Amazon Quantum Ledger Database (Amazon QLDB)

Developer Guide
Table of Contents

What Is Amazon QLDB? ......................................................................................................................... 1
Amazon QLDB Video ................................................................................................................................. 1
Amazon QLDB Pricing .............................................................................................................................. 1
Getting Started with QLDB .................................................................................................................... 1
Overview .................................................................................................................................................. 2
   Journal First ......................................................................................................................................... 2
   Immutable .............................................................................................................................................. 3
   Cryptographically Verifiable ................................................................................................................. 3
   SQL Compatible and Document Flexible .............................................................................................. 4
QLDB Glossary ........................................................................................................................................ 8
Accessing Amazon QLDB ......................................................................................................................... 11
   Signing Up for AWS ............................................................................................................................ 11
   Getting an AWS Access Key ............................................................................................................... 11
   Configuring Your Credentials ........................................................................................................... 12
   Using the Console ............................................................................................................................... 12
   Using the AWS CLI (Control Plane Only) .......................................................................................... 13
      Downloading and Configuring the AWS CLI ................................................................................. 13
   Using the Amazon QLDB Shell (Data Plane Only) (Preview) ............................................................ 14
      Prerequisites ....................................................................................................................................... 14
      Setting Up the Shell .......................................................................................................................... 15
      Invoking the Shell ............................................................................................................................ 15
      Connection Parameters .................................................................................................................. 15
      Exiting the Shell ............................................................................................................................... 16
      Example ............................................................................................................................................ 16
   Using the API ....................................................................................................................................... 17
Getting Started with the Console .................................................................................................................. 18
   Prerequisites and Considerations ....................................................................................................... 18
   Step 1: Create a New Ledger ............................................................................................................... 18
   Step 2: Create Tables, Indexes, and Sample Data .............................................................................. 19
   Step 3: Query the Tables ...................................................................................................................... 25
   Step 4: Modify Documents .................................................................................................................. 25
   Step 5: View the Revision History ...................................................................................................... 27
   Step 6: Verify a Document .................................................................................................................. 29
      To Request a Digest ........................................................................................................................... 29
      To Verify a Document Revision ...................................................................................................... 30
   Step 7: Clean Up .................................................................................................................................. 31
   Next Steps ............................................................................................................................................ 32
Getting Started with the Driver .................................................................................................................. 33
   Java Driver ........................................................................................................................................... 33
      See Also ............................................................................................................................................. 33
      Prerequisites ..................................................................................................................................... 33
      Setting Your AWS Credentials ........................................................................................................ 34
      Setting the AWS Region and Endpoint ........................................................................................... 34
<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTCNOW</td>
<td>323</td>
</tr>
<tr>
<td>Timestamp Format Strings</td>
<td>323</td>
</tr>
<tr>
<td>PartiQL Operators</td>
<td>324</td>
</tr>
<tr>
<td>Arithmetic Operators</td>
<td>324</td>
</tr>
<tr>
<td>Comparison Operators</td>
<td>325</td>
</tr>
<tr>
<td>Logical Operators</td>
<td>325</td>
</tr>
<tr>
<td>String Operators</td>
<td>325</td>
</tr>
<tr>
<td>Reserved Words</td>
<td>326</td>
</tr>
<tr>
<td>API Reference</td>
<td>331</td>
</tr>
<tr>
<td>Actions</td>
<td>331</td>
</tr>
<tr>
<td>Amazon QLDB</td>
<td>332</td>
</tr>
<tr>
<td>Amazon QLDB Session</td>
<td>370</td>
</tr>
<tr>
<td>Data Types</td>
<td>375</td>
</tr>
<tr>
<td>Amazon QLDB</td>
<td>375</td>
</tr>
<tr>
<td>Amazon QLDB Session</td>
<td>381</td>
</tr>
<tr>
<td>Common Errors</td>
<td>397</td>
</tr>
<tr>
<td>Common Parameters</td>
<td>399</td>
</tr>
<tr>
<td>DBQMS API Reference</td>
<td>401</td>
</tr>
<tr>
<td>CreateFavoriteQuery</td>
<td>401</td>
</tr>
<tr>
<td>CreateQueryHistory</td>
<td>401</td>
</tr>
<tr>
<td>DeleteFavoriteQueries</td>
<td>401</td>
</tr>
<tr>
<td>DeleteQueryHistory</td>
<td>401</td>
</tr>
<tr>
<td>DescribeFavoriteQueries</td>
<td>401</td>
</tr>
<tr>
<td>DescribeQueryHistory</td>
<td>401</td>
</tr>
<tr>
<td>GetQueryString</td>
<td>402</td>
</tr>
<tr>
<td>UpdateFavoriteQuery</td>
<td>402</td>
</tr>
<tr>
<td>UpdateQueryHistory</td>
<td>402</td>
</tr>
<tr>
<td>Quotas</td>
<td>403</td>
</tr>
<tr>
<td>Default Quotas</td>
<td>403</td>
</tr>
<tr>
<td>Fixed Quotas</td>
<td>403</td>
</tr>
<tr>
<td>Transaction Size</td>
<td>404</td>
</tr>
<tr>
<td>Naming Constraints</td>
<td>404</td>
</tr>
<tr>
<td>Release History</td>
<td>405</td>
</tr>
</tbody>
</table>
What Is Amazon QLDB?

Amazon Quantum Ledger Database (Amazon QLDB) is a fully managed ledger database owned by a central trusted authority that provides a transparent, immutable, and cryptographically verifiable transaction log of all of your application changes. Amazon QLDB tracks every application data change and maintains a complete and verifiable history of changes over time.

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 cannot be altered, updated, or deleted. And using cryptography, you can easily 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 tracks each application data change and maintains a complete and verifiable history of changes over time.

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 sections:

- Overview of Amazon QLDB (p. 2)—To get a high-level overview of QLDB.
- Core Concepts and Terminology in Amazon QLDB (p. 7)—To learn fundamental QLDB concepts and terminology.
- Accessing Amazon QLDB (p. 11)—To learn how to access QLDB using the AWS Management Console, API, or AWS Command Line Interface (AWS CLI).

To get started quickly with the QLDB console, see Getting Started with the Amazon QLDB Console (p. 18).

To learn about application development with QLDB, see Getting Started with the Amazon QLDB Driver (p. 33).
Overview of Amazon QLDB

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

Topics
- Journal First (p. 2)
- Immutable (p. 3)
- Cryptographically Verifiable (p. 3)
- SQL Compatible and Document Flexible (p. 4)
- Open Source Developer Ecosystem (p. 4)
- Highly Available and Scalable (p. 5)
- Enterprise Grade (p. 5)

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 are not immutable and are not 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 transaction 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.
• In this example, an application connects to a ledger and executes 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 cannot be deleted, modified, or overwritten. There are no APIs or other methods to alter any committed data. This enables you to access and query the full history of your ledger.

QLDB writes one or more blocks 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 executed 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. This feature enables the journal 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 were executed in the transaction.

For more information, see Data Verification in Amazon QLDB (p. 205).

SQL Compatible 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 PartiQL, see the Amazon QLDB PartiQL Reference (p. 272).

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, see the Amazon QLDB Data Format Reference (p. 269).

For a high-level comparison of the core components and features in traditional relational databases versus QLDB, see From Relational to Ledger (p. 5).

Open Source Developer Ecosystem

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 (p. 33).

Amazon Ion also provides client libraries that serialize and deserialize Ion data for you. For code examples of common use cases for processing Ion data, see the Amazon Ion Cookbook on GitHub.
Highly Available and Scalable

QLDB is 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 ACID (atomicity, consistency, isolation, and durability) 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 (p. 200).

From Relational to Ledger

If you are 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.

![Diagram showing mapping constructs between Relational and QLDB](image-url)
The following table shows the primary high-level similarities and differences of built-in operational features between a traditional RDBMS and QLDB.

<table>
<thead>
<tr>
<th>Operation</th>
<th>RDBMS</th>
<th>QLDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating tables</td>
<td>CREATE TABLE statement that defines all column names and data types</td>
<td>CREATE TABLE statement that doesn't define any table attributes or data types to allow schemaless and open content</td>
</tr>
<tr>
<td>Creating indexes</td>
<td>CREATE INDEX statement</td>
<td>CREATE INDEX statement that can be executed on empty tables and on a single field</td>
</tr>
<tr>
<td>Inserting data</td>
<td>INSERT statement that specifies values within a new row or tuple that adheres to the schema as defined by the table</td>
<td>INSERT statement that specifies values within a new document in any valid Amazon Ion format regardless of the existing documents in the table</td>
</tr>
<tr>
<td>Querying data</td>
<td>SELECT-FROM-WHERE statement</td>
<td>SELECT-FROM-WHERE statement in the same syntax as SQL when querying flat documents</td>
</tr>
<tr>
<td>Updating data</td>
<td>UPDATE-SET-WHERE statement</td>
<td>UPDATE-SET-WHERE statement in the same syntax as SQL when updating flat documents</td>
</tr>
<tr>
<td>Deleting data</td>
<td>DELETE-FROM-WHERE statement</td>
<td>DELETE-FROM-WHERE statement in the same syntax as SQL when deleting flat documents</td>
</tr>
<tr>
<td>Nested and semistructured data</td>
<td>Flat rows or tuples only</td>
<td>Documents that can have any structured, semistructured, or nested data as supported by the Amazon Ion data format and the PartiQL query language</td>
</tr>
<tr>
<td>Querying metadata</td>
<td>No built-in metadata</td>
<td>SELECT statement that queries from the built-in committed view of a table</td>
</tr>
<tr>
<td>Querying revision history</td>
<td>No built-in data history</td>
<td>SELECT statement that queries from the built-in history function</td>
</tr>
<tr>
<td>Cryptographic verification</td>
<td>No built-in cryptography or immutability</td>
<td>APIs that return a digest of a journal and a proof that verifies the integrity of any document revision relative to that digest</td>
</tr>
</tbody>
</table>

For an overview of the core concepts and terminology in QLDB, see Core Concepts (p. 7).

For detailed information about the process of creating, querying, and managing your data on a ledger, see Working with Data and History (p. 187).
Core Concepts and Terminology in Amazon QLDB

This section provides an overview of the core concepts and terminology in Amazon QLDB, including ledger structure and how a ledger manages data. After you read this section, see Working with Data and History (p. 187) and follow the examples that walk you through the process of creating tables, inserting data, and running basic queries.

As a ledger database, QLDB differs from other document-based databases when it comes to the following key concepts.

Ledger Structure

Fundamentally, QLDB data is organized into tables of Amazon Ion (p. 269) documents. More precisely, tables are collections of document revisions. A document revision represents a single iteration of the document's full dataset. Because QLDB stores the complete change history of your data, a table contains not only the latest revision of its documents, but also all prior iterations. For details on revisions, see Updating and Deleting Documents (p. 196) and Querying Revision History (p. 196).

Note
For brevity, this guide often refers to document revisions as simply documents. This shorthand term keeps a consistent meaning because we typically think in terms of documents when inserting, updating, and deleting elements of a collection. However, when querying document history, it's important to specify that we are referring to revisions.

Write Transactions

When an application needs to modify data in a document, it does so in a database transaction. Within a transaction, data is read from the ledger, updated, and committed (p. 200) to the journal. The journal represents a complete and immutable history of all the changes to your data. QLDB writes one or more chained blocks 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 (p. 272) statements that committed them.

The following diagram illustrates this journal structure.
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 (p. 205). 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.

**Querying Your Data**

QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. A ledger provides queryable views of your data based on the transaction information that is committed to the journal. Similar to views in relational databases, a view in QLDB is a projection 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 views using PartiQL `SELECT` statements:

- **User**—The latest non-deleted revision of your application-defined data only. This is the default view in QLDB.
- **Committed**—The latest non-deleted revision of both your data and the system-generated metadata. This is the full system-defined table that corresponds directly to your user table.

In addition to these views, you can query the revision history of your data by using the built-in History Function (p. 197). The history function returns both your 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 is materialized into the tables that you define. These tables enable faster and more efficient queries. When an application reads data, it accesses the tables and indexes that are stored in your indexed storage.

**Next Steps**

To help you understand how to use a ledger with your data, see Working with Data and History in Amazon QLDB (p. 187). This guide explains how these concepts work in QLDB, 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 (p. 18).

For a list of the key terms and definitions described in this section, see the Amazon QLDB Glossary (p. 8).

**Amazon QLDB Glossary**

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

An object that is committed to the journal in a transaction. A single transaction writes one or more blocks in the journal, but 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 (p. 272) statements that committed them.

Each block also has a hash value for verification. A block hash consists of the full hash chain of every entry 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 generated from your ledger’s full hash chain as of the latest committed block in the journal at that time.

QLDB enables you to 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 (p. 269) 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 revision

A structure that represents a single iteration of a document's full dataset. A revision includes both your application-defined data and QLDB-generated metadata. Each revision is stored in 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 consists of the full hash chain of every revision and statement within that entry combined with the hash of the previous chained entry.

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 storage

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

journal strand

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

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. A proof enables you to verify the integrity of your revisions relative to the digest. For more information, see Data Verification in Amazon QLDB (p. 205).

table

An unordered collection of document revisions.
view

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

You can access Amazon QLDB using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the QLDB API. Before accessing QLDB, you must do the following:

1. **Sign up for AWS.** (p. 11)
2. Use AWS Identity and Access Management (IAM) to manage access to QLDB. For information, see *How Amazon QLDB Works with IAM* (p. 239).
3. **Get an AWS access key** (p. 11) (used to access QLDB programmatically).
   
   **Note**
   
   If you plan to interact with QLDB only through the console, you don't need an AWS access key, and you can skip ahead to **Accessing Amazon QLDB Using the Console** (p. 12).
4. **Configure your credentials** (p. 12) (used to access QLDB programmatically).

After completing these steps, see the following topics to learn more about how to access QLDB:

- Using the Console (p. 12)
- Using the AWS CLI (Control Plane Only) (p. 13)
- Using the Amazon QLDB Shell (Data Plane Only) (Preview) (p. 14)
- Using the API (p. 17)

Signing Up for AWS

To use the QLDB service, you must have an AWS account. If you don't already have an account, you are prompted to create one when you sign up. You're not charged for any AWS services that you sign up for unless you use them.

**To sign up for AWS**

2. Follow the online instructions.
   
   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Getting an AWS Access Key

Before you can access Amazon QLDB programmatically or through the AWS CLI, you must have an AWS access key. You don't need an access key if you plan to use the QLDB console only.

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, create a new administrator IAM user with access keys for yourself.

The only time that you can view or download the secret access key is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see *Permissions Required to Access IAM Resources* in the *IAM User Guide*. 

To create access keys for an IAM user

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, choose Users.
3. Choose the name of the user whose access keys you want to create, and then choose the Security credentials tab.
4. In the Access keys section, choose Create access key.
5. To view the new access key pair, choose Show. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:
   - Access key ID: AKIAIOSFODNN7EXAMPLE
   - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose Download .csv file. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

   Keep the keys confidential in order to protect your AWS account and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.
7. After you download the .csv file, choose Close. When you create an access key, the key pair is active by default, and you can use the pair right away.

Related topics
- What Is IAM? in the IAM User Guide
- AWS Security Credentials in AWS General Reference

Configuring Your Credentials

Before you can access Amazon QLDB programmatically or through the AWS CLI, you must configure your credentials to enable authorization for your applications.

There are several ways to do this, including the following methods:
- Manually create a credentials file to store your AWS access key ID and secret access key.
- Use the aws configure command of the AWS CLI to automatically create the file.
  To install and configure the AWS CLI, see Accessing Amazon QLDB Using the AWS CLI (Control Plane Only) (p. 13).
- Use environment variables.

For instructions on configuring your credentials, see Getting Started with the Amazon QLDB Driver (p. 33). Each programming language has specific prerequisites with links to instructions in their respective AWS SDK developer guide.

Accessing Amazon QLDB Using the Console

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

You can use the console to do the following in QLDB:
• Create, delete, describe, and list ledgers.
• Create a table and indexes.
• Insert, update, and delete data.
• Run queries.
• Cryptographically verify documents.
• Export 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 (p. 18).

Accessing Amazon QLDB Using the AWS CLI (Control Plane Only)

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 ad hoc operations. You can also use it to embed Amazon QLDB operations within utility scripts.

Before you can use the AWS CLI with QLDB, you must get an access key ID and secret access key. For more information, see Getting an AWS Access Key (p. 11).

For a complete listing 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 control plane API operations that are listed in the Amazon QLDB API Reference (p. 331). These APIs are used only for managing ledgers and for non-transactional data operations.
To execute data transactions with the qldb-session API using a command line interface, see Accessing Amazon QLDB Using the QLDB Shell (Data Plane Only) (Preview) (p. 14).

Topics
• Downloading and Configuring the AWS CLI (p. 13)
• Using the AWS CLI with QLDB (p. 13)

Downloading and Configuring the AWS CLI

The AWS CLI runs on Windows, macOS, or Linux. To download, install, and configure it, follow these steps:

2. Follow the instructions for Installing the AWS CLI and Configuring the AWS CLI in the AWS Command Line Interface User Guide.

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:
Accessing Amazon QLDB Using the QLDB Shell (Data Plane Only) (Preview)

This is prerelease documentation for a feature in preview release. It is subject to change.

Amazon QLDB provides a command line shell for interaction with the transactional data plane. This shell is written in Python and enables you to execute PartiQL (p. 272) statements on ledger data.

**Note**
The Amazon QLDB Shell only supports the `qldb-session` transactional data API. This API is used only for executing PartiQL statements on a QLDB ledger.

To interact with the `qldb` control plane APIs using a command line interface, see Accessing Amazon QLDB Using the AWS CLI (Control Plane Only) (p. 13).

This tool is not intended to be incorporated into an application or adopted for production purposes. The objective of this tool is to give developers, DevOps, database administrators, and others who are interested the opportunity to rapidly experiment with QLDB and PartiQL.

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

**Topics**
- Prerequisites (p. 14)
- Setting Up the Shell (p. 15)
- Invoking the Shell (p. 15)
- Connection Parameters (p. 15)
- Exiting the Shell (p. 16)
- Example (p. 16)

**Prerequisites**

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

1. Follow the AWS setup instructions in Accessing Amazon QLDB (p. 11). This includes signing up for AWS and getting an AWS access key for development.
2. Install Python version 3.4 or later from the Python downloads site.
3. Set up your AWS credentials and your default AWS Region. For instructions, see Quickstart in the AWS SDK for Python (Boto 3) documentation.
Setting Up the Shell

To install the QLDB Shell from PyPI using `pip3` (a package manager for Python 3), run the following at your command line terminal.

```
$ pip3 install qldbshell
```

Installing the shell also installs the following required package dependencies:

- pyqldb
- amazon.ion
- boto3

Invoking the Shell

To invoke the QLDB Shell on your command line terminal for a specific ledger, run the following command. Replace `test-ledger` with your ledger name.

```
$ qldbshell --ledger test-ledger
```

**Note**

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

Each successive line that you enter in your `qldbshell` session is treated as a separate PartiQL statement. In the current version of the QLDB Shell, a PartiQL statement cannot span more than one line.

Connection Parameters

To see a list of available input arguments, run the following command before you invoke a `qldbshell` session.

```
$ qldbshell --help
```

The following connection parameters are available for the `qldbshell` command. You can add the optional arguments to override the AWS Region, credentials profile, and endpoint that the shell uses.

**Usage syntax:**

```
$ qldbshell -l LEDGER_NAME [-v] [-p PROFILE] [-r REGION_CODE] [-s QLDB_SESSION_ENDPOINT]
```

**Required arguments**

- `-l LEDGER_NAME`, `--ledger LEDGER_NAME`

  Specifies the name of the ledger that the shell connects to. The ledger name must already exist and must be active.
Optional arguments

-\(v\), --verbose

   Increases output verbosity in your shell session.

-\(p\) PROFILE, --profile PROFILE

   Specifies the location of your AWS credentials profile that the shell uses for authentication.
   
   If not provided, the shell uses your default AWS profile, which is located at ~/.aws/credentials.

-\(r\) REGION_CODE, --region REGION_CODE

   Specifies the AWS Region code of the QLDB ledger that the shell connects 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 QLDB_SESSION_ENDPOINT

   Specifies the qldb-session API endpoint that the shell connects to.
   
   For a complete list of available QLDB Regions and endpoints, see Amazon QLDB Endpoints and Quotas in the AWS General Reference.

Parameter File

As an alternative to passing arguments on the command line, you can also save input parameters in a file. Put each parameter on a separate line. Then, pass the file name to the qldbshell as follows.

```
$ qldbshell @params.conf
```

Exiting the Shell

To exit the current qldbshell session and close the current ledger database connection, run the exit command.

```
qldbshell > exit
```

Example

For information about writing PartiQL statements in QLDB, see the Amazon QLDB PartiQL Reference (p. 272).

Example

The following example shows a common sequence of basic commands.

Note

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

```
$ qldbshell --ledger test-ledger --region us-east-1
qldbshell > CREATE TABLE TestTable
qldbshell > INSERT INTO TestTable `{"Name": "John Doe"}`
qldbshell > SELECT * FROM TestTable
```
Accessing Amazon QLDB Using the API

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 and its underlying AWS SDK to interact with your ledger using the APIs.

The QLDB Driver provides support for QLDB in Java, .NET, Node.js, and Python. To get started quickly with these languages, see Getting Started with the Amazon QLDB Driver (p. 33).

Before you can use the QLDB Driver in your application, you must get an AWS access key ID and secret access key. For more information, see Getting an AWS Access Key (p. 11).
Getting Started with the Amazon QLDB Console

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. QLDB enables you to track and verify the full history of changes in an easily accessible and flexible database.

As you work through this tutorial, the following sections explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to cryptographically verify a registration document, 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 (p. 18)
- Step 1: Create a New Ledger (p. 18)
- Step 2: Create Tables, Indexes, and Sample Data (p. 19)
- Step 3: Query the Tables in a Ledger (p. 25)
- Step 4: Modify Documents in a Ledger (p. 25)
- Step 5: View the Revision History for a Document (p. 27)
- Step 6: Verify a Document in a Ledger (p. 29)
- Step 7 (Optional): Clean Up Resources (p. 31)
- Getting Started with Amazon QLDB: Next Steps (p. 32)

**Tutorial Prerequisites and Considerations**

Before you start this tutorial, you must follow the AWS setup instructions in Accessing Amazon QLDB (p. 11). These steps include signing up for AWS and creating an AWS Identity and Access Management (IAM) user with QLDB access.

**Step 1: Create a New Ledger**

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.
   - 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 (p. 232).

   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 (p. 19).

Step 2: Create Tables, Indexes, and Sample Data

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>
<thead>
<tr>
<th>Table Name</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleRegistration</td>
<td>VIN</td>
</tr>
<tr>
<td>VehicleRegistration</td>
<td>LicensePlateNumber</td>
</tr>
<tr>
<td>Vehicle</td>
<td>VIN</td>
</tr>
<tr>
<td>Person</td>
<td>GovId</td>
</tr>
</tbody>
</table>
Step 2: Create Tables, Indexes, and Sample Data

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriversLicense</td>
<td>LicenseNumber</td>
</tr>
<tr>
<td>DriversLicense</td>
<td>PersonId</td>
</tr>
</tbody>
</table>

You can use the QLDB console to automatically create these tables with indexes and load them with sample data. Or, you can use the Query editor on the console to manually run each PartiQL (p. 272) statement step-by-step.

### Automatic Option

**To create tables, indexes, and sample data**

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.

**Note**

As a best practice, use this document id metadata field as a foreign key. For more information, see **Querying Document Metadata** (p. 193).

**To create tables, indexes, and sample data**

2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.

4. Start by creating four tables. QLDB supports open content and does not enforce schema, so you don't specify attributes or data types.

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

```sql
CREATE TABLE VehicleRegistration
```

Repeat this step for each of the following.

```sql
CREATE TABLE Vehicle
CREATE TABLE Person
CREATE TABLE DriversLicense
```

5. Next, create indexes that help speed up queries against each table.

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

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

Repeat this step for the following.

```sql
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
CREATE INDEX ON Vehicle (VIN)
CREATE INDEX ON Person (GovId)
CREATE INDEX ON DriversLicense (LicenseNumber)
CREATE INDEX ON DriversLicense (PersonId)
```

6. 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.

```sql
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',
```
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
<<
{ 'LicenseNumber' : 'LEWISR261LL',
  'LicenseType' : 'Learner',
  'ValidFromDate' : '2016-12-20T',
  'ValidToDate' : '2020-11-15T',
  'PersonId' : '' },
{ 'LicenseNumber' : 'LOGANB486CG',
  'LicenseType' : 'Probationary',
  'ValidFromDate' : '2016-04-06T',
  'ValidToDate' : '2020-11-15T',
  'PersonId' : '' },
{ 'LicenseNumber' : '744 849 301',
  'LicenseType' : 'Full',
  'ValidFromDate' : '2017-12-06T',
  'ValidToDate' : '2022-10-15T',
  'PersonId' : '' },
{ 'LicenseNumber' : 'P626-168-229-765',
  'LicenseType' : 'Learner',
  'ValidFromDate' : '2017-08-16T',
  'ValidToDate' : '2021-11-15T',
  'PersonId' : '' },
{ 'LicenseNumber' : 'S152-780-97-415-0',
  'LicenseType' : 'Passport',
  'ValidFromDate' : '2017-11-15T',
  'ValidToDate' : '2022-11-15T',
  'PersonId' : '' },
}
>>
```
8. 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.

```sql
INSERT INTO VehicleRegistration
<<
{ 'VIN': '1N4AL11D75C109151',
  'LicensePlateNumber': 'LEWISR261LL',
  'State': 'WA',
  'City': 'Seattle',
  'PendingPenaltyTicketAmount': 90.25,
  'ValidFrom': '2017-08-21T00:00:00Z',
  'ValidTo': '2020-05-11T00:00:00Z',
  'Owners': {
    'PrimaryOwner': { 'PersonId': '' },
    'SecondaryOwners': []
  } },
{ 'VIN': 'KM8SRDF6EU074761',
  'LicensePlateNumber': 'CA762X',
  'State': 'WA',
  'City': 'Kent',
  'PendingPenaltyTicketAmount': 130.75,
  'ValidFrom': '2017-09-14T00:00:00Z',
  'ValidTo': '2020-06-25T00:00:00Z',
  'Owners': {
    'PrimaryOwner': { 'PersonId': '' },
    'SecondaryOwners': []
  } },
{ 'VIN': '3HPGK5GF53FM761765',
  'LicensePlateNumber': 'CD892Z',
  'State': 'WA',
  'City': 'Everett',
  'PendingPenaltyTicketAmount': 442.30,
  'ValidFrom': '2011-03-17T00:00:00Z',
  'ValidTo': '2021-03-24T00:00:00Z',
  'Owners': {
    'PrimaryOwner': { 'PersonId': '' },
    'SecondaryOwners': []
  } },
{ 'VIN': '1HVBAAANXWH544237',
  'LicensePlateNumber': 'LS572D',
  'State': 'WA',
  'City': 'Tacoma',
  'PendingPenaltyTicketAmount': 42.20,
  'ValidFrom': '2011-10-26T00:00:00Z',
  'ValidTo': '2023-09-25T00:00:00Z',
  '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' : '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 (p. 25).
Step 3: Query the Tables in a Ledger

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.

To query the tables

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose Query 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.

   ```sql
   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.

   ```sql
   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.

   ```sql
   SELECT * FROM Person AS p, DriversLicense as l
   WHERE p.GovId = l.LicenseNumber
   ```

To learn about modifying documents in the tables in the vehicle-registration ledger, see Step 4: Modify Documents in a Ledger (p. 25).

Step 4: Modify Documents in a Ledger

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.

**Important**
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 and Terminology in Amazon QLDB (p. 7). To learn more about metadata, see Querying Document Metadata (p. 193).

To modify documents
1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose Query 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.

   ```sql
   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.

   ```sql
   UPDATE VehicleRegistration AS r
   SET r.Owners.PrimaryOwner.PersonId = '294jJ3YUoH1IEEm8GSabOs' --replace with your id
   WHERE r.VIN = '1N4AL11D75C109151'
   ```

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

   ```sql
   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.

   ```sql
   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.
Step 5: View the Revision History for a Document

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 (p. 197).

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 Query 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 (p. 195).
In the query editor window, enter the following statement, and then choose **Run**.

```sql
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 in the following syntax.

```sql
SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h WHERE h.metadata.id = 'id'
```

- Best practice is to qualify a history query with a document `id`. This helps to avoid inefficient queries.
- The `start-time` and `end-time` are optional parameters that are both inclusive. They must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC).
- The `start-time` and `end-time` are Amazon Ion literal values that can be denoted with backticks (`...`). To learn more, see Querying Ion with PartiQL (p. 289).

The history function returns documents with the same schema as the committed view. For example, enter the following statement, and then choose **Run**. Be sure to replace the `id` value with your own document ID as appropriate.

```sql
SELECT h.data.VIN, h.data.City, h.data.Owners FROM history(VehicleRegistration) AS h WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
```

This first history query projects your modified vehicle registration data. The output includes all revisions of the document and should look similar to the following.

<table>
<thead>
<tr>
<th>VIN</th>
<th>City</th>
<th>Owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>&quot;Seattle&quot;</td>
<td>{PrimaryOwner: {PersonId:&quot;&quot;}, SecondaryOwners:[]}</td>
</tr>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>&quot;Seattle&quot;</td>
<td>{PrimaryOwner: {PersonId:&quot;294jJ3YUoH1IEEm8GSabOs&quot;}, SecondaryOwners:[]}</td>
</tr>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>&quot;Everett&quot;</td>
<td>{PrimaryOwner: {PersonId:&quot;7NmE8YLPbXc0IqesJy1rpR&quot;}, SecondaryOwners:[]}</td>
</tr>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>&quot;Everett&quot;</td>
<td>{PrimaryOwner: {PersonId:&quot;7NmE8YLPbXc0IqesJy1rpR&quot;}, SecondaryOwners: [{PersonId:&quot;5Ufgd1nK06gF5CWc0Iu64s&quot;}]}</td>
</tr>
</tbody>
</table>

**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 (p. 25).

6. 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.
Step 6: Verify a Document

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 (p. 205).

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 (p. 205) 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 (p. 269) 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.

   ```json
   {
   "digest": "42zaJOfV8iGutVGNaIuzgWhD5Xb/5B91ScHnvxFXm9E=",
   "digestTipAddress": "{strandId:"BlFTjlSXze9B1h1KOSzcE3",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 as a fingerprint 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 real scenario for verification, you should use a digest that was previously saved rather than doing the two tasks consecutively. Ideally, the digest is saved soon after the document revision that you want to verify is written to the ledger.

**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 identifier. The **blockAddress** specifies the block location where the revision was committed.

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

2. Choose the **vehicle-registration** ledger.

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

   ```sql
   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 (p. 212).

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** (p. 31).

---

**Step 7 (Optional): Clean Up Resources**

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 or the AWS Command Line Interface (AWS CLI). The QLDB console disables deletion protection for you when you use it to delete a ledger.

**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. In the list of ledgers, select `vehicle-registration`.
4. Choose **Delete ledger**. Confirm this by entering `vehicle-registration` in the field provided.
5. If deletion protection is enabled for this ledger, you must also choose the option to **Override deletion protection**.

To learn more about working with ledgers in QLDB, see Getting Started with Amazon QLDB: Next Steps (p. 32).
Getting Started with Amazon QLDB: Next Steps

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

- Working with Data and History in Amazon QLDB (p. 187)
- Getting Started with the Amazon QLDB Driver (p. 33)
- Amazon QLDB Concurrency Model (p. 200)
- Data Verification in Amazon QLDB (p. 205)
- Exporting Your Journal in Amazon QLDB (p. 216)
- Amazon QLDB PartiQL Reference (p. 272)
Getting Started with the Amazon QLDB Driver

This section contains hands-on tutorials to help you learn about developing with Amazon QLDB using the QLDB Driver. The driver and its underlying AWS SDK enable your application to interact with QLDB using both the control plane API (qldb) and the transactional data API (qldbsession).

The driver provides a high-level abstraction layer over the transactional data plane. It streamlines the execution of PartiQL (p. 272) statements on ledger data by managing SendCommand API calls for you. The driver is open-sourced on GitHub and is available for the following programming languages.

Topics
- Amazon QLDB Driver for Java (p. 33)
- Amazon QLDB Driver for .NET (Preview) (p. 101)
- Amazon QLDB Driver for Node.js (Preview) (p. 104)
- Amazon QLDB Driver for Python (p. 146)
- Common Errors for the Amazon QLDB Driver (p. 185)

Amazon QLDB Driver for Java

To work with data in your ledger, you can connect to Amazon QLDB from your Java application using an AWS-provided driver. Then you can code your application to use the transactional data API. The following sections describe how to get started with the QLDB Driver for Java.

Topics
- See Also (p. 33)
- Prerequisites (p. 33)
- Setting Your AWS Credentials (p. 34)
- Setting the AWS Region and Endpoint (p. 34)
- Quick Start Installation (Driver Only) (p. 35)
- Amazon QLDB Java Tutorial (p. 36)
- Best Practices for the Amazon QLDB Java Driver (p. 100)

See Also

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

- API Reference
- Common Errors (p. 185)
- Driver Source Code
- Sample Application Source Code
- Amazon Ion Cookbook

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 (p. 11).

2. Choose a Java integrated development environment (IDE):
   - **AWS Cloud9 (recommended)** – Use AWS Cloud9 as your Java IDE and automate setup of the QLDB sample application with a provided AWS CloudFormation template. This option requires the least amount of manual installation and configuration work.
     
     For this option, skip the rest of this section and proceed to the Java Tutorial (p. 36). Then, follow the AWS Cloud9 steps in Installing the Sample Application (p. 37).
   - **Other IDE** – Use your own development resources to manually install your preferred IDE.
     
     For this option, continue the following setup steps.

3. Set up a Java development environment by downloading and installing the following:
   - Java SE Development Kit 8 (such as Amazon Corretto 8).
   - Java IDE (such as Eclipse or IntelliJ).

4. Configure your AWS credentials and Region for development:
   - Set up your AWS security credentials (p. 34) for use with the AWS SDK for Java.
   - Set your AWS Region (p. 34) to determine your default QLDB endpoint.

5. Next, you can download the complete tutorial sample application, or you can install only the driver in an existing Java project.
   - To run the complete tutorial with the QLDB sample application, proceed to the Java Tutorial (p. 36).
   - To install only the QLDB Driver and the AWS SDK for Java in an existing project, see Quick Start Installation (Driver Only) (p. 35).

## Setting Your AWS Credentials

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 are using an AWS credentials file, as described in Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.

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 = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

## Setting the AWS Region and Endpoint

As part of the previous step to set up your AWS credentials, you should also set your default AWS Region. The sample application connects to QLDB in your default AWS Region. You can also change the Region in your code by modifying the AmazonQldb client object (control plane) or the AmazonQldbSessionClientBuilder object (transactional data plane).

The following code example instantiates a new AmazonQldb client object.

```java
import com.amazonaws.regions.Regions;
import com.amazonaws.services.qldb.AmazonQldb;
import com.amazonaws.services.qldb.AmazonQldbClientBuilder;

// This client will default to US East (Ohio)
```
You can use the `withRegion` method to run your code against QLDB in any Region where it is available. For a complete list, see Amazon QLDB Endpoints and Quotas in the Amazon Web Services General Reference.

Quick Start Installation (Driver Only)

To install only the QLDB Driver and the AWS SDK for Java, add the following artifacts as dependencies in your Java project:

1. `amazon-qldb-driver-java` – The QLDB Driver for Java. The latest version is 1.1.0.
   This artifact automatically includes the AWS SDK for Java core module, Amazon Ion (p. 269) libraries, and other required dependencies.

2. `aws-java-sdk-qldb` – The QLDB module of the AWS SDK for Java. The minimum QLDB-supported version is 1.11.649.
   This module enables your application to interact with the control plane API operations listed in the Amazon QLDB API Reference (p. 331).

3. `jackson-dataformat-ion` – (Optional) The open source Jackson data format module for Ion. You need this artifact to run the code in the Java Tutorial (p. 36). The sample application requires version 2.10.0.pr1 or later.
   If you are using your own application to interact with QLDB, you don't need to add this Jackson dependency to your project.

For example, if you use Maven or Gradle as your build tool, add the following dependencies in your `build.gradle` (Gradle) or `pom.xml` (Maven) configuration file.

**Gradle**

```gradle
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version: '1.1.0'
    compile group: 'com.amazonaws', name: 'aws-java-sdk-qldb', version: '1.11.649'
    compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-ion', version: '2.10.0.pr1'
}
```

**Maven**

```xml
<dependencies>
<dependency>
  <groupId>software.amazon.qldb</groupId>
  <artifactId>amazon-qldb-driver-java</artifactId>
  <version>1.1.0</version>
</dependency>
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk-qldb</artifactId>
  <version>1.11.649</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-ion</artifactId>
  <version>2.10.0.pr1</version>
</dependency>
```
For code examples that demonstrate operations such as how to create a ledger and execute data transactions on a ledger, see the Java Tutorial (p. 36).

Amazon QLDB Java Tutorial

In this tutorial, you use the QLDB Driver for Java to create an Amazon QLDB ledger and populate it with tables and sample data. The driver enables your application to interact with QLDB using the transactional data 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 sections 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.

As you work through this tutorial, you can refer to the AWS SDK for Java API Reference.

Prerequisites

Before you get started, make sure that you complete the Prerequisites (p. 33). This includes signing up for AWS and, if you're using an IDE other than AWS Cloud9, getting an AWS access key for development.

Topics

- Installing the Amazon QLDB Java Sample Application (p. 36)
- Step 1: Create a New Ledger (p. 39)
- Step 2: Test Connectivity to the Ledger (p. 44)
- Step 3: Create Tables, Indexes, and Sample Data (p. 46)
- Step 4: Query the Tables in a Ledger (p. 67)
- Step 5: Modify Documents in a Ledger (p. 70)
- Step 6: View the Revision History for a Document (p. 76)
- Step 7: Verify a Document in a Ledger (p. 78)
- Step 8 (Optional): Clean Up Resources (p. 98)

Installing the Amazon QLDB Java Sample Application

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.

Prerequisites

Before you get started, make sure that you complete the Prerequisites (p. 33). This includes signing up for AWS and, if you're using an IDE other than AWS Cloud9, getting an AWS access key for development.

Installation

The following steps describe how to download and set up the sample application with a local development environment, or how to automate setup of the sample application by using AWS Cloud9 as your IDE.
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.

```bash
git clone -b v1.1.0 git@github.com:aws-samples/amazon-qldb-dmv-sample-java.git
```

This package includes the Gradle configuration and complete code from the Java Tutorial (p. 36).

2. Load and run the provided application.
   - If you are 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 are using IntelliJ:
     a. Start IntelliJ, and on the IntelliJ menu, choose File and then Open.
     b. From 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 'ListLedgers'.

3. Proceed to Step 1: Create a New Ledger (p. 39) 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 (p. 37)
- Part 2: Set Up Your IDE (p. 38)
- Part 3: Run the QLDB DMV Sample Application (p. 39)

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

Switch to a Region that supports QLDB. For a complete list, see Amazon QLDB Endpoints and Quotas in the AWS General Reference.

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.


5. Enter a Stack name (such as qldb-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 amazon-qldb-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

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.

   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 the `dmv-setup.sh` script to install OpenJDK 8. In the AWS Cloud9 terminal that you created in the previous step, run the following commands in order:

```
aws s3 cp s3://amazon-qldb-assets/setup-scripts/dmv-setup.sh .
sh dmv-setup.sh
```

5. Upon completion, you should see the following message printed in the terminal:

```
** DMV Sample App setup completed , enjoy!! **
```

6. 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 are 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 (p. 39) 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](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**

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.

```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.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 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 IonObjectMapper MAPPER = new 
        IonValueMapper(IonSystemBuilder.standard().build());
    public static final IonSystem SYSTEM = IonSystemBuilder.standard().build();

    private Constants() { }

    static {
        MAPPER.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    }
}
```
Note
This Constants class also includes an instance of the open source Jackson IonValueMapper class. You can use this mapper to process your Amazon Ion (p. 269) 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.

```java
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;

public final class DescribeLedger {
    public static AmazonQLDB client = CreateLedger.getClient();
    public static final Logger log = LoggerFactory.getLogger(DescribeLedger.class);

    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.
     */
    public static void describe(String name) {
        DescribeLedgerRequest describeLedgerRequest = new DescribeLedgerRequest(name);
        DescribeLedgerResult result = client.describeLedger(describeLedgerRequest);
    }
}
```
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.

```java
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;

/**
 * Build a low-level QLDB client.
 * *
 * @return {link AmazonQLDB} control plane client.
 */
public static AmazonQLDB getClient() {
  return null;
}
```
Note

- In the `createLedger` call, you must specify a ledger name and a permissions mode. The only permissions mode that is currently supported for a ledger is `ALLOW_ALL`.
- 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).
To verify your connection to the new ledger, proceed to Step 2: Test Connectivity to the Ledger (p. 44).

**Step 2: Test Connectivity to the Ledger**

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).

```java
package software.amazon.qldb.tutorial;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.qldbsession.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:
 * <p>
 */
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() {
    }
}
```
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();
}

public static PooledQldbDriver getDriver() {
    if (driver == null) {
        driver = createQldbDriver();
    }
    return driver;
}

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 control plane API operations
listed in the Amazon QLDB API Reference (p. 331).
We recommended that you use PooledQldbDriver unless you need to implement a custom session pool with QldbDriver. The default pool size for PooledQldbDriver is the maximum number of concurrent sessions (p. 403) that the driver 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 an Optimistic Concurrency Control (OCC) conflict retry limit. By default, the execute method transparently retries OCC conflict exceptions up to this configurable limit. If the transaction still fails after the limit is reached, then the driver throws the exception. To learn more, see Handling OCC Conflict Exceptions (p. 201).

2. Compile and run the 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 the AWS Region and Endpoint (p. 34). You can also change the Region by modifying the AmazonQLDBSessionClientBuilder properties.

The following code example instantiates a new AmazonQLDBSessionClientBuilder object.

```java
import com.amazonaws.regions.Regions;
import com.amazonaws.services.qldbsession.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 is 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 (p. 46).

Step 3: Create Tables, Indexes, and Sample Data

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>
<thead>
<tr>
<th>Table Name</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleRegistration</td>
<td>VIN</td>
</tr>
</tbody>
</table>

To create tables in the vehicle-registration ledger, proceed to Step 3: Create Tables, Indexes, and Sample Data (p. 46).
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.

**Important**

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 and Terminology in Amazon QLDB (p. 7). To learn more about metadata, see Querying Document Metadata (p. 193).

**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 are serializable to and from Amazon Ion format.

   **Note**
   
   QLDB Documents (p. 273) are stored in Ion format. So, you can use the open source Jackson library to model the data in JSON.

   **1. DriversLicense.java**

   ```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;
   ```
public final class DriversLicense {
    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;
    }
}

2. Person.java

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
*/
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;

/**
 * Represents a person, serializable to (and from) Ion.
 */
public final class Person {
    private final String firstName;
    private final String lastName;
    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);
}

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 java.time.LocalDate;
import com.amazon.ion.Decimal;
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;

/**
 * Represents a vehicle registration, serializable to (and from) Ion.
 */
public final class VehicleRegistration {
    private final String vin;
    private final String licensePlateNumber;
    private final String state;
    private final String city;
    private final Decimal 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 Decimal 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 Decimal getPendingPenaltyTicketAmount() {
        return pendingPenaltyTicketAmount;
    }
```java
@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);
}
```

4. Vehicle.java

```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;

/**
 * Represents a vehicle, serializable to (and from) Ion.
 */
public final class Vehicle {
    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;
    }
}

5. Owner.java

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
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;
    }
}

6. Owners.java
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;
    }
}

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;
    }
}
8. IonLocalDateDeserializer.java

```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());
    }
}
```
9. IonLocalDateSerializer.java

```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.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.

```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
 */

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.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);
    }
}
```
package software.amazon.qldb.tutorial.model;

import java.io.IOException;
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 com.amazon.ion.Decimal;
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;

/**<*
 * 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",
                convertToDecimal(90.25), convertToLocalDate("2017-08-21"),
                convertToLocalDate("2020-05-11"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("KM8SRDHF6EU074761", "CA762X", "WA", "Kent",
                convertToDecimal(130.75), convertToLocalDate("2017-09-14"),
                convertToLocalDate("2020-06-25"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("3HGGK5G53FM761765", "CD820Z", "WA", "Everett",
                convertToDecimal(442.30), convertToLocalDate("2011-03-17"),
                convertToLocalDate("2021-03-24"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1HVBBAANXWH544237", "LS477D", "WA", "Tacoma",
                convertToDecimal(42.20), convertToLocalDate("2011-10-26"),
                convertToLocalDate("2024-03-19"),
                new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1C4RJFAG0FC625797", "TH393F", "WA", "Olympia",
                convertToDecimal(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("1C4RJFAG0PC625797", "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("2022-10-15")),
    new DriversLicense(null, "744 849 301", "Full",
                       convertToLocalDate("2017-12-06"),
                       convertToLocalDate("2021-11-15")),
    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 'yyyyMMdd' into a java.util.Date object.
 * 
 * @param date The date string to convert.
 * @return java.time.LocalDate or null if there is a ParseException
 */
public static synchronized LocalDate convertToLocalDate(String date) {
    return LocalDate.parse(date, DATE_TIME_FORMAT);
}

/**
 * Convert the result set into a list of IonValues.
 * 
 * Convert a date string with the format 'yyyyMMdd' into a java.util.Date object.
 * 
 * @param date The date string to convert.
 * @return java.time.LocalDate or null if there is a ParseException
 */
public static synchronized LocalDate convertToLocalDate(String date) {
    return LocalDate.parse(date, DATE_TIME_FORMAT);
}
* @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(Objects.requireNonNull(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 qldbSession
  * 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 QldbRevision object.
* @throws IllegalStateException if failed to convert parameter into IonValue.
*/
public static QldbRevision getDocumentById(QldbSession qldbSession, String
  tableName, String documentId) {
    try {
        final List<IonValue> parameters =
          Collections.singletonList(Objects.requireNonNull(documentId));
        final String query = String.format("SELECT c.* FROM _ql_committed_%s AS c
          BY docId WHERE docId = ?", tableName);
        Result result = qldbSession.execute(query, parameters);
        if (result.isEmpty()) {
            throw new IllegalStateException("Unable to retrieve document ID using " + documentId);
        }
        return getDocument(result.iterator().next(), documentId);
    } catch (IOException ioe) {
        throw new IllegalStateException(ioe);
    }
}
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();
}

/**
 * Convert the given double to a decimal value.
 * @param num The double to convert.
 * @return the decimal value of the double.
 */
private static synchronized Decimal convertToDecimal(final double num) {
    return Decimal.valueOf(num);
}

/**
 * Return a copy of the given driver's license with updated person Id.

public static DriversLicense updatePersonIdDriversLicense(final DriversLicense oldLicense, final String personId) {
    return new DriversLicense(personId, oldLicense.getLicenseNumber(),
        oldLicense.getLicenseType(),
        oldLicense.getValidFromDate(), oldLicense.getValidToDate());
}

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 also uses the Amazon Ion library 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.

/*
 * 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
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:
*/
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);
         }, (retryAttempt) -> log.info("Retrying due to OCC conflict...");
   }
}

Note

• You can use the pooled driver’s execute method to run a single PartiQL statement
  on your ledger. You can also use this method with an executor to create more complex
  execution sequences with other method calls.
• This program demonstrates how to pass a TransactionExecutor lambda to the
  execute method. In this example, you run multiple CREATE TABLE PartiQL statements
  with a single lambda expression.
• By default, the execute method implicitly starts and auto-commits a transaction. You
  can also take control over transactions to group multiple executions within a transaction.
For more information about this and how to handle OCC conflict exceptions, see Handling OCC Conflict Exceptions (p. 201) in the Concurrency Model section.

4. Compile and run the following program (CreateIndex.java) to create indexes on the tables, as previously described.

```java
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:
 */
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 = txm.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.

```java
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:
 */
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,
```java
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 a list of IonValue 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.

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 (p. 67).

**Step 4: Query the Tables in a Ledger**

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 (p. 272) as its query language and Amazon Ion (p. 269) 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.

**To query the tables**

1. Compile and run the following program (`ScanTable.java`) to scan all documents in all tables in your ledger.

```java
package software.amazon.qldb.tutorial;
import com.amazon.ion.IonStruct;
import org.slf4j.Logger;
```

/*
 * 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.
 */
```
public final class ScanTable {
    private static final Logger log = LoggerFactory.getLogger(ScanTable.class);

    private ScanTable() { }

    /**
     * Scan the table with the given {@code tableName} for all documents.
     * 
     * @param txn  The {@link TransactionExecutor} for lambda execute.
     * @param tableName  Name of the table to scan.
     * @return  a list of documents in {@link IonStruct}.
     */
    public static List<IonStruct> scanTableForDocuments(final TransactionExecutor txn, final String tableName) {
        log.info("Scanning '{}'...", tableName);
        final String scanTable = String.format("SELECT * FROM %s", tableName);
        List<IonStruct> documents = toIonStructs(txn.execute(scanTable));
        log.info("Scan successful!");
        printDocuments(documents);
        return documents;
    }

    /**
     * Pretty print all elements in the provided {@link Result}.
     *
     * @param result  {@link Result} from executing a query.
     */
    public static void printDocuments(final Result result) {
        result.iterator().forEachRemaining(row -> log.info(row.toPrettyString()));
    }

    /**
     * Pretty print all elements in the provided list of {@link IonStruct}.
     *
     * @param documents  List of documents to print.
     */
    public static void printDocuments(final List<IonStruct> documents) {
        documents.forEach(row -> log.info(row.toPrettyString()));
    }

    /**
     * Convert the result set into a list of {@link IonStruct}.
     *
     * @param result  {link Result} from executing a query.
     * @return  a list of documents in IonStruct.
     */
public static List<IonStruct> toIonStructs(final Result result) {
    final List<IonStruct> documentList = new ArrayList<>();
    result.iterator().forEachRemaining(row -> documentList.add((IonStruct) row));
    return documentList;
}

public static void main(final String... args) {
    ConnectToLedger.getDriver().execute(txn -> {
        List<String> tableNames = scanTableForDocuments(txn, Constants.USER_TABLES)
            .stream()
            .map((s) -> s.get("name").toString())
            .collect(Collectors.toList());
        for (String tableName : tableNames) {
            scanTableForDocuments(txn, tableName);
        }
    }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
}

2. Compile and run the following program (FindVehicles.java) to query all vehicles registered under a person in your ledger.

```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;

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:
 */
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);
    }
}

Step 5: Modify Documents in a Ledger

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.

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 (p. 70).
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:
 *
 */

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.
     * 
     */
    public static Person findPersonFromDocumentId(final TransactionExecutor txn, final String documentId) {
        try {
            log.info("Finding person for documentId: {}...", documentId);
            
            IonReader reader = IonReaderBuilder.create()
                    .withBaseSchema(SampleData.SAMPLE_SCHEMA)
                    .build()
                    .open()
                    
            IonValue doc = reader.nextObject();
            
            String id = doc.getString("id");
            if (id == null) {
                throw new ClassNotFoundException();
            }
            
            Person person = new Person(
                    doc.getString("name"),
                    doc.getString("email"),
                    doc.getString("phone"),
                    doc.getString("address"),
                    doc.getString("profession"),
                    doc.getString("additionalDetails"))
                    
            return person;
        }
    }
}
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);

/**
 * 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 RM8SRDF6EU074761 in your ledger.
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;

public final class AddSecondaryOwner {
    public static final Logger log = LoggerFactory.getLogger(AddSecondaryOwner.class);

    private AddSecondaryOwner() { }

    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;
                    }
                }
            }
        } catch (IOException e) {
            log.error("IOException occurred while trying to convert VIN to an IonValue.", e);
        }
        return false;
    }

    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;
                    }
                }
            }
        } catch (IOException e) {
            log.error("IOException occurred while trying to convert VIN to an IonValue.", e);
        }
        return false;
    }
}

/**
 * Finds and adds secondary owners for a vehicle.
 * This code expects that you have AWS credentials setup per:
 */
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;
                    }
                }
            }
        } catch (IOException e) {
            log.error("IOException occurred while trying to convert VIN to an IonValue.", e);
        }
        return false;
    }
To review these changes in the vehicle-registration ledger, see Step 6: View the Revision History for a Document (p. 76).
Step 6: View the Revision History for a Document

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).

```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;

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:
 */

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 (p. 197) in the following syntax.

```sql
SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h WHERE h.metadata.id = 'id'
```

- Best practice is to qualify a history query with a document id. This helps to avoid inefficient queries.

- The start-time and end-time are both optional. They are Amazon Ion literal values that can be denoted with backticks (`...`). To learn more, see Querying Ion with PartiQL (p. 289).

2. Compile and run the 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 (p. 78).
Step 7: Verify a Document in a Ledger

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 (p. 205).

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 five .java files, which represent QLDB objects that are required for verification and a utility class with helper methods to pretty-print certain QLDB response types.

```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;

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");
    }
}

2. QldbRevision.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.
 * */

package software.amazon.qldb.tutorial.qldb;

import java.io.IOException;
import java.math.BigDecimal;
import java.time.ZoneId;
import java.util.Arrays;
import java.util.Date;
import java.util.Objects;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.amazon.ion.IonBlob;
import com.amazon.ion.IonInt;
import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonTimestamp;
import com.amazon.ion.IonValue;
import com.amazon.ion.hash.IonHashReader;
import com.amazon.ion.hash.IonHashReaderBuilder;
import com.amazon.ionhash.MessageDigestIonHasherProvider;
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 software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.Verifier;

/**
 * Represents a QldbRevision including both user data and metadata.
 */
public final class QldbRevision {
    private static final Logger log = LoggerFactory.getLogger(QldbRevision.class);
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    private static MessageDigestIonHasherProvider ionHasherProvider = new
            MessageDigestIonHasherProvider("SHA-256");
    private static final ZoneId UTC = ZoneId.of("UTC");
    private static final BigDecimal ONE_THOUSAND = BigDecimal.valueOf(1000L);

    /**
     * Represents the metadata field of a QLDB Document
     */
    public static class Metadata {
        private final String id;
        private final long version;
        @JsonSerialize(using =
                IonTimestampSerializers.IonTimestampJavaDateSerializer.class)
        private final Date txTime;
        private final String txId;

        @JsonCreator
        public Metadata(@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 Metadata 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 Metadata(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 Metadata} 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 Metadata} 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; }
  Metadata metadata = (Metadata) o;
  return version == metadata.version
      && id.equals(metadata.id)
      && txTime.equals(metadata.txTime)
      && txId.equals(metadata.txId);
}

/**
 * Generate a hash code for the {@link Metadata} 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
}
}

private final BlockAddress blockAddress;
private final Metadata metadata;
private final byte[] hash;
private final IonStruct data;

@JsonCreator
public QldbRevision(@JsonProperty("blockAddress") final BlockAddress blockAddress,
    @JsonProperty("metadata") final Metadata metadata,
    @JsonProperty("hash") final byte[] hash,
    @JsonProperty("data") final IonStruct data) {
  this.blockAddress = blockAddress;
  this.metadata = metadata;
  this.hash = hash;
  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 Metadata} object.
 */
public Metadata getMetadata() {
  return metadata;
}
* Gets the SHA-256 hash value of the data.
* @return the byte array representing the hash.
*/
public byte[] getHash() {
    return hash;
}

/**
 * Gets the revision data.
 * @return the revision data.
*/
public IonStruct getData() {
    return data;
}

/**
 * Constructs a new {@link QldbRevision} from an IonStruct.
 * The specified IonStruct must include the following fields
 * - blockAddress -- a @link BlockAddress,
 * - metadata -- a @link Metadata,
 * - hash -- the document's hash calculated by QLDB,
 * - data -- an IonStruct containing user data in the document.
 * If any of these fields are missing or are malformed, then throws IllegalArgumentException.
 * If the document hash calculated from the members of the specified IonStruct does not match
 * the hash member of the IonStruct then throws IllegalArgumentException.
 * @param ionStruct The IonStruct that contains a QldbRevision object.
 * @return the converted QldbRevision object.
 * @throws IOException if failed to parse parameter IonStruct.
*/
public static QldbRevision fromIon(final IonStruct ionStruct) throws IOException {
    try {
        BlockAddress blockAddress =
            Constants.MAPPER.readValue(ionStruct.get("blockAddress"), BlockAddress.class);
        IonBlob hash = (IonBlob) ionStruct.get("hash");
        IonStruct metadataStruct = (IonStruct) ionStruct.get("metadata");
        IonStruct data = (IonStruct) ionStruct.get("data");
        if (hash == null || data == null) {
            throw new IllegalArgumentException("Document is missing required fields");
        }
        byte[] candidateHash = computeHash(metadataStruct, data);
        if (!Arrays.equals(candidateHash, hash.getBytes())) {
            throw new IllegalArgumentException("Hash entry of QLDB revision and computed hash "+ "of QLDB revision do not match");
        }
        Metadata metadata = Metadata.fromIon(metadataStruct);
        return new QldbRevision(blockAddress, metadata, hash.getBytes(), data);
    } catch (ClassCastException e) {
        log.error("Failed to parse ion document");
        throw new IllegalArgumentException("Document members are not of the correct type", e);
    }
}
public static byte[] computeHash(final IonStruct metadata, final IonStruct data) {
    byte[] metaDataHash = hashIonValue(metadata);
    byte[] dataHash = hashIonValue(data);
    return Verifier.joinHashesPairwise(metaDataHash, dataHash);
}

private static byte[] hashIonValue(final IonValue ionValue) {
    IonReader reader = SYSTEM.newReader(ionValue);
    IonHashReader hashReader = IonHashReaderBuilder.standard()
        .withHasherProvider(ionHasherProvider)
        .withReader(reader)
        .build();
    while (hashReader.next() != null) {  }
    return hashReader.digest();
}

public String toString() {
    return "QldbRevision{" +
            "blockAddress=" + blockAddress +
            ", metadata=" + metadata +
            ", hash=" + Arrays.toString(hash) +
            ", data=" + data +
    "}';
}

public boolean equals(final Object o) {
    if (this == o) {
        return true;
    }
    if (!(o instanceof QldbRevision)) {
        return false;
    }
    final QldbRevision that = (QldbRevision) o;
    return this.equal(that);
return Objects.equals(getBlockAddress(), that.getBlockAddress()) &&
Objects.equals(getMetadata(),
    that.getMetadata()) && Arrays.equals(getHash(), that.getHash()) &&
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;
}

3. 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;
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
}

4. Verifier.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
*/

86
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.Arrays;
import java.util.Comparator;
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:
 */
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 <em>signed</em> 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
     * 2. Check that the {code candidateLedgerDigest} is equal to the {code ledgerDigest}.
     */
    public final class Verifier { }
public static boolean verify(byte[] documentHash, byte[] digest, String proofBlob) {
    Proof proof = Proof.fromBlob(proofBlob);
    byte[] candidateDigest = buildCandidateDigest(proof, documentHash);
    return Arrays.equals(digest, candidateDigest);
}

private static byte[] buildCandidateDigest(Proof proof, byte[] leafHash) {
    return calculateRootHashFromInternalHashes(proof.getInternalHashes(), leafHash);
}

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.
 */
public static byte[] joinHashesPairwise(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.
 */
public static byte[] calculateRootHashFromInternalHashes(final List<byte[]> internalHashes, final byte[] leafHash) {
    return internalHashes.stream().reduce(leafHash, Verifier::joinHashesPairwise);
}

/**
 * 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) {
    // Implementation...
}
```java
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;
}
```

5. QldbStringUtils.java

```java
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;
import java.util.Base64;
import java.util.Base64.Encoder;

/**
 * Helper methods to pretty-print certain QLDB response types.
 */
public class QldbStringUtils {
    private static final Encoder ENCODER = Base64.getEncoder();

    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}`.

```java
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}`.
*/

```java
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}`.
*/

```java
public static String toUnredactedString(GetDigestResult getDigestResult) {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    if (getDigestResult.getDigest() != null) {
        String digest = ENCODER.encodeToString(getDigestResult.getDigest().array());
        sb.append("Digest:
        ").append(ENCODER.encodeToString(getDigestResult.getDigest().array()));
    }
    sb.append("}*
    return sb.toString();
}
```
2. Review the following two .java programs, which contain code examples that demonstrate 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.

1. GetDigest.java

```java
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:
 */
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 {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.

2. GetRevision.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.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:
 */
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 qldbSession 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 qldbSession, final String ledgerName, final String vin) {
        // Code to verify the integrity of a document revision...
    }
}
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<>();
        qldbSession.execute(txn -> 
                documentsWithMetadataList.addAll(queryRegistrationsByVin(txn, vin)), 
                (retryAttempt) -> log.info("Retrying due to OCC conflict...");
        log.info("Registrations queried successfully!");
    }

    for (IonStruct ionStruct : documentsWithMetadataList) {
        QldbRevision document = QldbRevision.fromIon(ionStruct);
        log.info(String.format("Let's verify the document: %s", 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.toByteArray(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.toByteArray(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
 * @param documentId
 * @param digestTipAddress
 * @param blockAddress
 */
return the requested revision.
public static GetRevisionResult getRevision(final String ledgerName, final String documentId, final BlockAddress blockAddress, final ValueHolder digestTipAddress) {
    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 TransactionExecutor for lambda execute.
 * @param vin The unique VIN to query.
 * @return a list of IonStruct representing the registration history.
 * @throws IllegalStateException if failed to convert parameters into 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));
    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 (p. 213).

3. Compile and run the GetRevision.java program to cryptographically verify the VehicleRegistration document with VIN 1N4AL11D75C109151.

If you no longer need to use the vehicle-registration ledger, proceed to Step 8 (Optional): Clean Up Resources (p. 98).
Step 8 (Optional): Clean Up Resources

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

To delete the ledger

- Compile and run the following program (DeleteLedger.java) to delete your vehicle-registration ledger and all of its contents.

```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.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:
 */
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.
Best Practices for the Amazon QLDB Java Driver

This section describes best practices for configuring and using the Amazon QLDB Driver 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 PooledQldbDriver (p. 100)
- Retrying Exceptions (p. 100)

Configuring the PooledQldbDriver

We recommend using the PooledQldbDriver, which is a variation of the driver object that is included with the QLDB Driver for every language. This driver manages connections to your ledger by maintaining a pool of sessions that are reused across transactions.

Configure the PooledQldbDriver as a singleton

To help ensure that only one instance of the PooledQldbDriver exists in your application instance, you can use dependency injection frameworks such as Spring, Google Guice, or Dagger. The following code example shows how to configure PooledQldbDriver as a singleton.

```java
@Singleton
public PooledQldbDriver qldbDriver (AWSCredentialsProvider credentialsProvider,
        @Named(LEDGER_NAME_CONFIG_PARAM) String ledgerName) {
        AmazonQldbSessionClientBuilder builder = AmazonQldbSessionClientBuilder.standard();
        if (null != credentialsProvider) {
            builder.setCredentials(credentialsProvider);
        }
        return PooledQldbDriver.builder()
                .withLedger(ledgerName)
                .withRetryLimit(3)
                .withSessionClientBuilder(builder)
                .build();
}
```

Configure the retry attempts

The PooledQldbDriver automatically retries transactions when common transient exceptions (such as SocketTimeoutException or NoHttpResponseException) occur. You can use the RetryLimit configuration parameter when creating an instance of PooledQldbDriver to set the maximum number of retry attempts.

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 sessions and connections

The maximum number of ledger connections that are used by PooledQldbDriver is determined by its poolLimit and the maximum number of allowed open HTTP connections, as defined by the AWS SDK for Java. The pool limit must be greater than zero and less or equal to ClientConfiguration.getMaxConnections().

The limit of concurrent driver sessions is defined in Quotas in Amazon QLDB (p. 403).

Retrying Exceptions

When retrying exceptions that occur in QLDB, consider the following recommendations.
Retrying 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 PooledQldbDriver retries the transaction as many times as the retryLimit is configured.

For more information about OCC, see Amazon QLDB Concurrency Model (p. 200).

Retrying a transaction outside the PooledQldbDriver for other exceptions

To retry a transaction outside of the driver when custom, application-defined exceptions are thrown during execution, you must wrap the transaction. For example, the following code shows how to use the Reslience4J library to retry a transaction in QLDB.

```java
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 (QldbSession qldbSession = driver.getSession()) {
        qldbSession.execute(txn -> {
            // Transaction code
            }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
    return null;
}
```

Note

Retrying a transaction outside of the QLDB Driver has a multiplier effect. For example, if the PooledQldbDriver 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.

Amazon QLDB Driver for .NET (Preview)

This is prerelease documentation for a feature in preview release. It is subject to change.

To work with data in your ledger, you can connect to Amazon QLDB from your Microsoft .NET application using an AWS-provided driver. Then you can code your application to use the transactional data API.

The driver targets .NET Standard 2.0. For more information on compatibility, see .NET Standard on the Microsoft Docs site.
The following sections describe how to get started with the QLDB Driver for .NET.

Topics
- See Also (p. 102)
- Prerequisites (p. 102)
- Quick Start Installation (p. 102)
- Example (p. 102)

See Also
For more information about the functionality supported by the .NET driver, see the following:
- API Reference
- Common Errors (p. 185)
- Driver Source Code
- Amazon Ion Cookbook

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 (p. 11). This includes signing up for AWS and getting an AWS access key for development.
2. Install a current version of Microsoft Visual Studio. You can download a free copy of the Community edition from the Microsoft Visual Studio site.
3. To install the QLDB Driver and the AWS SDK for .NET in an existing project, see Quick Start Installation (p. 102).

Quick Start Installation
To install the QLDB Driver for .NET, use the NuGet package manager. The driver package name is Amazon.QLDB.Driver.

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

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

Installing the driver also installs its dependencies, including the AWS SDK for .NET and Amazon Ion (p. 269) libraries.

Example
Then, using the driver's namespace, you can use the driver in your application to connect to a ledger. The following C# code example shows how to do the following:

1. Create a ledger named my-ledger.
2. Describe the ledger to confirm that it's active.
3. Create a driver session to connect to the ledger.
4. Use the session to create a table named MyTable1 on the ledger.

```csharp
using System;
namespace Hello
{
    using Amazon.QLDB.Driver;
    using Amazon.QLDB;
    using Amazon.QLDB.Model;
    using Amazon.QLDBSession;
    using Amazon.QLDBSession.Model;
    using Amazon.Runtime;
    using System.Threading;
    using System.Text;
    class Program
    {
        static void Main(string[] args)
        {
            string ledgerName = "my-ledger";
            Console.WriteLine("Create the ledger '{ledgerName}'");
            AmazonQLDBClient qldbClient = new AmazonQLDBClient();
            CreateLedgerRequest createLedgerRequest = new CreateLedgerRequest
            {
                Name = ledgerName,
                PermissionsMode = PermissionsMode.ALLOW_ALL
            };
            qldbClient.CreateLedgerAsync(createLedgerRequest).GetAwaiter().GetResult();
            Console.WriteLine("Waiting for ledger to be active");
            DescribeLedgerRequest describeLedgerRequest = new DescribeLedgerRequest
            {
                Name = ledgerName
            };
            while (true)
            {
                DescribeLedgerResponse describeLedgerResponse =
                qldbClient.DescribeLedgerAsync(describeLedgerRequest).GetAwaiter().GetResult();
                if (describeLedgerResponse.State.Equals(LedgerState.ACTIVE.Value))
                {
                    Console.WriteLine("'{ledgerName}' ledger created sucessfully.");
                    break;
                }
                Console.WriteLine("Creating the '{ledgerName}' ledger...");
                Thread.Sleep(1000);
            }
            AmazonQLDBSessionConfig amazonQldbSessionConfig = new AmazonQLDBSessionConfig();
            Console.WriteLine("Create the QLDB Driver");
            IQldbDriver driver = PooledQldbDriver.Builder()
                .WithQLDBSessionConfig(amazonQldbSessionConfig)
                .WithLedger(ledgerName)
                .Build();
            string tableName = "MyTable1";
            using (IQldbSession qldbSession = driver.GetSession())
            {
                // qldbSession.Execute will start a transaction and commit it.
                IResult result = qldbSession.Execute("CREATE TABLE {tableName}"");
                Console.WriteLine("Table '{tableName}' created");
            }
        }
    }
}
Amazon QLDB Driver for Node.js (Preview)

This is prerelease documentation for a feature in preview release. It is subject to change.

To work with data in your ledger, you can connect to Amazon QLDB from your Node.js application using an AWS-provided driver. Then you can code your application to use the transactional data API. The following sections describe how to get started with the QLDB Driver for Node.js.

Topics

• See Also (p. 104)
• Prerequisites (p. 104)
• Quick Start Installation (Driver Only) (p. 105)
• Amazon QLDB Node.js Tutorial (p. 105)

See Also

For more information about the functionality supported by the Node.js driver, see the following:

• API Reference
• Common Errors (p. 185)
• Driver Source Code
• Sample Application Source Code
• Amazon Ion Cookbook

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 (p. 11). This includes signing up for AWS and getting an AWS access key for development.
2. Set up the AWS SDK for JavaScript in Node.js by completing the Prerequisite Tasks and Step 1 in the AWS SDK for JavaScript in Node.js Getting Started Guide. This includes the following:
   • Install Node.js. The QLDB Driver for Node.js requires version 10.x or later.
   • Configure your development environment for the AWS SDK for JavaScript in Node.js.
   • Set your AWS Region. For a complete list of available Regions, see Amazon QLDB Endpoints and Quotas in the AWS General Reference.
3. Next, you can download the complete tutorial sample application, or you can install only the driver in an existing Node.js project.
   • To run the complete tutorial with the QLDB sample application, proceed to the Node.js Tutorial (p. 105).
Quick Start Installation (Driver Only)

To install only the QLDB Driver using npm (the Node.js package manager), enter the following command from your project's root directory.

```
npm install amazon-qldb-driver-nodejs
```

Installing the driver also installs TypeScript 3.5.x as a dependency. In addition, the driver has peer dependencies on the `aws-sdk` (AWS SDK for JavaScript) and `ion-js` (Amazon Ion data format) packages. So, you must also install these packages as dependencies in your project.

```
npm install aws-sdk

npm install ion-js
```

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 session for a specified ledger name and AWS Region.

```typescript
import { PooledQldbDriver, QldbSession } from "amazon-qldb-driver-nodejs";

const testServiceConfigOptions = {
  region: "us-east-1",
};

const qldbDriver: PooledQldbDriver = new PooledQldbDriver("testLedger",
  testServiceConfigOptions);

const qldbSession: QldbSession = await qldbDriver.getSession();

for (const table of await qldbSession.getTableNames()) {
  console.log(table);
}
```

For code examples that demonstrate other operations such as how to create a ledger and execute data transactions on a ledger, see the Node.js Tutorial (p. 105).

Amazon QLDB Node.js Tutorial

This is prerelease documentation for a feature in preview release. It is subject to change.

In this tutorial, you use the QLDB Driver for Node.js to create an Amazon QLDB ledger and populate it with tables and sample data. The driver enables your application to interact with QLDB using the transactional data 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 sections 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.

As you work through this tutorial, you can refer to the AWS SDK for JavaScript API Reference.

**Prerequisites**

Before you get started, make sure that you complete the **Prerequisites (p. 104)**. This includes signing up for AWS and getting an AWS access key for development.

**Topics**

- Installing the Amazon QLDB Node.js Sample Application (p. 106)
- Step 1: Create a New Ledger (p. 107)
- Step 2: Test Connectivity to the Ledger (p. 109)
- Step 3: Create Tables, Indexes, and Sample Data (p. 111)
- Step 4: Query the Tables in a Ledger (p. 124)
- Step 5: Modify Documents in a Ledger (p. 128)
- Step 6: View the Revision History for a Document (p. 133)
- Step 7: Verify a Document in a Ledger (p. 135)
- Step 8 (Optional): Clean Up Resources (p. 144)

**Installing the Amazon QLDB Node.js Sample Application**

This is prerelease documentation for a feature in preview release. It is subject to change.

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.

**Prerequisites**

Before you get started, make sure that you complete the **Prerequisites (p. 104)**. This includes signing up for AWS and getting an AWS access key for development.

**Installation**

**To install the sample application**

1. Enter the following command to clone the sample application from GitHub.

   ```bash
   git clone 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.

   ```bash
   cd amazon-qldb-dmv-sample-nodejs
   ```

3. Do a clean install of the dependencies.
4. Transpile the package.

```bash
npm run build
```

The transpiled JavaScript files are written in the `./dist` directory.

5. Proceed to Step 1: Create a New Ledger (p. 107) to start the tutorial and create a ledger.

## Step 1: Create a New Ledger

This is prerelease documentation for a feature in preview release. It is subject to change.

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.

```typescript
/*
 * 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;
```
2. Use the following program (CreateLedger.ts) to create a ledger named vehicle-registration.

```javascript
/*
 * 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/qldb";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { sleep } from "./qldb/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 qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a CreateLedgerResponse.
 */
export async function createLedger(ledgerName: string, qldbClient: QLDB): Promise<CreateLedgerResponse> {
    log(`Creating a ledger named: ${ledgerName}...`);
    const request: CreateLedgerRequest = {
        Name: ledgerName,
        PermissionsMode: "ALLOW_ALL"
    }
    const result: CreateLedgerResponse = await qldbClient.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 qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a DescribeLedgerResponse.
 */
```
export async function waitForActive(ledgerName: string, qldbClient: QLDB): Promise<DescribeLedgerResponse> {
    log(`Waiting for ledger ${ledgerName} to become active...`);
    const request: DescribeLedgerRequest = {
        Name: ledgerName
    }
    while (true) {
        const result: DescribeLedgerResponse = await qldbClient.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.

var main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        await createLedger(LEDGER_NAME, qldbClient);
        await waitForActive(LEDGER_NAME, qldbClient);
    } 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. The only permissions mode that is currently supported for a ledger is ALLOW_ALL.
- 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 (p. 109).

Step 2: Test Connectivity to the Ledger

This is prerelease documentation for a feature in preview release. It is subject to change.
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.

```javascript
import { PooledQldbDriver, QldbDriver, QldbSession } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/qldbsession";
import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

const pooledQldbDriver: QldbDriver = createQldbDriver();

/**
 * Close a QLDB session object.
 * @param session The session to close.
 */
export function closeQldbSession(session: QldbSession): void {
    if (null !== session) {
        session.close();
    }
}

/**
 * Create a pooled 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 pooled driver for creating sessions.
 */
export function createQldbDriver(
    ledgerName: string = LEDGER_NAME,
    serviceConfigurationOptions: ClientConfiguration = {}): QldbDriver {
    const qldbDriver = new PooledQldbDriver(ledgerName,
        serviceConfigurationOptions);
    return qldbDriver;
}

/**
 * Retrieve a QLDB session object.
 */
```
Note

To run data transactions on your ledger, you must create a QLDB session object to connect to a specific ledger. This is a different client object than the qldbClient object that you used in the previous step (p. 107) to create the ledger. That previous client is only used to run the control plane API operations listed in the Amazon QLDB API Reference (p. 331). First, you 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 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 (p. 111).

**Step 3: Create Tables, Indexes, and Sample Data**

This is prerelease documentation for a feature in preview release. It is subject to change.

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
You also create the following indexes.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleRegistration</td>
<td>VIN</td>
</tr>
<tr>
<td>VehicleRegistration</td>
<td>LicensePlateNumber</td>
</tr>
<tr>
<td>Vehicle</td>
<td>VIN</td>
</tr>
<tr>
<td>Person</td>
<td>GovId</td>
</tr>
<tr>
<td>DriversLicense</td>
<td>LicenseNumber</td>
</tr>
<tr>
<td>DriversLicense</td>
<td>PersonId</td>
</tr>
</tbody>
</table>

When inserting sample data, you first insert documents into the Person table. Then, use the system-assigned id from each Person document to populate the corresponding fields in the appropriate VehicleRegistration and DriversLicense documents.

**Important**

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 and Terminology in Amazon QLDB (p. 7). To learn more about metadata, see Querying Document Metadata (p. 193).

**To create tables and indexes**

1. Use the following program (CreateTable.ts) to create the previously mentioned tables.

```javascript
/*
 * 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 { QldbSession, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { closeQldbSession, createQldbSession } from "./ConnectToLedger";
```
Note

This program demonstrates how to use the `executeLambda` function in a QLDB session. In this example, you run multiple `CREATE TABLE` PartiQL statements with a single Lambda expression.

2. To run the transpiled program, enter the following command.

   ```bash
   node dist/CreateTable.js
   ```

3. Use the following program (`CreateIndex.ts`) to create indexes on the tables, as previously described.

   ```typescript
   /*
   * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
   * SPDX-License-Identifier: MIT-0
   */
   ```
import { QldbSession, TransactionExecutor } from "amazon-qldb-driver-nodejs";

import { closeQldbSession, createQldbSession } 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 {TransactionExecutor} tx The lambda executor.
 * @param {string} tableName Name of the table to add indexes for.
 * @returns {Promise<number>} Promise which fulfills with the number of changes to the database.
 */
export async function createIndex(txn, tableName, indexAttribute) {
    const statement = `CREATE INDEX on ${tableName} (${indexAttribute})`;
    return await txn.executeInline(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<void>} Promise which fulfills with void.
 */
var main = async function() {
    let session: QldbSession;
    try {
        session = await createQldbSession();
        await session.executeLambda(async (txn) => {
            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_NUMBER_INDEX_NAME),
                createIndex(txn, DRIVERS_LICENSE_TABLE_NAME, PERSON_ID_INDEX_NAME),
            ]);
        });
    } catch (error) {
        log(`Error occurred during lambda function execution: ${error}`);
    }
}

114
createIndex(txn, DRIVERS_LICENSE_TABLE_NAME, LICENSE_NUMBER_INDEX_NAME)
]);
}, () => log("Retrying due to OCC conflict...");
} catch (e) {
    error(`Unable to create indexes: ${e}`);
} finally {
    closeQldbSession(session);
}

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("2016-12-20")
    }
];
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 : "F626-168-229-765",
  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 : "F626-168-229-765",
  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
        }
    }];
2. Util.ts — A utility module that imports from the ion-js package to provide helper functions that convert, parse, and print Amazon Ion (p. 269) 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
 */
import { createQldbWriter, QldbWriter, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { GetBlockResponse, GetDigestResponse, ValueHolder } from "aws-sdk/clients/qldb";
import {
  Decimal,
  decodeUtf8,
  IonTypes,
  makePrettyWriter,
  makeReader,
  Reader,
  Timestamp,
  toBase64,
  Writer
} from "ion-js";

import { error } from "./.LogUtil";

/**
 * 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());
}

/**
 * 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());
}

/**
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}
  = ?`;
  const parameter: QldbWriter = createQldbWriter();
  parameter.writeString(value);
  let documentId: string;
  await txn.executeInline(query, [parameter]).then((result: Result) => {
    const resultList: Reader[] = result.getResultList();
    if (resultList.length === 0) {
      throw new Error(`Unable to retrieve document ID using ${value}.`);
    }
    documentId = getFieldValue(resultList[0], ["id"]) props
    .catch((err) => {
      error(`Error getting documentId: ${err}`);
    });
    return documentId;
  });
}

/**
 * Function which, given a reader and a path, traverses through the reader using the
 * path to find the value.
 * @param ionReader The reader to operate on.
 * @param path The path to find the value.
 * @returns The value obtained after traversing the path.
 */
export function getFieldValue(ionReader: any, path: string[]): any {
  ionReader.next();
  ionReader.stepIn();
  return recursivePathLookup(ionReader, path);
}

/**
 * Helper method that traverses through the reader using the path to find the value.
 * @param ionReader The reader to operate on.
 * @param path The path to find the value.
 * @returns The value, or undefined if the provided path does not exist.
 */
export function recursivePathLookup(ionReader: Reader, path: string[]): any | undefined {
  if (path.length === 0) {
    // If the path's length is 0, the current ionReader node is the value which
    // should be returned.
    if (ionReader.type() === IonTypes.LIST) {
      const list: any[] = [];
      ionReader.stepIn(); // Step into the list.
      while (ionReader.next() !== null) {
        const itemInList: any = recursivePathLookup(ionReader, []);
        list.push(itemInList);
      }
      return list;
    } else if (ionReader.type() === IonTypes.STRUCT) {
      const structToReturn: any = {};
      // Process the struct.
    }
  }
}
let type: any;
const currentDepth: any = ionReader.depth();
ionReader.stepIn();
while (ionReader.depth() > currentDepth) {
  // In order to get all values within the struct, we need to visit every node.
  type = ionReader.next();
  if (type === null) {
    // End of the container indicates that we need to step out.
    ionReader.stepOut();
  } else {
    structToReturn[ionReader.fieldName()] = recursivePathLookup(ionReader, []);
  }
}
return structToReturn;
return ionReader.value();
}

export function sleep(ms: number): Promise<void> {
  return new Promise(resolve => setTimeout(resolve, ms));
}

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 covert 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 (p. 195).

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
import { createQldbWriter, QldbSession, QldbWriter, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { Reader } from "ion-js";
import { closeQldbSession, createQldbSession } 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";
import { getFieldValue, writeValueAsIon } from "./qldb/Util";

/**
 * 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 documentsWriter: QldbWriter = createQldbWriter();
  writeValueAsIon(documents, documentsWriter);
  let result: Result = await txn.executeInline(statement, [documentsWriter]);
  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: Reader[] = documentIds.getResultList();"
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: Reader[]): void {
  documentIds.forEach((reader: Reader, i: number) => {
    const documentId: string = getFieldValue(reader, ["documentId"]) || 
      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.
 */
var main = async function(): Promise<void> {
  let session: QldbSession;
  try {
    session = await createQldbSession();
    await session.executeLambda(async (txn) => {
      await updateAndInsertDocuments(txn);
    }, () => log("Retrying due to OCC conflict..."));
  } catch (e) {
    error("Unable to insert documents: ${e}");
  }
  finally {
    closeQldbSession(session);
  }
}

if (require.main === module) {
  main();
}

Note

- This program demonstrates how to call the executeInline function with
  parameterized values. You can pass a list of Ion 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 (p. 124).

**Step 4: Query the Tables in a Ledger**

This is prerelease documentation for a feature in preview release. It is subject to change.
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 (p. 272) as its query language and Amazon Ion (p. 269) 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.

To query the tables

1. Use the following program (ScanTable.ts) to scan all the documents in all the tables in your ledger.

```typescript
import { QldbSession, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { decodeUtf8, makePrettyWriter, Reader, Writer } from "ion-js";
import { closeQldbSession, createQldbSession } from "./ConnectToLedger";
import { log } from "./qldb/LogUtil";

/**
 * Pretty print the Readers in the provided result list.
 * @param resultList The result list containing the Readers to pretty print.
 */
export function prettyPrintResultList(resultList: Reader[]): void {
    const writer: Writer = makePrettyWriter();
    resultList.forEach((reader: Reader) => {
        writer.writeValues(reader);
    });
    log(decodeUtf8(writer.getBytes()));
}

/**
 * Scan for all the documents in a table.
 * @param tx The TransactionExecutor for lambda execute.
 * @param tableName The name of the table to operate on.
 * @returns Promise which fulfills with a Result object.
 */
export async function scanTableForDocuments(txn: TransactionExecutor, tableName: string): Promise<Result> {
    log(`Scanning ${tableName}...`);
    const query: string = `SELECT * FROM ${tableName}`;
    ```
```javascript
return await txn.executeInline(query).then((result: Result) => {
    return result;
});

/**
* Retrieve the list of table names.
* @param session The session to retrieve table names from.
* @returns Promise which fulfills with a list of table names.
*/
export async function scanTables(session: QldbSession): Promise<string[]> {
    return await session.getTableNames();
}

/**
* Scan for all the documents in a table.
* @returns Promise which fulfills with void.
*/
var main = async function(): Promise<void> {
    let session: QldbSession;
    try {
        session = await createQldbSession();
        await scanTables(session).then(async (listofTables: string[]) => {
            for (const tableName of listofTables) {
                await session.executeLambda(async (txn) => {
                    const result: Result = await scanTableForDocuments(txn, tableName);
                    prettyPrintResultList(result.getResultList());
                });
            }, () => log("Retrying due to OCC conflict...");
        } catch (e) {
            log('Error displaying documents: ' + e);
        } finally {
            closeQldbSession(session);
        }
    }

    if (require.main === module) {
        main();
    }
}
```

2. To run the transpiled program, enter the following command.

```
node dist/ScanTable.js
```

3. Use the following program (`FindVehicles.ts`) to query all vehicles registered under a person in your ledger.

```typescript
/*
* 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
```

126
import { createQldbWriter, QldbSession, QldbWriter, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { Reader } from "ion-js";
import { closeQldbSession, createQldbSession } from "./ConnectToLedger";
import { PERSON } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qlldb/Constants";
import { error, log } from "./qlldb/LogUtil";
import { getDocumentId, writeValueAsIon } from "./qlldb/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 = ?";
  const qldbWriter: QldbWriter = createQldbWriter();
  writeValueAsIon(documentId, qldbWriter);
  await txn.executeInline(query, [qldbWriter]).then((result: Result) => {
    const resultList: Reader[] = 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.
 */
var main = async function(): Promise<void> {
  let session: QldbSession;
  try {
    session = await createQldbSession();
    await session.executeLambda(async (txn) => {
      await findVehiclesForOwner(txn, PERSON[0].GovId);
    }, () => log("Retrying due to OCC conflict..."));
  } catch (e) {
    error("Error getting vehicles for owner: ${e}");
  } finally {
    closeQldbSession(session);
  }
}

if (require.main === module) {
  main();
}

Note
First, this program queries the Person table for the document with GovId LEWISR2611LL

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.

4. To run the transpiled program, enter the following command.

```bash
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 (p. 128).

**Step 5: Modify Documents in a Ledger**

This is prerelease documentation for a feature in preview release. It is subject to change.

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.

```javascript
/*
 * 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 { createQldbWriter, QldbSession, QldbWriter, Result, TransactionExecutor } from
"amazon-qldb-driver-nodejs";
import { Reader } from "ion-js";
import { closeQldbSession, createQldbSession } from ".//ConnectToLedger";
import { PERSON, VEHICLE } from ".//model/SampleData";
import { PERSON_TABLE_NAME } from ".//qldb/Constants";
import { error, log } from ".//qldb/LogUtil";
import { getDocumentId, getFieldValue, writeValueAsIon } from ".//qldb/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 a Reader containing the person. */

export async function findPersonFromDocumentId(txn: TransactionExecutor, documentId: string): Promise<Reader> {
    const query: string = "SELECT p.* FROM Person AS p BY pid WHERE pid = ?";
    const qldbWriter: QldbWriter = createQldbWriter();
    writeValueAsIon(documentId, qldbWriter);

    let personId: Reader;
    await txn.executeInline(query, [qldbWriter]).then((result: Result) => {
        const resultList: Reader[] = 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 a Reader containing the primary owner.
 */

export async function findPrimaryOwnerForVehicle(txn: TransactionExecutor, vin: string): Promise<Reader> {
    log('Finding primary owner for vehicle with VIN: ' + vin);
    const query: string = "SELECT Owners.PrimaryOwner.PersonId FROM VehicleRegistration AS v WHERE v.VIN = ?";
    const vinWriter: QldbWriter = createQldbWriter();
    writeValueAsIon(vin, vinWriter);

    let documentId: string;
    await txn.executeInline(query, [vinWriter]).then((result: Result) => {
        const resultList: Reader[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error('Unable to retrieve document ID using ' + vin + '.');
        }
        documentId = getFieldValue(resultList[0], ['PersonId']);
    });
    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 = ?";
    const personIdWriter: QldbWriter = createQldbWriter();
    writeValueAsIon(documentId, personIdWriter);

    const vinWriter: QldbWriter = createQldbWriter();
    writeValueAsIon(vin, vinWriter);

    log('Updating the primary owner for vehicle with VIN: #' + vin + '...');
    await txn.executeInline(statement, [personIdWriter, vinWriter]).then((result: Result) => {
```javascript
const resultList: Reader[] = 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: Reader = await findPrimaryOwnerForVehicle(txn, vin);
    if (getFieldValue(primaryOwner, ["GovId"]) !== 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.
 */
var main = async function(): Promise<void> {
    let session: QldbSession;
    try {
        session = await createQldbSession();
        const vin: string = VEHICLE[0].VIN;
        const previousOwnerGovId: string = PERSON[0].GovId;
        const newPrimaryOwnerGovId: string = PERSON[1].GovId;
        await session.executeLambda(async (txn) => {
            await validateAndUpdateRegistration(txn, vin, previousOwnerGovId, newPrimaryOwnerGovId);
        }, () => log("Retrying due to OCC conflict..."));
        catch (e) {
            error("Unable to connect and run queries: ${e}");
        } finally {
            closeQldbSession(session);
        }
    } catch (e) {
        log("Unable to transfer vehicle, could not find registration.");
        throw new Error("Unable to transfer vehicle, could not find registration.");
    }
}

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.

```typescript
/*
 * 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 { createQldbWriter, QldbSession, QldbWriter, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { makeReader, Reader } from "ion-js";
import { closeQldbSession, createQldbSession } from "./ConnectToLedger";
import { PERSON, VEHICLE_REGISTRATION } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId, getFieldValue, writeValueAsIon } 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 qldbWriter: QldbWriter = createQldbWriter();
    const personToInsert = { PersonId: secondaryOwnerId };
    writeValueAsIon(personToInsert, qldbWriter);
    await txn.executeInline(query, [qldbWriter]).then(async (result: Result) => {
        const resultList: Reader[] = result.getResultList();
        log(`VehicleRegistration Document IDs which had secondary owners added: ": ${resultList}`);
        prettyPrintResultList(resultList);
    });
}
```
/**
 * Query for a document ID with a government ID.
 * @param txn The 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 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 = ?";
    const qldbWriter: QldbWriter = createQldbWriter();
    writeValueAsIon(vin, qldbWriter);
    let doesExist: boolean = false;
    await txn.executeInline(query, [qldbWriter]).then((result: Result) => {
        const resultList: Reader[] = result.getResultList();
        resultList.forEach((reader: Reader) => {
            const secondaryOwnersList: any[] = getFieldValue(reader, ["SecondaryOwners"]);
            secondaryOwnersList.forEach((secondaryOwner) => {
                const secondaryOwnerReader: Reader = makeReader(JSON.stringify(secondaryOwner));
                if (getFieldValue(secondaryOwnerReader, ["PersonId"]) === secondaryOwnerId) {
                    doesExist = true;
                }
            });
        });
        return doesExist;
    });

    /**
     * Finds and adds secondary owners for a vehicle.
     * @returns Promise which fulfills with void.
     */
    var main = async function(): Promise<void> {
        let session: QldbSession;
        try {
            session = await createQldbSession();
            const vin: string = VEHICLE_REGISTRATION[1].VIN;
            const govId: string = PERSON[0].GovId;
            await session.executeLambda(async (txn) => {
                const documentId: string = await getDocumentIdByGovId(txn, govId);
            });
        } catch (e) {
            console.error(e);
        }
    }
}
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(`Retrying due to OCC conflict...`));

log(`Secondary owners successfully updated.`);
}

} finally {
    closeQldbSession(session);
}

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 (p. 133).

Step 6: View the Revision History for a Document

This is prerelease documentation for a feature in preview release. It is subject to change.

After modifying the registration data for a vehicle in the previous step (p. 128), 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.
    */
Amazon Quantum Ledger Database
(Amazon QLDB) Developer Guide
Node.js Tutorial
import { createQldbWriter, QldbSession, QldbWriter, Result, TransactionExecutor } from
"amazon-qldb-driver-nodejs";
import { Reader } from "ion-js";
import
import
import
import
import
import

{
{
{
{
{
{

closeQldbSession, createQldbSession } from "./ConnectToLedger";
VEHICLE_REGISTRATION } from "./model/SampleData";
VEHICLE_REGISTRATION_TABLE_NAME } from "./qldb/Constants";
prettyPrintResultList } from "./ScanTable";
error, log } from "./qldb/LogUtil";
getDocumentId, writeValueAsIon } 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();
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 = ?`;
const qldbWriter: QldbWriter = createQldbWriter();
writeValueAsIon(documentId, qldbWriter);

}

await txn.executeInline(query, [qldbWriter]).then((result: Result) => {
log(`Querying the 'VehicleRegistration' table's history using VIN: ${vin}.`);
const resultList: Reader[] = result.getResultList();
prettyPrintResultList(resultList);
});

/**
* Query a table's history for a particular set of documents.
* @returns Promise which fulfills with void.
*/
var main = async function(): Promise<void> {
let session: QldbSession;
try {
session = await createQldbSession();
const vin: string = VEHICLE_REGISTRATION[0].VIN;
await session.executeLambda(async (txn) => {
await previousPrimaryOwners(txn, vin);
}, () => log("Retrying due to OCC conflict..."));
} catch (e) {
error(`Unable to query history to find previous owners: ${e}`);
} finally {
closeQldbSession(session);
}
}
if (require.main === module) {
main();
}

134


Note

- You can view the revision history of a document by querying the built-in `History Function (p. 197)` in the following syntax.

```sql
SELECT * FROM history(
  table [, "start-time", "end-time"]
) AS h
WHERE h.metadata.id = 'id'
```

- Best practice is to qualify a history query with a document `id`. This helps to avoid inefficient queries.
- The `start-time` and `end-time` are both optional. They are Amazon Ion literal values that can be denoted with backticks (`...`). For more information, see `Querying Ion with PartiQL (p. 289)`.

2. To run the transpiled program, enter the following command.

```bash
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 (p. 135).

**Step 7: Verify a Document in a Ledger**

This is prerelease documentation for a feature in preview release. It is subject to change.

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 (p. 205).

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`

```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
```
import { ValueHolder } from "aws-sdk/clients/qldb";
import { Reader } 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 Reader into a ValueHolder.
 * Shape of the ValueHolder must be: `{IonText: "{strandId: <strandId>,
sequenceNo: <sequenceNo>}"}
 * @param reader The Reader that contains the block address values to convert.
 * @returns The ValueHolder that contains the strandId and sequenceNo.
 */
export function blockAddressToValueHolder(reader: Reader): ValueHolder {
    const strandId: string = getStrandId(reader);
    const sequenceNo: number = getSequenceNo(reader);
    const valueHolder: string = `{strandId: "${strandId}", sequenceNo: ${sequenceNo}}`;
    const blockAddress: ValueHolder = {IonText: valueHolder};
    return blockAddress;
}

/**
 * Helper method that steps into the provided Reader to get the Metadata ID.
 * @param reader The Reader to step into.
 * @returns The Metadata ID.
 */
export function getMetadataId(reader: Reader): string {
    reader.stepOut();
    reader.next();
    return reader.stringValue();
}

/**
 * Helper method that steps into the provided Reader to get the Sequence No.
 * @param reader The Reader to step into.
 * @returns The Sequence No.
 */
export function getSequenceNo(reader: Reader): number {
    reader.next();
    const fieldName: string = reader.fieldName();
    if (fieldName !== "sequenceNo") {
        throw new Error('Expected field name sequenceNo, found ${fieldName}.');
    }
    return reader.numberValue();
}

/**
 * Helper method that steps into the provided Reader to get the Strand ID.
 * @param reader The Reader to step into.
 * @returns The Strand ID.
 */
export function getStrandId(reader: Reader): string {
    reader.next();
reader.stepIn();
const type = reader.next();
if (type.name !== "struct") {
    throw new Error(`Unexpected format: expected struct, but got IonType: 
#{type.name}`);
}
reader.stepIn();
reader.next();
const fieldName: string = reader.fieldName();
if (fieldName !== "strandId") {
    throw new Error(`Expected field name strandId, found 
#{fieldName}.`);
}
return reader.stringValue();

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 { makeReader, Reader, toBase64 } from "ion-js";

import { getFieldValue } 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.
 */
function calculateRootHashFromInternalHash(internalHashes: Uint8Array[], leafHash: Uint8Array): Uint8Array {
  const rootHash: Uint8Array = internalHashes.reduce(joinHashesPairwise, leafHash);
  return rootHash;
}

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;
}

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;
}

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)) | ((1 << (bitShift - 1)) & (~alteredHash[bytePos]));
  return alteredHash;
}
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 blockText: string = valueHolder.IonText;
  const r: Reader = makeReader(blockText);
  const blockHash: Uint8Array = getFieldValue(r, "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`: '[]', `field`: '[]'}
 * @param valueHolder A structure containing an Ion string value.
 * @returns A list of hash values.
 */
function parseProof(valueHolder: ValueHolder): Uint8Array[] {
  let proofList: string = valueHolder.IonText;
  const r = makeReader(proofList);
  r.next();
  r.stepIn();
  proofList = proofList.replace("\]," ACTIVE\],");
  proofList = proofList.replace("[", "[");
  const array: string[] = proofList.split("","\[\]");

  const byteArray: Uint8Array[] = [];
  for (let i = 0; i < array.length; i++) {
    r.next();
    byteArray.push(r.byteValue());
  }
  return byteArray;
}
/**
 *  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 "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { digestResponseToString } from "./qldb/Util";

/**
 * Get the digest of a ledger's journal.
 * @param ledgerName Name of the ledger to operate on.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a GetDigestResponse.
 */
export async function getDigestResult(ledgerName: string, qldbClient: QLDB): Promise<GetDigestResponse> {
    const request: GetDigestRequest = {
        Name: ledgerName
    };
    const result: GetDigestResponse = await qldbClient.getDigest(request).promise();
    return result;
}
```javascript
/**
 * This is an example for retrieving the digest of a particular ledger.
 * @returns Promise which fulfills with void.
 */
var main = async function(): Promise<void> {
  try {
    const qldbClient: QLDB = new QLDB();
    log(`Retrieving the current digest for ledger: 
\n${LEDGER_NAME}.\n`);
    const digest: GetDigestResponse = await getDigestResult(LEDGER_NAME,
    qldbClient);
    log(`Success. Ledger digest: 
\n${digestResponseToString(digest)}.\n`);
  } catch (e) {
    error(`Unable to get a ledger digest: 
\n${e}\n`);
  }
}

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.

```
async function getRevision(
    ledgerName: string,
    documentId: string,
    blockAddress: ValueHolder,
    digestTipAddress: ValueHolder,
    qldbClient: QLDB
): Promise<GetRevisionResponse> {
    const request: GetRevisionRequest = {
        Name: ledgerName,
        BlockAddress: blockAddress,
        DocumentId: documentId,
        DigestTipAddress: digestTipAddress
    };    
    const result: GetRevisionResponse = await qldbClient.getRevision(request).promise();
    return result;
}

export async function lookupRegistrationForVin(txn: TransactionExecutor, vin: string):
    Promise<Reader[]> {
    log(`Querying the 'VehicleRegistration' table for VIN: ${vin}...`);
    let resultList: Reader[];
    const query: string = "SELECT blockAddress, metadata.id FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?";
    const qldbWriter: QldbWriter = createQldbWriter();
    writeValueAsIon(vin, qldbWriter);
    await txn.executeInline(query, [qldbWriter]).then(function(result) {
        resultList = result.getResultList();
    });
    return resultList;
}

export async function verifyRegistration(
    txn: TransactionExecutor,
    ledgerName: string,
    vin: string,
    qldbClient: QLDB
): Promise<void> {
    log(`Verifying the revision history for VIN: ${vin}...`);
    const query: string = "SELECT blockAddress, digestTipAddress FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?";
    const qldbWriter: QldbWriter = createQldbWriter();
    writeValueAsIon(vin, qldbWriter);
    await txn.executeInline(query, [qldbWriter]).then(function(result) {
        const resultList = result.getResultList();
        for (let i = 0; i < resultList.length; i++) {
            const revision = resultList[i];
            const digestTipAddress = revision.DigestTipAddress;
            const blockAddress = revision.BlockAddress;
            const documentId = revision.DocumentId;
            const ledgerName = revision.Name;
            const proof = await qldbClient.proveRevisionsForBlock(ledgerName, documentId, blockAddress, digestTipAddress);
`: Promise<void> {`n
  log(`Let’s verify the registration with VIN = ${vin}, in ledger = ${ledgerName}.`);
  const digest: GetDigestResponse = await getDigestResult(ledgerName, qldbClient);
  const digestBytes: Digest = digest.Digest;
  const digestTipAddress: ValueHolder = digest.DigestTipAddress;

  log(`Got a ledger digest: digest tip address = 
  ${(valueHolderToString(digestTipAddress)),
  digest = \${toBase64(<Uint8Array> digestBytes)}.`);
  log(`Querying the registration with VIN = ${vin} to verify each version of the`n
  registration...`);
  const resultList: Reader[] = 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,
      qldbClient
    );

    const revision: string = revisionResponse.Revision.IonText;
    const revisionReader: Reader = makeReader(revision);

    const documentHash: Uint8Array = getFieldValue(revisionReader, ["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`n
    NOT verified.
    The altered document hash is: $(toBase64(alteredDocumentHash))`n
    );
    verified = verifyDocument(alteredDocumentHash, digestBytes, proof);

    if (verified) {
      throw new Error("Expected altered document hash to not be verified against`n
      digest.");
    } else {
      log("Success! As expected flipping a bit in the document hash causes`n
      verification to fail.");
    }
    log(`Finished verifying the registration with VIN = ${vin} in ledger =`n
    ${ledgerName}.`);
  }
} `n

/** `n * Verify the integrity of a document revision in a QLDB ledger. `n * @returns Promise which fulfills with void. `n */
```javascript
var main = async function(): Promise<void> {
  let session: QldbSession;
  try {
    const qldbClient: QLDB = new QLDB();
    session = await createQldbSession();
    const registration = VEHICLE_REGISTRATION[0];
    const vin: string = registration.VIN;
    await session.executeLambda(async (txn) => {
      await verifyRegistration(txn, LEDGER_NAME, vin, qldbClient);
    });
  } catch (e) {
    error(`Unable to verify revision: ${e}`);
  } finally {
    closeQldbSession(session);
  }
}

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 (p. 144)**.

### Step 8 (Optional): Clean Up Resources

**This is prerelease documentation for a feature in preview release. It is subject to change.**

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.

```typescript
/*
 * 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,
 */
```
import { isResourceNotFoundException } from "amazon-qldb-driver-nodejs";
import { QLDB } from "aw-sdk";
import { DeleteLedgerRequest, DescribeLedgerRequest } from "aw-sdk/clients/qldb";

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 qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */
export async function deleteLedger(ledgerName: string, qldbClient: QLDB): Promise<void> {
  log(`Attempting to delete the ledger with name: ${ledgerName}`);
  const request: DeleteLedgerRequest = {
    Name: ledgerName
  };
  await qldbClient.deleteLedger(request).promise();
  log("Success.");
}

/**
 * Wait for the ledger to be deleted.
 * @param ledgerName Name of the ledger to be deleted.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */
export async function waitForDeleted(ledgerName: string, qldbClient: QLDB): Promise<void> {
  log("Waiting for the ledger to be deleted...");
  const request: DescribeLedgerRequest = {
    Name: ledgerName
  };
  while (true) {
    try {
      await qldbClient.describeLedger(request).promise();
      log("The ledger is still being deleted. Please wait...");
      await sleep(LEDGER_DELETION_POLL_PERIOD_MS);
    } catch (e) {
      if (isResourceNotFoundException(e)) {
        log("Success. Ledger is deleted.");
        break;
      } else {
        throw e;
      }
    }
  }
}

/**
 * Delete a ledger.
 * @returns Promise which fulfills with void.
 */
var main = async function(): Promise<void> {
  try {
    const qldbClient: QLDB = new QLDB();
    await setDeletionProtection(LEDGER_NAME, qldbClient, false);
    await deleteLedger(LEDGER_NAME, qldbClient);
    await waitForDeleted(LEDGER_NAME, qldbClient);
  } 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 Driver for Python

To work with data in your ledger, you can connect to Amazon QLDB from your Python application using an AWS-provided driver. Then you can code your application to use the transactional data API. The following sections describe how to get started with the QLDB Driver for Python.

Topics
- See Also (p. 146)
- Prerequisites (p. 146)
- Quick Start Installation (Driver Only) (p. 147)
- Amazon QLDB Python Tutorial (p. 147)

See Also
For more information about the functionality supported by the Python driver, see the following:

- API Reference
- Common Errors (p. 185)
- Driver Source Code
- Sample Application Source Code
- Amazon Ion API Reference

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 (p. 11). This includes signing up for AWS and getting an AWS access key for development.
2. Install Python version 3.4 or later from the Python downloads site.
3. Set up your AWS credentials and your default AWS Region. For instructions, see Quickstart in the AWS SDK for Python (Boto 3) documentation.

For a complete list of available Regions, see Amazon QLDB Endpoints and Quotas in the AWS General Reference.

4. Next, you can download the complete tutorial sample application, or you can install only the driver in an existing Python project.
   - To run the complete tutorial with the QLDB sample application, proceed to the Python Tutorial (p. 147).
   - To install only the QLDB Driver and the AWS SDK for Python (Boto 3) in an existing project, see Quick Start Installation (Driver Only) (p. 147).

**Quick Start Installation (Driver Only)**

To install only the QLDB Driver from PyPI using `pip` (a package manager for Python), enter the following at the command line.

```
pip install pyqldb
```

Installing the driver also installs its dependencies, including the AWS SDK for Python (Boto 3).

**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.

```
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 code examples that demonstrate other operations such as how to create a ledger and execute data transactions on a ledger, see the Python Tutorial (p. 147).

**Amazon QLDB Python Tutorial**

In this tutorial, you use the QLDB Driver for Python to create an Amazon QLDB ledger and populate it with tables and sample data. The driver enables your application to interact with QLDB using the transactional data 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 sections 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.

As you work through this tutorial, you can refer to the QLDB low-level client section of the AWS SDK for Python (Boto 3) API Reference.

**Prerequisites**

Before you get started, make sure that you complete the Prerequisites (p. 146). This includes signing up for AWS and getting an AWS access key for development.
Installing the Amazon QLDB Python Sample Application

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.

Prerequisites

Before you get started, make sure that you complete the Prerequisites (p. 146). This includes signing up for AWS and getting an AWS access key for development.

Installation

To install the sample application

1. Enter the following `pip` command to clone and install the sample application from GitHub.

   ```bash
   pip install https://github.com/aws-samples/amazon-qldb-dmv-sample-python/archive/master.zip
   
   The sample application packages the complete source code from this tutorial and its dependencies, including the Python driver and the AWS SDK for Python (Boto 3).
   
   2. Before you start running code at the command line, switch your current working directory to the location where the `pyqldbsamples` package is installed. Enter the following command.

   ```bash
   cd $(python -c "import pyqldbsamples; print(pyqldbsamples.__path__[0])")
   
   3. Proceed to Step 1: Create a New Ledger (p. 148) to start the tutorial and create a ledger.

Step 1: Create a New Ledger

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.

   ```python
   # Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
   # SPDX-License-Identifier: MIT-0
   #
   ```
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.

```
from logging import basicConfig, getLogger, INFO
from time import sleep
from boto3 import client
from pyqldbsamples.constants import Constants

# 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:
from logging import basicConfig, getLogger, INFO
from time import import sleep
from boto3 import import client
from pyqldbsamples.constants import 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 = qldb_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)  

if __name__ == '__main__':  
    """
    Create a ledger and wait for it to be active.
    """
    try:  
        create_ledger(Constants.LEDGER_NAME)  
        wait_for_active(Constants.LEDGER_NAME)  
    except Exception as e:  
        logger.exception('Unable to create the ledger!')  
        raise e  

**Note**

- In the `create_ledger` call, you must specify a ledger name and a permissions mode. The only permissions mode that is currently supported for a ledger is `ALLOW_ALL`.

- 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).
3. To run the program, enter the following command.

```python
python create_ledger.py
```

To verify your connection to the new ledger, proceed to Step 2: Test Connectivity to the Ledger (p. 151).

**Step 2: Test Connectivity to the Ledger**

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.

```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:

from logging import basicConfig, getLogger, INFO
from botocore.exceptions import ClientError
from botocore.config import Config
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
from pyqldbamples.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].
    """

    if boto3_session is None:
        boto3_session = boto3.Session()

    if region_name is None:
        region_name = boto3_session.region_name

    if endpoint_url is None:
        endpoint_url = boto3_session._session.config.convert_region_to_endpoint(region_name)

    if region_name is None:
        region_name = Constants.REGION_NAME

    if endpoint_url is None:
        endpoint_url = Constants.ENDPOINT_URL

    config = Config(RegionName=region_name, ParamDataSource="local")

    return PooledQldbDriver(ledger_name, boto3_session=boto3_session, config=config)
```

151
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 session 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 control plane API operations listed in the Amazon QLDB API
  Reference (p. 331).

- 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 (p. 153).
Step 3: Create Tables, Indexes, and Sample Data

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>
<thead>
<tr>
<th>Table Name</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleRegistration</td>
<td>VIN</td>
</tr>
<tr>
<td>VehicleRegistration</td>
<td>LicensePlateNumber</td>
</tr>
<tr>
<td>Vehicle</td>
<td>VIN</td>
</tr>
<tr>
<td>Person</td>
<td>GovId</td>
</tr>
<tr>
<td>DriversLicense</td>
<td>LicenseNumber</td>
</tr>
<tr>
<td>DriversLicense</td>
<td>PersonId</td>
</tr>
</tbody>
</table>

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.

**Important**

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 and Terminology in Amazon QLDB (p. 7). To learn more about metadata, see Querying Document Metadata (p. 193).

To create tables and indexes

1. Use the following program (create_table.py) to create the previously mentioned tables.

```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:
from logging import basicConfig, getLogger, INFO
from pyqlibsamples.constants import Constants
from pyqlibsamples.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.
    
    :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 a QLDB session.
In this example, you run multiple CREATE TABLE PartiQL statements with a single Lambda
expression.

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.

```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:
from logging import basicConfig, getLogger, INFO
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)  # type: ignore
basicConfig(level=INFO)

def create_index(transaction_executor, table_name, index_attribute):
    """
    Create an index for a particular table.
    """
    :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 '{0}'...".format(index_attribute))
    statement = 'CREATE INDEX on `{0}` ({1})'.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 (p. 269) data.

```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.
from datetime import datetime
from decimal import Decimal
from logging import basicConfig, getLogger, INFO
logger = getLogger(__name__)
basicConfig(level=INFO)
class SampleData:
Sample domain objects for use throughout this tutorial.

```py
# 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': '3HGGK5GS3FM761765',
'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': 'KM8SRDF6EU074761',
  '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': '1HVBAANXWH544237',
  '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_iion(py_object):
    
    Convert a Python object into an Ion object.

    :type py_object: 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.
    """
    :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)
    list_of_ids = map(lambda table: table.get('id'), cursor)
    return list_of_ids

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.stream_cursor.StreamCursor`
    :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
```python
def print_result(cursor):
    """
    Pretty print the result set. Returns the number of documents in the result set.
    :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.
        logger.info(dumps(row, binary=False, indent='  ', omit_version_marker=True))
        result_counter += 1
    return result_counter
```

**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 (p. 195).

2. Use the following program (`insert_document.py`) to insert the sample data into your tables.

```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:
from logging import basicConfig, getLogger, INFO
from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion, SampleData,
    get_document_ids_from_dml_results
from pyqldbsamples.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
    """
```

---

161
**Python Tutorial**

```python
: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.
    :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 {} table...'.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.
    :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)
```

```python
if __name__ == '__main__':
    """
```
Insert documents into a table in a QLDB ledger.

```python
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 a list of Ion 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.

   ```bash
   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 (p. 163).

### Step 4: Query the Tables in a Ledger

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 (p. 272) as its query language and Amazon Ion (p. 269) 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.

**To query the tables**

1. Use the following program (scan_table.py) to scan all documents in all tables in your ledger.

   ```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,
   ```
from pyqldbamples.model.sample_data import print_result
from pyqldbamples.connect_to_ledger import create_qldb_session

logger = get_logger(__name__)
basicConfig(level=INFO)

def scan_table(transaction_executor, table_name):
    
    :param transaction_executor: An Executor object allowing for execution of statements within a transaction.

    :type table_name: str
    :param table_name: The name of the table to operate on.

    :return: Cursor on the result set of a statement query.

    logger.info('Scanning {}...'.format(table_name))
    query = 'SELECT * FROM {}'.format(table_name)
    return transaction_executor.execute_statement(query)

if __name__ == '__main__':
    
    try:
        with create_qldb_session() as session:
            # Scan all the tables and print their documents.
            tables = session.list_tables()
            for table in tables:
                cursor = session.execute_lambda(
                    lambda executor: scan_table(executor, table),
                    retry_indicator=lambda retry_attempt: logger.info('Retrying due to OCC conflict...'))
                logger.info('Scan successful!')
                print_result(cursor)
    except Exception:
        logger.exception('Unable to scan tables.')

2. To run the program, enter the following command.

    python scan_table.py

3. Use the following program (find_vehicles.py) 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
    #
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 = get_logger(__name__)
basicConfig(level=INFO)

def find_vehicles_for_owner(transaction_executor, gov_id):
    """
    Find vehicles registered under a driver using their government ID.
    """
    :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.

4. 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 (p. 166).

Step 5: Modify Documents in a Ledger

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.

```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:
from logging import basicConfig, getLogger, INFO
from pyqldbsamples.add_secondary_owner import get_document_ids, print_result, SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion
from pyqldbsamples.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.
```
def find_primary_owner_for_vehicle(transaction_executor, vin):
    """
    Find the primary owner of a vehicle given its VIN.
    """
    :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.
    """
    :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.

    :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)
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.

    :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.
from pyqldbsamples.model.sample_data import to_ion_struct, get_document_ids, print_result, SampleData, convert_object_to_ion
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session
logger = get_logger(__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.
    :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: \{0\}.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.
    :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: \{0\}...'.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.
    :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.
    :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_qldb_session() as session:
            session.execute_lambda(lambda executor: register_secondary_owner(executor,
                vin, gov_id),
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 (p. 171).

**Step 6: View the Revision History for a Document**

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.

```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:
from datetime import datetime, timedelta
from logging import basicConfig, getLogger, INFO
from pyqldbamples.model.sample_data import print_result, get_document_ids, SampleData
from pyqldbamples.constants import Constants
from pyqldbamples.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
    
    
    lambda retry_attempt: logger.info('Retrying due to OCC conflict...
    logger.info('Secondary owners successfully updated.
    except Exception:
        logger.exception('Error adding secondary owner.
```

171
```python
: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.
    """
    :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 (p. 197) in the following syntax.

    SELECT * FROM history( table [`, `start-time`, `, `end-time`] ) AS h
    WHERE h.metadata.id = `id`

- Best practice is to qualify a history query with a document id. This helps to avoid
  inefficient queries.
```
- The start-time and end-time are both optional. They are Amazon Ion literal values that can be denoted with backticks (`...`). To learn more, see Querying Ion with PartiQL (p. 289).

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 (p. 173).

Step 7: Verify a Document in a Ledger

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 (p. 205).

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`

   ```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
   # 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': {}}
   ```

   173
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:
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.
```

```python

```text

```python

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
    :type hash2: bytes
    :param hash1: The first hash value.
    :param hash2: The second hash value.
    
    :rtype: bytes
    :return: The concatenated and altered hash value.
    """
    hash1 = bytes(hash1).copy()
    hash2 = bytes(hash2).copy()
    hash1 = array('b', hash1)
    hash2 = array('b', hash2)
    for i in range(len(hash2) - 1, -1, -1):
        hash1[i] = hash2[i]
    return bytes(hash1)
```python
: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 calculated from the document revision.
    :return: True if the document revision is verified, False otherwise.
    """
    calculated_root_hash = calculate_root_hash_from_internal_hashes(parsed_proof, leaf_hash)
    return calculated_root_hash == document_hash
```

.. code-block:: python

    :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:`pyqldbsamples.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. qldb_string_utils.py

.. code-block:: 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.

    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.
# 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:
from logging import basicConfig, getLogger, INFO
from boto3 import client
from pyqldbsamples.constants import Constants
from pyqldbsamples.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

if __name__ == '__main__':
    """
    This is an example for retrieving the digest of a particular ledger.
    """
    try:
        digest = get_digest_result(Constants.LEDGER_NAME)
    except Exception as e:
        logger.exception('Unable to get a ledger digest!')
        raise e

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.
# 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:

from logging import basicConfig, getLogger, INFO
from amazon.ion.simpleion import loads
from boto3 import client
from pyqldbsamples.constants import Constants
from pyqldbsamples.get_digest import get_digest_result
from pyqldbsamples.model.sample_data import SampleData, convert_object_to_ion
from pyqldbsamples.block_address import block_address_to_dictionary
from pyqldbsamples.verifier import verify_document, flip_random_bit, to_base_64
from pyqldbsamples.connect_to_ledger import create_qldb_session
from pyqldbsamples.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 = qldb_client.get_revision(Name=ledger_name, BlockAddress=block_address,
                                       DocumentId=document_id,
                                       DigestTipAddress=digest_tip_address)
    return result

def lookup_registration_for_vin(qldb_session, vin):
    """
    Query revision history for a particular vehicle for verification.

    :type qldb_session: :py:class:`pyqldb.session.qldb_session.QldbSession`
    :param qldb_session: An instance of the QldbSession class.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration with.

    :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)]
    result = qldb_session.run_sql(query, parameters=parameters)
```
cursor = qldb_session.execute_statement(query, parameters)
return cursor

def verify_registration(qldb_session, ledger_name, vin):
    """
    Verify each version of the registration for the given VIN.
    :type qldb_session: :py:class:`pyqldb.session.qldb_session.QldbSession`
    :param qldb_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(qldb_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 (p. 182).

**Step 8 (Optional): Clean Up Resources**

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.

```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:
from logging import basicConfig, getLogger, INFO
from time import sleep
from boto3 import client
from pyqldbsamples.constants import Constants
```

182
```python
from pyqldbsamples.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)
            sleep(LEDGER_DELETION_POLL_PERIOD_SEC)
        except qldb_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 = qldb_client.update_ledger(Name=ledger_name,
    DeletionProtection=deletion_protection)
    logger.info('Success. Ledger updated: {}').format(result)

if __name__ == '__main__':
```

Delete a ledger.

```python
try:
    set_deletion_protection(Constants.LEDGER_NAME, False)
    delete_ledger(Constants.LEDGER_NAME)
    wait_for_deleted(Constants.LEDGER_NAME)
except Exception as e:
    logger.exception('Unable to delete the ledger.')
    raise e
```

**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`).

```python
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
# This code expects that you have AWS credentials setup per:

from boto3 import client
from pyqldbsamples.constants 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

if __name__ == '__main__':
    
    Describe a QLDB ledger.
```
2. To run the program, enter the following command.

```
python delete_ledger.py
```

Common Errors for the Amazon QLDB Driver

This section describes runtime errors that can be thrown by the Amazon QLDB Driver when calling the `qldbsession` APIs.

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.

**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 limit, any work done on the transaction is rejected, and Amazon QLDB discards the session. This limit protects the client from leaking sessions by starting transactions and not committing or aborting them.

If this is a common exception in your application, it is likely that transactions are simply taking too long to execute. If transaction execution is taking longer than 30 seconds, optimize your statements to speed up the transactions. Examples of statement optimization include executing fewer statements per transaction and tuning your table indexes.

**InvalidSessionException**

Message: Exceeded the session idle time limit: Session `sessionId`

QLDB has discarded the session because it remained idle past the idle time limit. QLDB defines an idle session as a session that is not in an active transaction. A session can be idle for up to 60 seconds, after which QLDB discards the session to protect the client from leaking sessions.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the `PooledQldbDriver`, which is a variation of the driver object that is included with the QLDB Driver for every language. This driver manages the session pool and its health on the application's behalf.

**InvalidSessionException**

Message: Session `sessionId` has expired

QLDB discarded the session because it exceeded its maximum total lifetime. QLDB discards sessions after approximately 15 minutes, regardless of idleness. 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 `PooledQldbDriver`, which is a variation of the driver object that is included with the QLDB Driver for every language. This driver 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.
- Sessions that expire or exceed the idle time limit 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 PooledQldbDriver, which is a variation of the driver object that is included with the QLDB Driver for every language. This driver 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. If your application is consistently getting throttled by QLDB for StartSessionRequest calls, we recommend using the PooledQldbDriver. This is a variation of the driver object that is included with the QLDB Driver for every language. This 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 in Amazon QLDB (p. 403). 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 withPoolLimit method of the Amazon QLDB Driver for Java to configure the maximum number of available sessions in a pool.

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 aborted.

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 aborting a transaction. To mitigate this issue, the client should read the data before closing the transaction.
Creating Tables and Inserting Documents

After creating an Amazon QLDB ledger, your first step is to create a table with a basic CREATE TABLE (p. 294) statement. The fundamental data model in QLDB is described as follows:

1. Ledgers consist of user-defined tables. A table can be thought of as a **bag**, which is an unordered collection in PartiQL.
2. Tables consist of QLDB Documents (p. 273), which are datasets in Amazon Ion (p. 269) struct format.

Topics
- Creating Tables and Indexes (p. 187)
- Table Metadata (p. 188)
- Inserting Documents (p. 189)

**Creating Tables and Indexes**

Tables have simple names with no namespaces. QLDB supports open content and does not enforce schema, so you don't define attributes or data types when creating tables.

```
CREATE TABLE VehicleRegistration
```
CREATE TABLE Vehicle

You can also create indexes on tables to help speed up queries.

**Note**
In QLDB, indexes are optional. They can improve query performance. However, note the following:

- Indexes can only be created on empty tables.
- Indexes can only be created on a single field. Composite indexes are currently not supported.
- Indexes can't be dropped after they are created.
- Query performance is improved only when you use an equality predicate; for example, `fieldName = 123456789`.

QLDB does not currently honor inequalities in query predicates. One implication of this is that range filtered scans are not implemented.
- Names of indexed fields can have a maximum of 128 characters.

CREATE INDEX ON VehicleRegistration (VIN)

CREATE INDEX ON VehicleRegistration (LicensePlateNumber)

CREATE INDEX ON Vehicle (VIN)

**Table Metadata**

Each table that you create in a ledger has a system-assigned unique ID. You can find a table's ID and other metadata by querying the system table `information_schema.user_tables`.

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: "5PLf9SXwdd631PaSia006",
    name: "VehicleRegistration",
    indexes: [{ expr: "[VIN]" }, { expr: "[LicensePlateNumber]" }],
    status: "ACTIVE"
}
```

**Table metadata fields**

- **tableId**—The table ID.
- **name**—The table name.
- **indexes**—The list of indexes on the table.
- **status**—The table's current status (ACTIVE or INACTIVE). A table becomes INACTIVE when you DROP it.
Inserting Documents

Then you can insert documents into your tables. QLDB documents are stored in Amazon Ion format. The following PartiQL INSERT (p. 299) statements include a subset of the vehicle registration sample data used in Getting Started with the Amazon QLDB Console (p. 18).

```sql
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': '294jJ3YUoH1IEEm8GSabOs'},
     'SecondaryOwners': []
   }},

  {'VIN': 'KM8SRDHF6EU074761',
   'LicensePlateNumber': 'CA762X',
   'State': 'WA',
   'City': 'Kent',
   'PendingPenaltyTicketAmount': 130.75,
   'ValidFromDate': '2017-09-14T',
   'ValidToDate': '2020-06-25T',
   'Owners': {
     'PrimaryOwner': {'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9'},
     'SecondaryOwners': []
   } }
>>
```

```sql
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 PartiQL's syntax and semantics, see Querying Ion with PartiQL (p. 289).

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.

**Important**
Because QLDB does not 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 (p. 190).

---

### Querying Your Data

The user view returns the latest non-deleted revision of your 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 (p. 300) in the QLDB PartiQL Reference.

**Topics**
- Basic Queries (p. 190)
- Projections and Filters (p. 191)
- Joins (p. 192)
- Nested Data (p. 192)

### Basic Queries

Basic SELECT queries return the documents that you inserted into the table. The following queries show results for the vehicle registration documents previously inserted in Creating Tables and Inserting Documents (p. 187).

**Note**
PartiQL is SQL compatible, so it supports table scans such as this example. But we generally don’t recommend running SELECT queries without a WHERE predicate clause. QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. The best practice is to run queries that filter on a document ID or a table's indexed fields.

```
SELECT * FROM VehicleRegistration
```

```
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  PendingPenaltyTicketAmount: 90.25,
  ValidFrom: "2017-08-21T",
  ValidTo: "2020-05-11T",
  Owners: {
    PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOu64s" }]
  }
},
```
Projections and Filters

You can do projections (targeted `SELECT`) and 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 is 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.City = 'Seattle' --string
AND r.PendingPenaltyTicketAmount < 100.0 --decimal
AND r.ValidToDate >= '2019-09-04T' --timestamp with day precision
```
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
```

```
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  PendingPenaltyTicketAmount: 90.25,
  ValidFrom: 2017-08-21T,
  ValidTo: 2020-05-11T,
  Owners: {
    PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }],
  },
  Type: "Sedan",
  Year: 2011,
  Make: "Audi",
  Model: "A5",
  Color: "Silver"
},
{
  VIN: "KM8SRDHF6EU074761",
  LicensePlateNumber: "CA762X",
  State: "WA",
  City: "Kent",
  PendingPenaltyTicketAmount: 130.75,
  ValidFrom: 2017-09-14T,
  ValidTo: 2020-06-25T,
  Owners: {
    PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
    SecondaryOwners: []
  },
  Type: "Sedan",
  Year: 2015,
  Make: "Tesla",
  Model: "Model S",
  Color: "Blue"
}
```

Nested Data

PartiQL has Amazon Ion extensions that enable you 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,
```
The following shows a subquery in the `SELECT` list that projects nested data, in addition to an inner join.

```sql
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
```

To learn how to query your document metadata, proceed to **Querying Document Metadata (p. 193)**.

**Querying Document Metadata**

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 IDs, commit timestamps, and journal attributes.

**Committed View**

You can access this 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 QLDB-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.

```sql
SELECT * FROM _ql_committed_VehicleRegistration
```
Using the data previously inserted in **Creating Tables and Inserting Documents (p. 187)**, the output of this query shows the system contents of each document's latest non-deleted revision. The system document includes the vehicle registration data nested in the `data` field, metadata nested in the `metadata` field, its SHA-256 hash value, and the location in the journal where it was committed.

```
{
  blockAddress:
    
      
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14

  hash:{"wCsmM6qD4STXz0WYmE+47nZwWtcCz9D6zNtCiM5GoWg="},
  data:
    
      
        VIN: "1N4AL11D5C109151",
        LicensePlateNumber: "LEWISR261LL",
        State: "WA",
        City: "Seattle",
        PendingPenaltyTicketAmount: 90.25,
        ValidFrom: 2017-08-21T,
        ValidTo: 2020-05-11T,
        Owners: 
          
            
              PrimaryOwner: { PersonId: "294jJ3YUoHlIEEm8GSab0s" },
            SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcO1u64s" }]
          

  },
  metadata:
    
      
        id:"3Qv67yjXEwB9Sjmikuug6Gc",
        version:0,
        txId:"HgXAKLjAtV0Hq41NpYdX60"
}
},
{
  blockAddress:
    
      
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14

  hash:{"wPuwH60TtccVg/23BFp+redRXuCALkbDihkEvCXx22Jk="},
  data:
    
      
        VIN: "KM8SRDHF6EU074761",
        LicensePlateNumber: "CA762X",
        State: "WA",
        City: "Kent",
        PendingPenaltyTicketAmount: 130.75,
        ValidFrom: 2017-09-14T,
        ValidTo: 2020-06-25T,
        Owners: 
          
            
              PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
              SecondaryOwners: []
          

  },
  metadata:
    
      
        id:"J0zfB3IwGU727mpPwWyxg",
        version:0,
        txId:"HgXAKLjAtV0Hq41NpYdX60"
}
```

These documents include the following system attributes:

- **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 value that uniquely represents the document revision. The hash covers the data and metadata fields and can be used for cryptographic verification.

• **metadata**—The document's metadata attributes.
  • **id**—The system-assigned document ID.
  • **version**—The 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.

To learn how to query the document ID field in the default user view, proceed to Using the BY Clause to Query Document ID (p. 195).

### Using the BY Clause to Query Document ID

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 explained in Creating Tables and Inserting Documents (p. 187). 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 rid, r.VIN, r.LicensePlateNumber, r.State, r.City, r.Owners
FROM VehicleRegistration AS r BY rid
WHERE rid = '3Qv67yjXEWb9SjmvkuG6Cp'
```

```
{  
  rid: "3Qv67yjXEWb9SjmvkuG6Cp",
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR26ILL",
  State: "WA",
  City: "Seattle",
  Owners: {  
    PrimaryOwner: { PersonId: "294j3YUoH1IEEm8GSabOs" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWOu64s" }]  
  }
}
```

In this query, **rid** is a user-defined alias that is declared in the **FROM** clause, using the **BY** keyword. This **rid** 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.

To learn how to make changes to a document in your table, proceed to Updating and Deleting Documents (p. 196).
Updating and Deleting Documents

In Amazon QLDB, a document revision is a structure that represents a single iteration of a document's full dataset and includes both your data and system-generated metadata. Each revision is uniquely identified by a combination of the document ID and a zero-based version number.

The lifecycle of a document ends when it's deleted from a table. This means that no document with the same document ID can be created again in the same ledger.

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.

```sql
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' : []
    }
}
```

Before you continue, find the document's id with the following query. Save this id because you need it for the history function in the next section.

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

```sql
UPDATE VehicleRegistration AS r
SET r.City = 'Bellevue'
WHERE r.VIN = '1HVBBAANXWH544237'
```

```sql
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

For information about the syntax and parameters of these DML statements, see UPDATE (p. 303) and DELETE (p. 295) in the Qldb PartiQL 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 (p. 196).

Querying Revision History

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 (p. 196) by querying the built-in history function.
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.

The history function has the following syntax.

```
SELECT * FROM history(table [, `start-time`, `end-time`] ) AS h
WHERE h.metadata.id = 'id'
```

**Note**
As a best practice, qualify a history query with a document id. This helps to avoid inefficient queries.

The start-time and end-time are Ion timestamp literals that can be denoted with backticks (`...`). To learn more, see Querying Ion with PartiQL (p. 289).

The table parameter can be either a table name or a 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, see Querying the History of Dropped Tables (p. 199).

The start and end times specify the time range during which any revisions were active. They don't specify the transaction time range during which revisions were committed to the journal. The start and end time parameters have the following behavior:

- The start-time and end-time are optional parameters that 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.

To query the document's history, use the id that you previously saved in Updating and Deleting Documents (p. 196). For example, the following query returns the revision history for id = ADR2LL1fGslUS4jr4EqTdnQF between 2000T and 2019-06-05T23:59:59Z. Be sure to replace the id, start time, and end time with your own values as appropriate.

```
WHERE h.metadata.id = 'ADR2LL1fGslUS4jr4EqTdnQF' --replace with your id
```

Your query results should look similar to the following.

```json
{
    blockAddress:
        strandId:"JdxjK9bSYB5jMHWcI464T",
    sequenceNo:14
}
```
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: ADR2Ll1fGsU4Jr4EqTdnQF.
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.

Querying the History of Dropped Tables

In addition to a table name, you can also query the QLDB history function with a table ID as the first input parameter. This enables you to query the history of dropped tables. After a table is dropped, you can no longer query its history with the table name.

First, find the table ID by querying the Table Metadata (p. 188). 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 the previous section. The following is an example that queries the history of document ID ADR2Ll1fGsU4Jr4EqTdnQF 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
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

Undropping Tables

When you drop a table in QLDB, you are just inactivating it. You can still use the UNDROP 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. The table ID is a unique identifier that should be enclosed in double quotation marks.

```
UNDROP TABLE "5PLf9SXwndd631PaSIa006"
```

The status of VehicleRegistration should now be ACTIVE.
Amazon QLDB Concurrency Model

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, QLDB makes it easy to access the complete and verifiable history of all changes to your data, and to stream coherent transactions to other data services as needed.

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 executes concurrent transactions in a serial manner, such that it produces the same effect as if those transactions were executed serially.

Before committing, each transaction performs a validation check to ensure that no other committed transaction has modified the snapshot of data that it's accessing. If this check reveals conflicting modifications, or the state of the data snapshot 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.

For a code example that shows how to detect and handle OCC conflicts, see Handling OCC Conflict Exceptions (p. 201).

OCC Example

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 execute the following `SELECT` and `INSERT` statements within a single transaction. Their application executes the `INSERT` statement only if the `SELECT` statement returns no results.

**SELECT** from Vehicle v WHERE v.VIN = 'ABCDE12345EXAMPLE'

**INSERT** into Vehicle VALUES
{
    '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.

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 execute the `INSERT`.

### Handling OCC Conflict Exceptions

As described in Amazon QLDB Concurrency Model (p. 200), Amazon QLDB performs validation checks when trying to write transactions to the journal. If this verification phase fails, QLDB returns an optimistic concurrency control (OCC) conflict exception to the application layer. The application can then retry transactions that were rejected.
Note
The QLDB driver also provides an execution method that transparently retries OCC conflict exceptions up to a configurable limit by default. If the transaction still fails after the limit is reached, the driver throws the exception.

This section provides a Java example that produces an OccConflictException. It demonstrates how to handle this conflict by catching the exception and then retrying the failed transaction.

Prerequisites

Before running the demo, make sure that you complete the following tasks if you haven't already done so:

1. Follow the steps in Installing the Amazon QLDB Java Sample Application (p. 36) to set up your Java environment and install the sample application.
2. Do steps 1–3 in the Java Tutorial (p. 36) to create a ledger named vehicle-registration and load it with sample data.

Running the Demo

Compile and run the following program (OCCConflictDemo.java), which simulates an OCC conflict exception.

```java
package software.amazon.qldb.tutorial;

import com.amazon.ion.IonValue;
import com.amazonaws.services.qldbsession.model.OccConflictException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Transaction;
import software.amazon.qldb.tutorial.model.SampleData;
import java.io.IOException;
import java.util.Collections;
import java.util.List;

/**
 * Demonstrates how to handle OCC conflicts, where two users try to execute and commit changes to the same document.
 * When OCC conflict occurs on execute or commit, implicitly handled by restarting the transaction.
 */
```
In this example, two sessions on the same ledger try to access the registration city for the same Vehicle Id.

This code expects that you have AWS credentials setup per:

```
public final class OccConflictDemo {

    public static final Logger log = LoggerFactory.getLogger(OccConflictDemo.class);

    private OccConflictDemo() {
    }

    /**
     * Commit the transaction and retry up to a constant number of times.
     * @param qldbSession A QLDB session.
     * @param transaction An open transaction.
     * @param statement The statement to execute.
     * @param parameters
     * @param retries.
     * @throws IOException if failed to convert parameter into an IonValue.
     */
    public static void commitTransaction(final QldbSession qldbSession, Transaction transaction, final String statement, final String parameters)
        throws IOException {
        for (int i = 0; i < Constants.RETRY_LIMIT; i++) {
            try {
                transaction.commit();
                log.info("Commit successful after {} retries.", i);
                break;
            } catch (OccConflictException e) {
                log.info("Commit failed due to an OCC conflict. Restart transaction.");
                transaction = qldbSession.startTransaction();
                executeTransaction(qldbSession, transaction, statement, parameters);
            }
        }
    }

    /**
     * Start a new transaction and execute it with the given statement.
     * @param qldbSession A QLDB session.
     * @param transaction An open transaction.
     * @param statement The statement to execute.
     * @param parameters
     * @param retries.
     * @throws IOException if failed to convert parameter into an IonValue.
     */
    public static void executeTransaction(final QldbSession qldbSession, Transaction transaction, final String statement, final String parameters)
        throws IOException {
        for (int i = 0; i < Constants.RETRY_LIMIT; i++) {
            try {
                final List<IonValue> parameter = Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(parameters));
                transaction.execute(statement, parameter);
                log.info("Execute successful after {} retries.", i);
                break;
            } catch (OccConflictException e) {
                log.info("Execute failed due to an OCC conflict. Restart transaction.");
                transaction = qldbSession.startTransaction();
                executeTransaction(qldbSession, transaction, statement, parameters);
            }
        }
    }
}
```
From this Java code example, you can observe the following:

- When using the `execute()` method to execute a single statement, QLDB implicitly starts and commits a transaction, and any OCC conflicts are transparently handled. You can take full control over when the transaction is committed by explicitly starting a transaction, as follows.

```
final Transaction t1 = session.startTransaction();
```

- When you explicitly start a transaction and execute a statement, you must manually commit the transaction in your session.

```
t1.commit();
```

- You also assume responsibility of OCC conflict handling. To catch an OCC exception, put the `commit()` statement inside a `try` block. Then in the `catch` block, check for an OCC conflict exception as follows.

```
} catch (OccConflictsException e) {
```

- If an OCC conflict exception is caught, you can print an error message and retry the failed transaction, as shown in the code example.
Data Verification in Amazon QLDB

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 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. It enables you to look back and verify the integrity of your document revisions relative to that signature.

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. These entries are the objects whose integrity you can verify 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.

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 (p. 206)
- Digest (p. 207)
- Merkle Tree (p. 207)
- Proof (p. 208)

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 SHA-256 hash function is one-way, meaning that it’s not mathematically feasible to compute the input when given an output.
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.

QLDB enables you to generate a digest as a secure output file. Then, you can 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 enables efficient verification of large datasets with an audit proof mechanism. For more information about Merkle trees, see the [Merkle tree Wikipedia page](https://en.wikipedia.org/wiki/Merkle_tree). To learn more about Merkle audit proofs and for an example use case, see [How Log Proofs Work](https://www.certificate-transparency.org/logproofs.html) 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.

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.

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 do this, start with hash A and concatenate it with hash B. Then, hash the result to compute D. Next, use D and E to compute F. Finally, 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.
What Is the Verification Process in QLDB?

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.

Note
What you are specifically proving is that the document revision was not altered between the time that you saved this digest and when you run the verification.

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 (p. 209)
- Step 2: Verifying Your Data in QLDB (p. 210)

Step 1: Requesting a Digest in QLDB

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.

Topics
- AWS Management Console (p. 209)
- QLDB API (p. 210)

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 **Query 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** (p. 269) 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": "42zaJOfV8iGutVGNaIuzQWhDSXb/5B91ScHnvPXm9E=",
  "digestTipAddress": "{strandId:"BlFTjlSXze9B1h1K0szcE3",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 as a fingerprint 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 using the Amazon QLDB API with an AWS SDK. The QLDB API provides the following action 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 Java code examples, see **Installing the Amazon QLDB Java Sample Application** (p. 36), and follow the steps to download and install the sample application. The demo code in class **GetDigest** provides an example of requesting a digest from a sample ledger named **vehicle-registration**.

Before requesting a digest, make sure that you follow at least Steps 1–3 in the **Java Tutorial** (p. 36) to create a ledger named **vehicle-registration** and load it with sample data.

To verify a document revision using the digest that you saved, proceed to **Step 2: Verifying Your Data in QLDB** (p. 210).

---

**Step 2: Verifying Your Data in QLDB**

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** (p. 209).

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**
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 (p. 209). 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 identifier. The blockAddress specifies the block location where the revision was committed.

   In the navigation pane, choose Query 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.

   ```sql
   SELECT metadata.id, blockAddress FROM _ql_committed_table
   WHERE criteria
   ```

   For example, the following query returns a document from the vehicle-registration sample ledger created in Getting Started with the Amazon QLDB Console (p. 18).

   ```sql
   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 (p. 269), string data types are delimited with double quotes. For example, you must copy only the alphanumeric text in the following snippet.

   "LtMN3YjSwzBlqf7sLifrG"

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 (p. 212) to download the example code.

   The console displays the results of your request in the **Verification results** card. For more information, see Verification Results (p. 212).

### QLDB API

You can also verify a document revision using the Amazon QLDB API with an AWS SDK. The QLDB API provides the following actions 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 journal block object at a specified address in a ledger. 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 (p. 331).

For Java code examples, see Installing the Amazon QLDB Java Sample Application (p. 36), and follow the steps to download and install the sample application. The demo 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.

Before trying a verification, make sure that you follow at least Steps 1–3 in the Java Tutorial (p. 36) to create a ledger named **vehicle-registration** and load it with sample data.

### Verification Results

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 (p. 210).

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 are 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 (p. 213).

• **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 are 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 executed to commit entries in this block.

  **Note**
  If you run parameterized statements programmatically, they 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 ?`

• **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 (p. 214) for information about possible causes.

### Using a Proof to Recalculate Your Digest

After QLDB returns a proof for your document verification request, you have the option of trying the hash calculations yourself. This section describes the 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.
   For the QLDB implementation of this comparison logic, see Step 7: Verify a Document in a Ledger (p. 78) in Getting Started with the Driver for the hashComparator method in Verifier.java.
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.
Common Errors for Verification

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 actions 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: GetDigest, GetBlock, GetRevision

Make sure that you provide a valid Amazon Ion (p. 269) value before retrying your request.

**IllegalArgumentException**

Message: The provided block address is not valid.

API: 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: "BlFTjISXze9Bi1h1KOszcE3", sequenceNo: 14}`

**IllegalArgumentException**

Message: The sequence number of the provided digest tip address is beyond the strand's latest committed record.

API: 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: 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: 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: 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: 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: 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 Your Journal in Amazon QLDB

Amazon QLDB uses an immutable transactional log, known as a journal, for data storage. The journal tracks every change to your 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 sections describe how to export journal blocks (p. 7) 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 Amazon Ion format (p. 269) (.ion).

For information about Amazon S3, see the Amazon Simple Storage Service Developer Guide or the Amazon Simple Storage Service Console User Guide.

Topics

- Requesting a Journal Export in QLDB (p. 216)
- Journal Export Output in QLDB (p. 219)
- Journal Export Permissions in QLDB (p. 223)
- Common Errors for Journal Export (p. 225)

Requesting a Journal Export in QLDB

Amazon QLDB provides an API to request an export of your journal blocks for a specified date and time range and a specified Amazon Simple Storage Service (Amazon S3) bucket destination.

Topics

- AWS Management Console (p. 216)
- QLDB API (p. 218)
- Export Job Expiration (p. 218)

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 cannot 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.

\[s3://aws-s3-bucket/prefix/\]

You must specify an S3 bucket name and an optional prefix name for the output objects. The following is an example.

\[s3://awsExampleBucket/journalExport/\]

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 Configuration**—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 **Customer managed CMK** option, you must also specify a symmetric customer master key (CMK) in the following Amazon Resource Name (ARN) format.

\[arn:aws:kms:aws-region:account-id:key/key-id\]

• **Permission**—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 CMK in AWS KMS. To learn how to create this role, see Export Permissions (p. 223).

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/](https://console.aws.amazon.com/s3/).

   To learn more about the format of these output objects, see Journal Export Output in QLDB (p. 219).

**Note**

Export jobs expire seven days after they complete. For more information, see Export Job Expiration (p. 218).
QLDB API

You can also request a journal export using the Amazon QLDB API with an AWS SDK. The QLDB API provides the following actions 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. The data is written as objects in Amazon Ion 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 (p. 331).

For Java code examples, see Installing the Amazon QLDB Java Sample Application (p. 36) and follow the steps to download and install the sample application. The demo code in the following classes provide examples of creating an export, checking the status of an export, and processing the output of an export.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExportJournal</td>
<td>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.</td>
</tr>
<tr>
<td>DescribeJournalExport</td>
<td>Describes a journal export job for a specified exportId in the vehicle-registration sample ledger.</td>
</tr>
<tr>
<td>ListJournalExports</td>
<td>Returns a list of journal export job descriptions for the vehicle-registration sample ledger.</td>
</tr>
<tr>
<td>ValidateQldbHashChain</td>
<td>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.</td>
</tr>
</tbody>
</table>

Before requesting an export, make sure that you follow at least Steps 1–3 in the Java Tutorial (p. 36) to create a ledger named vehicle-registration and load it with sample data.

**Export Job Expiration**

Journal export jobs expire seven days after they complete. This expiration period is a hard limit and cannot be changed. This means that after an export job expires, you can no longer use the QLDB console or the following APIs 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 APIs 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

An Amazon QLDB journal export job writes two manifest files in addition to the Amazon Ion data objects containing your journal blocks. These are all saved in the Amazon Simple Storage Service (Amazon S3) bucket that you provided in your export request (p. 216). The following sections describe the format and contents of each output object.

Topics
- Manifest Files (p. 219)
- Ion Data Objects (p. 220)

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.

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://aws-s3-bucket/prefix/exportId.started.manifest
```

The following is an example of an initial manifest file and its contents.

```
s3://awsExampleBucket/journalExport/8UyXulxccYLAsbN1aon7e4.started.manifest
```

```
{
   ledgerName:"my-ledger",
   exportId:"8UyXulxccYLAsbN1aon7e4",
   inclusiveStartTime:"2019-04-15T00:00:00.000Z",
   exclusiveEndTime:"2019-04-15T22:00:00.000Z",
   bucket:"awsExampleBucket",
   prefix:"journalExport",
   objectEncryptionType:"NO_ENCRYPTION"
}
```

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://aws-s3-bucket/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.

```
s3://awsExampleBucket/journalExport/8UyXulxccYLAsbN1aon7e4.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"
  ]
}
```

Ion Data Objects

Amazon QLDB writes journal data objects in the text representation of the Amazon Ion format (.ion) in the provided Amazon S3 bucket.

Data Object Names

A journal export job writes these data objects with the following naming convention.

```
s3://aws-s3-bucket/prefix/yyyy/mm/dd/hh/strandId.startSn-endSn.ion
```

- 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 being exported.
- `startSn-endSn`—The sequence number range included in the object. Sequence number specifies the location of a block within a strand.

For example, suppose that you specify the following path.

```
s3://awsExampleBucket/journalExport/
```

Your export job creates an Amazon S3 data object that looks similar to the following.

```
s3://awsExampleBucket/journalExport/2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.1-5.ion
```
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 (p. 272) statements that committed them.

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

**Note**
This block example is for informational purposes only. The hashes shown are not real calculated hash values.
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 journal's full hash chain also includes the hashes of these revisions. The full hash chain is required for cryptographic verification.
Journal Export Permissions in QLDB

Before submitting a journal export request in Amazon QLDB, you must provide QLDB with write permissions in your specified Amazon Simple Storage Service (Amazon S3) bucket. If you choose AWS Key Management Service (AWS KMS) as the Amazon S3 object encryption type for your export job, you must also provide QLDB with permissions to use your specified symmetric customer master key (CMK). QLDB does not support asymmetric CMKs. For more information, see Using Symmetric and Asymmetric Keys in the AWS Key Management Service Developer Guide.

To provide your export job with the necessary permissions, you can make QLDB assume an IAM role with the appropriate permissions policies. An IAM role is an entity within your AWS account that has specific permissions. You can use an IAM role in your account to grant an AWS service permissions to access your account's resources.

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 are 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. Otherwise, you can choose a role that you previously created.

Topics
- Create a Permissions Policy (p. 223)
- Create an IAM Role (p. 224)

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 customer master key (CMK) in AWS KMS.

For more information about Amazon S3 bucket policies, see Using Bucket Policies and User Policies in the Amazon Simple Storage Service 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 Amazon S3 bucket and CMK must both be in the same AWS Region as your 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 AWS KMS as the Amazon S3 object encryption type and providing your own CMK, use the following example policy document. To use this policy, replace aws-s3-bucket, aws-region, account-id, and key-id in the example with your own information.
• For other encryption types, use the following example policy document. To use this policy, replace \texttt{aws-s3-bucket} in the example with your own Amazon S3 bucket name.


document

6. Choose \textbf{Review policy}.

\textbf{Note}
You can switch between the Visual editor and \texttt{JSON} tabs at 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.

\section*{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.

\section*{To create the service role for an QLDB (IAM console)}

1. Sign in to the AWS Management Console and open the IAM console at \url{https://console.aws.amazon.com/iam/}.
2. In the navigation pane of the IAM console, choose Roles, and then choose Create role.
3. Choose the AWS service role type, and then choose QLDB.
4. Choose the QLDB use case. Then choose Next: Permissions.
5. Select the box next to the policy that you created in the previous steps.
6. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

   Expand the Set permissions boundary section and 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. Select the policy to use for the permissions boundary or choose Create policy to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure Creating IAM policies in the IAM User Guide. After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.

7. Choose Next: Tags.
8. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the IAM User Guide.
9. Choose Next: Review.
10. If possible, enter a role name or role name suffix to help you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both PRODROLE and prodrole. Because various entities might reference the role, you cannot edit the name of the role after it has been created.
11. (Optional) For Role description, enter a description for the new role.
12. 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.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "qldb.amazonaws.com"
         },
         "Action": [ "sts:AssumeRole" ]
      }
   ]
}
```

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

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.

**IllegalArgumentException**

Message: QLDB encountered an error validating S3 configuration: errorCode errorMessage
A possible cause for this error is that the provided Amazon Simple Storage Service (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 Developer 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 (p. 223).

**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 Developer 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 (KMS) key

You provided a KMSKeyArn with an ObjectEncryptionType of either NO_ENCRYPTION or SSE_S3. You can only provide a customer 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.
Ledger Management in Amazon QLDB

This section describes how to use the QLDB API and the AWS Command Line Interface (AWS CLI) 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 (p. 12).

Topics
- Basic Operations for Amazon QLDB Ledgers (p. 227)
- Tagging Amazon QLDB Resources (p. 232)

Basic Operations for Amazon QLDB Ledgers

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 are the common steps for ledger operations using the AWS SDK for Java and the AWS CLI. For full Java code examples that demonstrate these operations, see Installing the Amazon QLDB Java Sample Application (p. 36) to download the complete QLDB sample application. Or, follow the step-by-step instructions in the Getting Started with the Amazon QLDB Driver (p. 33) tutorial for Java.

Topics
- Creating a Ledger (p. 227)
- Describing a Ledger (p. 229)
- Updating a Ledger (p. 230)
- Deleting a Ledger (p. 230)
- Listing Ledgers (p. 231)

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.
- **Permissions mode** — The permissions mode to assign to the ledger. The only permissions mode that is currently supported for a ledger is ALLOW_ALL.
- **Deletion protection** — (Optional) The flag that prevents a ledger from being deleted by any user. If not provided on 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 using the QLDB API or the AWS CLI. You can disable it by using the UpdateLedger operation to set the flag to false.
Creating a Ledger

- **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 (p. 232).

The ledger is not 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. Execute 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**

```java
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
CreateLedgerRequest request = new CreateLedgerRequest()
    .withName(ledgerName)
    .withPermissionsMode(PermissionsMode.ALLOW_ALL);
CreateLedgerResult result = client.createLedger(request);
```

**Note**
Deletion protection is enabled by default if you don't specify it.

**Example — Disable deletion protection and attach specified tags**

```java
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.ALLOW_ALL)
    .withDeletionProtection(false)
    .withTags(tags);
CreateLedgerResult result = client.createLedger(request);
```

**Creating a Ledger (AWS CLI)**

Create a new ledger named `vehicle-registration`.

**Example**

```bash
aws qldb create-ledger --name vehicle-registration --permissions-mode ALLOW_ALL
```
Note
Deletion protection is enabled by default if you don't specify it.

Or, create a new ledger named vehicle-registration with deletion protection disabled and with specified tags.

Example

```
aws qldb create-ledger --name vehicle-registration --no-deletion-protection --permissions-mode ALLOW_ALL --tags IsTest=true,Domain=Test
```

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

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. Execute 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);
```
Describing a Ledger (AWS CLI)

Describe the vehicle-registration ledger that you just created.

Example

aws qldb describe-ledger --name vehicle-registration

Updating a Ledger

The UpdateLedger operation currently enables you to change the DeletionProtection flag only for an existing ledger. 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 and a new Boolean value for deletion protection.
3. Execute the updateLedger method by providing the request object as a parameter.

The following code example demonstrates the preceding steps. The updateLedger request returns an UpdateLedgerResult object that has updated information about the ledger.

Example

```java
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest() 
    .withName(ledgerName) 
    .withDeletionProtection(false);
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 qldb update-ledger --name vehicle-registration --no-deletion-protection

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 using the QLDB API or the AWS CLI. When you use the QLDB console to delete a ledger, the console disables deletion protection for you.

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. Execute the `deleteLedger` method by providing the request object as a parameter.

The following code example demonstrates the preceding steps.

**Example**

```java
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 qldb 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. This enables you to get the next page of results.

3. Execute 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**

```java
AmazonQldbClient qldbClient = AmazonQldbClientBuilder.standard().build();
List<LedgerSummary> ledgerSummaries = new ArrayList<>();
String nextToken = null;
do {  
    ListLedgersRequest request = new ListLedgersRequest().withNextToken(nextToken);
    ListLedgersResult result = qldbClient.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**

```bash
aws qldb list-ledgers
```

## Tagging Amazon QLDB Resources

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 Simple Storage Service (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 and Cost Management 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 (p. 233)
- Tag Naming and Usage Conventions (p. 233)
Supported Resources in Amazon QLDB

The following resources in Amazon QLDB support tagging:

- ledger

For information about adding and managing tags, see Managing Tags (p. 233).

Amazon QLDB supports the tag-based access control feature of AWS Identity and Access Management (IAM). For information, see Authorization Based on QLDB Tags (p. 240).

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). Amazon EC2 resources allow any characters.
- 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 prohibited for tags; it's reserved for AWS use. You can't edit or delete tag keys or values with this 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. For information about working with tags, see the following:

- ListTagsForResource in the Amazon QLDB API Reference
- TagResource in the Amazon QLDB API Reference
- UntagResource in the Amazon QLDB API Reference
- Step 1: Create a New Ledger (p. 18) in Getting Started with the Amazon QLDB Console

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.
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 (p. 234)
- Identity and Access Management in Amazon QLDB (p. 235)
- Logging and Monitoring in Amazon QLDB (p. 247)
- Compliance Validation for Amazon QLDB (p. 265)
- Resilience in Amazon QLDB (p. 265)
- Infrastructure Security in Amazon QLDB (p. 266)

**Data Protection in Amazon QLDB**

Amazon QLDB conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that 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.
- Set up API and user activity logging with AWS CloudTrail.

We strongly recommend that you never put sensitive identifying information, such as your customers’ account numbers, into free-form 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 QLDB
or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don’t include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

Data Encryption in Amazon QLDB

By default, all Amazon QLDB data in transit and at rest is encrypted. QLDB does not currently support customer managed CMKs (customer master keys). QLDB uses AWS-owned keys to encrypt customer data.

Identity and Access Management in Amazon QLDB

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 (p. 235)
- Authenticating with Identities (p. 235)
- Managing Access Using Policies (p. 237)
- How Amazon QLDB Works with IAM (p. 239)
- Amazon QLDB Identity-Based Policy Examples (p. 241)
- Troubleshooting Amazon QLDB Identity and Access (p. 245)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work 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 (p. 245).

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 employees 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 (p. 239).

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 Amazon QLDB Identity-Based Policy Examples (p. 241).

Authenticating with Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the IAM User Guide.
You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don’t use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see *Signature Version 4 Signing Process* in the AWS General Reference.

Regardless of the authentication method that you use, you might also 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 *Using Multi-Factor Authentication (MFA) in AWS* in the *IAM User Guide*.

**AWS Account Root User**

When you first create an AWS account, you begin with a single 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 do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

**IAM Users and Groups**

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see *Managing Access Keys for IAM Users* in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

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 *When to Create an IAM User (Instead of a Role)* 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. You can temporarily assume an IAM role in the AWS Management Console by *switching roles*. 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 *Using IAM Roles* in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
• **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.

• **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 How IAM Roles Differ from Resource-based Policies in the IAM User Guide.

• **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

• **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 Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the IAM User Guide.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the IAM User Guide.

### Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) 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.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

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, role, or group. These policies control what actions that identity can perform, on which
resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM 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 Choosing 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 such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access Control Lists

Access control lists (ACLs) are a type of policy that controls 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 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 How SCPs Work 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 AWS Identity and Access Management (IAM) to manage access to QLDB, you should understand what IAM features are available to use with QLDB. To get a high-level view of how QLDB and other AWS services work with IAM, see AWS Services That Work with IAM in the IAM User Guide.

Topics
- QLDB Identity-Based Policies (p. 239)
- QLDB Resource-Based Policies (p. 240)
- Authorization Based on QLDB Tags (p. 240)
- QLDB IAM Roles (p. 241)

QLDB Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources in addition to the conditions under which actions are allowed or denied. QLDB supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the IAM User Guide.

Actions

The Action element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in QLDB use a qldb: prefix before the action. For example, to grant someone permission to describe a QLDB ledger with the QLDB DescribeLedger API operation, you include the qldb:DescribeLedger action in their policy. Policy statements must include either an Action element or a NotAction element. QLDB defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
    "qldb:CreateLedger",
    "qldb:UpdateLedger"
]
```

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 see a list of QLDB actions, see Actions Defined by Amazon QLDB in the IAM User Guide.

Note

To interact with the QLDB transactional data API by executing PartiQL statements on a ledger, you must grant permission to the SendCommand action as follows.

```
"Action": "qldb:SendCommand"
```

Resources

The Resource element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.
The QLDB ledger resource has the following ARN.

```
arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the exampleLedger ledger in your statement, use the following ARN.

```
"Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"
```

To specify all ledgers that belong to a specific account, use the wildcard (*).

```
"Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/*"
```

Some QLDB actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
  "exampleLedger1",
  "exampleLedger2"
]
```

To see a list of QLDB resource types and their ARNs, see Resources Defined by Amazon QLDB in the IAM User Guide. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon QLDB.

**Condition Keys**

QLDB does not provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the IAM User Guide.

**Examples**

To view examples of QLDB identity-based policies, see Amazon QLDB Identity-Based Policy Examples (p. 241).

**QLDB Resource-Based Policies**

QLDB does not support resource-based policies.

**Authorization Based on QLDB Tags**

You can attach tags to QLDB resources or pass tags in a request to QLDB. 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. For more information about tagging QLDB resources, see Tagging Amazon QLDB Resources (p. 232).

To view an example of an identity-based policy for limiting access to a resource based on the tags on that resource, see Updating QLDB Ledgers Based on Tags (p. 244).
QLDB IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with QLDB

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS Security Token Service (AWS STS) API operations such as AssumeRole or GetFederationToken.

QLDB supports using temporary credentials.

Service-Linked Roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but cannot edit the permissions for service-linked roles.

QLDB does not support service-linked roles.

Service Roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

QLDB supports service roles for the ExportJournalToS3 API operation, as described in the following section.

Choosing an IAM Role in QLDB

When you export journal blocks in QLDB, you must choose a role to allow QLDB to write objects into an Amazon S3 bucket on your behalf. If you have previously created a service role, then QLDB provides you with a list of roles to choose from. Make sure that you choose a role that allows access to write into your specified bucket. If you choose AWS Key Management Service (AWS KMS) as the server-side encryption type for your export job, you must choose a role that also allows access to your specified customer master key (CMK) in AWS KMS. For more information, see Journal Export Permissions in QLDB (p. 223).

Amazon QLDB Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify QLDB resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the IAM User Guide.

Topics

- Policy Best Practices (p. 242)
- Using the QLDB Console (p. 242)
- AWS Managed (Predefined) Policies for Amazon QLDB (p. 242)
- Executing Data Transactions (p. 243)
- Allow Users to View Their Own Permissions (p. 243)
- Exporting a Journal to an Amazon S3 Bucket (p. 244)
Policy Best Practices

Identity-based policies are very powerful. They 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 Using AWS Managed Policies** – To start using QLDB quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get Started Using Permissions With AWS Managed Policies](#) in the [IAM User Guide](#).
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant Least Privilege](#) in the [IAM User Guide](#).
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using Multi-Factor Authentication (MFA) in AWS](#).
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON Policy Elements: Condition](#) in the [IAM User Guide](#).

Using the QLDB Console

To access the Amazon QLDB console and use all of its features, you must have full access with read and write permissions to the QLDB resources in your AWS account. In addition to these QLDB permissions, the console requires permissions from the *Database Query Metadata Service* (dbqms). This service provides your recent and saved queries for the *Query editor* on the QLDB console. For more information, see [DBQMS API Reference](#).

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 (IAM users or roles) with that policy. To ensure that those entities can still use the QLDB console, also attach the following AWS managed policy to the entities, as described in [AWS Managed (Predefined) Policies for Amazon QLDB](#).

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 you’re trying to perform. You can also use one of the AWS managed policies described in the following section.

**AWS Managed (Predefined) Policies for Amazon QLDB**

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see [AWS Managed Policies](#) and [Adding Permissions to a User](#) in the [IAM User Guide](#).

The following AWS managed policies, which you can attach to users in your account, are specific to QLDB and are grouped by use case scenario:

- **AmazonQLDBReadOnly** — Grants read-only access to all QLDB resources.
• **AmazonQldbFullAccess** — Grants full access to all QLDB resources by using the QLDB API or the AWS CLI.

• **AmazonQldbConsoleFullAccess** — Grants full access to all QLDB resources by using the AWS Management Console.

**Note**
You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also create your own custom IAM policies to allow permissions for QLDB actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

### Executing Data Transactions

To interact with the QLDB transactional data API by executing PartiQL (p. 272) statements on a ledger, you must grant permission to the `SendCommand` action. The following JSON document is an example of a policy that grants permission to only the `SendCommand` action on the ledger `exampleLedger`. You can also use one of the AWS managed policies described in the previous section to grant full access to all QLDB resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBDataTransactionPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"
    }
  ]
}
```

### 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:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": [
        "arn:aws:iam::*:user/${aws:username}"
      ]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [  
    
```

243
Exporting a Journal to an Amazon S3 Bucket

In this example, you want to grant an IAM user in your AWS account access to write into one of your Amazon S3 buckets, `awsExampleBucket`. This is required for a QLDB journal export job.

In addition to granting the `s3:PutObject` permission to the user, the policy also grants `s3:PutObjectAcl` for the ability to set the access control list (ACL) permissions for an object. For an example walkthrough that grants permissions to users and tests them using the console, see An Example Walkthrough: Using User Policies to Control Access to Your Bucket.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBJournalExportS3Permission",
      "Effect": "Allow",
      "Action": ["s3:PutObject", "s3:PutObjectAcl"],
      "Resource": "arn:aws:s3:::awsExampleBucket/*"
    }
  ]
}
```

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 an IAM role with specific permissions attached to it.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "qldb.amazonaws.com"
      },
      "Action": ["sts:AssumeRole"]
    }
  ]
}
```

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 that user’s user name. This policy also grants the permissions necessary to complete this action on the console.

```json
{
   "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 IAM users in your account. If a user named `richard-roe` tries to view 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.

**Troubleshooting Amazon QLDB Identity and Access**

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 (p. 245)
- I Am Not Authorized to Perform `iam:PassRole` (p. 246)
- I Want to View My Access Keys (p. 246)
- I’m an Administrator and Want to Allow Others to Access QLDB (p. 246)
- I Want to Allow People Outside of My AWS Account to Access My QLDB Resources (p. 246)

**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 user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a ledger but does not have `qldb:DescribeLedger` permissions.

```text
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: qldb:DescribeLedger on resource: exampleLedger
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `exampleLedger` 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, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies 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 granted by a service role. Mary does not have permissions to pass the role to the service.

```plaintext
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the IAM User Guide.

I'm an Administrator and Want to Allow Others to Access QLDB

To allow others to access QLDB, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in QLDB.

To get started right away, see Creating Your First IAM Delegated User and Group in the IAM User Guide.

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 (p. 239).
Logging and Monitoring in Amazon QLDB

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 executing data operations.
- Exceptions, so that you can determine whether any requests resulted in an error.

Topics

- Monitoring Tools (p. 247)
- Monitoring with Amazon CloudWatch (p. 248)
- Automating Amazon QLDB with CloudWatch Events (p. 251)
- Logging Amazon QLDB API Calls with AWS CloudTrail (p. 252)

Monitoring Tools

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 (p. 248)
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 (p. 248).

- **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 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

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? (p. 249)
• Amazon QLDB Metrics and Dimensions (p. 249)
• Creating CloudWatch Alarms to Monitor Amazon QLDB (p. 251)

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.
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

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JournalStorage</td>
<td>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.</td>
</tr>
<tr>
<td>Units: Bytes</td>
<td></td>
</tr>
<tr>
<td>Dimensions: LedgerName</td>
<td></td>
</tr>
<tr>
<td>IndexedStorage</td>
<td>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.</td>
</tr>
<tr>
<td>Units: Bytes</td>
<td></td>
</tr>
<tr>
<td>Dimensions: LedgerName</td>
<td></td>
</tr>
<tr>
<td>ReadIOs</td>
<td>The number of disk read I/O operations, reported in one-minute intervals.</td>
</tr>
<tr>
<td>Units: Count</td>
<td></td>
</tr>
<tr>
<td>Dimensions: LedgerName</td>
<td></td>
</tr>
<tr>
<td>WriteIOs</td>
<td>The number of disk write I/O operations, reported in one-minute intervals.</td>
</tr>
<tr>
<td>Units: Count</td>
<td></td>
</tr>
<tr>
<td>Dimensions: LedgerName</td>
<td></td>
</tr>
<tr>
<td>CommandLatency</td>
<td>The amount of time taken for data operations, reported in one-minute intervals.</td>
</tr>
<tr>
<td>Units: Milliseconds</td>
<td></td>
</tr>
<tr>
<td>Dimensions: LedgerName, CommandType</td>
<td></td>
</tr>
<tr>
<td>OccConflictExceptions</td>
<td>The number of requests to QLDB that generate an OccConflictException. For information about optimistic concurrency control (OCC), see Amazon QLDB Concurrency Model (p. 200).</td>
</tr>
<tr>
<td>Units: Count</td>
<td></td>
</tr>
<tr>
<td>Session4xxExceptions</td>
<td>The number of requests to QLDB that generate an HTTP 4xx error.</td>
</tr>
<tr>
<td>Units: Count</td>
<td></td>
</tr>
<tr>
<td>Session5xxExceptions</td>
<td>The number of requests to QLDB that generate an HTTP 5xx error.</td>
</tr>
<tr>
<td>Units: Count</td>
<td></td>
</tr>
</tbody>
</table>
### Automation with CloudWatch Events

#### Dimensions for QLDB Metrics

The metrics for QLDB are qualified by the values for the account, ledger name, or command type. You can use the CloudWatch console to retrieve QLDB data along any of the dimensions in the following table.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LedgerName</td>
<td>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.</td>
</tr>
<tr>
<td>CommandType</td>
<td>This dimension limits the data to one of the following QLDB data API commands:</td>
</tr>
<tr>
<td></td>
<td>• AbortTransaction</td>
</tr>
<tr>
<td></td>
<td>• CommitTransaction</td>
</tr>
<tr>
<td></td>
<td>• EndSession</td>
</tr>
<tr>
<td></td>
<td>• ExecuteStatement</td>
</tr>
<tr>
<td></td>
<td>• FetchPage</td>
</tr>
<tr>
<td></td>
<td>• StartSession</td>
</tr>
<tr>
<td></td>
<td>• StartTransaction</td>
</tr>
</tbody>
</table>

#### 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

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:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionRateExceededExceptions</td>
<td>The number of requests to QLDB that generate a SessionRateExceededException.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
</tbody>
</table>

---

251
• 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 in your AWS account changes. The following is an example of an event that QLDB reported, in which a ledger's state changed to **DELETING**.

```json
{
    "version" : "0",
    "id" : "2f6557eb-e361-54ef-0f9f-99dd9f171c62",
    "detail-type" : "QLDB Ledger State Change",
    "source" : "aws.qldb",
    "account" : "123456789012",
    "region" : "us-east-2",
    "resources" : ["arn:aws:qldb:us-east-2:123456789012:ledger/exampleLedger"],
    "detail" : {
        "ledgerName" : "exampleLedger",
        "state" : "DELETING"
    }
}
```

Some examples of using CloudWatch Events with QLDB include 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**.

For more information, see the Amazon CloudWatch Events User Guide.

**Logging Amazon QLDB API Calls with AWS CloudTrail**

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 control plane 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, see the AWS CloudTrail User Guide.

**QLDB Information in CloudTrail**

CloudTrail is enabled on your AWS account when you create the account. When 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 control plane and non-transactional data plane actions are logged by CloudTrail and are documented in the Amazon QLDB API Reference (p. 331). 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 AWS Identity and Access Management (IAM) 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"
},
"eventSource": "qldb.amazonaws.com",
"eventName": "CreateLedger",
"awsRegion": "us-east-2",
"errorCode": null,
"requestParameters": {
"Name": "CloudtrailTest",
"PermissionsMode": "ALLOW_ALL"
},
"responseElements": {
"CreationDateTime": 1.561497687403E9,
"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\":\"AKIAIOSFODN7EXAMPLE:test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin-test-user\",\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHEXAMPLE\",\"sessionContext":{\"attributes\":{\"mfaAuthenticated\":false,\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODN7EXAMPLE\",\"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"
},
"eventSource": "qldb.amazonaws.com",
"requestParameters": {
"Name": "CloudtrailTest",
"PermissionsMode": "ALLOW_ALL"
},
"responseElements": {
"CreationDateTime": 1.561497687403E9,
"State": "CREATING",
"Name": "CloudtrailTest"
},
"requestID": "3135aec7-978f-11e9-b313-1dd92a14919e",
"eventId": "bf703ff9-676f-41dd-be6f-5f666c9f7852",
"readOnly": false,
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
"eventName": "DescribeLedger",
"awsRegion": "us-east-2",
"errorCode": null,
"requestParameters": {
  "name": "CloudtrailTest"
},
"responseElements": null,
"requestID": "3af51ba0-978f-11e9-8ae6-837dd17a19f8",
"eventID": "be12861-3e38-4503-83de-49fdec7fca0fb",
"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": "AKIAI44QH8DHEXAMPLE",
    "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"
      }
    },
    "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": "be12861-3e38-4503-83de-49fdec7fca0fb",
    "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"
  },
  "cloudtrailEvent": {
    "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": "3af51ba0-978f-11e9-8ae6-837dd17a19f8",
    "eventID": "be12861-3e38-4503-83de-49fdec7fca0fb",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  }
}
"accessKeyId":"AKIAI44QH8DBEXAMPLE","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:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest","Tags":{"TagKey":"TagValue"}},"responseElements":null,"requestID":"3b1d6638-9f8f-11e9-916b-b7d64ec76521","eventID":"6101c9a4-7683-4431-812b-9a91a8b8c849","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": "3b1d6638-9f8f-11e9-916b-b7d64ec76521" },
{ "cloudtrailEvent": {
 "userIdentity": {
 "arn": "arn:aws:st.s::123456789012:assumed-role/Admin/test-user"
 },
 "eventSource": "qldb.amazonaws.com",
 "eventName": "ListTagsForResource",
 "awsRegion": "us-east-2",
 "errorCode": null,
 "requestParameters": {
 "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest"
 },
 "responseElements": null,
 "requestID": "3b56c321-978f-11e9-916b-b7d64ec76521",
 "eventID": "375e57d7-cf94-495a-9a48-ac2192181c02",
 "readOnly": true,
 "eventType": "AwsApiCall",
 "recipientAccountId": "123456789012" }
],
"rawCloudtrailEvent": "{"eventVersion":"1.05","userIdIdentity":{"type ":"AssumedRole","principalId":"AKIAIOSFODNN7EXAMPLE:st.s-user"},"arn ":"arn:aws:sts::123456789012:assumed-role/Admin/test-user","accountId": "123456789012","AccessKeyId": "AKIAI44QH8DBEXAMPLE","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"}},
"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:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest"},
"responseElements":null,"requestID": "3b56c321-978f-11e9-8527-25175bfa8fd","eventID": "375e57d7-cf94-495a-9a48-ac2192181c02","readOnly": true,"eventType": "AwsApiCall",
"recipientAccountId": "123456789012" },
"name": "ListTagsForResource",
"request": ["com.amazonaws.services.qldb.model.ListTagsForResourceRequest",
{ "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest" },
"requestId": "3b56c321-978f-11e9-916b-b7d64ec76521" ]}
"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"
       },
       "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": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9",
       "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":"AKIAI44QH8DH8EXAMPLE","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%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest"},"responseElements":null,"requestId": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9","eventId": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9","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"
       }
   },
   "identity": {
       "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
   },
   "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user",
   "requestId": "123456789012"}
Amazon Quantum Ledger Database
(Amazon QLDB) Developer Guide
Logging Amazon QLDB API Calls with AWS CloudTrail

"arn:aws:iam::123456789012:role/Admin","accountID":"123456789012","userName":"Admin")),"eventTime":"2019-06-25T21:21:49Z","eventSource":"qlbd.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":"5cddd8a1-978f-11e9-a6de-dbe464b9dec"},"eventID":"a5cb60db-e6c5-4f5e-a5fc-0712249622b3","readOnly":true,"eventType":"AwsApiCall","recipientAccountId":"123456789012"},
"name": "GetDigest",
"request": [
  "com.amazonaws.services.qlbd.model.GetDigestRequest",
  {
    "customRequestHeaders": null,
    "customQueryParameters": null,
    "name": "CloudtrailTest",
    "digestTipAddress": null
  }
],
"requestId": "5cddd8a1-978f-11e9-a6de-dbe464b9dec"
},
{
  "cloudtrailEvent": {
    "userIdentity": {
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },
    "eventSource": "qlbd.amazonaws.com",
    "eventName": "GetBlock",
    "awsRegion": "us-east-2",
    "errorCode": null,
    "requestParameters": {
      "BlockAddress": {
        "IonText": "{strandId:"2P2nsG3K2RwHQccUbnAMAj",sequenceNo:0}"
      },
      "name": "CloudtrailTest",
      "DigestTipAddress": null
    }
  },
  "requestId": "5cddd8a1-978f-11e9-a6de-dbe464b9dec",
  "responseElements": null,
  "requestID": "5cddd8a1-978f-11e9-a6de-dbe464b9dec",
  "eventId": "a5cb60db-e6c5-4f5e-a5fc-0712249622b3",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
"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"
  },
  "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-df6dafa37ab",
"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\":\"AKIAI44QH8DHBEEXAMPLE\",\"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\",\"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\"},{\"BlockAddress\":{\"IonText\":\"{strandId:"2P2nsG3K2RwHQccUbnAMAj",sequenceNo:0}\"}}},\"responseElements":null,\"requestParameters":null,\"cloudtrailTest\":{\"name\":\"CloudtrailTest\",\"blockAddress\":{\"IonText\":\"{strandId:"2P2nsG3K2RwHQccUbnAMAj",sequenceNo:0}\"}}}},
"name": "CloudtrailTest",
"blockAddress": {
"ionText": "\"{strandId:"2P2nsG3K2RWHQccUbnAMA\"},sequenceNo:1\"\"
},
"documentId": "BUyXvDw6ApoFVOA2HPfUE",
"digestTipAddress": {
"ionText": "\"{strandId:"2P2nsG3K2RWHQccUbnAMA\"},sequenceNo:1\"\"
}
],
"requestId": "41e19139-978f-11e9-aaed-dfe1dafe37ab"
},

"cloudtrailEvent": {
"userIdentity": {
"arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
},
"eventSource": "qldb.amazonaws.com",
"eventName": "ExportJournalToS3",
"awsRegion": "us-east-2",
"errorCode": null,
"requestParameters": {
"InclusiveStartTime": 1.561497687254E9,
"name": "CloudtrailTest",
"S3ExportConfiguration": {
"Bucket": "cloudtrailtests-123456789012-us-east-2",
"Prefix": "CloudTrailTestsJournalExport",
"EncryptionConfiguration": {
"ObjectEncryptionType": "SSE_S3"
}
}
},
"ExclusiveEndTime": 1.561497715795E9
},
"responseElements": {
"ExportId": "BabQhsmJRYDCGMnA2xYBDG"
},
"requestID": "423815f8-978f-11e9-afcf-55f7d0f3583d",
"eventID": "1b5abdc4-52fa-435f-857e-8995efa19b7",
"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\":\"AKIAI44QH8DHEEXAMPLE\",\"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\":\"cloudtrailtests-123456789012-us-east-2\",\"Prefix\":\"\"},\"responseElements\":{\"ExportId\":\"BabQhsmJRYDCGMnA2xYBDG\"},\"requestID\":\"423815f8-978f-11e9-afcf-55f7d0f3583d\",\"eventID\":\"1b5abdc4-52fa-435f-857e-8995efa19b7\"},\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"},
"name": "ExportJournalToS3",
"request": [
"com.amazonaws.services.qldb.model.ExportJournalToS3Request",
{"value":"CloudtrailTest"}]
}
"customRequestHeaders": null,
"customQueryParameters": null,
"name": "CloudtrailTest",
"inclusiveStartTime": 1561497687254,
"exclusiveEndTime": 1561497715795,
"s3ExportConfiguration": {
  "bucket": "cloudtrailtests-123456789012-us-east-2",
  "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",
      "eventSource": "qldb.amazonaws.com",
      "eventName": "DescribeJournalS3Export",
      "awsRegion": "us-east-2",
      "errorCode": null,
      "requestParameters": {
        "name": "CloudtrailTest",
        "exportId": "BabQhsAJRYDCGMnA2xYBG"
      },
      "responseElements": null,
      "requestId": "427ebbbbc-978f-11e9-8888-e9894c94bb9",
      "eventID": "ca8ffcc88-16ff-4f5f-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","sourceUserAgent":"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":"BabQhsAJRYDCGMnA2xYBG"},"responseElements":null,"requestId":"427ebbbbc-978f-11e9-8888-e9894c94bb9","eventID":"ca8ffcc88-16ff-4f5f-9042-d94fadb389c3","readOnly":true,"eventType":"AwsApiCall","recipientAccountID":"123456789012"}",
  "name": "DescribeJournalS3Export",
  "request": [
    "com.amazonaws.services.qldb.model.DescribeJournalS3ExportRequest",
    {"customRequestHeaders": null,
     "customQueryParameters": null,
     "name": "CloudtrailTest",
     "exportId": "BabQhsAJRYDCGMnA2xYBG"}
  ],
  "requestId": "427ebbbbc-978f-11e9-8888-e9894c94bb9"
}
{  
  "cloudtrailEvent": {  
    "userIdentity": {  
      "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
    },  
    "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\":\"AKIAIOSFODN7EXAMPLE\":test-user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":\"123456789012\",\"accessKeyID\":\"AKIAI44QH8DH8EXAMPLE\",\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":false,\"creationDate\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",\"userId\":\"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\"},\n"name": "ListJournalS3ExportsForLedger",\n"request": [\n  {\n    "com.amazonaws.services.qldb.model.ListJournalS3ExportsForLedgerRequest": {\n      "customRequestHeaders": null,\n      "customQueryParameters": null,\n      "name": "CloudtrailTest",\n      "maxResults": null,\n      "nextToken": null
    }
  },\n  {\n    "cloudtrailEvent": {\n      "userIdentity": {\n        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
      },\n      "eventTime": "2019-06-25T21:21:56Z",\n      "eventSource": "qldb.amazonaws.com",\n      "eventName": "ListJournalS3ExportsForLedger",\n      "awsRegion": "us-east-2",\n      "errorCode": null,\n      "requestParameters": null,\n      "responseElements": null,\n      "requestID": "42cc1814-978f-11e9-befb-f5dbaa142118",\n      "eventID": "4e24d7df-810c-4cf4-884e-00482278b6ce",\n      "readOnly": true,\n      "eventType": "AwsApiCall",\n      "recipientAccountId": "123456789012"
    }
  }
}

263
Logging Amazon QLDB API Calls with AWS CloudTrail

/rawCloudtrailEvent": "{"eventVersion":"1.05","userIdentity":{"type":"AssumedRole","principalId":"AKIAIOSFODNN7EXAMPLE:assumed-role/Admin-test-user","arn": "arn:aws:sts::123456789012:assumed-role/Admin-test-user","accountId": "123456789012","accessKeyId": "AKIAI44QH8DH8EXAMPLE","sessionContext": {"attributes": {"mfaAuthenticated": "false","creationDate": "2019-06-25T21:21:57Z"},"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","userId": "CloudTrailTest"},"request": ["com.amazonaws.services.qldb.model.DeleteLedgerRequest", {"name": "DeleteLedger"}],"requestId": "42f439b9-978f-11e9-8b2c-69ef598d6e9","responseElements": null,"requestID": "42f439b9-978f-11e9-8b2c-69ef598d6e9","eventID": "42f439b9-978f-11e9-8b2c-69ef598d6e9","readOnly": false,"eventType": "AwsApiCall","recipientAccountId": "123456789012"},
"rawCloudtrailEvent": "{"eventVersion":"1.05","userIdentity":{"type":"AssumedRole","principalId":"AKIAIOSFODNN7EXAMPLE:assumed-role/Admin-test-user","arn": "arn:aws:sts::123456789012:assumed-role/Admin-test-user","accountId": "123456789012","accessKeyId": "AKIAI44QH8DH8EXAMPLE","sessionContext": {"attributes": {"mfaAuthenticated": "false","creationDate": "2019-06-25T21:21:57Z"},"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": "ListJournalS3Exports","awsRegion": "us-east-2","sourceIPAddress": "192.0.2.01","userId": "CloudTrailTest"},"request": ["com.amazonaws.services.qldb.model.ListJournalS3ExportsRequest", {"maxResults": null,"nextToken": null}],"requestId": "42cc1814-978f-11e9-befb-f5dbaa142118","eventID": "42cc1814-978f-11e9-befb-f5dbaa142118","readOnly": true,"eventType": "AwsApiCall","recipientAccountId": "123456789012"}
Compliance Validation for Amazon QLDB

Amazon QLDB is currently not certified for any AWS compliance programs.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. 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 QLDB 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 and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **AWS Config** – This AWS 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 that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon QLDB

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 Simple Storage Service (Amazon S3) bucket. You can use this data for various purposes such as data retention, analytics, and auditing. For more information, see Exporting Your Journal in Amazon QLDB (p. 216).

**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 does not provide a dedicated backup and related restore feature at this time.

**Journal streams**

QLDB also provides a continuous journal stream capability with QLDB Streams (currently in preview release). You can integrate QLDB Streams with the Amazon Kinesis streaming platform to process real-time journal data.

**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 does not provide an automated recovery feature for logical corruption scenarios at this time.

**Infrastructure Security in Amazon QLDB**

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 an access key ID and a secret access key that is 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 Using Amazon QLDB with Interface VPC Endpoints (p. 267).
Using Amazon QLDB with Interface VPC Endpoints

You can use an interface VPC endpoint to keep traffic between your Amazon Virtual Private Cloud (Amazon VPC) and Amazon QLDB from leaving the AWS network. Interface VPC endpoints don’t require an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Interface VPC endpoints are powered by AWS PrivateLink. AWS PrivateLink enables private communication between AWS services using an elastic network interface with private IPs in your Amazon VPC. For more information, see What Is Amazon VPC? and Interface VPC Endpoints (AWS PrivateLink) in the Amazon VPC User Guide.

Topics
- Using Interface VPC Endpoints for QLDB (p. 267)
- Controlling Access to VPC Endpoints for QLDB (p. 267)

Using Interface VPC Endpoints for QLDB

To get started, you don’t need to change the settings for your QLDB ledgers. Simply create an interface VPC endpoint to enable your QLDB traffic from and to your Amazon VPC resources to start flowing through the interface VPC endpoint. For more information, see Creating an Interface Endpoint.

For example, consider a QLDB ledger application that runs on an Amazon Elastic Compute Cloud (Amazon EC2) instance in a VPC. If you enable a QLDB interface VPC endpoint, calls between the ledger application and QLDB flow through the private interface VPC endpoint instead of public endpoints.

Controlling Access to VPC Endpoints for QLDB

You can use VPC endpoint policies to control access by attaching a policy to a VPC endpoint. Or, you can use additional fields in a policy that is attached to an IAM user, group, or role to allow access only from a specified VPC endpoint. When used together, VPC endpoint policies and IAM policies can restrict access to specific QLDB actions on specified ledgers to a specified VPC endpoint.

The following are example endpoint policies for accessing QLDB ledgers.

- **VPC policy example: Read-only access** — You can attach the following sample policy to a VPC endpoint. (For more information, see Controlling Access to Amazon VPC Resources in the Amazon VPC User Guide). This policy restricts actions to only listing and describing QLDB ledgers through the VPC endpoint that it’s attached to.

```json
{
   "Statement": [
      {
         "Sid": "ReadOnly",
         "Principal": "*",
         "Action": [
            "qldb:List*",
            "qldb:Describe*"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

- **VPC policy example: Restrict access to a specific QLDB ledger** — You can attach the following sample policy to a VPC endpoint. This policy restricts access to a specific ledger resource through the VPC endpoint that it’s attached to.
• IAM policy example: Restrict access to a specific QLDB ledger from a specific VPC endpoint only —
You can attach the following sample policy to an IAM user, role, or group. The policy allows access to a
specified QLDB ledger only from a specified VPC endpoint.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessFromSpecificEndpoint",
      "Action": "qldb:*",
      "Effect": "Deny",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger",
      "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-1a2b3c4d" } }
    }
  ]
}
```
Amazon QLDB Data Format Reference

Amazon QLDB uses a data notation model that unifies Amazon Ion with a subset of PartiQL (p. 272) 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 information about how to query Amazon Ion data with PartiQL in QLDB, see Querying Ion with PartiQL (p. 289) in the QLDB PartiQL Reference.

**Topics**
- Ion Specification (p. 269)
- What Is Amazon Ion? (p. 269)
- JSON Compatible (p. 270)
- Extensions From JSON (p. 270)
- Ion Text Example (p. 271)
- API References (p. 271)

**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 serialize and deserialize Ion data for you. For code examples of common use cases for processing Ion data, see the Amazon Ion Cookbook on GitHub.

**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 is based on an abstract data model that lets you store both structured and unstructured data. It is 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 are not 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 enables applications to optimize how it processes data for different use cases.

**Note**

The Ion data model is strictly value-based and does not support references. Thus, the data model can represent data hierarchies that can be nested to arbitrary depth, but not directed graphs.
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-java
- ion-js
- ion-python
Amazon QLDB PartiQL Reference

Amazon QLDB supports a subset of the PartiQL query language. The following sections describe the QLDB implementation of PartiQL.

Note
QLDB does not support all PartiQL operations.

Topics
• What is PartiQL? (p. 272)
• PartiQL in Amazon QLDB (p. 272)
• Data Types (p. 272)
• QLDB Documents (p. 273)
• Querying Ion with PartiQL (p. 289)
• PartiQL Statements (p. 292)
• PartiQL Functions (p. 305)
• PartiQL Operators (p. 324)
• Reserved Words (p. 326)

What is PartiQL?

PartiQL provides SQL-compatible query access across multiple data stores containing structured data, semistructured data, and nested data. It is 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 QLDB Data Format Reference (p. 269).

PartiQL in Amazon QLDB

To run PartiQL queries in QLDB, you can do one of the following:

• Use the Query editor on the AWS Management Console for QLDB.
• Use the command line QLDB Shell.
• Run them programmatically using a provided QLDB driver.

For information about using these methods to access QLDB, see Accessing Amazon QLDB (p. 11).

Data Types

Amazon QLDB stores documents in Amazon Ion (p. 269) 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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>A generic null value</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean values</td>
</tr>
<tr>
<td>int</td>
<td>Signed integers of arbitrary size</td>
</tr>
<tr>
<td>decimal</td>
<td>Decimal-encoded real numbers of arbitrary precision</td>
</tr>
<tr>
<td>float</td>
<td>Binary-encoded floating point numbers (64-bit IEEE)</td>
</tr>
<tr>
<td>timestamp</td>
<td>Date/time/timezone moments of arbitrary precision</td>
</tr>
<tr>
<td>string</td>
<td>Unicode text literals</td>
</tr>
<tr>
<td>symbol</td>
<td>Unicode symbolic atoms (identifiers)</td>
</tr>
<tr>
<td>blob</td>
<td>Binary data of user-defined encoding</td>
</tr>
<tr>
<td>clob</td>
<td>Text data of user-defined encoding</td>
</tr>
<tr>
<td>struct</td>
<td>Unordered collections of name-value pairs</td>
</tr>
<tr>
<td>list</td>
<td>Ordered heterogeneous collections of values</td>
</tr>
</tbody>
</table>

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.

---

## QLDB Documents

QLDB stores data records as documents, which are just Amazon Ion (p. 269) `struct` objects that are inserted into a table. For the Ion specification, see the Amazon Ion GitHub site.

### Topics

- Ion Document Structure (p. 273)
- PartiQL-Ion Type Mapping (p. 289)

### Ion Document Structure

Like JSON, QLDB documents are composed of name-value pairs in the following structure.

```json
{
  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 (p. 272), including container types: nested structures, lists, and lists of structures.

Also like JSON, a struct is denoted with curly braces ({...}), and a list is denoted with square brackets ([...]). The following example is a document from the sample data in

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. QLDB enables you to track and verify the full history of changes in an easily accessible and flexible database.

As you work through this tutorial, the following sections explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to cryptographically verify a registration document, 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 (p. 18)
- Step 1: Create a New Ledger (p. 18)
- Step 2: Create Tables, Indexes, and Sample Data (p. 19)
- Step 3: Query the Tables in a Ledger (p. 25)
- Step 4: Modify Documents in a Ledger (p. 25)
- Step 5: View the Revision History for a Document (p. 27)
- Step 6: Verify a Document in a Ledger (p. 29)
- Step 7 (Optional): Clean Up Resources (p. 31)
- Getting Started with Amazon QLDB: Next Steps (p. 32)

Tutorial Prerequisites and Considerations

Before you start this tutorial, you must follow the AWS setup instructions in Accessing Amazon QLDB (p. 11). These steps include signing up for AWS and creating an AWS Identity and Access Management (IAM) user with QLDB access.

Step 1: Create a New Ledger

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.
   - 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 (p. 232). 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 (p. 19).

---

### Step 2: Create Tables, Indexes, and Sample Data

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>
<thead>
<tr>
<th>Table</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleRegistration</td>
<td>VIN</td>
</tr>
<tr>
<td>VehicleRegistration</td>
<td>LicensePlateNumber</td>
</tr>
</tbody>
</table>
You can use the QLDB console to automatically create these tables with indexes and load them with sample data. Or, you can use the Query editor on the console to manually run each PartiQL (p. 272) 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.

**Note**

As a best practice, use this document id metadata field as a foreign key. For more information, see Querying Document Metadata (p. 193).
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 Query editor.
3. Choose the vehicle-registration ledger.
4. Start by creating four tables. QLDB supports open content and does not enforce schema, so you don't specify attributes or data types.
   In the query editor window, enter the following statement, and then choose Run.

   ```sql
   CREATE TABLE VehicleRegistration
   ```
   Repeat this step for each of the following.

   ```sql
   CREATE TABLE Vehicle
   CREATE TABLE Person
   CREATE TABLE DriversLicense
   ```

5. Next, create indexes that help speed up queries against each table.
   In the query editor window, enter the following statement, and then choose Run.

   ```sql
   CREATE INDEX ON VehicleRegistration (VIN)
   ```
   Repeat this step for the following.

   ```sql
   CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
   CREATE INDEX ON Vehicle (VIN)
   CREATE INDEX ON Person (GovId)
   CREATE INDEX ON DriversLicense (LicenseNumber)
   CREATE INDEX ON DriversLicense (PersonId)
   ```

6. 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.

   ```sql
   INSERT INTO Person
   << {
       'FirstName' : 'Raul',
       'LastName' : 'Lewis',
       'DOB' : '1963-08-19T',
       'GovId' : 'LEWISR261LL',
       'GovIdType' : 'Driver License',
       'Address' : '1719 University Street, Seattle, WA, 98109'
   }
   ```
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
<<
{ 
    'LicenseNumber': 'LEWISR261LL',
    'LicenseType': 'Learner',
    'ValidFromDate': '2016-12-20T',
    'ValidToDate': '2020-11-15T',
    'PersonId': ''
},
{ 
    'LicenseNumber': 'LOGANB486CG',
    'LicenseType': 'Probationary',
    'ValidFromDate': '2016-04-06T',
    'ValidToDate': '2020-11-15T',
    'PersonId': ''
},
{ 
    'LicenseNumber': '744 849 301',
    'LicenseType': 'Full',
    'ValidFromDate': '2017-12-06T',
    'ValidToDate': '2022-10-15T',
    'PersonId': ''
},
{ 
    'LicenseNumber': 'P626-168-229-765',
    'LicenseType': 'Learner',
    'ValidFromDate': '2017-08-16T',
    'ValidToDate': '2020-11-15T',
    'PersonId': ''
} >>
```
8. 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.

```sql
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' : '1HVBBAAAXWH544237',
    'LicensePlateNumber' : 'LS477D',
    'State' : 'WA',
    'City' : 'Tacoma',
    'PendingPenaltyTicketAmount' : 42.20,
    'ValidFromDate' : '2011-10-26T',
}>>
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.

```sql
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' : '3HGGK5S3FM761765',
  'Type' : 'Motorcycle',
  'Year' : 2011,
  'Make' : 'Ducati',
  'Model' : 'Monster 1200',
  'Color' : 'Yellow'
},
{
  'VIN' : '1HVBBANXWH5844237',
  '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 (p. 25).

**Step 3: Query the Tables in a Ledger**

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.

**To query the tables**

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose Query 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.
   
   ```sql
   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.

   ```sql
   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.

   ```sql
   SELECT * FROM Person AS p, DriversLicense as l
   WHERE p.GovId = l.LicenseNumber
   ```

   To learn about modifying documents in the tables in the vehicle-registration ledger, see Step 4: Modify Documents in a Ledger (p. 25).
Step 4: Modify Documents in a Ledger

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.

Important
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 and Terminology in Amazon QLDB (p. 7). To learn more about metadata, see Querying Document Metadata (p. 193).

To modify documents

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose Query 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.

   ```sql
   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.

   ```sql
   UPDATE VehicleRegistration AS r
   SET r.Owners.PrimaryOwner.PersonId = '294jJ3YUoH1IEEm8GSobOs' -- replace with your id
   WHERE r.VIN = '1N4AL11D75C109151'
   ```

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

```sql
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.

```sql
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = 'IN7MVYtUjkp1GMZu0F6CG9', --replace with your id
    r.City = 'Everett'
WHERE r.VIN = '1N4AL11D75C109151'
```

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

```sql
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.

```sql
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 (p. 296) DML statement.

```sql
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners
    VALUE { 'PersonId' : '5Ufgdlnj06gF5CWcOIu64s' } --replace with your id
```

Confirm that you modified SecondaryOwners by issuing this statement.

```sql
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 (p. 27).
of a document that you inserted, updated, and deleted by querying the built-in History Function (p. 197).

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 Query 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 (p. 195).
   In the query editor window, enter the following statement, and then choose Run.

   ```sql
   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 in the following syntax.

   ```sql
   SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h
   WHERE h.metadata.id = 'id'
   ```

   • Best practice is to qualify a history query with a document id. This helps to avoid inefficient queries.
   • The start-time and end-time are optional parameters that are both inclusive. They must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC).
   • The start-time and end-time are Amazon Ion literal values that can be denoted with backticks (``...``). To learn more, see Querying Ion with PartiQL (p. 289).

The history function returns documents with the same schema as the committed view. For example, enter the following statement, and then choose Run. Be sure to replace the id value with your own document ID as appropriate.

   ```sql
   SELECT h.data.VIN, h.data.City, h.data.Owners
   FROM history(VehicleRegistration) AS h
   WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
   ```

This first history query projects your modified vehicle registration data. The output includes all revisions of the document and should look similar to the following.

<table>
<thead>
<tr>
<th>VIN</th>
<th>City</th>
<th>Owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>Seattle</td>
<td>{PrimaryOwner: {PersonId:&quot;&quot;},SecondaryOwners: []}</td>
</tr>
</tbody>
</table>
### Step 5: View the Revision History

<table>
<thead>
<tr>
<th>VIN</th>
<th>City</th>
<th>Owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>Seattle</td>
<td>{PrimaryOwner: {PersonId:&quot;294jJ3YUoH1IEEm8GSabOs&quot;}, SecondaryOwners:[]}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIN</th>
<th>City</th>
<th>Owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>Everett</td>
<td>{PrimaryOwner: {PersonId:&quot;7NmE8YLbXc0IqesJy1rpR&quot;}, SecondaryOwners:[]}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIN</th>
<th>City</th>
<th>Owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>Everett</td>
<td>{PrimaryOwner: {PersonId:&quot;7NmE8YLbXc0IqesJy1rpR&quot;}, SecondaryOwners:[]}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIN</th>
<th>City</th>
<th>Owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1N4AL11D75C109151&quot;</td>
<td>Everett</td>
<td>SecondaryOwners: [{PersonId:&quot;5Ufgdlnj06gF5CWc0Iu64s&quot;}]</td>
</tr>
</tbody>
</table>

### 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 (p. 25).

6. 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.

```sql
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.

<table>
<thead>
<tr>
<th>version id</th>
<th>txTime</th>
<th>txId</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;2019-05-23T19:20:360d-3Z&quot;</td>
<td>FMrOvPuPxJ3k4661z475</td>
</tr>
<tr>
<td>1</td>
<td>&quot;2019-05-23T21:40:199d-3Z&quot;</td>
<td>KFByxe842x88DNHcvAPQOt</td>
</tr>
<tr>
<td>2</td>
<td>&quot;2019-05-23T21:44:432d-3Z&quot;</td>
<td>FViDOJRwzHFrMvAyJ2Kd9</td>
</tr>
<tr>
<td>3</td>
<td>&quot;2019-05-23T21:49:254d-3Z&quot;</td>
<td>Z6C1Zd7vCmJ6RAv0vT24YA</td>
</tr>
</tbody>
</table>

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.
- The txTime shows that the initial revision of the document (version 0) was created at 2019-05-23T19:20:360-3Z.
- Each subsequent transaction creates a new revision with the same document id, an incremented version number, and an updated txId and txTime.

To cryptographically verify a document revision in the vehicle-registration ledger, proceed to Step 6: Verify a Document in a Ledger (p. 29).
Step 6: Verify a Document in a Ledger

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 (p. 205).

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 (p. 205) 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 (p. 269) 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.

```json
{
"digest": "42zaJ0Fv8iGutVGNaIuzQWhD5Xb/5B9lScHnvxPXm9E=",
"digestTipAddress": "{strandId:\"BlFTj1sXZe9B1h1KoszcE3\",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 as a fingerprint 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 real scenario for verification, you should use a digest that was previously saved rather than doing the two tasks consecutively. Ideally, the digest is saved soon after the document revision that you want to verify is written to the ledger.

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 identifier. The blockAddress specifies the block location where the revision was committed.
   In the navigation pane of the QLDB console, choose Query editor.
2. Choose the vehicle-registration ledger.
3. In the query editor window, enter the following statement, and then choose Run.

   ```sql
   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 (p. 212).
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 (p. 31).

**Step 7 (Optional): Clean Up Resources**

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 or the AWS Command Line Interface (AWS CLI). The QLDB console disables deletion protection for you when you use it to delete a ledger.

**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. In the list of ledgers, select **vehicle-registration**.
4. Choose **Delete ledger**. Confirm this by entering **vehicle-registration** in the field provided.
5. If deletion protection is enabled for this ledger, you must also choose the option to **Override deletion protection**.

To learn more about working with ledgers in QLDB, see Getting Started with Amazon QLDB: Next Steps (p. 32).

**Getting Started with Amazon QLDB: Next Steps**

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

- Working with Data and History in Amazon QLDB (p. 187)
- Getting Started with the Amazon QLDB Driver (p. 33)
- Amazon QLDB Concurrency Model (p. 200)
- Data Verification in Amazon QLDB (p. 205)
- Exporting Your Journal in Amazon QLDB (p. 216)
- Amazon QLDB PartiQL Reference (p. 272)

(p. 288)

```json
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  PendingPenaltyTicketAmount: 90.25,
  ValidFrom: 2017-08-21T,
  ValidTo: 2020-05-11T,
  Owners: {
    PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWc0Iu64s" }]
  }
}```
Important
In Ion, double quotation marks denote string values, and bare 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 PartiQL's syntax and semantics when using the QLDB console to query Ion data, see Querying Ion with PartiQL (p. 289).

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 an SQL tuple, which traditionally represents a table row.
  - However, with open content and without schema, queries that rely on the ordered nature of an 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.

Querying Ion with PartiQL
When you use the AWS Management Console to query data in Amazon QLDB, you write statements in PartiQL, but results are shown in Amazon Ion. PartiQL is intended to be SQL-compatible, whereas Ion is an extension of JSON. This leads to syntactic differences with how you notate data in your queries, compared to how the QLDB console presents your query results.

This section describes PartiQL semantics for running manual statements with the Query editor on the console.

Note
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 performant. To learn more, see Step 3: Create Tables, Indexes, and Sample Data (p. 46) in the Getting Started with the Driver Java tutorial.

Topics
- Syntax and Semantics (p. 289)
- Backtick Notation (p. 291)
- Path Navigation (p. 292)
- Aliasing (p. 292)
- PartiQL Specification (p. 292)

Syntax and Semantics
When using the QLDB console to query Ion data, the following are PartiQL's fundamental query syntax and semantics:
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 in bare text without any quotes.

In PartiQL, double quotation marks (" . . ") denote a quoted PartiQL identifier. Unquoted text represents a regular PartiQL identifier, such as a table name.

Ion literals

Any Ion literals can be denoted with backticks (` . . `) in a PartiQL statement.

Field names

Ion field names are case-sensitive symbols. PartiQL enables you to 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 is 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 the prior 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 cannot 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.

```ion
INSERT INTO VehicleRegistration VALUE
{
    'VIN' : 'KM8SRDHFP6EU074761', --string
    'RegNum' : 1722, --integer
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75, --decimal
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': '1nmeDdLo3AhGswBtyM1eYh' }
        ]
    },
    '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.

```ion
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.

```ion
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.

```ion
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 are not equal. For example, the following PartiQL expressions evaluate to different Ion values.

```
'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 are 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 Statements

PartiQL extends SQL-92 to support documents in the Amazon Ion data format. Amazon QLDB supports the following PartiQL statements.
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 (p. 293)
- CREATE TABLE (p. 294)
- DROP TABLE (p. 295)
- UNDROP TABLE (p. 305)

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 (p. 295)
- FROM (p. 296)
- INSERT (p. 299)
- SELECT (p. 300)
- UPDATE (p. 303)

CREATE INDEX

Use the CREATE INDEX statement to create a new index on a table field in your Amazon QLDB ledger.

In QLDB, indexes are optional. They can improve query performance. However, note the following:

- Indexes can only be created on empty tables.
- Indexes can only be created on a single field. Composite indexes are currently not supported.
- Indexes can't be dropped after they are created.
- Query performance is improved only when you use an equality predicate; for example, fieldName = 123456789.

QLDB does not currently honor inequalities in query predicates. One implication of this is that range filtered scans are not implemented.

- Names of indexed fields can have a maximum of 128 characters.

Topics

- Syntax (p. 293)
- Parameters (p. 294)
- Examples (p. 294)

Syntax

```
CREATE INDEX ON table (field)
```
CREATE TABLE

Parameters

table

The table where you want to create the index. The table must already exist.

field

The table field name on which to create the index. The field must be a top-level attribute.

Examples

<table>
<thead>
<tr>
<th>Create Index on VehicleRegistration (LicensePlateNumber)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Index on Vehicle (VIN)</td>
</tr>
</tbody>
</table>

CREATE TABLE

Use the CREATE TABLE statement to create a new table in your Amazon QLDB ledger.

Tables have simple names with no namespaces. QLDB supports open content and does not enforce schema, so you don't define attributes or data types when creating tables.

Topics

- Syntax (p. 294)
- Parameters (p. 294)
- Examples (p. 294)

Syntax

CREATE TABLE table

Parameters

table

The name of the table to create. The following are the naming constraints:

- Must only contain 1–128 alphanumeric characters or underscores.
- The first character must be a letter or an underscore.
- The remaining characters can be any combination of alphanumeric characters and underscores.
- The table names are case sensitive.
- The name must not be a QLDB PartiQL reserved word (p. 326).

Examples

CREATE TABLE VehicleRegistration
DELETE

Use the `DELETE` statement to delete an existing document from your Amazon QLDB ledger.

Topics
- Syntax (p. 295)
- Parameters (p. 295)
- Examples (p. 295)

Syntax

```
DELETE FROM table [ AS table_alias ]
[ WHERE condition ]
```

Parameters

`table`

The user table containing the data to be deleted. DML statements are only supported in the default user view (p. 190). Each statement can only be executed 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.

`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.

Examples

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

DROP TABLE

Use the `DROP TABLE` statement to inactivate an existing table in your Amazon QLDB ledger. You can use the `UNDROP TABLE` (p. 305) statement to reactivate it.

Topics
- Syntax (p. 295)
- Parameters (p. 296)
- Examples (p. 296)

Syntax

```
DROP TABLE table
```
Parameters

table

The name of the table to drop. The table must already exist and have a status of ACTIVE.

Examples

```
DROP TABLE VehicleRegistration
```

FROM

A statement that starts with FROM is an Amazon Ion extension that enables you to insert and remove specific elements within a document in Amazon QLDB. You can also use this statement to update existing elements in a document, similar to the UPDATE (p. 303) statement.

Topics

- Syntax (p. 296)
- Parameters (p. 296)
- Nested Collections (p. 297)
- Examples (p. 297)

Syntax

Insert a new element within an existing document. To insert a new top-level document into a table, you must use INSERT (p. 299).

```
FROM table [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
INSERT INTO element VALUE data [ AT key_name ]
```

Remove an existing element within a document, or remove an entire top-level document. The latter is semantically the same as the traditional DELETE (p. 295) syntax.

```
FROM table [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
REMOVE element
```

Update one or more elements within a document. If an element does not exist, it is inserted. This is semantically the same as the traditional UPDATE (p. 303) syntax.

```
FROM table [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
SET element = data [, element = data, ... ]
```

Parameters

**table**

The user table containing the data to be modified. DML statements are only supported in the default user view (p. 190). Each statement can only be executed 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 (p. 297).

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 project or filter on the document ID (p. 193) while querying the default user view. For more information, see Using the BY Clause to Query Document ID (p. 195).

condition

The selection criteria for the documents to be modified.

element

An element within the documents 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.

Note

If you omit the WHERE clause, then all of the documents in the table are modified.

Nested Collections

While you can execute a DML statement on a single table only, you can also include one or more collections that are nested within that table. You can declare an alias for each collection and then use it in both the SET and WHERE clauses.

For example, the FROM sources of the following statement include both the VehicleRegistration table and its nested Owners 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 Owners.SecondaryOwners list that has a PersonId of 'abc123' within the VehicleRegistration document that has a VIN of '1N4AL11D75C109151'.

Examples

Modify an element within a document. If the element does not exist, it is inserted.

```
FROM Vehicle AS v
WHERE 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 = 'document_id'
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.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'
REMOVE p.Address

Remove a whole document from a table.

FROM Person AS p
WHERE p.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'
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]

Add \{'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'} >>

**INSERT**

Use the `INSERT` statement to add one or more Amazon Ion documents to a table in Amazon QLDB.

**Topics**
- Syntax (p. 299)
- Parameters (p. 299)
- Examples (p. 300)

**Syntax**

Insert a single document.

```
INSERT INTO table VALUE doc
```

Insert multiple documents.

```
INSERT INTO table << doc, doc, ... >>
```

**Parameters**

**table**

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 (p. 190).

**doc**

A valid QLDB document (p. 273). You must specify at least one document. Multiple documents must be separated by commas.

The document must be denoted with curly braces ({ ... }).

Each field name in the document is a case-sensitive Ion symbol that can be denoted with single quotation marks (‘...’) in PartiQL.

String values are also denoted with 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.
Examples

Insert a single document.

```sql
INSERT INTO VehicleRegistration VALUE
{
    'VIN': 'KM8SRDH6EU074761', --string
    'RegNum': 1722, --integer
    'State': 'WA',
    'City': 'Kent',
    'PendingPenaltyTicketAmount': 130.75, --decimal
    'Owners': { --nested struct
        'PrimaryOwner': { 'PersonId': '294J3YU6H1IEEm8GSabOs' },
        'SecondaryOwners': [ --list of structs
            { 'PersonId': '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': '1nmeDdLo3AhGswBtyM1eYh' }
        ]
    },
    'ValidFromDate': `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate': `2020-06-25T`
}
```

Insert multiple documents (that is, a bag of documents).

```sql
INSERT INTO Person <<
{
    'FirstName': 'Raul',
    'LastName': 'Lewis',
    'DOB': `1963-08-19T`,
    'GovId': 'LEWSIR261LL',
    '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': `1974-02-26T`,
    'GovId': '744 849 301',
    'GovIdType': 'SSN',
    'Address': '4058 Melrose Street, Spokane Valley, WA, 99206'
}
>>
```

SELECT

Use the SELECT statement to retrieve data from one or more tables in Amazon QLDB.

Topics

- Syntax (p. 301)
- Parameters (p. 301)
- Joins (p. 302)
• Nested Query Limitations (p. 302)
• Examples (p. 302)

Syntax

```
SELECT [ VALUE ] expression [ AS field_alias ] [, ...]
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 (p. 305) or fields that are
modified by PartiQL Operators (p. 324).

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.

source

A source to be queried. The only sources currently supported are table names, inner joins (p. 302)
between tables, nested SELECT queries (subject to Nested Query Limitations (p. 302)), and history
function (p. 196) 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 number of an element within a list structure.
In other words, this specifies the position of an element within a list. 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 (p. 193) while querying the default user view. For more
information, see Using the BY Clause to Query Document ID (p. 195).

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.

```sql
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.

```sql
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.

```sql
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.

```sql
SELECT r.VIN,
     (SELECT o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM VehicleRegistration AS r
```

Examples

The following code shows a basic SELECT all wildcard with no filter criteria.

```sql
SELECT * FROM Vehicle
```

Note
PartiQL is SQL compatible, so it supports table scans such as this example. But we generally don't recommend running SELECT queries without a WHERE predicate clause. QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. The best practice is to run queries that filter on a document id or a table's indexed fields.

The following shows SELECT projections with a string filter.

```sql
SELECT FirstName, LastName, Address
FROM Person
WHERE Address LIKE '%Seattle%'
```
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 within VehicleRegistration, not a different collection named Owners (if one existed).

```sql
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
```

The following shows a subquery in the SELECT list that projects nested data, and an implicit inner join.

```sql
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
```

The following shows an explicit inner join.

```sql
SELECT
    v.Make,
    v.Model,
    r.Owners
FROM
    VehicleRegistration AS r JOIN Vehicle AS v
ON
    r.VIN = v.VIN
```

The following shows a projection of the document id metadata field, using the BY clause.

```sql
SELECT
    r_id,
    r.VIN
FROM
    VehicleRegistration AS r BY r_id
WHERE
    r_id = 'document_id'
```

The following returns the PersonId and index number of each person in the Owners.SecondaryOwners list for a document in table VehicleRegistration.

```sql
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = 'KM8SRDHF6EU074761'
```

**UPDATE**

Use the UPDATE statement to modify the value of one or more elements within a document in Amazon QLDB. If an element does not exist, it is inserted.

**Topics**

- Syntax (p. 304)
- Parameters (p. 304)
Examples (p. 304)

Syntax

```
UPDATE table [ AS table_alias ] [ BY id_alias ]
SET element = data [, element = data, ... ]
[ WHERE condition ]
```

Parameters

table

The user table containing the data to be modified. DML statements are only supported in the default user view (p. 190). Each statement can only be executed 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 FROM clause using the BY keyword. This is useful when you want to project or filter on the document ID (p. 193) while querying the default user view. For more information, see Using the BY Clause to Query Document ID (p. 195).

element

An element within the documents to be created or modified.

data

A new value for the element.

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.

Examples

Update a field in a document. If the field does not exist, it is inserted.

```
UPDATE Person AS p
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE p.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'
```

Filter on the system-assigned document id metadata field.

```
UPDATE Person AS p BY pid
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE pid = 'document_id'
```

Overwrite an entire document.
UPDATE Person AS p
SET p = {
    'FirstName' : 'Rosemarie',
    'LastName' : 'Holloway',
    'DOB' : '1977-06-18T',
    'GovId' : 'LEWISR261LL',
    'GovIdType' : 'Driver License',
    'Address' : '4637 Melrose Street, Ellensburg, WA, 98926'
} 
WHERE p.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'

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'

UNDROP TABLE

Use the UNDROP TABLE statement to reactivate a table that you previously dropped (that is, inactivated) in your Amazon QLDB ledger.

Topics
- Syntax (p. 305)
- Parameters (p. 305)
- Examples (p. 305)

Syntax

UNDROP TABLE "table-id"

Parameters

**table-id**

The unique table ID to undrop, denoted by double quotation marks.

The table must have been previously dropped, meaning that it exists in the system table information_schema.user_tables and has a status of INACTIVE. There must also be no active, existing table with the same name.

Examples

UNDROP TABLE "5PLf9SXwdd631PaSIa006"

PartiQL Functions

PartiQL in Amazon QLDB supports the following built-in variants of SQL standard functions.
Note
Any SQL functions that are not included in this list are not currently supported in QLDB.

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.

<table>
<thead>
<tr>
<th>Function Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR_LENGTH(null)</td>
<td>null</td>
</tr>
<tr>
<td>CHAR_LENGTH(missing)</td>
<td>null (also returns null)</td>
</tr>
</tbody>
</table>

Aggregate Functions

- AVG Function (p. 307)
- COUNT Function (p. 311)
- MAX Function (p. 315)
- MIN Function (p. 316)
- SIZE Function (p. 317)
- SUM Function (p. 319)

Conditional Functions

- COALESCE Function (p. 310)
- EXISTS Function (p. 313)
- NULLIF Function (p. 316)

Date and Time Functions

- DATE_ADD Function (p. 312)
- DATE_DIFF Function (p. 312)
- EXTRACT Function (p. 314)
- UTCNOW Function (p. 323)

Scalar Functions

- TXID Function (p. 322)

String Functions

- CHAR_LENGTH Function (p. 310)
- CHARACTER_LENGTH Function (p. 310)
- LOWER Function (p. 315)
- SUBSTRING Function (p. 318)
- TRIM Function (p. 321)
Data Type Formatting Functions

- UPPER Function (p. 322)
- CAST Function (p. 307)
- TO_STRING Function (p. 319)
- TO_TIMESTAMP Function (p. 320)

AVG Function

Returns the average (arithmetic mean) of the input expression values. The AVG 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.
```

CAST Function

Evaluates a given expression to a value and converts the value to a specified target data type. If the conversion cannot 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 (p. 272).

type

The name of the target data type for conversion. This parameter can be one of the supported Data Types (p. 272).

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).

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(null AS null)</td>
<td>null</td>
</tr>
<tr>
<td>CAST(missing AS null)</td>
<td>null</td>
</tr>
<tr>
<td>CAST(missing AS missing)</td>
<td>null</td>
</tr>
<tr>
<td>CAST(null AS missing)</td>
<td>null</td>
</tr>
<tr>
<td>CAST(null AS boolean)</td>
<td>null (null AS any data type name results in null)</td>
</tr>
<tr>
<td>CAST(missing AS boolean)</td>
<td>null (missing AS any data type name results in null)</td>
</tr>
</tbody>
</table>

Any value that is not an unknown type cannot be cast to NULL or MISSING.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(true AS null)</td>
<td>error</td>
</tr>
<tr>
<td>CAST(true AS missing)</td>
<td>error</td>
</tr>
<tr>
<td>CAST(1 AS null)</td>
<td>error</td>
</tr>
</tbody>
</table>

The following examples show casting AS boolean.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(true AS boolean)</td>
<td>true no-op</td>
</tr>
<tr>
<td>CAST(0 AS boolean)</td>
<td>false</td>
</tr>
<tr>
<td>CAST(1 AS boolean)</td>
<td>true</td>
</tr>
<tr>
<td>CAST(<code>1e0</code> AS boolean)</td>
<td>true (float)</td>
</tr>
<tr>
<td>CAST(<code>1d0</code> AS boolean)</td>
<td>true (decimal)</td>
</tr>
<tr>
<td>CAST('a' AS boolean)</td>
<td>false</td>
</tr>
<tr>
<td>CAST('true' AS boolean)</td>
<td>true (SqlName string 'true')</td>
</tr>
<tr>
<td>CAST(<code>true</code> AS boolean)</td>
<td>true (Ion symbol <code>true</code>)</td>
</tr>
<tr>
<td>CAST(<code>false</code> AS boolean)</td>
<td>false (Ion symbol <code>false</code>)</td>
</tr>
</tbody>
</table>

The following examples show casting AS integer.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(true AS integer)</td>
<td>1</td>
</tr>
<tr>
<td>CAST(false AS integer)</td>
<td>0</td>
</tr>
<tr>
<td>CAST(1 AS integer)</td>
<td>1</td>
</tr>
<tr>
<td>CAST(<code>1d0</code> AS integer)</td>
<td>1</td>
</tr>
<tr>
<td>CAST(<code>1d3</code> AS integer)</td>
<td>1000</td>
</tr>
<tr>
<td>CAST(1.00 AS integer)</td>
<td>1</td>
</tr>
<tr>
<td>CAST(1.45 AS integer)</td>
<td>1</td>
</tr>
<tr>
<td>CAST(1.75 AS integer)</td>
<td>1</td>
</tr>
<tr>
<td>CAST('12' AS integer)</td>
<td>12</td>
</tr>
<tr>
<td>CAST('aa' AS integer)</td>
<td>error</td>
</tr>
<tr>
<td>CAST(<code>22</code> AS integer)</td>
<td>22</td>
</tr>
</tbody>
</table>
The following examples show casting as **float**.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(true AS float)</td>
<td>1e0</td>
</tr>
<tr>
<td>CAST(false AS float)</td>
<td>0e0</td>
</tr>
<tr>
<td>CAST(1 AS float)</td>
<td>1e0</td>
</tr>
<tr>
<td>CAST('1d0` AS float)</td>
<td>1e0</td>
</tr>
<tr>
<td>CAST('1d3` AS float)</td>
<td>1000e0</td>
</tr>
<tr>
<td>CAST(1.00 AS float)</td>
<td>1e0</td>
</tr>
<tr>
<td>CAST('12` AS float)</td>
<td>12e0</td>
</tr>
<tr>
<td>CAST('aa` AS float)</td>
<td>error</td>
</tr>
<tr>
<td>CAST('22` AS float)</td>
<td>22e0</td>
</tr>
<tr>
<td>CAST(<code>'x</code> AS float)</td>
<td>error</td>
</tr>
</tbody>
</table>

The following examples show casting as **decimal**.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(true AS decimal)</td>
<td>1.</td>
</tr>
<tr>
<td>CAST(false AS decimal)</td>
<td>0.</td>
</tr>
<tr>
<td>CAST(1 AS decimal)</td>
<td>1.</td>
</tr>
<tr>
<td>CAST('1d0` AS decimal)</td>
<td>1. (REPL printer serialized to 1.)</td>
</tr>
<tr>
<td>CAST('1d3` AS decimal)</td>
<td>1d3</td>
</tr>
<tr>
<td>CAST(1.00 AS decimal)</td>
<td>1.00</td>
</tr>
<tr>
<td>CAST('12` AS decimal)</td>
<td>12.</td>
</tr>
<tr>
<td>CAST('aa` AS decimal)</td>
<td>error</td>
</tr>
<tr>
<td>CAST('22` AS decimal)</td>
<td>22.</td>
</tr>
<tr>
<td>CAST(<code>'x</code> AS decimal)</td>
<td>error</td>
</tr>
</tbody>
</table>

The following examples show casting as **timestamp**.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(<code>'2001T</code> AS timestamp)</td>
<td>2001T</td>
</tr>
<tr>
<td>CAST(<code>'2001-01-01T00:00:00.000Z</code> AS timestamp)</td>
<td>2001-01-01T00:00:00.000Z</td>
</tr>
<tr>
<td>CAST(true AS timestamp)</td>
<td>error</td>
</tr>
<tr>
<td>CAST(2001 AS timestamp)</td>
<td>error</td>
</tr>
</tbody>
</table>

The following examples show casting as **symbol**.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(<code>'xx</code> AS symbol)</td>
<td>xx (<code>'xx</code> is an Ion symbol)</td>
</tr>
<tr>
<td>CAST('xx` AS symbol)</td>
<td>xx ('xx' is a string)</td>
</tr>
<tr>
<td>CAST(42 AS symbol)</td>
<td>'42'</td>
</tr>
<tr>
<td>CAST('le0` AS symbol)</td>
<td>'1'</td>
</tr>
<tr>
<td>CAST('1d0` AS symbol)</td>
<td>'1.0'</td>
</tr>
<tr>
<td>CAST(true AS symbol)</td>
<td>'true'</td>
</tr>
<tr>
<td>CAST(false AS symbol)</td>
<td>'false'</td>
</tr>
<tr>
<td>CAST(<code>'2001T</code> AS symbol)</td>
<td>'2001T'</td>
</tr>
<tr>
<td>CAST(<code>'2001-01-01T00:00:00.000Z</code> AS symbol)</td>
<td>'2001-01-01T00:00:00.000Z'</td>
</tr>
</tbody>
</table>

The following examples show casting as **string**.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(<code>'xx</code> AS string)</td>
<td>&quot;xx&quot; (<code>'xx</code> is an Ion symbol)</td>
</tr>
<tr>
<td>CAST('xx` AS string)</td>
<td>&quot;xx&quot; ('xx' is a string)</td>
</tr>
<tr>
<td>CAST(42 AS string)</td>
<td>&quot;42&quot;</td>
</tr>
<tr>
<td>CAST('le0` AS string)</td>
<td>&quot;1.0&quot;</td>
</tr>
<tr>
<td>CAST('1d0` AS string)</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>CAST(true AS string)</td>
<td>&quot;true&quot;</td>
</tr>
<tr>
<td>CAST(false AS string)</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>CAST(<code>'2001T</code> AS string)</td>
<td>&quot;2001T&quot;</td>
</tr>
<tr>
<td>CAST(<code>'2001-01-01T00:00:00.000Z</code> AS string)</td>
<td>&quot;2001-01-01T00:00:00.000Z&quot;</td>
</tr>
</tbody>
</table>
The following examples show casting AS `struct`.

<table>
<thead>
<tr>
<th>Cast Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CAST(\{ a: 1 \}</code> AS struct)</td>
<td>{a:1}</td>
</tr>
<tr>
<td>CAST(true AS struct)</td>
<td>error</td>
</tr>
</tbody>
</table>

The following examples show casting AS `list`.

<table>
<thead>
<tr>
<th>Cast Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CAST(</code>[1, 2, 3]` AS list)</td>
<td>[1,2,3]</td>
</tr>
<tr>
<td><code>CAST(\&quot;a\&quot;, \{ \&quot;b\&quot;:2 \}\) AS list)</code></td>
<td>[&quot;a&quot;,&quot;b&quot;:2]</td>
</tr>
<tr>
<td><code>CAST(\&quot;b\&quot;:2\) AS list)</code></td>
<td>error</td>
</tr>
</tbody>
</table>

**CHAR_LENGTH Function**

Returns the number of characters in the specified string, where `character` is defined as a single unicode code point.

**Syntax**

```
CHAR_LENGTH ( string )
```

**Arguments**

- `string`
  
  The field name or expression of data type `string` that the function evaluates.

**Return Type**

`int`

**Examples**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR_LENGTH(&quot;&quot;)</td>
<td>0</td>
</tr>
<tr>
<td>CHAR_LENGTH(abcdefg)</td>
<td>7</td>
</tr>
<tr>
<td>CHAR_LENGTH('e#')</td>
<td>2</td>
</tr>
</tbody>
</table>

**CHARACTER_LENGTH Function**

Synonym of the `CHAR_LENGTH` function.

See `CHAR_LENGTH` Function (p. 310).

**COALESCE Function**

Given a list of one or more arguments, evaluates the arguments in order from left to right and returns 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 does not propagate NULL and MISSING.
Syntax

\[
\text{COALESCE} ( \text{expression} [, \text{expression}, \ldots ] )
\]

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 (p. 272).

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

\[
\begin{align*}
\text{COALESCE(missing, missing)} & \rightarrow \text{null} \\
\text{COALESCE(1, null)} & \rightarrow 1 \\
\text{COALESCE(null, null, 1)} & \rightarrow 1 \\
\text{COALESCE(null, 'string')} & \rightarrow \text{'string'} \\
\text{COALESCE(missing, 1)} & \rightarrow 1
\end{align*}
\]

COUNT Function

Returns the number of documents that are defined by the expression.

The COUNT function has two variations. \(\text{COUNT} \ (\ast)\) counts all the documents in the target table whether or not they include null or missing values. \(\text{COUNT} \ (\text{expression})\) computes the number of documents with non-null values in a specific, existing field or expression.

Syntax

\[
\text{COUNT} \ (\ast | \text{expression})
\]

Arguments

expression

The field name or expression that the function operates on. This parameter can be any of the supported Data Types (p. 272).

Return Type

int

Examples

\[
\begin{align*}
\text{SELECT COUNT(*) FROM VehicleRegistration r} & \rightarrow 5 \\
\text{SELECT COUNT(r.VIN) FROM VehicleRegistration r WHERE r.City = 'Tacoma'} & \rightarrow 1
\end{align*}
\]
DATE_ADD Function

Increments a given timestamp value by a specified interval.

Syntax

```
DATEADD( datepart, interval, timestamp )
```

Arguments

datepart

The date 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.

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`) -- 2017-01-02T03:04:05.006Z
DATE_ADD(second, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:04:06.006Z
```

DATE_DIFF Function

Returns the difference between the specified date parts of two given timestamps.

Syntax

```
DATE_DIFF( datepart, timestamp1, timestamp2 )
```
Arguments

datepart

The date 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.

Return Type

int

Examples

<table>
<thead>
<tr>
<th>Date Difference Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE_DIFF(year, '2010-01-01T', '2011-01-01T')</td>
<td>1</td>
</tr>
<tr>
<td>DATE_DIFF(year, '2010-12T', '2011-01T')</td>
<td>0 (must be at least 12 months apart to evaluate as a 1 year difference)</td>
</tr>
<tr>
<td>DATE_DIFF(month, '2010T', '2010-05T')</td>
<td>4 (2010T is equivalent to 2010-01-01T00:00:00Z)</td>
</tr>
<tr>
<td>DATE_DIFF(month, '2010T', '2011T')</td>
<td>12</td>
</tr>
<tr>
<td>DATE_DIFF(month, '2011T', '2010T')</td>
<td>-12</td>
</tr>
<tr>
<td>DATE_DIFF(month, '2010-12-31T', '2011-01-01T')</td>
<td>0 (must be at least a full month apart to evaluate as a 1 month difference)</td>
</tr>
<tr>
<td>DATE_DIFF(day, '2010-01-01T23:00T', '2010-01-02T01:00T')</td>
<td>0 (must be at least 24 hours apart to evaluate as a 1 day difference)</td>
</tr>
</tbody>
</table>

EXISTS Function

Given a value, returns TRUE if the value is a non-empty collection. Otherwise, returns FALSE. If the input to EXISTS is not a container, the result is FALSE.

The EXISTS function does not propagate NULL and MISSING.

Syntax

EXISTS ( value )

Arguments

datepart

The field name or expression that the function evaluates. This parameter can be any of the supported Data Types (p. 272).
Return Type

bool

Examples

<table>
<thead>
<tr>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXISTS(<code>[]</code>)</td>
<td>false (empty list)</td>
</tr>
<tr>
<td>EXISTS(<code>[1, 2, 3]</code>)</td>
<td>true (non-empty list)</td>
</tr>
<tr>
<td>EXISTS(<code>[missing]</code>)</td>
<td>true (non-empty list)</td>
</tr>
<tr>
<td>EXISTS(<code>{}</code>)</td>
<td>false (empty struct)</td>
</tr>
<tr>
<td>EXISTS(<code>{'a': 1}</code>)</td>
<td>true (non-empty struct)</td>
</tr>
<tr>
<td>EXISTS(<code>()</code></td>
<td>false (empty s-expression)</td>
</tr>
<tr>
<td>EXISTS(<code>(+ 1 2)</code></td>
<td>true (non-empty s-expression)</td>
</tr>
<tr>
<td>EXISTS(<code>&lt;&lt;&gt;&gt;</code>)</td>
<td>false (empty bag)</td>
</tr>
<tr>
<td>EXISTS(<code>&lt;&lt;null&gt;&gt;</code>)</td>
<td>true (non-empty bag)</td>
</tr>
<tr>
<td>EXISTS(1)</td>
<td>false</td>
</tr>
<tr>
<td>EXISTS(<code>2017T</code>)</td>
<td>false</td>
</tr>
<tr>
<td>EXISTS(null)</td>
<td>false</td>
</tr>
<tr>
<td>EXISTS(missing)</td>
<td>false</td>
</tr>
</tbody>
</table>

EXTRACT Function

Returns the integer value of a specified date part from a given timestamp.

Syntax

```
EXTRACT ( datepart FROM timestamp )
```

Arguments

**datepart**

The date 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.

Return Type

int

Examples

```
EXTRACT(YEAR FROM `2010-01-01T`) -- 2010
```
LOWER Function

Converts 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

```
LOWER('AbCdEfG!@#$')  -- 'abcdefg!@#$'
```

MAX Function

Returns the maximum value in a set of documents.

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
```

MIN Function

Syntax

```
MIN ( expression )
```

Returns the minimum value in a set of documents.

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
```

NULLIF Function

Given two expressions, returns NULL if the two expressions evaluate to the same value. Otherwise, returns the result of evaluating the first expression.
The **NULLIF** function does not propagate **NULL** and **MISSING**.

**Syntax**

```plaintext
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 (p. 272).

**Return Type**

Any supported data type. The return type is either **NULL** or the same as the type of the first expression.

**Examples**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULLIF(1, 1)</td>
<td>-- null</td>
</tr>
<tr>
<td>NULLIF(1, 2)</td>
<td>-- 1</td>
</tr>
<tr>
<td>NULLIF(1.0, 1)</td>
<td>-- null</td>
</tr>
<tr>
<td>NULLIF(1, '1')</td>
<td>-- 1</td>
</tr>
<tr>
<td>NULLIF([1], [1])</td>
<td>-- null</td>
</tr>
<tr>
<td>NULLIF(1, NULL)</td>
<td>-- null</td>
</tr>
<tr>
<td>NULLIF(NULL, 1)</td>
<td>-- null</td>
</tr>
<tr>
<td>NULLIF(null, null)</td>
<td>-- null</td>
</tr>
<tr>
<td>NULLIF(missing, null)</td>
<td>-- null</td>
</tr>
<tr>
<td>NULLIF(missing, missing)</td>
<td>-- null</td>
</tr>
</tbody>
</table>

**SIZE Function**

Returns the number of elements in a given container data type (list, structure, or bag).

**Syntax**

```plaintext
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**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
</table>
| `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  |

### SUBSTRING Function

Returns 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 `(start-index + length) - 1`. In other words, the length of the substring. This number cannot be negative.
  
  If this parameter is not provided, the function proceeds until the end of the `string`.

**Return Type**

`string`

**Examples**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SUBSTRING('123456789', 0)</code></td>
<td>'123456789'</td>
</tr>
<tr>
<td><code>SUBSTRING('123456789', 1)</code></td>
<td>'123456789'</td>
</tr>
<tr>
<td><code>SUBSTRING('123456789', 2)</code></td>
<td>'23456789'</td>
</tr>
<tr>
<td><code>SUBSTRING('123456789', -4)</code></td>
<td>'123456789'</td>
</tr>
</tbody>
</table>
SUM Function

Returns the sum of the input field or expression values. The SUM 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
```

TO_STRING Function

Returns 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.

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 (p. 323).

Return Type

string

Examples

| TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y')  | -- "July 20, 1969" |
| TO_STRING(`1969-07-20T20:18Z`, 'M-d-yy')   | -- "7-20-69" |
| TO_STRING(`1969-07-20T20:18:00+08:00Z`, 'y-MM-dd''T''H:m:ssXXXX') | -- "1969-07-20T20:18:00+0800" |

TO_TIMESTAMP Function

Given a string that represents a timestamp, converts 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 (p. 323).

If this argument is omitted, the function assumes that the string is in the format of a standard Ion timestamp. This is the recommended way to parse an Ion timestamp using this function.

Zero padding is optional when using a single-character format symbol (such as y, M, d, 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**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TO_TIMESTAMP('2007T')</code></td>
<td><code>2007T</code></td>
</tr>
<tr>
<td><code>TO_TIMESTAMP('2007-02-23T12:14:33.079-08:00')</code></td>
<td><code>2007-02-23T12:14:33.079-08:00</code></td>
</tr>
<tr>
<td><code>TO_TIMESTAMP('2016', 'y')</code></td>
<td><code>2016T</code></td>
</tr>
<tr>
<td><code>TO_TIMESTAMP('2016', 'yyyy')</code></td>
<td><code>2016T</code></td>
</tr>
<tr>
<td><code>TO_TIMESTAMP('02-2016', 'MM-yyyy')</code></td>
<td><code>2016-02T</code></td>
</tr>
<tr>
<td><code>TO_TIMESTAMP('Feb 2016', 'MMM yyyy')</code></td>
<td><code>2016-02T</code></td>
</tr>
<tr>
<td><code>TO_TIMESTAMP('February 2016', 'MMMM yyyy')</code></td>
<td><code>2016-02T</code></td>
</tr>
</tbody>
</table>

**TRIM Function**

Trims 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**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TRIM(' foobar ')</code></td>
<td><code>'foobar'</code></td>
</tr>
</tbody>
</table>
TXID Function

Returns the unique transaction ID of the current statement that you are executing. 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 >>  -- "L7S9iJqcn9W2M4qOEn27ay"
```

UPPER Function

Converts 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

```
UPPER('AbCdEfG1@##')  -- 'ABCDEFG1@##'
```
UTCNOW Function

Returns the current time in Coordinated Universal Time (UTC) as a timestamp.

Syntax

UTCNOW()

Arguments

None

Return Type

timestamp

Examples

```
SELECT UTCNOW() FROM << 0 >>  -- 2019-12-27T20:12:16.999Z
```

Timestamp Format Strings

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.

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yy</td>
<td>70</td>
<td>Two-digit year</td>
</tr>
<tr>
<td>y</td>
<td>1970</td>
<td>Four-digit year</td>
</tr>
<tr>
<td>yyyy</td>
<td>1970</td>
<td>Zero-padded four-digit year</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>Month integer</td>
</tr>
<tr>
<td>MM</td>
<td>01</td>
<td>Zero-padded month integer</td>
</tr>
<tr>
<td>MMM</td>
<td>Jan</td>
<td>Abbreviated month name</td>
</tr>
<tr>
<td>MMMM</td>
<td>January</td>
<td>Full month name</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>Day of the month (1–31)</td>
</tr>
<tr>
<td>dd</td>
<td>02</td>
<td>Zero-padded day of the month (01–31)</td>
</tr>
<tr>
<td>a</td>
<td>AM or PM</td>
<td>Meridian indicator (for 12-hour clock)</td>
</tr>
<tr>
<td>h</td>
<td>3</td>
<td>Hour (12-hour clock, 1–12)</td>
</tr>
<tr>
<td>hh</td>
<td>03</td>
<td>Zero-padded hour (12-hour clock, 01–12)</td>
</tr>
<tr>
<td>H</td>
<td>3</td>
<td>Hour (24-hour clock, 0–23)</td>
</tr>
</tbody>
</table>
### PartiQL Operators

PartiQL in Amazon QLDB supports the following SQL standard operators.

**Note**
Any SQL operators that are not included in this list are not currently supported in QLDB.

#### Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Add</td>
</tr>
<tr>
<td>-</td>
<td>Subtract</td>
</tr>
<tr>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
</tbody>
</table>
### Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Modulo</td>
</tr>
</tbody>
</table>

### Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>TRUE if all the conditions separated by AND are TRUE</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>TRUE if the operand is within the range of comparisons</td>
</tr>
<tr>
<td>IN</td>
<td>TRUE if the operand is equal to one of a list of expressions</td>
</tr>
<tr>
<td>IS</td>
<td>TRUE if the operand is a given data type, including NULL or MISSING</td>
</tr>
<tr>
<td>LIKE</td>
<td>TRUE if the operand matches a pattern</td>
</tr>
<tr>
<td>NOT</td>
<td>Reverses the value of a given Boolean expression</td>
</tr>
<tr>
<td>OR</td>
<td>TRUE if any of the conditions separated by OR is TRUE</td>
</tr>
</tbody>
</table>

### String Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reserved Words

The following is a list of PartiQL reserved words in Amazon QLDB. You can use a reserved word as a quoted identifier with double quotation marks (for example, "user").

**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 sections:

- PartiQL Functions (p. 305)
- PartiQL Operators (p. 324)
- PartiQL Statements (p. 292)

<table>
<thead>
<tr>
<th>Reserved Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSOLUTE</td>
</tr>
<tr>
<td>ACTION</td>
</tr>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>ALL</td>
</tr>
<tr>
<td>ALLOCATE</td>
</tr>
<tr>
<td>ALTER</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>ANY</td>
</tr>
<tr>
<td>ARE</td>
</tr>
<tr>
<td>AS</td>
</tr>
<tr>
<td>ASC</td>
</tr>
<tr>
<td>ASSERTION</td>
</tr>
<tr>
<td>AT</td>
</tr>
<tr>
<td>AUTHORIZATION</td>
</tr>
<tr>
<td>AVG</td>
</tr>
<tr>
<td>BAG</td>
</tr>
<tr>
<td>BEGIN</td>
</tr>
<tr>
<td>BETWEEN</td>
</tr>
<tr>
<td>BIT</td>
</tr>
<tr>
<td>BIT_LENGTH</td>
</tr>
<tr>
<td>BLOB</td>
</tr>
<tr>
<td>BOOL</td>
</tr>
<tr>
<td>BOOLEAN</td>
</tr>
<tr>
<td>BOTH</td>
</tr>
<tr>
<td>BY</td>
</tr>
<tr>
<td>CASCADE</td>
</tr>
<tr>
<td>CASCADED</td>
</tr>
<tr>
<td>CASE</td>
</tr>
<tr>
<td>CAST</td>
</tr>
<tr>
<td>CATALOG</td>
</tr>
<tr>
<td>CHAR</td>
</tr>
<tr>
<td>CHARACTER</td>
</tr>
<tr>
<td>CHARACTER_LENGTH</td>
</tr>
<tr>
<td>CHAR_LENGTH</td>
</tr>
<tr>
<td>CHECK</td>
</tr>
<tr>
<td>CLOB</td>
</tr>
<tr>
<td>CLOSE</td>
</tr>
<tr>
<td>COALESCE</td>
</tr>
<tr>
<td>COLLATE</td>
</tr>
<tr>
<td>COLLATION</td>
</tr>
<tr>
<td>COLUMN</td>
</tr>
<tr>
<td>COMMIT</td>
</tr>
<tr>
<td>CONNECT</td>
</tr>
<tr>
<td>CONNECTION</td>
</tr>
<tr>
<td>CONSTRAINT</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
</tr>
<tr>
<td>CONTINUE</td>
</tr>
</tbody>
</table>

326
<table>
<thead>
<tr>
<th>CONVERT</th>
<th>CORRESPONDING</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>CROSS</td>
<td>CURRENT</td>
</tr>
<tr>
<td>CURRENT_DATE</td>
<td>CURRENT_TIME</td>
<td>CURRENT_TIMESTAMP</td>
</tr>
<tr>
<td>CURRENT_USER</td>
<td>CURSOR</td>
<td>DATE</td>
</tr>
<tr>
<td>DATE_ADD</td>
<td>DATE_DIFF</td>
<td>DAY</td>
</tr>
<tr>
<td>DEALLOCATE</td>
<td>DEC</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>DECLARE</td>
<td>DEFAULT</td>
<td>DEFERRABLE</td>
</tr>
<tr>
<td>DEFERRED</td>
<td>DELETE</td>
<td>DESC</td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>DESCRIPTOR</td>
<td>DIAGNOSTICS</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>DISTINCT</td>
<td>DOMAIN</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DROP</td>
<td>ELSE</td>
</tr>
<tr>
<td>END</td>
<td>END-EXEC</td>
<td>ESCAPE</td>
</tr>
<tr>
<td>EXCEPT</td>
<td>EXCEPTION</td>
<td>EXEC</td>
</tr>
<tr>
<td>EXECUTES</td>
<td>EXISTS</td>
<td>EXTERNAL</td>
</tr>
<tr>
<td>EXTRACT</td>
<td>FALSE</td>
<td>FETCH</td>
</tr>
<tr>
<td>FIRST</td>
<td>FLOAT</td>
<td>FOR</td>
</tr>
<tr>
<td>FOREIGN</td>
<td>FOUND</td>
<td>FROM</td>
</tr>
<tr>
<td>FULL</td>
<td>GET</td>
<td>GLOBAL</td>
</tr>
<tr>
<td>GO</td>
<td>GOTO</td>
<td>GRANT</td>
</tr>
<tr>
<td>GROUP</td>
<td>HAVING</td>
<td>HOUR</td>
</tr>
<tr>
<td>HOUR</td>
<td>IDENTITY</td>
<td>IMMEDIATE</td>
</tr>
<tr>
<td>IN</td>
<td>INDEX</td>
<td>INDICATOR</td>
</tr>
<tr>
<td>INITIAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Patched Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIVE</td>
<td></td>
</tr>
<tr>
<td>REMOVE</td>
<td></td>
</tr>
<tr>
<td>RESTRICT</td>
<td></td>
</tr>
<tr>
<td>REVOKE</td>
<td></td>
</tr>
<tr>
<td>RIGHT</td>
<td></td>
</tr>
<tr>
<td>ROLLBACK</td>
<td></td>
</tr>
<tr>
<td>RONS</td>
<td></td>
</tr>
<tr>
<td>SCHEMA</td>
<td></td>
</tr>
<tr>
<td>SCROLL</td>
<td></td>
</tr>
<tr>
<td>SECOND</td>
<td></td>
</tr>
<tr>
<td>SECTION</td>
<td></td>
</tr>
<tr>
<td>SELECT</td>
<td></td>
</tr>
<tr>
<td>SESSION</td>
<td></td>
</tr>
<tr>
<td>SESSION_USER</td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>SEXP</td>
<td></td>
</tr>
<tr>
<td>SIZE</td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>SOME</td>
<td></td>
</tr>
<tr>
<td>SPACE</td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td></td>
</tr>
<tr>
<td>SQLCODE</td>
<td></td>
</tr>
<tr>
<td>SQLERROR</td>
<td></td>
</tr>
<tr>
<td>SQLSTATE</td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>STRUCT</td>
<td></td>
</tr>
<tr>
<td>SUBSTRING</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td></td>
</tr>
<tr>
<td>SYMBOL</td>
<td></td>
</tr>
<tr>
<td>SYSTEM_USER</td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td></td>
</tr>
<tr>
<td>TEMPORARY</td>
<td></td>
</tr>
<tr>
<td>THEN</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>TIMEZONE_HOUR</td>
<td></td>
</tr>
<tr>
<td>TIMEZONE_MINUTE</td>
<td></td>
</tr>
<tr>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>TO_STRING</td>
<td></td>
</tr>
<tr>
<td>TO_TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>TRAILING</td>
<td></td>
</tr>
<tr>
<td>TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>TRANSLATE</td>
<td></td>
</tr>
<tr>
<td>TRANSLATION</td>
<td></td>
</tr>
<tr>
<td>TRIM</td>
<td></td>
</tr>
<tr>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>TUPLE</td>
<td></td>
</tr>
<tr>
<td>TXID</td>
<td></td>
</tr>
<tr>
<td>UNDROP</td>
<td></td>
</tr>
<tr>
<td>UNION</td>
<td></td>
</tr>
<tr>
<td>UNIQUE</td>
<td></td>
</tr>
<tr>
<td>UNKNOWN</td>
<td></td>
</tr>
<tr>
<td>UNPIVOT</td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td></td>
</tr>
<tr>
<td>UPPER</td>
<td></td>
</tr>
<tr>
<td>USAGE</td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
</tr>
<tr>
<td>USING</td>
<td></td>
</tr>
<tr>
<td>UTCNOW</td>
<td></td>
</tr>
<tr>
<td>VALUE</td>
<td></td>
</tr>
<tr>
<td>VALUES</td>
<td></td>
</tr>
<tr>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>VARYING</td>
<td></td>
</tr>
<tr>
<td>VIEW</td>
<td></td>
</tr>
<tr>
<td>WHEN</td>
<td></td>
</tr>
<tr>
<td>WHENEVER</td>
<td></td>
</tr>
</tbody>
</table>
Reserved Words

WHERE
WITH
WORK
WRITE
YEAR
ZONE
Amazon QLDB API Reference

This section 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 control plane. These APIs are used only for managing ledgers and for non-transactional data operations. You can use these APIs to create, delete, modify, and describe ledgers. You can also cryptographically verify a document and export journal blocks.

- **Amazon QLDB Session** – The QLDB transactional data plane. You can use these APIs to execute data transactions on a ledger with PartiQL (p. 272) statements.

  Note
  We highly recommend that you use the Amazon QLDB Driver to execute data transactions on a ledger. The driver provides a high-level abstraction layer over the qldbsession data plane and manages SendCommand API calls for you.
  For information and a list of supported programming languages, see *Getting Started with the Driver* (p. 33).

Topics
- Actions (p. 331)
- Data Types (p. 375)
- Common Errors (p. 397)
- Common Parameters (p. 399)

Actions

The following actions are supported by Amazon QLDB:

- CreateLedger (p. 333)
- DeleteLedger (p. 337)
- DescribeJournalS3Export (p. 339)
- DescribeLedger (p. 341)
- ExportJournalToS3 (p. 344)
- GetBlock (p. 347)
- GetDigest (p. 350)
- GetRevision (p. 352)
- ListJournalS3Exports (p. 355)
- ListJournalS3ExportsForLedger (p. 357)
- ListLedgers (p. 360)
- ListTagsForResource (p. 362)
- TagResource (p. 364)
- UntagResource (p. 366)
- UpdateLedger (p. 368)

The following actions are supported by Amazon QLDB Session:

- SendCommand (p. 371)
Amazon QLDB

The following actions are supported by Amazon QLDB:

- CreateLedger (p. 333)
- DeleteLedger (p. 337)
- DescribeJournalS3Export (p. 339)
- DescribeLedger (p. 341)
- ExportJournalToS3 (p. 344)
- GetBlock (p. 347)
- GetDigest (p. 350)
- GetRevision (p. 352)
- ListJournalS3Exports (p. 355)
- ListJournalS3ExportsForLedger (p. 357)
- ListLedgers (p. 360)
- ListTagsForResource (p. 362)
- TagResource (p. 364)
- UntagResource (p. 366)
- UpdateLedger (p. 368)
CreateLedger
Service: Amazon QLDB

Creates a new ledger in your AWS account.

Request Syntax

```
POST /ledgers HTTP/1.1
Content-type: application/json

{
   "DeletionProtection": boolean,
   "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 (p. 333)**

The flag that prevents a ledger from being deleted by any user. If not provided on 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 using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the `UpdateLedger` operation to set the flag to `false`. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Required: No

**Name (p. 333)**

The name of the ledger that you want to create. The name must be unique among all of your ledgers in the current AWS Region.

Type: String


Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

Required: Yes

**PermissionsMode (p. 333)**

The permissions mode to assign to the ledger that you want to create.

Type: String

Valid Values: `ALLOW_ALL`
Required: Yes

**Tags (p. 333)**

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

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,
   "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 (p. 334)**

The Amazon Resource Name (ARN) for the ledger.

Type: String


**CreationDateTime (p. 334)**

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 (p. 334)**

The flag that prevents a ledger from being deleted by any user. If not provided on 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 using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the UpdateLedger operation to set the flag to false. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean
**Name (p. 334)**

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 (p. 334)**

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 (p. 397)](#).

- **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](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [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 using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the UpdateLedger operation to set the flag to false. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Request Syntax

DELETE /ledgers/name HTTP/1.1

URI Request Parameters

The request requires the following URI parameters.

name (p. 337)

The name of the ledger that you want to delete.


Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9]+$

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 (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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, when it was created, current status, and its start and end time export parameters.

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 requires the following URI parameters.

exportId (p. 339)

The unique ID of the journal export job that you want to describe.

Length Constraints: Fixed length of 22.

Pattern: ^[A-Za-z-0-9]+$ name (p. 339)

The name of the ledger.


Pattern: (?!.*-)(?!^[0-9]+$)(?!^-)(?!.*$)^[A-Za-z0-9-]+$

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",
        "RoleArn": "string",
        "S3ExportConfiguration": {
            "Bucket": "string",
            "EncryptionConfiguration": {
                "KmsKeyArn": "string",
                "ObjectEncryptionType": "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 (p. 339)

Information about the journal export job returned by a DescribeJournalS3Export request.

Type: JournalS3ExportDescription (p. 376) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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 and when it was created.

Request Syntax

GET /ledgers/name HTTP/1.1

URI Request Parameters

The request requires the following URI parameters.

`name` (p. 341)

The name of the ledger that you want to describe.


Pattern: `(?!^.*--)(?!^\[0-9\]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

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,
   "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` (p. 341)

The Amazon Resource Name (ARN) for the ledger.

Type: String


`CreationDateTime` (p. 341)

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 (p. 341)

The flag that prevents a ledger from being deleted by any user. If not provided on 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 using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the UpdateLedger operation to set the flag to false. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Name (p. 341)

The name of the ledger.

Type: String


Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$  

State (p. 341)

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 (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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. The data is written as files in Amazon Ion 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,
  "RoleArn": "string",
  "S3ExportConfiguration": {
    "Bucket": "string",
    "EncryptionConfiguration": {
      "KmsKeyArn": "string",
      "ObjectEncryptionType": "string"
    },
    "Prefix": "string"
  }
}

URI Request Parameters

The request requires the following URI parameters.

name (p. 344)

The name of the ledger.


Pattern: (?![^.*-])(?!^[0-9]+$)(?!^-)(?!.*$)^[A-Za-z0-9-]+$

Request Body

The request accepts the following data in JSON format.

ExclusiveEndTime (p. 344)

The exclusive end date and time for the range of journal contents that you want 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
**InclusiveStartTime (p. 344)**

The inclusive start date and time for the range of journal contents that you want 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

**RoleArn (p. 344)**

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 master key (CMK) in AWS Key Management Service (AWS KMS) for server-side encryption of your exported data.

Type: String


Required: Yes

**S3ExportConfiguration (p. 344)**

The configuration settings of the Amazon S3 bucket destination for your export request.

Type: `S3ExportConfiguration (p. 380)` object

Required: Yes

**Response Syntax**

```json
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 (p. 345)**

The unique ID 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-z0-9]+$  

Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
GetBlock
Service: Amazon QLDB

Returns a journal block object at a specified address in a ledger. Also returns a proof of the specified block for verification if DigestTipAddress is provided.

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 requires the following URI parameters.

name (p. 347)

The name of the ledger.


Pattern: (?![^.*--](?!^[-]+)(?!^[-]+[0-9]+$)(?![^.-])(?!^[-]+.[^-]+$)^[A-Za-z0-9-]+$)

Request Body

The request accepts the following data in JSON format.

BlockAddress (p. 347)

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:"BlFTj1SXze9Bi1K0ScE3",sequenceNo:14}

Type: ValueHolder (p. 381) object

Required: Yes

DigestTipAddress (p. 347)

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":"BlFTjLSXze9Blih1K0szcE3",sequenceNo:49}

Type: ValueHolder (p. 381) 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 (p. 348)

The block data object in Amazon Ion format.

Type: ValueHolder (p. 381) object

Proof (p. 348)

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 (p. 381) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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 requires the following URI parameters.

name (p. 350)

The name of the ledger.


Pattern: (?![^\^]+[0-9]+$)(?!^(-)(?!.*-$)^[A-Za-z0-9-]+$)

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 (p. 350)

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 (p. 350)

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 (p. 381) object
Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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 requires the following URI parameters.

**name (p. 352)**

The name of the ledger.


Pattern: (?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

Request Body

The request accepts the following data in JSON format.

**BlockAddress (p. 352)**

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: "BlFTjISZe9B1h1K0szeE3", sequenceNo: 14}`

Type: ValueHolder (p. 381) object

Required: Yes

**DigestTipAddress (p. 352)**

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: "BlFTjISZe9B1h1K0szeE3", sequenceNo: 49}`

Type: ValueHolder (p. 381) object

Required: No

**DocumentId (p. 352)**

The unique ID 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 (p. 353)**

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 (p. 381) object

**Revision (p. 353)**

The document revision data object in Amazon Ion format.

Type: ValueHolder (p. 381) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

**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
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListJournalS3Exports

Service: Amazon QLDB

Returns an array of journal export job descriptions 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 requires the following URI parameters.

MaxResults (p. 355)

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 (p. 355)

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.


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",
      "RoleArn": "string",
      "S3ExportConfiguration": {
         "Bucket": "string",
         "EncryptionConfiguration": {
            "KmsKeyArn": "string",
            "ServerSideEncryption": "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 (p. 355)**

The array of journal export job descriptions for all ledgers that are associated with the current AWS account and Region.

Type: Array of JournalS3ExportDescription (p. 376) objects

**NextToken (p. 355)**

- 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


Pattern: `^[A-Za-z-0-9+/=]+$`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListJournalS3ExportsForLedger
Service: Amazon QLDB

Returns an array of journal export job descriptions 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 requires the following URI parameters.

MaxResults (p. 357)

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 (p. 357)

The name of the ledger.


Pattern: (?![^.*--])(?![0-9]+$)(?!^-)(?!.*$)^[A-Za-z0-9-]+$ NextToken (p. 357)

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.


Pattern: ^[A-Za-z0-9-0-9+/=]+$ 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",
   

357
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 (p. 357)**

The array of journal export job descriptions that are associated with the specified ledger.

Type: Array of **JournalS3ExportDescription (p. 376)** objects

**NextToken (p. 357)**

- 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


Pattern: ^[A-Za-z0-9+/=]+$
ListLedgers
Service: Amazon QLDB

Returns an array of ledger summaries that are associated with the current AWS account and Region. This action returns a maximum of 100 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 requires the following URI parameters.

MaxResults (p. 360)

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 (p. 360)

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.


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 (p. 360)

The array of ledger summaries that are associated with the current AWS account and Region.

Type: Array of LedgerSummary (p. 378) objects

NextToken (p. 360)

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


Pattern: ^[A-Za-z-0-9+/=]+$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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 requires the following URI parameters.

```
resourceArn (p. 362)
```

The Amazon Resource Name (ARN) for which you want to list the tags. For example:

```
arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger
```


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 (p. 362)
```

The tags that are currently associated with the specified Amazon QLDB resource.

Type: String to string map

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 (p. 397).
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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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

```plaintext
POST /tags/<resourceArn> HTTP/1.1
Content-type: application/json

{
   "Tags": {
      "string": "string"
   }
}
```

URI Request Parameters

The request requires the following URI parameters.

resourceArn (p. 364)

The Amazon Resource Name (ARN) to which you want to add the tags. For example:

```plaintext
arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger
```


Request Body

The request accepts the following data in JSON format.

Tags (p. 364)

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

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

```plaintext
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 (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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 requires the following URI parameters.

resourceArn (p. 366)

The Amazon Resource Name (ARN) from which you want to remove the tags. For example:

arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger


TagKeys (p. 366)

The list of tag keys that you want to remove.

Array Members: Minimum number of 0 items. Maximum number of 200 items.


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 (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- 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
}
```

URI Request Parameters

The request requires the following URI parameters.

**name (p. 368)**

The name of the ledger.


Pattern: (?!^.*--)(?!^\[0-9\]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$

Request Body

The request accepts the following data in JSON format.

**DeletionProtection (p. 368)**

The flag that prevents a ledger from being deleted by any user. If not provided on 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 using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the UpdateLedger operation to set the flag to false. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json
{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "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 (p. 368)

The Amazon Resource Name (ARN) for the ledger.

Type: String


CreationDateTime (p. 368)

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 (p. 368)

The flag that prevents a ledger from being deleted by any user. If not provided on 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 using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the UpdateLedger operation to set the flag to false. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Name (p. 368)

The name of the ledger.

Type: String


State (p. 368)

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 (p. 397).

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
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Amazon QLDB Session

The following actions are supported by Amazon QLDB Session:

- SendCommand (p. 371)
SendCommand
Service: Amazon QLDB Session

Sends a command to an Amazon QLDB ledger.

Note
We highly recommend that you use the Amazon QLDB Driver to execute data transactions on a ledger. The driver provides a high-level abstraction layer over this qldbsession data plane and manages SendCommand API calls for you.
For information and a list of supported programming languages, see Getting Started with the Amazon QLDB Driver in the Amazon QLDB Developer Guide.

Request Syntax

```json
{
  "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 (p. 399).

The request accepts the following data in JSON format.

AbortTransaction (p. 371)

Command to abort the current transaction.

Type: AbortTransactionRequest (p. 382) object

Required: No

CommitTransaction (p. 371)

Command to commit the specified transaction.
Type: `CommitTransactionRequest` (p. 384) object
Required: No

**EndSession** (p. 371)
Command to end the current session.
Type: `EndSessionRequest` (p. 386) object
Required: No

**ExecuteStatement** (p. 371)
Command to execute a statement in the specified transaction.
Type: `ExecuteStatementRequest` (p. 388) object
Required: No

**FetchPage** (p. 371)
Command to fetch a page.
Type: `FetchPageRequest` (p. 390) object
Required: No

**SessionToken** (p. 371)
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
Pattern: `^[A-Za-z-0-9+/=]+$`
Required: No

**StartSession** (p. 371)
Command to start a new session. A session token is obtained as part of the response.
Type: `StartSessionRequest` (p. 393) object
Required: No

**StartTransaction** (p. 371)
Command to start a new transaction.
Type: `StartTransactionRequest` (p. 395) object
Required: No

**Response Syntax**

```json
{
   "AbortTransaction": {
   }
}
```
"CommitTransaction": {
  "CommitDigest": blob,
  "TransactionId": "string"
},
"EndSession": {
},
"ExecuteStatement": {
  "FirstPage": {
    "NextPageToken": "string",
    "Values": [
      {
        "IonBinary": blob,
        "IonText": "string"
      }
    ]
  },
  "FetchPage": {
    "Page": {
      "NextPageToken": "string",
      "Values": [
        {
          "IonBinary": blob,
          "IonText": "string"
        }
      ]
    }
  },
  "StartSession": {
    "SessionToken": "string"
  },
  "StartTransaction": {
    "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 (p. 372)

Contains the details of the aborted transaction.

Type: AbortTransactionResult (p. 383) object

CommitTransaction (p. 372)

Contains the details of the committed transaction.

Type: CommitTransactionResult (p. 385) object

EndSession (p. 372)

Contains the details of the ended session.

Type: EndSessionResult (p. 387) object

ExecuteStatement (p. 372)

Contains the details of the executed statement.

Type: ExecuteStatementResult (p. 389) object
FetchPage (p. 372)

Contains the details of the fetched page.

Type: FetchPageResult (p. 391) object

StartSession (p. 372)

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 (p. 394) object

StartTransaction (p. 372)

Contains the details of the started transaction.

Type: StartTransactionResult (p. 396) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 397).

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

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
Data Types

The following data types are supported by Amazon QLDB:

- JournalS3ExportDescription (p. 376)
- LedgerSummary (p. 378)
- S3EncryptionConfiguration (p. 379)
- S3ExportConfiguration (p. 380)
- ValueHolder (p. 381)

The following data types are supported by Amazon QLDB Session:

- AbortTransactionRequest (p. 382)
- AbortTransactionResult (p. 383)
- CommitTransactionRequest (p. 384)
- CommitTransactionResult (p. 385)
- EndSessionRequest (p. 386)
- EndSessionResult (p. 387)
- ExecuteStatementRequest (p. 388)
- ExecuteStatementResult (p. 389)
- FetchPageRequest (p. 390)
- FetchPageResult (p. 391)
- Page (p. 392)
- StartSessionRequest (p. 393)
- StartSessionResult (p. 394)
- StartTransactionRequest (p. 395)
- StartTransactionResult (p. 396)
- ValueHolder (p. 397)

Amazon QLDB

The following data types are supported by Amazon QLDB:

- JournalS3ExportDescription (p. 376)
- LedgerSummary (p. 378)
- S3EncryptionConfiguration (p. 379)
- S3ExportConfiguration (p. 380)
- ValueHolder (p. 381)
JournalS3ExportDescription
Service: Amazon QLDB

The information about a journal export job, including the ledger name, export ID, when it was created, current status, and its start and end time export parameters.

Contents

ExclusiveEndTime
The exclusive end date and time for the range of journal contents that are 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 unique ID 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 are specified in the original export request.

Type: Timestamp
Required: Yes

LedgerName
The name of the ledger.

Type: String
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 master key (CMK) in AWS Key Management Service (AWS KMS) for server-side encryption of your exported data.

Type: String


Required: Yes

**S3ExportConfiguration**

The Amazon Simple Storage Service (Amazon S3) bucket location in which a journal export job writes the journal contents.

Type: [S3ExportConfiguration](p. 380) object

Required: Yes

**Status**

The current state of the journal export job.

Type: String

Valid Values: `IN_PROGRESS | COMPLETED | CANCELLED`

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 Go
- AWS SDK for Java
- 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


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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
Amazon Quantum Ledger Database
(Amazon QLDB) Developer Guide
Amazon QLDB

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

KmsKeyArn

The Amazon Resource Name (ARN) for a symmetric customer master key (CMK) in AWS Key Management Service (AWS KMS). Amazon QLDB does not support asymmetric CMKs.

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


Required: No

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

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 Go
- AWS SDK for Java
- 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


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 (p. 379) 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 Go
• AWS SDK for Java
• AWS SDK for Ruby V3
ValueHolder
Service: Amazon QLDB

A structure that can contain an Amazon Ion value in multiple encoding formats.

Contents
IonText

An Amazon Ion plaintext value contained in a ValueHolder structure.

Type: String
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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3

Amazon QLDB Session

The following data types are supported by Amazon QLDB Session:

- AbortTransactionRequest (p. 382)
- AbortTransactionResult (p. 383)
- CommitTransactionRequest (p. 384)
- CommitTransactionResult (p. 385)
- EndSessionRequest (p. 386)
- EndSessionResult (p. 387)
- ExecuteStatementRequest (p. 388)
- ExecuteStatementResult (p. 389)
- FetchPageRequest (p. 390)
- FetchPageResult (p. 391)
- Page (p. 392)
- StartSessionRequest (p. 393)
- StartSessionResult (p. 394)
- StartTransactionRequest (p. 395)
- StartTransactionResult (p. 396)
- ValueHolder (p. 397)
AbortTransactionRequest
Service: Amazon QLDB Session

Contains the details of the transaction to abort.

Contents

The members of this 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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
AbortTransactionResult
Service: Amazon QLDB Session

Contains the details of the aborted transaction.

Contents
The members of this 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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
CommmitTransactionRequest
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.

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 Go
- AWS SDK for Java
- 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

TransactionId

The transaction ID of the committed 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 Go
• AWS SDK for Java
• AWS SDK for Ruby V3
EndSessionRequest
Service: Amazon QLDB Session

Specifies a request to end the session.

Contents
The members of this 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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
EndSessionResult
Service: Amazon QLDB Session

Contains the details of the ended session.

Contents

The members of this 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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
**ExecuteStatementRequest**
Service: Amazon QLDB Session

Specifies a request to execute a statement.

**Contents**

**Parameters**

Specifies the parameters for the parameterized statement in the request.

Type: Array of *ValueHolder* (p. 397) objects

Required: No

**Statement**

Specifies the statement of the request.

Type: String


Required: Yes

**TransactionId**

Specifies the transaction ID of the request.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
ExecuteStatementResult
Service: Amazon QLDB Session

Contains the details of the executed statement.

Contents

FirstPage

Contains the details of the first fetched page.

Type: Page (p. 392) 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 Go
- AWS SDK for Java
- 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


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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
FetchPageResult
Service: Amazon QLDB Session

Contains the page that was fetched.

Contents

Page

Contains details of the fetched page.

Type: Page (p. 392) 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 Go
- AWS SDK for Java
- 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
Pattern: ^[\w-0-9+/=]+$
Required: No

**Values**

A structure that contains values in multiple encoding formats.
Type: Array of `ValueHolder (p. 397)` 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 Go
- AWS SDK for Java
- 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


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 Go
- AWS SDK for Java
- 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


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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
StartTransactionRequest
Service: Amazon QLDB Session

Specifies a request to start a transaction.

Contents

The members of this 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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
StartTransactionResult
Service: Amazon QLDB Session

Contains the details of the started transaction.

Contents

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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3
**ValueHolder**

Service: Amazon QLDB Session

A structure that can contain an Amazon Ion value in multiple encoding formats.

**Contents**

**IonBinary**

An Amazon Ion binary value contained in a ValueHolder structure.

Type: Base64-encoded binary data object


Required: No

**IonText**

An Amazon Ion plaintext value contained in a ValueHolder structure.

Type: String


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 Go
- AWS SDK for Java
- AWS SDK for Ruby V3

**Common Errors**

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

InvalidParameterCombination
Parameters that must not be used together were used together.
HTTP Status Code: 400

InvalidParameterValue
An invalid or out-of-range value was supplied for the input parameter.
HTTP Status Code: 400

InvalidQueryParameter
The AWS query string is malformed or does not adhere to AWS standards.
HTTP Status Code: 400

MalformedQueryString
The query string contains a syntax error.
HTTP Status Code: 404

MissingAction
The request is missing an action or a required parameter.
HTTP Status Code: 400

MissingAuthenticationToken
The request must contain either a valid (registered) AWS access key ID or X.509 certificate.
HTTP Status Code: 403

MissingParameter
A required parameter for the specified action is not supplied.
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

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 Signature Version 4 Signing Process in the Amazon Web Services General Reference.

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 Task 2: Create a String to Sign for Signature Version 4 in the Amazon Web Services General Reference.
Common Parameters

**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 Handling Dates in Signature Version 4 in the Amazon Web Services General Reference.

**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 Security Token Service, go to AWS Services That Work with IAM in the IAM User Guide.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

**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.

**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 Task 1: Create a Canonical Request For Signature Version 4 in the Amazon Web Services General Reference.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.
The Database Query Metadata Service (dbqms) provides your recent and saved queries for the Query editor on the AWS Management Console for Amazon QLDB.

The following DBQMS actions are supported:

Topics
- CreateFavoriteQuery (p. 401)
- CreateQueryHistory (p. 401)
- DeleteFavoriteQueries (p. 401)
- DeleteQueryHistory (p. 401)
- DescribeFavoriteQueries (p. 401)
- DescribeQueryHistory (p. 401)
- GetQueryString (p. 402)
- UpdateFavoriteQuery (p. 402)
- UpdateQueryHistory (p. 402)

CreateFavoriteQuery

Save a new favorite query. Each IAM user can create up to 1000 saved queries. This limit is subject to change in the future.

CreateQueryHistory

Save a new query history entry.

DeleteFavoriteQueries

Allows to delete multiple saved queries.

DeleteQueryHistory

Delete query history entries.

DescribeFavoriteQueries

List of saved queries created by an IAM user in a given account.

DescribeQueryHistory

List query history entries.
GetQueryString

Retrieve full query text from a query ID.

UpdateFavoriteQuery

Update the query string, description, name, or expiration date.

UpdateQueryHistory

Update the status of query history.
Quotas in Amazon QLDB

This section describes the current quotas, also referred to as limits, in Amazon QLDB.

Topics
- Default Quotas (p. 403)
- Fixed Quotas (p. 403)
- Transaction Size (p. 404)
- Naming Constraints (p. 404)

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, open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default Quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>The maximum number of active ledgers that you can create in this account in the current Region</td>
<td>5</td>
</tr>
</tbody>
</table>

Fixed Quotas

In addition to default quotas, QLDB has the following fixed quotas per ledger. These quotas cannot be increased:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Fixed Quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of concurrent driver sessions</td>
<td>1500</td>
</tr>
<tr>
<td>Number of concurrent journal export jobs</td>
<td>2</td>
</tr>
<tr>
<td>Number of active tables</td>
<td>20</td>
</tr>
<tr>
<td>Number of total tables (active and inactive)</td>
<td>40</td>
</tr>
<tr>
<td>Number of indexes per table</td>
<td>5</td>
</tr>
<tr>
<td>Number of documents in a transaction</td>
<td>40</td>
</tr>
<tr>
<td>Document size</td>
<td>128 KB</td>
</tr>
<tr>
<td>Transaction size</td>
<td>4 MB</td>
</tr>
<tr>
<td>Expiration period for completed journal export jobs</td>
<td>7 days</td>
</tr>
</tbody>
</table>
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 QLDB-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 does not 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 does not 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.

<table>
<thead>
<tr>
<th>Ledger name</th>
<th>Must only contain 1–32 alphanumeric characters or hyphens.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Must not be all numbers.</td>
</tr>
<tr>
<td></td>
<td>Can’t contain two consecutive hyphens.</td>
</tr>
<tr>
<td></td>
<td>The first and last characters must be a letter or number.</td>
</tr>
<tr>
<td></td>
<td>The ledger names are case sensitive.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table name</th>
<th>Must only contain 1–128 alphanumeric characters or underscores.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The first character must be a letter or an underscore.</td>
</tr>
<tr>
<td></td>
<td>The remaining characters can be any combination of alphanumeric characters and underscores.</td>
</tr>
<tr>
<td></td>
<td>The table names are case sensitive.</td>
</tr>
<tr>
<td></td>
<td>Must not be a QLDB PartiQL reserved word (p. 326).</td>
</tr>
</tbody>
</table>
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**: March 27, 2020

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon QLDB Shell (p. 405)</td>
<td>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 Plane Only) (Preview).</td>
<td>March 23, 2020</td>
</tr>
<tr>
<td>Amazon QLDB Driver for Java v1.1 (p. 405)</td>
<td>The Amazon QLDB Driver for Java version 1.1 is now generally available. For release notes, see the amazon-qldb-driver-java repository on the AWS Labs GitHub site.</td>
<td>March 20, 2020</td>
</tr>
<tr>
<td>Amazon QLDB Driver for .NET (p. 405)</td>
<td>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).</td>
<td>March 13, 2020</td>
</tr>
<tr>
<td>Amazon QLDB Driver for Python (p. 405)</td>
<td>The Amazon QLDB Driver for Python is now generally available. This driver is open-sourced on GitHub and enables you to use the AWS SDK for Python (Boto 3) to interact with QLDB's transactional data API. For more information, see Amazon QLDB Driver for Python. For release notes, see the amazon-qldb-driver-python repository on the AWS Labs GitHub site.</td>
<td>March 11, 2020</td>
</tr>
<tr>
<td>Topic</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>PartiQL UNDROP TABLE statement and nested DML (p. 405)</td>
<td>PartiQL in Amazon QLDB now supports <em>data manipulation language</em> (DML) statements in which nested collections are specified as sources in the <code>FROM</code> clause. For more information, see the <code>FROM</code> statement in the PartiQL Reference. QLDB PartiQL also supports the UNDROP TABLE statement. For more information, see Undropping Tables.</td>
<td>February 26, 2020</td>
</tr>
<tr>
<td>Expanded intro topic (p. 405)</td>
<td>Expanded the introductory <em>What is Amazon QLDB?</em> topic to include the new sections Overview of Amazon QLDB and From Relational to Ledger.</td>
<td>January 24, 2020</td>
</tr>
<tr>
<td>PartiQL Reference for supported functions (p. 405)</td>
<td>Expanded the Amazon QLDB PartiQL Reference to include detailed usage information about the supported SQL functions. For more information, see PartiQL Functions.</td>
<td>January 2, 2020</td>
</tr>
<tr>
<td>Driver Best Practices and Common Errors (p. 405)</td>
<td>Added new sections that describe best practices for the Amazon QLDB Driver for Java and common errors for all driver languages. For more information, see Best Practices and Common Errors.</td>
<td>January 2, 2020</td>
</tr>
<tr>
<td>New Regions launch (p. 405)</td>
<td>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.</td>
<td>November 19, 2019</td>
</tr>
<tr>
<td>Amazon QLDB Driver for Node.js (p. 405)</td>
<td>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).</td>
<td>November 13, 2019</td>
</tr>
<tr>
<td>Amazon QLDB Driver for Python (p. 405)</td>
<td>Amazon QLDB now provides a preview release of the Python driver. This driver enables you to use the AWS SDK for Python (Boto 3) to interact with QLDB's transactional data API. For more information, see Amazon QLDB Driver for Python (Preview).</td>
<td>October 29, 2019</td>
</tr>
<tr>
<td>Public release (p. 405)</td>
<td>This is the initial public release of Amazon QLDB. This release includes the Developer Guide and the integrated ledger management API Reference.</td>
<td>September 10, 2019</td>
</tr>
</tbody>
</table>