- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetDigest

Service: Amazon QLDB

Returns the digest of a ledger at the latest committed block in the journal. The response includes a 256-bit hash value and a block address.

Request Syntax

```
POST /ledgers/name/digest HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Digest": blob,
  "DigestTipAddress": {
    "IonText": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Digest

The 256-bit hash value representing the digest returned by a `GetDigest` request.

Type: Base64-encoded binary data object

Length Constraints: Fixed length of 32.

DigestTipAddress

The latest block location covered by the digest that you requested. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

Type: [ValueHolder](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetRevision

Service: Amazon QLDB

Returns a revision data object for a specified document ID and block address. Also returns a proof of the specified revision for verification if `DigestTipAddress` is provided.

Request Syntax

```
POST /ledgers/name/revision HTTP/1.1
Content-type: application/json

{
  "BlockAddress": {
    "IonText": "string"
  },
  "DigestTipAddress": {
    "IonText": "string"
  },
  "DocumentId": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

[name](#)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

[BlockAddress](#)

The block location of the document revision to be verified. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: {strandId:"B1FTj1SXze9BIh1K0szcE3",sequenceNo:14}.

Type: [ValueHolder](#) object

Required: Yes

[DigestTipAddress](#)

The latest block location covered by the digest for which to request a proof. An address is an Amazon Ion structure that has two fields: strandId and sequenceNo.

For example: {strandId:"B1FTj1SXze9BIh1K0szcE3",sequenceNo:49}.

Type: [ValueHolder](#) object

Required: No

[DocumentId](#)

The UUID (represented in Base62-encoded text) of the document to be verified.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Proof": {
    "IonText": "string"
  },
  "Revision": {
    "IonText": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Proof

The proof object in Amazon Ion format returned by a `GetRevision` request. A proof contains the list of hash values that are required to recalculate the specified digest using a Merkle tree, starting with the specified document revision.

Type: [ValueHolder](#) object

Revision

The document revision data object in Amazon Ion format.

Type: [ValueHolder](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListJournalKinesisStreamsForLedger

Service: Amazon QLDB

Returns all Amazon QLDB journal streams for a given ledger.

This action does not return any expired journal streams. For more information, see [Expiration for terminal streams](#) in the *Amazon QLDB Developer Guide*.

This action returns a maximum of `MaxResults` items. It is paginated so that you can retrieve all the items by calling `ListJournalKinesisStreamsForLedger` multiple times.

Request Syntax

```
GET /ledgers/name/journal-kinesis-streams?max_results=MaxResults&next_token=NextToken
HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

MaxResults

The maximum number of results to return in a single `ListJournalKinesisStreamsForLedger` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

NextToken

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListJournalKinesisStreamsForLedger` call, you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "NextToken": "string",
  "Streams": [
    {
      "Arn": "string",
      "CreationTime": number,
      "ErrorCause": "string",
      "ExclusiveEndTime": number,
      "InclusiveStartTime": number,
      "KinesisConfiguration": {
        "AggregationEnabled": boolean,
        "StreamArn": "string"
      },
      "LedgerName": "string",
      "RoleArn": "string",
      "Status": "string",
      "StreamId": "string",
      "StreamName": "string"
    }
  ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken

- If `NextToken` is empty, the last page of results has been processed and there are no more results to be retrieved.

- If `NextToken` is *not* empty, more results are available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListJournalKinesisStreamsForLedger` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

[Streams](#)

The QLDB journal streams that are currently associated with the given ledger.

Type: Array of [JournalKinesisStreamDescription](#) objects

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)

- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListJournalS3Exports

Service: Amazon QLDB

Returns all journal export jobs for all ledgers that are associated with the current AWS account and Region.

This action returns a maximum of `MaxResults` items, and is paginated so that you can retrieve all the items by calling `ListJournalS3Exports` multiple times.

This action does not return any expired export jobs. For more information, see [Export job expiration](#) in the *Amazon QLDB Developer Guide*.

Request Syntax

```
GET /journal-s3-exports?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

[MaxResults](#)

The maximum number of results to return in a single `ListJournalS3Exports` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

[NextToken](#)

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListJournalS3Exports` call, then you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "JournalS3Exports": [
    {
      "ExclusiveEndTime": number,
      "ExportCreationTime": number,
      "ExportId": "string",
      "InclusiveStartTime": number,
      "LedgerName": "string",
      "OutputFormat": "string",
      "RoleArn": "string",
      "S3ExportConfiguration": {
        "Bucket": "string",
        "EncryptionConfiguration": {
          "KmsKeyArn": "string",
          "ObjectEncryptionType": "string"
        },
        "Prefix": "string"
      },
      "Status": "string"
    }
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JournalS3Exports

The journal export jobs for all ledgers that are associated with the current AWS account and Region.

Type: Array of [JournalS3ExportDescription](#) objects

NextToken

- If NextToken is empty, then the last page of results has been processed and there are no more results to be retrieved.
- If NextToken is *not* empty, then there are more results available. To retrieve the next page of results, use the value of NextToken in a subsequent `ListJournalS3Exports` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListJournalS3ExportsForLedger

Service: Amazon QLDB

Returns all journal export jobs for a specified ledger.

This action returns a maximum of `MaxResults` items, and is paginated so that you can retrieve all the items by calling `ListJournalS3ExportsForLedger` multiple times.

This action does not return any expired export jobs. For more information, see [Export job expiration](#) in the *Amazon QLDB Developer Guide*.

Request Syntax

```
GET /ledgers/name/journal-s3-exports?max_results=MaxResults&next_token=NextToken
HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

[MaxResults](#)

The maximum number of results to return in a single `ListJournalS3ExportsForLedger` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

[name](#)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

[NextToken](#)

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListJournalS3ExportsForLedger` call, then you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "JournalS3Exports": [
    {
      "ExclusiveEndTime": number,
      "ExportCreationTime": number,
      "ExportId": "string",
      "InclusiveStartTime": number,
      "LedgerName": "string",
      "OutputFormat": "string",
      "RoleArn": "string",
      "S3ExportConfiguration": {
        "Bucket": "string",
        "EncryptionConfiguration": {
          "KmsKeyArn": "string",
          "ObjectEncryptionType": "string"
        },
        "Prefix": "string"
      },
      "Status": "string"
    }
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JournalS3Exports

The journal export jobs that are currently associated with the specified ledger.

Type: Array of [JournalS3ExportDescription](#) objects

NextToken

- If NextToken is empty, then the last page of results has been processed and there are no more results to be retrieved.
- If NextToken is *not* empty, then there are more results available. To retrieve the next page of results, use the value of NextToken in a subsequent `ListJournalS3ExportsForLedger` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListLedgers

Service: Amazon QLDB

Returns all ledgers that are associated with the current AWS account and Region.

This action returns a maximum of `MaxResults` items and is paginated so that you can retrieve all the items by calling `ListLedgers` multiple times.

Request Syntax

```
GET /ledgers?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

MaxResults

The maximum number of results to return in a single `ListLedgers` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

NextToken

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListLedgers` call, then you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200  
Content-type: application/json
```

```
{
  "Ledgers": [
    {
      "CreationDateTime": number,
      "Name": "string",
      "State": "string"
    }
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Ledgers](#)

The ledgers that are associated with the current AWS account and Region.

Type: Array of [LedgerSummary](#) objects

[NextToken](#)

A pagination token, indicating whether there are more results available:

- If `NextToken` is empty, then the last page of results has been processed and there are no more results to be retrieved.
- If `NextToken` is *not* empty, then there are more results available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListLedgers` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListTagsForResource

Service: Amazon QLDB

Returns all tags for a specified Amazon QLDB resource.

Request Syntax

```
GET /tags/resourceArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn

The Amazon Resource Name (ARN) for which to list the tags. For example:

arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Tags": {
    "string" : "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Tags

The tags that are currently associated with the specified Amazon QLDB resource.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)

- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

StreamJournalToKinesis

Service: Amazon QLDB

Creates a journal stream for a given Amazon QLDB ledger. The stream captures every document revision that is committed to the ledger's journal and delivers the data to a specified Amazon Kinesis Data Streams resource.

Request Syntax

```
POST /ledgers/name/journal-kinesis-streams HTTP/1.1
Content-type: application/json
```

```
{
  "ExclusiveEndTime": number,
  "InclusiveStartTime": number,
  "KinesisConfiguration": {
    "AggregationEnabled": boolean,
    "StreamArn": "string"
  },
  "RoleArn": "string",
  "StreamName": "string",
  "Tags": {
    "string" : "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

ExclusiveEndTime

The exclusive date and time that specifies when the stream ends. If you don't define this parameter, the stream runs indefinitely until you cancel it.

The `ExclusiveEndTime` must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC). For example: `2019-06-13T21:36:34Z`.

Type: Timestamp

Required: No

InclusiveStartTime

The inclusive start date and time from which to start streaming journal data. This parameter must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC). For example: `2019-06-13T21:36:34Z`.

The `InclusiveStartTime` cannot be in the future and must be before `ExclusiveEndTime`.

If you provide an `InclusiveStartTime` that is before the ledger's `CreationDateTime`, QLDB effectively defaults it to the ledger's `CreationDateTime`.

Type: Timestamp

Required: Yes

KinesisConfiguration

The configuration settings of the Kinesis Data Streams destination for your stream request.

Type: [KinesisConfiguration](#) object

Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal stream to write data records to a Kinesis Data Streams resource.

To pass a role to QLDB when requesting a journal stream, you must have permissions to perform the `iam:PassRole` action on the IAM role resource. This is required for all journal stream requests.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

StreamName

The name that you want to assign to the QLDB journal stream. User-defined names can help identify and indicate the purpose of a stream.

Your stream name must be unique among other *active* streams for a given ledger. Stream names have the same naming constraints as ledger names, as defined in [Quotas in Amazon QLDB](#) in the *Amazon QLDB Developer Guide*.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Tags

The key-value pairs to add as tags to the stream that you want to create. Tag keys are case sensitive. Tag values are case sensitive and can be null.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "StreamId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

StreamId

The UUID (represented in Base62-encoded text) that QLDB assigns to each QLDB journal stream.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

TagResource

Service: Amazon QLDB

Adds one or more tags to a specified Amazon QLDB resource.

A resource can have up to 50 tags. If you try to create more than 50 tags for a resource, your request fails and returns an error.

Request Syntax

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
  "Tags": {
    "string" : "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn

The Amazon Resource Name (ARN) to which you want to add the tags. For example:

arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

Request Body

The request accepts the following data in JSON format.

Tags

The key-value pairs to add as tags to the specified QLDB resource. Tag keys are case sensitive. If you specify a key that already exists for the resource, your request fails and returns an error. Tag values are case sensitive and can be null.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: Yes

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)

- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UntagResource

Service: Amazon QLDB

Removes one or more tags from a specified Amazon QLDB resource. You can specify up to 50 tag keys to remove.

Request Syntax

```
DELETE /tags/resourceArn?tagKeys=TagKeys HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn

The Amazon Resource Name (ARN) from which to remove the tags. For example:

```
arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger
```

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

TagKeys

The list of tag keys to remove.

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateLedger

Service: Amazon QLDB

Updates properties on a ledger.

Request Syntax

```
PATCH /ledgers/name HTTP/1.1
Content-type: application/json

{
  "DeletionProtection": boolean,
  "KmsKey": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

DeletionProtection

Specifies whether the ledger is protected from being deleted by any user. If not defined during ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger. You can disable it by calling the `UpdateLedger` operation to set this parameter to `false`.

Type: Boolean

Required: No

KmsKey

The key in AWS Key Management Service (AWS KMS) to use for encryption of data at rest in the ledger. For more information, see [Encryption at rest](#) in the *Amazon QLDB Developer Guide*.

Use one of the following options to specify this parameter:

- **AWS_OWNED_KMS_KEY:** Use an AWS KMS key that is owned and managed by AWS on your behalf.
- **Undefined:** Make no changes to the KMS key of the ledger.
- **A valid symmetric customer managed KMS key:** Use the specified symmetric encryption KMS key in your account that you create, own, and manage.

Amazon QLDB does not support asymmetric keys. For more information, see [Using symmetric and asymmetric keys](#) in the *AWS Key Management Service Developer Guide*.

To specify a customer managed KMS key, you can use its key ID, Amazon Resource Name (ARN), alias name, or alias ARN. When using an alias name, prefix it with "alias/". To specify a key in a different AWS account, you must use the key ARN or alias ARN.

For example:

- Key ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- Key ARN: arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
- Alias name: alias/ExampleAlias
- Alias ARN: arn:aws:kms:us-east-2:111122223333:alias/ExampleAlias

For more information, see [Key identifiers \(KeyId\)](#) in the *AWS Key Management Service Developer Guide*.

Type: String

Length Constraints: Maximum length of 1600.

Required: No

Response Syntax

```
HTTP/1.1 200
```

Content-type: application/json

```
{
  "Arn": "string",
  "CreationDateTime": number,
  "DeletionProtection": boolean,
  "EncryptionDescription": {
    "EncryptionStatus": "string",
    "InaccessibleKmsKeyDateTime": number,
    "KmsKeyArn": "string"
  },
  "Name": "string",
  "State": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Arn

The Amazon Resource Name (ARN) for the ledger.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

CreationDateTime

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

DeletionProtection

Specifies whether the ledger is protected from being deleted by any user. If not defined during ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger. You can disable it by calling the `UpdateLedger` operation to set this parameter to `false`.

Type: Boolean

EncryptionDescription

Information about the encryption of data at rest in the ledger. This includes the current status, the AWS KMS key, and when the key became inaccessible (in the case of an error).

Type: [LedgerEncryptionDescription](#) object

Name

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-(?!.*-$)^[A-Za-z0-9-]+$`

State

The current status of the ledger.

Type: String

Valid Values: CREATING | ACTIVE | DELETING | DELETED

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateLedgerPermissionsMode

Service: Amazon QLDB

Updates the permissions mode of a ledger.

Important

Before you switch to the STANDARD permissions mode, you must first create all required IAM policies and table tags to avoid disruption to your users. To learn more, see [Migrating to the standard permissions mode](#) in the *Amazon QLDB Developer Guide*.

Request Syntax

```
PATCH /ledgers/name/permissions-mode HTTP/1.1
Content-type: application/json

{
  "PermissionsMode": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

name

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

PermissionsMode

The permissions mode to assign to the ledger. This parameter can have one of the following values:

- **ALLOW_ALL**: A legacy permissions mode that enables access control with API-level granularity for ledgers.

This mode allows users who have the `SendCommand` API permission for this ledger to run all PartiQL commands (hence, **ALLOW_ALL**) on any tables in the specified ledger. This mode disregards any table-level or command-level IAM permissions policies that you create for the ledger.

- **STANDARD**: (*Recommended*) A permissions mode that enables access control with finer granularity for ledgers, tables, and PartiQL commands.

By default, this mode denies all user requests to run any PartiQL commands on any tables in this ledger. To allow PartiQL commands to run, you must create IAM permissions policies for specific table resources and PartiQL actions, in addition to the `SendCommand` API permission for the ledger. For information, see [Getting started with the standard permissions mode](#) in the *Amazon QLDB Developer Guide*.

Note

We strongly recommend using the **STANDARD** permissions mode to maximize the security of your ledger data.

Type: String

Valid Values: **ALLOW_ALL** | **STANDARD**

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Arn": "string",
```

```
"Name": "string",  
"PermissionsMode": "string"  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Arn

The Amazon Resource Name (ARN) for the ledger.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Name

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

The current permissions mode of the ledger.

Type: String

Valid Values: ALLOW_ALL | STANDARD

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

Amazon QLDB Session

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following actions are supported by Amazon QLDB Session:

- [SendCommand](#)

SendCommand

Service: Amazon QLDB Session

Sends a command to an Amazon QLDB ledger.

Note

Instead of interacting directly with this API, we recommend using the QLDB driver or the QLDB shell to run data transactions on a ledger.

- If you're working with an AWS SDK, use the QLDB driver. The driver provides a high-level abstraction layer above this *QLDB Session* data API and manages the `SendCommand` operation for you. For information and a list of supported programming languages, see [Getting started with the driver](#) in the *Amazon QLDB Developer Guide*.
- If you're working with the AWS Command Line Interface (AWS CLI), use the QLDB shell. The shell is a command line interface that uses the QLDB driver to interact with a ledger. For information, see [Accessing Amazon QLDB using the QLDB shell](#).

Request Syntax

```
{
  "AbortTransaction": {
  },
  "CommitTransaction": {
    "CommitDigest": blob,
    "TransactionId": "string"
  },
  "EndSession": {
  },
  "ExecuteStatement": {
    "Parameters": [
      {
        "IonBinary": blob,
        "IonText": "string"
      }
    ],
    "Statement": "string",
    "TransactionId": "string"
  },
  "FetchPage": {
```

```
    "NextPageToken": "string",
    "TransactionId": "string"
  },
  "SessionToken": "string",
  "StartSession": {
    "LedgerName": "string"
  },
  "StartTransaction": {
  }
}
```

Request Parameters

For information about the parameters that are common to all actions, see [Common Parameters](#).

The request accepts the following data in JSON format.

[AbortTransaction](#)

Command to abort the current transaction.

Type: [AbortTransactionRequest](#) object

Required: No

[CommitTransaction](#)

Command to commit the specified transaction.

Type: [CommitTransactionRequest](#) object

Required: No

[EndSession](#)

Command to end the current session.

Type: [EndSessionRequest](#) object

Required: No

[ExecuteStatement](#)

Command to execute a statement in the specified transaction.

Type: [ExecuteStatementRequest](#) object

Required: No

[FetchPage](#)

Command to fetch a page.

Type: [FetchPageRequest](#) object

Required: No

[SessionToken](#)

Specifies the session token for the current command. A session token is constant throughout the life of the session.

To obtain a session token, run the `StartSession` command. This `SessionToken` is required for every subsequent command that is issued during the current session.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: No

[StartSession](#)

Command to start a new session. A session token is obtained as part of the response.

Type: [StartSessionRequest](#) object

Required: No

[StartTransaction](#)

Command to start a new transaction.

Type: [StartTransactionRequest](#) object

Required: No

Response Syntax

```
{
  "AbortTransaction": {
    "TimingInformation": {
      "ProcessingTimeMilliseconds": number
    }
  },
  "CommitTransaction": {
    "CommitDigest": blob,
    "ConsumedIOs": {
      "ReadIOs": number,
      "WriteIOs": number
    },
    "TimingInformation": {
      "ProcessingTimeMilliseconds": number
    },
    "TransactionId": "string"
  },
  "EndSession": {
    "TimingInformation": {
      "ProcessingTimeMilliseconds": number
    }
  },
  "ExecuteStatement": {
    "ConsumedIOs": {
      "ReadIOs": number,
      "WriteIOs": number
    },
    "FirstPage": {
      "NextPageToken": "string",
      "Values": [
        {
          "IonBinary": blob,
          "IonText": "string"
        }
      ]
    },
    "TimingInformation": {
      "ProcessingTimeMilliseconds": number
    }
  },
  "FetchPage": {
    "ConsumedIOs": {
```

```

    "ReadIOs": number,
    "WriteIOs": number
  },
  "Page": {
    "NextPageToken": "string",
    "Values": [
      {
        "IonBinary": blob,
        "IonText": "string"
      }
    ]
  },
  "TimingInformation": {
    "ProcessingTimeMilliseconds": number
  }
},
"StartSession": {
  "SessionToken": "string",
  "TimingInformation": {
    "ProcessingTimeMilliseconds": number
  }
},
"StartTransaction": {
  "TimingInformation": {
    "ProcessingTimeMilliseconds": number
  },
  "TransactionId": "string"
}
}

```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AbortTransaction

Contains the details of the aborted transaction.

Type: [AbortTransactionResult](#) object

CommitTransaction

Contains the details of the committed transaction.

Type: [CommitTransactionResult](#) object

[EndSession](#)

Contains the details of the ended session.

Type: [EndSessionResult](#) object

[ExecuteStatement](#)

Contains the details of the executed statement.

Type: [ExecuteStatementResult](#) object

[FetchPage](#)

Contains the details of the fetched page.

Type: [FetchPageResult](#) object

[StartSession](#)

Contains the details of the started session that includes a session token. This `SessionToken` is required for every subsequent command that is issued during the current session.

Type: [StartSessionResult](#) object

[StartTransaction](#)

Contains the details of the started transaction.

Type: [StartTransactionResult](#) object

Errors

For information about the errors that are common to all actions, see [Common Errors](#).

BadRequestException

Returned if the request is malformed or contains an error such as an invalid parameter value or a missing required parameter.

HTTP Status Code: 400

CapacityExceededException

Returned when the request exceeds the processing capacity of the ledger.

HTTP Status Code: 400

InvalidSessionException

Returned if the session doesn't exist anymore because it timed out or expired.

HTTP Status Code: 400

LimitExceededException

Returned if a resource limit such as number of active sessions is exceeded.

HTTP Status Code: 400

OccConflictException

Returned when a transaction cannot be written to the journal due to a failure in the verification phase of *optimistic concurrency control* (OCC).

HTTP Status Code: 400

RateExceededException

Returned when the rate of requests exceeds the allowed throughput.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go v2](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for JavaScript V3](#)
- [AWS SDK for Kotlin](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)

- [AWS SDK for Ruby V3](#)

Data Types

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following data types are supported by Amazon QLDB:

- [JournalKinesisStreamDescription](#)
- [JournalS3ExportDescription](#)
- [KinesisConfiguration](#)
- [LedgerEncryptionDescription](#)
- [LedgerSummary](#)
- [S3EncryptionConfiguration](#)
- [S3ExportConfiguration](#)
- [ValueHolder](#)

The following data types are supported by Amazon QLDB Session:

- [AbortTransactionRequest](#)
- [AbortTransactionResult](#)
- [CommitTransactionRequest](#)
- [CommitTransactionResult](#)
- [EndSessionRequest](#)
- [EndSessionResult](#)
- [ExecuteStatementRequest](#)
- [ExecuteStatementResult](#)
- [FetchPageRequest](#)

- [FetchPageResult](#)
- [IOUsage](#)
- [Page](#)
- [StartSessionRequest](#)
- [StartSessionResult](#)
- [StartTransactionRequest](#)
- [StartTransactionResult](#)
- [TimingInformation](#)
- [ValueHolder](#)

Amazon QLDB

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following data types are supported by Amazon QLDB:

- [JournalKinesisStreamDescription](#)
- [JournalS3ExportDescription](#)
- [KinesisConfiguration](#)
- [LedgerEncryptionDescription](#)
- [LedgerSummary](#)
- [S3EncryptionConfiguration](#)
- [S3ExportConfiguration](#)
- [ValueHolder](#)

JournalKinesisStreamDescription

Service: Amazon QLDB

Information about an Amazon QLDB journal stream, including the Amazon Resource Name (ARN), stream name, creation time, current status, and the parameters of the original stream creation request.

Contents

KinesisConfiguration

The configuration settings of the Amazon Kinesis Data Streams destination for a QLDB journal stream.

Type: [KinesisConfiguration](#) object

Required: Yes

LedgerName

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^--)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal stream to write data records to a Kinesis Data Streams resource.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

Status

The current state of the QLDB journal stream.

Type: String

Valid Values: ACTIVE | COMPLETED | CANCELED | FAILED | IMPAIRED

Required: Yes

StreamId

The UUID (represented in Base62-encoded text) of the QLDB journal stream.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z0-9]+$`

Required: Yes

StreamName

The user-defined name of the QLDB journal stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Arn

The Amazon Resource Name (ARN) of the QLDB journal stream.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: No

CreationTime

The date and time, in epoch time format, when the QLDB journal stream was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

Required: No

ErrorCause

The error message that describes the reason that a stream has a status of IMPAIRED or FAILED. This is not applicable to streams that have other status values.

Type: String

Valid Values: KINESIS_STREAM_NOT_FOUND | IAM_PERMISSION_REVOKED

Required: No

ExclusiveEndTime

The exclusive date and time that specifies when the stream ends. If this parameter is undefined, the stream runs indefinitely until you cancel it.

Type: Timestamp

Required: No

InclusiveStartTime

The inclusive start date and time from which to start streaming journal data.

Type: Timestamp

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

JournalS3ExportDescription

Service: Amazon QLDB

Information about a journal export job, including the ledger name, export ID, creation time, current status, and the parameters of the original export creation request.

Contents

ExclusiveEndTime

The exclusive end date and time for the range of journal contents that was specified in the original export request.

Type: Timestamp

Required: Yes

ExportCreationTime

The date and time, in epoch time format, when the export job was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

Required: Yes

ExportId

The UUID (represented in Base62-encoded text) of the journal export job.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

InclusiveStartTime

The inclusive start date and time for the range of journal contents that was specified in the original export request.

Type: Timestamp

Required: Yes

LedgerName

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^--)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal export job to do the following:

- Write objects into your Amazon Simple Storage Service (Amazon S3) bucket.
- (Optional) Use your customer managed key in AWS Key Management Service (AWS KMS) for server-side encryption of your exported data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

S3ExportConfiguration

The Amazon Simple Storage Service (Amazon S3) bucket location in which a journal export job writes the journal contents.

Type: [S3ExportConfiguration](#) object

Required: Yes

Status

The current state of the journal export job.

Type: String

Valid Values: IN_PROGRESS | COMPLETED | CANCELLED

Required: Yes

OutputFormat

The output format of the exported journal data.

Type: String

Valid Values: ION_BINARY | ION_TEXT | JSON

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

KinesisConfiguration

Service: Amazon QLDB

The configuration settings of the Amazon Kinesis Data Streams destination for an Amazon QLDB journal stream.

Contents

StreamArn

The Amazon Resource Name (ARN) of the Kinesis Data Streams resource.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

AggregationEnabled

Enables QLDB to publish multiple data records in a single Kinesis Data Streams record, increasing the number of records sent per API call.

Default: True

Important

Record aggregation has important implications for processing records and requires de-aggregation in your stream consumer. To learn more, see [KPL Key Concepts](#) and [Consumer De-aggregation](#) in the *Amazon Kinesis Data Streams Developer Guide*.

Type: Boolean

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

LedgerEncryptionDescription

Service: Amazon QLDB

Information about the encryption of data at rest in an Amazon QLDB ledger. This includes the current status, the key in AWS Key Management Service (AWS KMS), and when the key became inaccessible (in the case of an error).

For more information, see [Encryption at rest](#) in the *Amazon QLDB Developer Guide*.

Contents

EncryptionStatus

The current state of encryption at rest for the ledger. This can be one of the following values:

- **ENABLED**: Encryption is fully enabled using the specified key.
- **UPDATING**: The ledger is actively processing the specified key change.

Key changes in QLDB are asynchronous. The ledger is fully accessible without any performance impact while the key change is being processed. The amount of time it takes to update a key varies depending on the ledger size.

- **KMS_KEY_INACCESSIBLE**: The specified customer managed KMS key is not accessible, and the ledger is impaired. Either the key was disabled or deleted, or the grants on the key were revoked. When a ledger is impaired, it is not accessible and does not accept any read or write requests.

An impaired ledger automatically returns to an active state after you restore the grants on the key, or re-enable the key that was disabled. However, deleting a customer managed KMS key is irreversible. After a key is deleted, you can no longer access the ledgers that are protected with that key, and the data becomes unrecoverable permanently.

Type: String

Valid Values: `ENABLED` | `UPDATING` | `KMS_KEY_INACCESSIBLE`

Required: Yes

KmsKeyArn

The Amazon Resource Name (ARN) of the customer managed KMS key that the ledger uses for encryption at rest. If this parameter is undefined, the ledger uses an AWS owned KMS key

for encryption. It will display `AWS_OWNED_KMS_KEY` when updating the ledger's encryption configuration to the AWS owned KMS key.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

InaccessibleKmsKeyDateTime

The date and time, in epoch time format, when the AWS KMS key first became inaccessible, in the case of an error. (Epoch time format is the number of seconds that have elapsed since 12:00:00 AM January 1, 1970 UTC.)

This parameter is undefined if the AWS KMS key is accessible.

Type: Timestamp

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

LedgerSummary

Service: Amazon QLDB

Information about a ledger, including its name, state, and when it was created.

Contents

CreationDateTime

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

Required: No

Name

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: No

State

The current status of the ledger.

Type: String

Valid Values: CREATING | ACTIVE | DELETING | DELETED

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

S3EncryptionConfiguration

Service: Amazon QLDB

The encryption settings that are used by a journal export job to write data in an Amazon Simple Storage Service (Amazon S3) bucket.

Contents

ObjectEncryptionType

The Amazon S3 object encryption type.

To learn more about server-side encryption options in Amazon S3, see [Protecting Data Using Server-Side Encryption](#) in the *Amazon S3 Developer Guide*.

Type: String

Valid Values: SSE_KMS | SSE_S3 | NO_ENCRYPTION

Required: Yes

KmsKeyArn

The Amazon Resource Name (ARN) of a symmetric encryption key in AWS Key Management Service (AWS KMS). Amazon S3 does not support asymmetric KMS keys.

You must provide a KmsKeyArn if you specify SSE_KMS as the ObjectEncryptionType.

KmsKeyArn is not required if you specify SSE_S3 as the ObjectEncryptionType.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

S3ExportConfiguration

Service: Amazon QLDB

The Amazon Simple Storage Service (Amazon S3) bucket location in which a journal export job writes the journal contents.

Contents

Bucket

The Amazon S3 bucket name in which a journal export job writes the journal contents.

The bucket name must comply with the Amazon S3 bucket naming conventions. For more information, see [Bucket Restrictions and Limitations](#) in the *Amazon S3 Developer Guide*.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 255.

Pattern: `^[A-Za-z-0-9-_.]+$`

Required: Yes

EncryptionConfiguration

The encryption settings that are used by a journal export job to write data in an Amazon S3 bucket.

Type: [S3EncryptionConfiguration](#) object

Required: Yes

Prefix

The prefix for the Amazon S3 bucket in which a journal export job writes the journal contents.

The prefix must comply with Amazon S3 key naming rules and restrictions. For more information, see [Object Key and Metadata](#) in the *Amazon S3 Developer Guide*.

The following are examples of valid Prefix values:

- JournalExports-ForMyLedger/Testing/
- JournalExports
- My:Tests/

Type: String

Length Constraints: Minimum length of 0. Maximum length of 128.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

ValueHolder

Service: Amazon QLDB

A structure that can contain a value in multiple encoding formats.

Contents

IonText

An Amazon Ion plaintext value contained in a ValueHolder structure.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1048576.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

Amazon QLDB Session

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following data types are supported by Amazon QLDB Session:

- [AbortTransactionRequest](#)
- [AbortTransactionResult](#)

- [CommitTransactionRequest](#)
- [CommitTransactionResult](#)
- [EndSessionRequest](#)
- [EndSessionResult](#)
- [ExecuteStatementRequest](#)
- [ExecuteStatementResult](#)
- [FetchPageRequest](#)
- [FetchPageResult](#)
- [IOUsage](#)
- [Page](#)
- [StartSessionRequest](#)
- [StartSessionResult](#)
- [StartTransactionRequest](#)
- [StartTransactionResult](#)
- [TimingInformation](#)
- [ValueHolder](#)

AbortTransactionRequest

Service: Amazon QLDB Session

Contains the details of the transaction to abort.

Contents

The members of this exception structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

AbortTransactionResult

Service: Amazon QLDB Session

Contains the details of the aborted transaction.

Contents

TimingInformation

Contains server-side performance information for the command.

Type: [TimingInformation](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

CommitTransactionRequest

Service: Amazon QLDB Session

Contains the details of the transaction to commit.

Contents

CommitDigest

Specifies the commit digest for the transaction to commit. For every active transaction, the commit digest must be passed. QLDB validates `CommitDigest` and rejects the commit with an error if the digest computed on the client does not match the digest computed by QLDB.

The purpose of the `CommitDigest` parameter is to ensure that QLDB commits a transaction if and only if the server has processed the exact set of statements sent by the client, in the same order that client sent them, and with no duplicates.

Type: Base64-encoded binary data object

Required: Yes

TransactionId

Specifies the transaction ID of the transaction to commit.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

CommitTransactionResult

Service: Amazon QLDB Session

Contains the details of the committed transaction.

Contents

CommitDigest

The commit digest of the committed transaction.

Type: Base64-encoded binary data object

Required: No

ConsumedIOs

Contains metrics about the number of I/O requests that were consumed.

Type: [IOUsage](#) object

Required: No

TimingInformation

Contains server-side performance information for the command.

Type: [TimingInformation](#) object

Required: No

TransactionId

The transaction ID of the committed transaction.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z0-9]+$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

EndSessionRequest

Service: Amazon QLDB Session

Specifies a request to end the session.

Contents

The members of this exception structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

EndSessionResult

Service: Amazon QLDB Session

Contains the details of the ended session.

Contents

TimingInformation

Contains server-side performance information for the command.

Type: [TimingInformation](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

ExecuteStatementRequest

Service: Amazon QLDB Session

Specifies a request to execute a statement.

Contents

Statement

Specifies the statement of the request.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100000.

Required: Yes

TransactionId

Specifies the transaction ID of the request.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

Parameters

Specifies the parameters for the parameterized statement in the request.

Type: Array of [ValueHolder](#) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)

- [AWS SDK for Ruby V3](#)

ExecuteStatementResult

Service: Amazon QLDB Session

Contains the details of the executed statement.

Contents

ConsumedIOs

Contains metrics about the number of I/O requests that were consumed.

Type: [IOUsage](#) object

Required: No

FirstPage

Contains the details of the first fetched page.

Type: [Page](#) object

Required: No

TimingInformation

Contains server-side performance information for the command.

Type: [TimingInformation](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

FetchPageRequest

Service: Amazon QLDB Session

Specifies the details of the page to be fetched.

Contents

NextPageToken

Specifies the next page token of the page to be fetched.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: Yes

TransactionId

Specifies the transaction ID of the page to be fetched.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

FetchPageResult

Service: Amazon QLDB Session

Contains the page that was fetched.

Contents

ConsumedIOs

Contains metrics about the number of I/O requests that were consumed.

Type: [IOUsage](#) object

Required: No

Page

Contains details of the fetched page.

Type: [Page](#) object

Required: No

TimingInformation

Contains server-side performance information for the command.

Type: [TimingInformation](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

IOUsage

Service: Amazon QLDB Session

Contains I/O usage metrics for a command that was invoked.

Contents

ReadIOs

The number of read I/O requests that the command made.

Type: Long

Required: No

WriteIOs

The number of write I/O requests that the command made.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

Page

Service: Amazon QLDB Session

Contains details of the fetched page.

Contents

NextPageToken

The token of the next page.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: No

Values

A structure that contains values in multiple encoding formats.

Type: Array of [ValueHolder](#) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

StartSessionRequest

Service: Amazon QLDB Session

Specifies a request to start a new session.

Contents

LedgerName

The name of the ledger to start a new session against.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-.)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

StartSessionResult

Service: Amazon QLDB Session

Contains the details of the started session.

Contents

SessionToken

Session token of the started session. This SessionToken is required for every subsequent command that is issued during the current session.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: No

TimingInformation

Contains server-side performance information for the command.

Type: [TimingInformation](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

StartTransactionRequest

Service: Amazon QLDB Session

Specifies a request to start a transaction.

Contents

The members of this exception structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

StartTransactionResult

Service: Amazon QLDB Session

Contains the details of the started transaction.

Contents

TimingInformation

Contains server-side performance information for the command.

Type: [TimingInformation](#) object

Required: No

TransactionId

The transaction ID of the started transaction.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

TimingInformation

Service: Amazon QLDB Session

Contains server-side performance information for a command. Amazon QLDB captures timing information between the times when it receives the request and when it sends the corresponding response.

Contents

ProcessingTimeMilliseconds

The amount of time that QLDB spent on processing the command, measured in milliseconds.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

ValueHolder

Service: Amazon QLDB Session

A structure that can contain a value in multiple encoding formats.

Contents

IonBinary

An Amazon Ion binary value contained in a `ValueHolder` structure.

Type: Base64-encoded binary data object

Length Constraints: Minimum length of 1. Maximum length of 131072.

Required: No

IonText

An Amazon Ion plaintext value contained in a `ValueHolder` structure.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1048576.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

Common Errors

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400

IncompleteSignature

The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

NotAuthorized

You do not have permission to perform this action.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action.

For more information about Signature Version 4, see [Signing AWS API requests](#) in the *IAM User Guide*.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: AWS4-HMAC-SHA256

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: *access_key/YYYYMMDD/region/service/aws4_request*.

For more information, see [Create a signed AWS API request](#) in the *IAM User Guide*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: 20120325T120000Z.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see [Elements of an AWS API request signature](#) in the *IAM User Guide*.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS STS, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Condition: If you're using temporary security credentials from AWS STS, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see [Create a signed AWS API request](#) in the *IAM User Guide*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

Quotas and limits in Amazon QLDB

⚠ Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

This section describes the current quotas, also referred to as limits, in Amazon QLDB.

Topics

- [Default quotas](#)
- [Fixed quotas](#)
- [Ledger quota](#)
- [Document size](#)
- [Transaction size](#)
- [Naming constraints](#)

Default quotas

QLDB has the following default quotas, as also listed at [Amazon QLDB endpoints and quotas](#) in the *AWS General Reference*. These quotas are per AWS account per Region. To request a quota increase for your account in a Region, use the Service Quotas console.

Sign in to the AWS Management Console and open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.

Resource	Default quota
The maximum number of active ledgers that you can create in this account in the current Region	5

Resource	Default quota
The maximum number of active journal exports to Amazon S3 per ledger	2
The maximum number of active journal streams to Kinesis Data Streams per ledger	5

Fixed quotas

In addition to default quotas, QLDB has the following fixed quotas per ledger. These quotas can't be increased by using Service Quotas:

Resource	Fixed quota
Number of concurrent active sessions	3000
Number of active tables	20
IndexedStorage – total amount of disk space used by the ledger's tables, indexes, and indexed history, reported in 15-minute intervals. For more information about Amazon CloudWatch metrics, see Monitoring with Amazon CloudWatch .	Maximum of 16 TiB
Number of total tables (active and inactive)  Note In QLDB, dropped tables are considered <i>inactive</i> and count against this total quota.	40
Number of indexes per table	5
Number of documents in a transaction	40

Resource	Fixed quota
Number of revisions to redact in a transaction	1
Document size (encoded in IonBinary format)	128 KB
Statement parameter size (IonBinary format)	128 KB
Statement parameter size (IonText format)	1 MB
Statement string length	100,000 characters
Transaction size	4 MB
Transaction timeout	30 seconds
Expiration period for completed journal export jobs	7 days
Expiration period for terminal journal streams	7 days

Ledger quota

To request a ledger quota increase for your account in a Region, you can use the Service Quotas console.

Open the Service Quotas console at <https://console.aws.amazon.com/servicequotas/>.

Some QLDB use cases require an expanding number of ledgers per AWS account per Region based on business growth. For example, you might need to create dedicated ledgers to isolate customers or data. In this case, consider leveraging a multi-account architecture to work with QLDB quotas. For more information, refer to *Account Silo Isolation* in the AWS whitepaper [SaaS Tenant Isolation Strategies](#).

Document size

The maximum size for a document that is encoded in the IonBinary format is 128 KB. We can't provide an exact limit for the size of a document in IonText format because the conversion from text to binary varies significantly based on the structure of each document. QLDB supports documents with open content, so each unique document structure alters the size calculation.

Transaction size

The maximum size for a transaction in QLDB is 4 MB. The size of a transaction is calculated based on the sum of the following factors.

Deltas

The document changes that are generated by *all the statements* within the transaction. In a transaction that impacts several documents, the total delta size is the sum of each affected document's individual delta.

Metadata

The system-generated transaction metadata that is associated with each affected document.

Indexes

If an index is defined on a table that is affected by the transaction, the associated index entry also generates a delta.

History

Because all document revisions are persisted in QLDB, all transactions also append to the history.

Inserts – Every document inserted into a table also has a copy inserted into its history table. For example, a newly inserted 100 KB document generates a *minimum* of 200 KB of deltas in a transaction. (This is a rough estimate that doesn't include metadata or indexes.)

Updates – Any document update, even for a single field, creates a new revision of the entire document in history, plus or minus the delta of the update. This means that a small update in a large document would still generate a large transaction delta. For example, adding 2 KB of data in an existing 100 KB document creates a new 102 KB revision in history. This adds up to at least 104 KB of total deltas in a transaction. (Again, this estimate doesn't include metadata or indexes.)

Deletes – Similar to updates, any delete transaction creates a new document revision in the history. However, the newly created DELETE revision is smaller than the original document because it has null user data and only contains metadata.

Naming constraints

The following table describes naming constraints within Amazon QLDB.

Ledger name	• Must only contain 1–32 alphanumeric characters or hyphens.
Journal stream name	• Must have a letter or number for the first and last characters. • Must not be all numbers. • Can't contain two consecutive hyphens. • Is case sensitive.
Table name	• Must only contain 1–128 alphanumeric characters or underscores. • Must have a letter or an underscore for the first character. • Can contain any combination of alphanumeric characters and underscores for the remaining characters. • Is case sensitive. • Must not be a QLDB PartiQL reserved word.

Amazon QLDB related information

Important

End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see [Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL](#).

The following related resources can help you as you work with this service.

Topics

- [Technical documentation](#)
- [GitHub repositories](#)
- [AWS blog posts and articles](#)
- [Media](#)
- [General AWS resources](#)

Technical documentation

- [Amazon QLDB FAQs](#) – Frequently asked questions about the product.
- [Amazon QLDB pricing](#) – AWS pricing information and examples.
- [AWS re:Post](#) – AWS community forum for questions and answers (Q&A).
- [Amazon Ion](#) – Developer guides, user guides, and references for the Amazon Ion data format.
- [PartiQL](#) – A specification document and general tutorials for the PartiQL query language.
- [QLDB Workshops](#) – Workshops that provide practical, hands-on examples of using Amazon QLDB to build system-of-record applications, including the following labs:
 - Learning QLDB fundamentals
 - Working with Amazon Ion, and converting Ion to and from JSON (Java)
 - Using AWS Glue and Amazon Athena to enable QLDB data for a data lake
 - Streaming QLDB data to an Amazon Aurora MySQL DB instance
- [Tamper Proof Quality Data Using Amazon QLDB](#) – An [AWS Solutions Implementation](#) that shows how to prevent attackers from tampering with quality data by using QLDB to maintain

an accurate history of data changes. AWS Solutions Implementations help you solve common problems and build faster using AWS.

GitHub repositories

Drivers

- [.NET driver](#) – A .NET implementation of the QLDB driver.
- [Go driver](#) – A Go implementation of the QLDB driver.
- [Java driver](#) – A Java implementation of the QLDB driver.
- [Node.js driver](#) – A Node.js implementation of the QLDB driver.
- [Python driver](#) – A Python implementation of the QLDB driver.

Command line shell

- [QLDB shell](#) – A Python and Rust implementation of a command line interface for the QLDB transactional data API.

Sample applications

- [Java DMV application](#) – A tutorial application that's based on a department of motor vehicles (DMV) use case. It demonstrates basic operations and best practices for using QLDB and the QLDB driver for Java.
- [.NET DMV application](#) – A DMV-based tutorial application that demonstrates basic operations and best practices for using QLDB and the QLDB driver for .NET.
- [Node.js DMV application](#) – A DMV-based tutorial application that demonstrates basic operations and best practices for using QLDB and the QLDB driver for Node.js.
- [Python DMV application](#) – A DMV-based tutorial application that demonstrates basic operations and best practices for using QLDB and the QLDB driver for Python.
- [Ledger loader](#) – A Java framework for asynchronous loading of data into a QLDB ledger at high velocity using a supported delivery channel (AWS DMS, Amazon SQS, Amazon SNS, Kinesis Data Streams, Amazon MSK, or EventBridge).
- [Export processor](#) – An extensible Java framework that handles the work of processing QLDB exports in Amazon S3 by reading the export's output and iterating through the exported blocks in sequence.

- [QLDB streams sample Lambda in Python](#) – An application that demonstrates how to consume QLDB streams by using an AWS Lambda function to send QLDB data to an Amazon SNS topic, which has an Amazon SQS queue subscribed to it.
- [QLDB streams OpenSearch integration sample](#) – A Python application that demonstrates how to integrate Amazon OpenSearch Service with QLDB by using streams.
- [Double-entry application](#) – A Java application that demonstrates how to model a double-entry financial ledger application using QLDB.
- [QLDB KVS for Node.js](#) – A simple key-value store interface library for QLDB with extra functions for document verification.

Amazon Ion and PartiQL

- [Amazon Ion libraries](#) – Ion team supported libraries, tools, and documentation.
- [PartiQL implementation](#) – The implementation, specification, and tutorials for PartiQL.

AWS blog posts and articles

- [How Earnin built their ledger service using Amazon QLDB](#) (February 16, 2023) – Describes how Earnin.com used QLDB to build a ledger service to provide their users with a modern and fully featured mobile financial application.
- [Export and analyze Amazon QLDB journal data using AWS Glue and Amazon Athena](#) (December 19, 2022) – Discusses how you can use the export feature in QLDB with AWS Glue and Athena to provide reporting and analytical capabilities to your ledger-based architectures.
- [How Shinsegae International enhances customer experience and prevents counterfeiting with Amazon QLDB](#) (August 3, 2022) – Describes how Shinsegae International built a digital authenticity verification service using Amazon QLDB to inform customers of the authenticity of luxury goods, and provide purchase and distribution history of products.
- [BungkusIT uses Amazon QLDB and VeriDoc Global's ISV technology to improve the customer and delivery agent experience](#) (April 29, 2022) – Shows how one logistics firm, BungkusIT, used QLDB to implement a centralized, transparent platform for inter-industry communication with an immutable, cryptographically verifiable transaction log.
- [Use Amazon QLDB as an immutable key-value store with a REST API and JSON](#) (February 14, 2022) – Introduces a way to work with QLDB as an immutable key-value store and use its audit features through a REST API.

- [**Building a core banking system with Amazon QLDB**](#) *(January 21, 2022)* – Shows how to use QLDB to build a core banking ledger system. It also shows why QLDB is a good fit-for-purpose database that suits the needs of the financial services industry.
- [**How Specright uses Amazon QLDB to create a traceable supply chain network**](#) *(January 21, 2022)* – Shows how Specright uses QLDB to create a traceable supply chain network that enables wholesale brands, retailers, and manufacturers to share critical supply chain data and packaging specifications data.
- [**How fEMR Delivers Cryptographically Secure and Verifiable Medical Data with Amazon QLDB**](#) *(December 23, 2021)* – Explores how Team fEMR used QLDB and other AWS managed services to enable their relief efforts.
- [**Monitor Amazon QLDB query access patterns**](#) *(November 8, 2021)* – Shows how to associate QLDB transactions and the PartiQL statements that were run in the transactions with the API requests received through Amazon API Gateway.
- [**Build a simple CRUD operation and data stream on Amazon QLDB using AWS Lambda**](#) *(September 28, 2021)* – Shows how to perform CRUD (create, read, update, and delete) operations on QLDB using AWS Lambda functions.
- [**Real-world cryptographic verification with Amazon QLDB**](#) *(August 26, 2021)* – Discusses the value of cryptographic verification in a QLDB ledger in the context of a realistic use case.
- [**Verify delivery conditions with the Accord Project and Amazon QLDB: Part 1, Part 2**](#) *(June 28, 2021)* – Discusses how to apply smart legal contracts technology to verify delivery conditions with the open-source [Accord Project](#) and QLDB.
- [**Amazon QLDB data streaming via AWS CDK**](#) *(June 7, 2021)* – Shows how to use the AWS Cloud Development Kit (AWS CDK) to set up QLDB, populate the QLDB data using AWS Lambda functions, and set up QLDB streaming to provide data resiliency of the ledger data.
- [**Demo fine grained access control in QLDB**](#) *(June 1, 2021)* – Shows how to get started with fine-grained AWS Identity and Access Management (IAM) permissions for a QLDB ledger.
- [**Stream Processing with DynamoDB Streams and QLDB Streams**](#) *(November 23, 2020)* – Provides a brief comparison between DynamoDB streams and QLDB streams, and tips on getting started with them.
- [**Streaming data from Amazon QLDB to OpenSearch**](#) *(August 19, 2020)* – Describes how to stream data from QLDB to Amazon OpenSearch Service to support rich text search and downstream analytics, such as aggregation or metrics across records.

- [How I streamed data from Amazon QLDB to DynamoDB using Nodejs in near-real time](#) (July 7, 2020) – Describes how to stream data from QLDB to DynamoDB to support single-digit latency and infinitely scalable key-value inquiries.
- [Building a GraphQL interface to Amazon QLDB with AWS AppSync: Part 1, Part 2](#) (May 4, 2020) – Discusses how to integrate QLDB and AWS AppSync to provide a versatile, GraphQL-powered API on top of a QLDB ledger.

Media

AWS videos

- [AWS Tutorials & Demos: QLDB to Aurora Streaming](#) (March 17, 2023; 21 minutes) – This video explains the fundamental concepts to implement the QLDB streaming capability, and demonstrates how to set up data streaming from QLDB to an Amazon Aurora MySQL downstream DB.
- [ArcBlock: Leveraging Amazon QLDB to Build a Decentralized Identity Solution](#) (May 31, 2022; 4 minutes) – ArcBlock walks through the decentralized identity solution that they built by using QLDB.
- [AWS re:Invent 2020: Using Amazon QLDB as a system-of-trust database for core business apps](#) (February 5, 2021; 30 minutes) – Learn how early Amazon QLDB users have applied the ledger database's unique properties for data provenance and cryptographic verifiability to implement systems of record with data integrity built in.
- [AWS re:Invent 2020: Building out a serverless application with Amazon QLDB](#) (February 5, 2021; 28 minutes) – Learn how to build, test, and optimize a fully functional serverless application by combining Amazon QLDB with services like AWS Lambda, Amazon Kinesis, and Amazon DynamoDB.
- [AWS re:Invent 2020: Building audit-based apps that maintain data integrity with Amazon QLDB](#) (February 5, 2021; 18 minutes) – This session dives into the problems that Amazon QLDB can solve, answers your questions about when and why you would use a ledger database, and shares use cases from customers such as Osano.
- [How to Centrally Store Immutable Logs using Amazon QLDB with .NET Applications](#) (December 7, 2020; 10 minutes) – A demonstration of how to use QLDB with .NET applications to centrally store immutable logs.
- [Virtual Workshop: Building Ledger-Based Systems of Record with QLDB and Java - AWS Online Tech Talks](#) (July 29, 2020; 87 minutes) – An instructor-led workshop that walks through

the *Working With Ion* Immersion Day lab to build a data loader program using the Amazon Ion library and the QLDB driver for Java.

- [**Developer Workshop: Using AWS's Immutable Ledger Database QLDB on ABT Node**](#) (June 22, 2020; 34 minutes) – A step-by-step guide on how to set up and configure QLDB on ABT Node. It also explains how to install and configure ArcBlock's Blockchain Explorer and Boarding Gate Blocklets to connect to QLDB.
- [**AWS Tech Talk: A Customer's Perspective on Building an Event-Triggered System-of-Record Application with Amazon QLDB**](#) (March 31, 2020; 29 minutes) – A tech talk by Matt Lewis—AWS Data Hero and Chief Architect of the Driver and Vehicle Licensing Agency (DVLA) in the UK—that explains DVLA's use case for QLDB and their application's event-driven architecture.
- [**AWS re:Invent 2019: Why you need a ledger database: BMW, DVLA, & Sage discuss use cases**](#) (December 5, 2019; 47 minutes) – A presentation by customers BMW, the UK government organization DVLA, and Sage that discusses reasons to use a ledger database and shares their use cases for QLDB.
- [**AWS re:Invent 2019: Amazon QLDB: An engineer's deep dive on why this is a game changer**](#) (December 5, 2019; 50 minutes) – A presentation by Andrew Certain (AWS Distinguished Engineer) that discusses the unique, journal-first architecture of QLDB, along with its various innovations. It includes cryptographic hashing, Merkle trees, multiple Availability Zone replication, and PartiQL support.
- [**Building Applications with Amazon QLDB, a First-of-Its-Kind Ledger Database - AWS Online Tech Talks**](#) (November 19, 2019; 51 minutes) – A deep-dive tech talk that describes the unique features of QLDB and specifics about how to use the core functionality. It includes querying your data's complete history, cryptographically verifying documents, and designing a data model.

Podcasts

- [**Why are customers choosing Amazon QLDB?**](#) (July 5, 2020; 33 minutes) – A discussion that explains the definition of a *ledger database*, how it's different from a blockchain, and how customers are using it today.

General AWS resources

- [**Classes & Workshops**](#) – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.

- [AWS Developer Center](#) – Explore tutorials, download tools, and learn about AWS developer events.
- [AWS Developer Tools](#) – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- [Getting Started Resource Center](#) – Learn how to set up your AWS account, join the AWS community, and launch your first application.
- [Hands-On Tutorials](#) – Follow step-by-step tutorials to launch your first application on AWS.
- [AWS Whitepapers](#) – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- [AWS Support Center](#) – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- [Support](#) – The primary webpage for information about Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- [Contact Us](#) – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- [AWS Site Terms](#) – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Release history for Amazon QLDB

The following table describes important changes in each release of Amazon QLDB and the corresponding updates in the *Amazon QLDB Developer Guide*. For notification about updates to this documentation, you can subscribe to the RSS feed.

- **API version:** 2019-01-02
- **Latest documentation update:** January 3, 2023

Change	Description	Date
Service deprecation	End of support notice: Existing customers will be able to use Amazon QLDB until end of support on 07/31/2025. For more details, see Migrate an Amazon QLDB Ledger to Amazon Aurora PostgreSQL .	July 18, 2024
Updated IAM guidance	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM .	January 3, 2023
Update to AWS managed policies	Amazon QLDB updated the existing AWS managed policies <code>AmazonQLDBFullAccess</code> and <code>AmazonQLDBConsoleFullAccess</code> . These policies have a new permission to allow principals to redact document revisions by using a PartiQL stored procedure. For more information, see AWS	November 4, 2022

[managed policies for Amazon QLDB.](#)

[Data redaction](#)

Amazon QLDB now supports the REDACT_REVISION PartiQL stored procedure for ledgers that were created on or after July 22, 2021. Using this stored procedure, you can permanently delete inactive document revisions in history and still maintain the overall data integrity of your ledger. For more information, see [Redacting document revisions](#).

November 3, 2022

[Node.js driver v3](#)

The Amazon QLDB driver for Node.js version 3.0 is now generally available. This version introduces support for the AWS SDK for JavaScript v3. For release notes, see the GitHub repository [awslabs/amazon-qldb-driver-nodejs](#).

September 26, 2022

[Go driver v3](#)

The Amazon QLDB driver for Go version 3.0 is now generally available. This version introduces support for the AWS SDK for Go v2. For release notes, see the GitHub repository [awslabs/amazon-qldb-driver-go](#).

August 11, 2022

[New PartiQL query editor](#)

A new PartiQL query editor on the Amazon QLDB console is now generally available . The new QLDB PartiQL editor provides an improved interface for authoring queries, debugging transactions, and exploring results. For information about opening and using the editor, see [Accessing Amazon QLDB using the console](#).

June 22, 2022

[Ion object mapper for .NET driver](#)

The Amazon QLDB driver for .NET version 1.3 introduces support for the Ion object mapper. This feature lets you completely bypass the need to manually convert between Amazon Ion types and native C# types. For the full change history of the Ion object mapper, see the [CHANGELOG.md](#) file in the GitHub repository `amzn/ion-object-mapper-dotnet` .

January 19, 2022

[JSON journal export format](#)

Amazon QLDB now supports the *JSON Lines* output format for journal exports. For more information, see [Exporting journal data from Amazon QLDB](#).

December 21, 2021

Troubleshooting Amazon QLDB	Added a new Troubleshooting topic that provides guidance for an aggregate list of common errors that you might encounter when using Amazon QLDB.	December 8, 2021
New Region launch	Amazon QLDB is now available in the Canada (Central) Region. For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the <i>Amazon Web Services General Reference</i> .	November 11, 2021
Cross-service confused deputy prevention	Amazon QLDB now supports using the <code>aws:SourceArn</code> and <code>aws:SourceAccount</code> global condition context keys in IAM resource policies to prevent the confused deputy problem. For more information, see Cross-service confused deputy prevention .	November 8, 2021
Amazon QLDB shell v2	Version 2.0 of the Amazon QLDB shell , which is written in Rust, is now generally available. For release notes, see the GitHub repository awslabs/amazon-qlldb-shell .	October 14, 2021

[Update to AWS managed policies](#)

Amazon QLDB updated the existing AWS managed policies `AmazonQLDBFullAccess` and `AmazonQLDBConsoleFullAccess`. These policies have newly added permissions to allow principals to pass any IAM role resource in your account to the QLDB service. This is required for all journal export and stream requests. For more information, see [AWS managed policies for Amazon QLDB](#).

September 2, 2021

[Customer managed AWS KMS keys](#)

Amazon QLDB now supports encryption at rest using customer managed keys in AWS Key Management Service (AWS KMS) for new ledger resources. For more information, see [Encryption at rest in Amazon QLDB](#).

July 22, 2021

[Update to AWS managed policy](#)

Amazon QLDB updated the existing AWS managed policy `AmazonQLDBReadOnly` to remove a duplicate `qldb:GetBlock` action that was previously listed twice. For more information, see [AWS managed policies for Amazon QLDB](#).

July 1, 2021

New Region launch	Amazon QLDB is now available in the Europe (London) Region. For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the <i>Amazon Web Services General Reference</i> .	June 24, 2021
Update to AWS managed policies	Amazon QLDB updated the existing AWS managed policies <code>AmazonQLDBFullAccess</code> and <code>AmazonQLDBConsoleFullAccess</code> . These policies have newly added permissions to allow principals to update the permissions mode in all ledgers, and to run all PartiQL commands in all STANDARD permission ledgers. For more information, see AWS managed policies for Amazon QLDB .	May 27, 2021
Standard permissions mode	Amazon QLDB now supports a STANDARD permissions mode for ledger resources. With the standard permissions mode, you can control access with finer granularity for ledgers, tables, and PartiQL commands. For more information, see Getting started with the standard permissions mode .	May 27, 2021

PartiQL statement statistics	The PartiQL statement statistics feature is now available in the <i>Query editor</i> on the Amazon QLDB console. For more information, see Getting statement statistics .	May 24, 2021
Tutorial: Verifying data using an AWS SDK	Added a step-by-step tutorial with code examples that demonstrate how to verify a revision hash and a block hash in Amazon QLDB using the QLDB API through an AWS SDK. To learn more, see Tutorial: Verifying data using an AWS SDK .	May 6, 2021
.NET driver v1.2	The Amazon QLDB driver for .NET version 1.2 is now generally available. This version introduces asynchronous APIs. For release notes, see the GitHub repository awslabs/amazon-qlldb-driver-dotnet .	April 1, 2021
PartiQL DROP INDEX statement	PartiQL in Amazon QLDB now supports the DROP INDEX statement. For more information, see Dropping indexes .	March 3, 2021

PartiQL statement statistics	The PartiQL statement statistics feature is now available in the latest version of the Amazon QLDB driver for all supported languages , including .NET, Go, and Python. For more information, see Getting statement statistics .	February 25, 2021
TypeScript quick start tutorial	Added TypeScript code examples in the Quick start tutorial for the Amazon QLDB driver for Node.js.	December 28, 2020
PartiQL statement statistics	The latest version of the Amazon QLDB driver for Java and Node.js now provides statement execution statistics that can help you run more efficient PartiQL statements. For more information, see Getting statement statistics .	December 22, 2020
Driver cookbook references and quick start tutorials	Added cookbook references for the Go and Node.js drivers, quick start tutorials for the Java and Python drivers, and a guide for working with Amazon Ion in QLDB. For more information, see Getting started with the Amazon QLDB driver .	November 24, 2020

Amazon QLDB shell v1.1	Version 1.1 of the Amazon QLDB shell is now generally available. For release notes, see the GitHub repository awslabs/amazon-qldb-shell .	October 26, 2020
Amazon QLDB driver for Go	The Amazon QLDB driver for Go is now generally available . This driver is open source on GitHub and enables you to use the AWS SDK for Go to interact with QLDB's transactional data API. For release notes, see the GitHub repository awslabs/amazon-qldb-driver-go .	October 20, 2020
Node.js driver setup recommendations	Added a new section that provides Setup recommendations for the Amazon QLDB driver for Node.js, including how to reduce latency by reusing connections with keep-alive.	October 16, 2020
Index creation on non-empty tables	Amazon QLDB now supports index creation on non-empty tables. For more information, see Managing indexes .	September 30, 2020
Optimizing query performance	Added a new section that describes query constraints in Amazon QLDB and provides guidance for Optimizing query performance given these constraints.	September 18, 2020

Cookbook reference for the Java driver	Added a Cookbook reference that shows code examples of common use cases for the Amazon QLDB driver for Java.	September 3, 2020
Session management with the driver	Added a new section that gives an overview of Session management with the driver in Amazon QLDB , and describes how the QLDB driver handles sessions when running data transactions.	September 1, 2020
Java driver v2.0	The Amazon QLDB driver for Java version 2.0 is now generally available. For release notes, see the GitHub repository awslabs/amazon-qlldb-driver-java .	August 28, 2020
Node.js driver v2.0	The Amazon QLDB driver for Node.js version 2.0 is now generally available. For release notes, see the GitHub repository awslabs/amazon-qlldb-driver-nodejs .	August 27, 2020
Related information	Added a new topic that contains links for Related information and additional resources to help you understand and work with Amazon QLDB.	August 24, 2020

Python driver v3.0	The Amazon QLDB driver for Python version 3.0 is now generally available. For release notes, see the GitHub repository awslabs/amazon-qlldb-driver-python .	August 20, 2020
Amazon QLDB driver for Go	A preview release of the Amazon QLDB driver for Go is now available. This driver enables you to use the AWS SDK for Go to interact with QLDB's transactional data API. For more information, see Amazon QLDB driver for Go (preview) .	August 6, 2020
Cookbook reference for the Python driver	Added a Cookbook reference that shows code examples of common use cases for the Amazon QLDB driver for Python.	July 24, 2020
Unique IDs in Amazon QLDB	Added a new section that describes the properties and usage guidelines of Unique IDs in Amazon QLDB .	July 9, 2020
AWS CloudFormation resource for journal streams	Amazon QLDB now supports a resource for creating journal streams using an AWS CloudFormation template. For more information, see the AWS::QLDB::Stream resource in the <i>AWS CloudFormation User Guide</i> .	July 9, 2020

[Cookbook reference for the .NET driver](#)

Added a [Cookbook](#) reference that shows code examples of common use cases for the Amazon QLDB driver for .NET.

July 1, 2020

[Amazon QLDB driver for .NET](#)

The [Amazon QLDB driver for .NET](#) is now generally available. This driver is open source on GitHub and enables you to use the AWS SDK for .NET to interact with QLDB's transactional data API. For release notes, see the GitHub repository [awslabs/amazon-qldb-driver-dotnet](#).

June 26, 2020

[Amazon QLDB driver for Node.js](#)

The [Amazon QLDB driver for Node.js](#) is now generally available. This driver is open source on GitHub and enables you to use the AWS SDK for JavaScript in Node.js to interact with QLDB's transactional data API. For release notes, see the [awslabs/amazon-qldb-driver-nodejs](#) GitHub repository.

June 5, 2020

[Amazon QLDB journal streams](#)

Amazon QLDB now enables you to create a *stream* that captures every document revision that is committed to your journal and delivers this data to [Amazon Kinesis Data Streams](#) in near-real time. For more information, see [Streaming journal data from Amazon QLDB](#).

May 19, 2020

[Amazon Ion code examples](#)

Added code examples that query and process Amazon Ion data in an Amazon QLDB ledger by using the QLDB driver for Java, Node.js, and Python. For more information, see [Ion code examples in Amazon QLDB](#).

May 12, 2020

[Concurrency model and journal contents](#)

Expanded the [Amazon QLDB concurrency model](#) topic to add information about using indexes to limit *optimistic concurrency control* (OCC) conflicts. Added a new section that describes [Journal contents in Amazon QLDB](#).

May 4, 2020

[Quick start guide for the Node.js driver](#)

Added a [Quick start](#) guide for the Amazon QLDB driver for Node.js.

May 1, 2020

[Amazon QLDB shell](#)

The [Amazon QLDB shell](#) is now generally available. This shell is open source on GitHub and provides a command line interface that enables you to execute PartiQL statements on ledger data. For release notes, see the [awslabs/amazon-qldb-shell](#) GitHub repository.

April 20, 2020

[Compliance certifications](#)

Amazon QLDB is now certified for AWS compliance programs, including HIPAA and ISO. For more information, see [Compliance validation for Amazon QLDB](#).

April 3, 2020

[Expanded driver recommendations](#)

Expanded the [Amazon QLDB driver recommendations](#) section to make it apply to all supported programming languages.

April 2, 2020

[Amazon QLDB shell](#)

A preview release of the Amazon QLDB shell is now available. This shell provides a command line interface that enables you to execute PartiQL statements on ledger data. For more information, see [Accessing Amazon QLDB using the QLDB shell \(data API only\) \(preview\)](#).

March 23, 2020

[Java driver v1.1](#)

The Amazon QLDB driver for Java version 1.1 is now generally available. For release notes, see the [awslabs/amazon-qlldb-driver-java](https://github.com/aws-labs/amazon-qlldb-driver-java) GitHub repository.

March 20, 2020

[Amazon QLDB driver for .NET](#)

Amazon QLDB now provides a preview release of the .NET driver. This driver enables you to use the AWS SDK for .NET to interact with QLDB's transactional data API. For more information, see [Amazon QLDB driver for .NET \(preview\)](#).

March 13, 2020

[Amazon QLDB driver for Python](#)

The [Amazon QLDB driver for Python](#) is now generally available. This driver is open source on GitHub and enables you to use the AWS SDK for Python (Boto3) to interact with QLDB's transactional data API. For release notes, see the [awslabs/amazon-qlldb-driver-python](https://github.com/aws-labs/amazon-qlldb-driver-python) GitHub repository.

March 11, 2020

[PartiQL UNDROP TABLE statement and nested DML](#)

PartiQL in Amazon QLDB now supports *data manipulation language* (DML) statements in which nested collections are specified as sources in the FROM clause. For more information, see the [FROM](#) statement in the *PartiQL reference*. QLDB PartiQL also supports the [UNDROP TABLE](#) statement. For more information, see [Undropping tables](#).

February 26, 2020

[Expanded intro topic](#)

Expanded the introductory *What is Amazon QLDB?* topic to include the new sections [Overview of Amazon QLDB](#) and [From relational to ledger](#).

January 24, 2020

[PartiQL reference for supported functions](#)

Expanded the *Amazon QLDB PartiQL reference* to include detailed usage information about the supported SQL functions. For more information, see [PartiQL functions](#).

January 2, 2020

[Driver recommendations and common errors](#)

Added new sections that describe [Driver recommendations](#) for the Amazon QLDB driver for Java and [Common errors](#) for all driver languages.

January 2, 2020

[New Regions launch](#)

Amazon QLDB is now available in the Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), and Europe (Frankfurt) Regions. For a complete list of available Regions, see [Amazon QLDB endpoints and quotas](#) in the *Amazon Web Services General Reference*.

November 19, 2019

[Amazon QLDB driver for Node.js](#)

Amazon QLDB now provides a preview release of the Node.js driver. This driver enables you to use the AWS SDK for JavaScript in Node.js to interact with QLDB's transactional data API. For more information, see [Amazon QLDB driver for Node.js \(preview\)](#).

November 13, 2019

[Amazon QLDB driver for Python](#)

Amazon QLDB now provides a preview release of the Python driver. This driver enables you to use the AWS SDK for Python (Boto3) to interact with QLDB's transactional data API. For more information, see [Amazon QLDB driver for Python \(preview\)](#).

October 29, 2019

[Public release](#)

This is the initial public release of Amazon QLDB. This release includes the [Developer Guide](#) and the integrated ledger management [API reference](#).

September 10, 2019