A QLDB Cheat Sheet for MySQL Users

Percona Live May 2021
Overview

● What is QLDB?
  ○ Features & Capabilities
● Why consider QLDB?
● MySQL - QLDB Syntax comparison
● Example migration/implemention
● Observations for general use

About Myself

- 20+ years MySQL experience in architecture and operations
- 10+ years AWS experience
- 15 years conference speaking
- Published author of 4 MySQL books

- Lead Data Architect/Engineer at Lifion by ADP

[http://ronaldbradford.com](http://ronaldbradford.com)
What is QLDB?

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

https://aws.amazon.com/qldb/
What is QLDB?

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

https://aws.amazon.com/qldb/
What is a Ledger?

- A history of financial transactions
- A verification that each line and column add up individually and collectively to the final balance
- The final balance matches an external reference (e.g. your bank account)
<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
<th>Balance</th>
<th>Category A</th>
<th>Category B</th>
<th>Category C</th>
<th>Category D</th>
<th>DR</th>
<th>CR</th>
<th>Income X</th>
<th>Income Y</th>
<th>Income Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/20</td>
<td>Opening Account</td>
<td>100</td>
<td></td>
<td>(1)</td>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>1/2/20</td>
<td></td>
<td>99</td>
<td>(1)</td>
<td>(2)</td>
<td></td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/3/20</td>
<td></td>
<td>97</td>
<td></td>
<td>(2)</td>
<td>(3)</td>
<td>(3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/4/20</td>
<td></td>
<td>94</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/5/20</td>
<td></td>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1/6/20</td>
<td></td>
<td>93</td>
<td></td>
<td>(5)</td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/7/20</td>
<td></td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/8/20</td>
<td></td>
<td>92</td>
<td></td>
<td>(7)</td>
<td>(7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/9/20</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/10/20</td>
<td></td>
<td>91</td>
<td></td>
<td></td>
<td>(9)</td>
<td>(9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/11/20</td>
<td></td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/12/20</td>
<td>Current Balance</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All lines & columns sum to external reference (i.e. Bank Account)
A single missing categorization (causes a totalling mismatch)
<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
<th>Balance</th>
<th>Category A</th>
<th>Category B</th>
<th>Category C</th>
<th>Category D</th>
<th>CR</th>
<th>Income X</th>
<th>Income Y</th>
<th>Income Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/20</td>
<td>Opening Account</td>
<td>100</td>
<td>(1)</td>
<td>(1)</td>
<td></td>
<td></td>
<td>100</td>
<td>100</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1/2/20</td>
<td></td>
<td>99</td>
<td>(2)</td>
<td></td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/3/20</td>
<td></td>
<td>97</td>
<td></td>
<td>(3)</td>
<td></td>
<td>(3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/4/20</td>
<td></td>
<td>94</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/5/20</td>
<td></td>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/6/20</td>
<td></td>
<td>93</td>
<td>(5)</td>
<td></td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/7/20</td>
<td></td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/8/20</td>
<td></td>
<td>92</td>
<td>(7)</td>
<td></td>
<td>(7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/9/20</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/10/20</td>
<td></td>
<td>91</td>
<td></td>
<td>(9)</td>
<td></td>
<td>(9)</td>
<td>(9)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/11/20</td>
<td></td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/12/20</td>
<td>Current Balance</td>
<td>101</td>
<td>(8)</td>
<td>(7)</td>
<td>(3)</td>
<td>(9)</td>
<td>(27)</td>
<td>128</td>
<td>100</td>
<td>16</td>
</tr>
<tr>
<td>Difference</td>
<td></td>
<td>0</td>
<td>(27)</td>
<td>0</td>
<td>4</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An incorrect column sum (causes mismatch)
<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
<th>Balance</th>
<th>Category A</th>
<th>Category B</th>
<th>Category C</th>
<th>Category D</th>
<th>DR</th>
<th>CR</th>
<th>Income X</th>
<th>Income Y</th>
<th>Income Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/20</td>
<td>Opening Account</td>
<td>100</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2/20</td>
<td></td>
<td>99</td>
<td>(1)</td>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/3/20</td>
<td></td>
<td>97</td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/4/20</td>
<td></td>
<td>94</td>
<td></td>
<td>(3)</td>
<td></td>
<td></td>
<td></td>
<td>(3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/5/20</td>
<td></td>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/6/20</td>
<td></td>
<td>93</td>
<td></td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/7/20</td>
<td></td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/8/20</td>
<td></td>
<td>92</td>
<td>(7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/9/20</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/10/20</td>
<td></td>
<td>91</td>
<td></td>
<td>(9)</td>
<td></td>
<td></td>
<td></td>
<td>(9)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/11/20</td>
<td></td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/12/20</td>
<td>Current Balance</td>
<td>99</td>
<td></td>
<td></td>
<td>(8)</td>
<td>(7)</td>
<td>(3)</td>
<td>(9)</td>
<td>128</td>
<td>104</td>
<td>16</td>
</tr>
</tbody>
</table>

**Difference**
2

Ledger total does not match external source (i.e. bank account balance)
QLDB Key Features

Key features

- Immutable
- Verifiable
- Supports SQL notation
- Supports smarter datatypes, object changes on the fly

https://aws.amazon.com/qldb/
Why consider a ledger database?

Limitations of RDBMS implementations

- No standard approach
- INSERT/UPDATE/DELETE trigger
  - Inserts data into audit table
  - Must use same structure
  - Additional audit information (who, when, what)
- Not a distributed transaction
- Auditing can be altered / disabled
- Code could miss new data (columns)
SQL Notation in QLDB

Subset of PartiQL

- Open source implementation
- Backward compatible with SQL-92
- Supports full nested data
- At V0.1.5-alpha

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.html
https://partiql.org/
PartiQL 101 Experiment

$ brew install openjdk
$ wget https://github.com/partiql/partiql-lang-kotlin/releases/download/v0.2.6-alpha/partiql-cli-0.2.6.tgz
$ tar xvfz partiql-cli-0.2.6.tgz
$ java --version
$ cd partiql-cli-0.2.6
$ bin/partiql -e Tutorial/code/q1.env
PartiQL> select * from hr.employees where id=4;
PartiQL> ^D

# client only, does not provide QLDB operations
PartiQL> CREATE TABLE example;
org.partiql.lang.eval.EvaluationException: DDL operations are not supported yet
   Evaluator Error: at line 1, column 1: Feature 'DDL Operations' not supported yet
Amazon Ion

- Rich text
- Hierarchical format
- Superset of JSON
- Supports binary representation
  - storage/transmission/skip scan

[https://amzn.github.io/ion-docs/](https://amzn.github.io/ion-docs/)
[https://github.com/amzn/ion-js](https://github.com/amzn/ion-js)
Ion 101 Experiment

$ npm install --save ion-js jsbi
$ cat example.js
let ion = require("ion-js");

// Reading
let ionData = '{ greeting: "Hello", name: "Ion" }';
let value = ion.load(ionData);
console.log(value.greeting + "", " + value.name + "!");

// Writing
let ionText = ion.dumpText(value);
console.log("Serialized Ion: " + ionText);

$ node example.js
Hello, Ion!
Serialized Ion: {greeting:"Hello",name:"Ion"}
First Observations of QLDB

- Slow (serverless)
- No uniqueness constraints (e.g. primary key)
- No bulk operations
- Limited client tooling
- Lacks strict syntax checking
- Lift & Shift capable (but stifles innovation)
- **Provides exactly the auditing you want**
Syntax Comparison

SELECT, INSERT, UPDATE, DELETE
CREATE, DROP <and more>
AWS Setup

# https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
$ unzip awscliv2.zip
$ sudo ./aws/install
$ export PATH=/usr/local/bin:$PATH
$ aws --version
# aws-cli/2.1.39 Python/3.9.4 Darwin/19.6.0 source/x86_64 prompt/off

# https://github.com/awslabs/amazon-qldb-shell
$ pip install virtualenv
$ virtualenv qldbshell
$ source qldbshell/bin/activate
$ pip install qldbshell
$ qldbshell

https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html
https://github.com/awslabs/amazon-qldb-shell
AWS IAM Policy Requirements

- AmazonQLDBConsoleFullAccess
- AmazonQLDBFullAccess
- AmazonQLDBReadOnly

Later reference
Create Ledger (QLDB) - Schema/Database (RDBMS)

$ aws qldb
$ aws qldb list-ledgers
$ aws qldb create-ledger --name demo --permissions-mode ALLOW_ALL
$ aws qldb describe-ledger --name demo
# State: CREATING -> ACTIVE

$ MYSQL_ROOT_PASSWD="$(date | md5sum | cut -c1-20)#"; echo $MYSQL_ROOT_PASSWD
$ docker run --name mysql-qldb-demo -e MYSQL_ROOT_PASSWORD=$MYSQL_ROOT_PASSWD -d mysql:8.0
$ docker exec -it mysql-qldb-demo mysql -uroot -p$MYSQL_ROOT_PASSWD
mysql> CREATE SCHEMA IF NOT EXISTS demo;
mysql> SHOW SCHEMAS;
mysql> SHOW CREATE SCHEMA demo;

QLDB Output (create-ledger)

$ aws qldb create-ledger --name demo --tags JIRA=DEMO-5826,Owner=RonaldBradford --permissions-mode ALLOW_ALL --no-deletion-protection
{
  "Name": "demo",
  "Arn": "arn:aws:qldb:us-east-1:999995630525:ledger/demo",
  "State": "CREATING",
  "CreationDateTime": "2021-03-06T22:46:41.760000+00:00",
  "DeletionProtection": false
}

$ aws qldb list-ledgers
{
  "Ledgers": [  
    {
      "Name": "demo",
      "State": "ACTIVE",
      "CreationDateTime": "2021-03-06T22:46:41.760000+00:00"
    }  
  ]
}

$ aws qldb describe-ledger --name demo
{
  "Name": "demo",
  "Arn": "arn:aws:qldb:us-east-1:999995630525:ledger/demo",
  "State": "ACTIVE",
  "CreationDateTime": "2021-03-06T22:46:41.760000+00:00",
  "DeletionProtection": false
}
Create Table

$ qldbshell --ledger demo
qldbshell > CREATE TABLE example
INFO:
{  
  tableId: "JuvoaxNSWT00uiZzGy8Gqw"
}

mysql> USE demo
mysql> CREATE TABLE example (  
    pkid INT UNSIGNED NOT NULL,
    name VARCHAR(100) NOT NULL,
    address_line1 VARCHAR(100) NOT NULL,
    address_line2 VARCHAR(100) NULL,
    city VARCHAR(100) NOT NULL,
    state_province VARCHAR(5) NULL,
    country CHAR(2) NOT NULL,
    PRIMARY KEY(pkid));

mysql> CREATE TABLE EXAMPLE(pkid INT);
Query OK, 0 rows affected (0.02 sec)

$ qldbshell --ledger demo
qldbshell > CREATE INDEX ON example(pkid);
INFO:
{  tableId: "JuvoaxNSWT00uiZzGy8Gqw"
}
qldbshell > CREATE INDEX ON example(name);
INFO:
{  tableId: "JuvoaxNSWT00uiZzGy8Gqw"
}

mysql> CREATE INDEX example_name ON example(name);
# or
mysql> ALTER TABLE example ADD INDEX (name);

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.create-index.html
Information_schema Metadata

qldbshell > SELECT * FROM information_schema.user_tables;

INFO:
{
  tableId: "JuvoaxNSWT00uiZzGy8Gqw",
  name: "example",
  indexes: [
    {
      expr: "[pkid]",
      indexId: "7iwC5k6bSWH8lMB1xdVbuN",
      status: "ONLINE"
    },
    ...
  ]
}

mysql> SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA=DATABASE();

#or
mysql> SELECT * FROM information_schema.tables WHERE table_schema=DATABASE();

https://docs.aws.amazon.com/qldb/latest/developerguide/working.catalog.html
Insert

qldbshell > INSERT INTO example VALUE {
  'pkid': 1,
  'name': 'Mickey Mouse',
  'address': {
    'line1': '123 Shoe Rd',
    'city': 'Mouseville',
    'state_province': 'ACME',
    'country': 'US'
  }
}
{
  documentId: "GZKrFFAAEWf9DER6GG51zU"
}

mysql> INSERT INTO example (pkid, name, address_line1, address_line2, city, state_province, country)
VALUES (1, 'Mickey Mouse', '123 Shoe Rd', NULL, 'Mouseville', 'ACME', 'US');

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.select.html
### Select

QLDB shell: 
```python
qlbshell > SELECT * FROM example;
{
  pkid: 1,
  name: "Mickey Mouse",
  address: {
    line1: "123 Shoe Rd",
    city: "Mouseville",
    state_province: "ACME",
    country: "US"
  }
}
...```

MySQL: 
```sql
mysql> SELECT * FROM example;
+------+--------------+---------------+---------------+------------+----------------+---------+
| pkid | name          | address_line1 | address_line2 | city       | state_province | country |
+------|---------------|---------------|---------------|------------|----------------+---------+
|  1   | Mickey Mouse  | 123 Shoe Rd   | NULL          | Mouseville | ACME           | US      |
+------|---------------|---------------|---------------|------------|----------------+---------+
```

Other Select Syntax

- Only inner joins are supported
- Nested queries (subqueries) - Limited
- Some SQL functions
  - [https://docs.aws.amazon.com/qldb/latest/developerguide/ql-functions.html](https://docs.aws.amazon.com/qldb/latest/developerguide/ql-functions.html)
- Limited observation of poor queries
  - PartiQL statement statistics
    - [https://docs.aws.amazon.com/qldb/latest/developerguide/working.statement-stats.html](https://docs.aws.amazon.com/qldb/latest/developerguide/working.statement-stats.html)
Update

```sql
qldbshell > UPDATE example SET name = 'Mickey A Mouse Sr' where pkid=1;
qldbshell > SELECT * FROM example;
{
  pkid: 1,
  address: {
    line1: "123 Shoe Rd",
    country: "US",
    city: "Mouseville",
    state_province: "ACME"
  },
  name: "Mickey A Mouse Sr"
}
```

```sql
qldbshell > UPDATE example SET suffix = 'Sr', name = 'Mickey A Mouse' where pkid=1;
qldbshell > SELECT * FROM example;
{
  pkid: 1,
  address: {
    line1: "123 Shoe Rd",
    country: "US",
    city: "Mouseville",
    state_province: "ACME"
  },
  suffix: "Sr",
  name: "Mickey A Mouse"
}
```

DELETE FROM example WHERE name='Minnie Mouse';
INFO:
{
  documentId: "JJfewUn1SWG0A0bsXkGVA2"
}

# No rows deleted gives no response
qldbshell > DELETE FROM example WHERE name='Minnie Mouse';
INFO:
INFO: (0.4258s)

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.delete.html
Insert - Handling Duplicate Constraints

qldbshell > INSERT INTO example VALUE { 'pkid' : 1, 'name' : 'Mickey Mouse', 'address' : { 'line1' : '123 Shoe Rd', 'city' : 'Mouseville', 'state_province' : 'ACME', 'country' : 'US'} }
   { documentId: "GZKrFFAAEWf9DER6GG51zU" }
qldbshell > INSERT INTO example VALUE { 'pkid' : 1, 'name' : 'Minnie Mouse', 'address' : { 'line1' : '123 Shoe Rd', 'city' : 'Mouseville', 'state_province' : 'ACME', 'country' : 'US'} }
   { documentId: "JJfewUnlSWG0AObsXkGVA2" }

mysql> INSERT INTO example (pkid, name, address_line1, address_line2, city, state_province, country)
   VALUES (1, 'Mickey Mouse', '123 Shoe Rd', NULL, 'Mouseville', 'ACME', 'US');
mysql> INSERT INTO example (pkid, name, address_line1, address_line2, city, state_province, country)
   VALUES (1, 'Minnie Mouse', '123 Shoe Rd', NULL, 'Mouseville', 'ACME', 'US');
ERROR 1062 (23000): Duplicate entry '1' for key 'example.PRIMARY'
Insert Verification (Select)

```
qldbshell > SELECT * FROM example;
{
    pkid: 1,
    name: "Minnie Mouse",
    address: {
        line1: "123 Shoe Rd",
        city: "Mouseville",
        state_province: "ACME",
        country: "US"
    }
},
...
```

```
mysql> SELECT * FROM example;
+-----+-------+---------------+---------------+------------+----------------+---------+
| pkid | name  | address_line1 | address_line2 | city       | state_province | country |
+-----+-------+---------------+---------------+------------+----------------+---------+
| 1   | Mickey Mouse | 123 Shoe Rd   | NULL          | Mouseville | ACME           | US      |
+-----+-------+---------------+---------------+------------+----------------+---------+
```

*No uniqueness at data level*
Insert Examples

qldbshell > INSERT INTO example VALUE { 'pkid' : 1, 'name' : 'Mickey Mouse', 'address' : { 'line1': '123 Shoe Rd', 'city' : 'Mouseville', 'state_province': 'ACME', country: 'US'}}

<Ledger:demo>

WARNING: Error while executing query: An error occurred (BadRequestException) when calling the SendCommand operation: Semantic Error: at line 1, column 153: No such variable named 'country'; No such variable named 'country'

# multi-row
Transactions

qldbshell > START TRANSACTION
Error in query: A PartiQL statement was used before a transaction was started
qldbshell > START
qldbshell(tx: 4ilTEihLfIRLa8rU6zK3c9) > UPDATE example SET name='xxx' WHERE pkid=1;
...
qldbshell(tx: 4ilTEihLfIRLa8rU6zK3c9) > ABORT
INFO: Transaction with transaction id 4ilTEihLfIRLa8rU6zK3c9 aborted
INFO: (0.0418s)

qldbshell > START
qldbshell(tx: 7ISCTGjk3L0FpHzKkMzk3y) > UPDATE example SET name='Mickey A Mouse' WHERE pkid=1;
...
qldbshell(tx: 7ISCTGjk3L0FpHzKkMzk3y) > COMMIT
INFO: Transaction with transaction id 7ISCTGjk3L0FpHzKkMzk3y committed
INFO: (0.0675s)

mysql> START TRANSACTION;
mysql> SAVEPOINT;
mysql> ROLLBACK;
mysql> COMMIT;

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.select.html
Transaction timeouts

qladbshell > START
INFO: (0.1469s)
qladbshell(tx: 4NTN9JUToVTDE92tAR0nIl) > UPDATE example SET name='xxx' WHERE pkid=1;
<Ledger:demo>
INFO: Query: UPDATE example SET name='xxx' WHERE pkid=1
INFO: Transaction with transaction id 4NTN9JUToVTDE92tAR0nIl aborted
Transaction expired.
INFO: (0.0911s)

https://docs.aws.amazon.com/qlldb/latest/developerguide/limits.html#limits.fixed
DROP/UNDROP table

qldbshell > DROP TABLE dropme

qldbshell > select * from information_schema.user_tables;
...  
  status: "INACTIVE"

qldbshell > select tableId from information_schema.user_tables where name='dropme';
tableId: "3Lb4t5pATh4971cvm5ggegB"

qldbshell > undrop table "3Lb4t5pATh4971cvm5ggegB";

Undrop requires tableId in quotes, not the tablename

mysql> DROP TABLE

FROM (Insert/Remove/Set)

qldbshell > select * from example;
{
  pkid: 1,
  address: {
    line1: "123 Shoe Rd",
    country: "US",
    city: "Mouseville",
    state_province: "ACME Inc"
  },
  name: "Mickey A Mouse",
  city: null,
  _state_province: null
}
INFO: (0.4630s)
qldbshell > FROM example AS e WHERE e.pkid=1 REMOVE e.city
qldbshell > FROM example AS e WHERE e.pkid=1 REMOVE e.state_province;

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.from.html
Syntax Summary

- CREATE SCHEMA|DATABASE
- CREATE TABLE
- CREATE INDEX
- INSERT
- UPDATE
- DELETE
- SELECT
- START TRANSACTION/ROLLBACK/COMMIT
- DROP TABLE

Differences

- CREATE LEDGER
- FROM [INSERT|REMOVE|SET]
- START/ABORT/COMMIT
- UNDROP TABLE
QLDB Audit Functionality
Select `<tablename>`

```ql
qldbshell > select * from example where pkid=1;
{
    pkid: 1,
    address: {
        line1: "123 Shoe Rd",
        country: "US",
        city: "Mouseville",
        state_province: "ACME"
    },
    name: "Mickey Mouse"
}
```
Select `_ql_committed_<tablename>`

```sql
cpyldbshell > select * from example where pkid=1;
qldbshell > select * from `_ql_committed_`example where data.pkid=1;
{
  blockAddress: {
    strandId: "HwVFArvqn1uKvQAhhkWcBk",
    sequenceNo: 63
  },
  hash: {{JAl800mIsIjZhV6HK1JNg6o4Exp++PdwGfcFYy1z+qU=}},
  data: {
    pkid: 1,
    address: {
      line1: "123 Shoe Rd",
      country: "US",
      city: "Mouseville",
      state_province: "ACME"
    },
    name: "Mickey Mouse"
  },
  metadata: {
    id: "GZKrFFAAEWf9DER6GG51zU",
    version: 6,
    txTime: 2021-05-06T18:49:45.035Z,
    txId: "DdM1VI6LEn00Pq0xekSBeh"
  }
}
qldbshell > select * from example where pkid=1;
{
  pkid: 1,
  address: {
    line1: "123 Shoe Rd",
    country: "US",
    city: "Mouseville",
    state_province: "ACME"
  },
  name: "Mickey Mouse"
}
```

[https://docs.aws.amazon.com/qldb/latest/developerguide/working.history.html](https://docs.aws.amazon.com/qldb/latest/developerguide/working.history.html)
Select By

qlbshell > SELECT __id, e.pkid, e.name FROM example AS e BY __id WHERE pkid = 1;
INFO:
{
    __id: "GZKrFFAAEWf9DER6GG51zU",
    pkid: 1,
    name: "Mickey Mouse"
}

qlbshell > select metadata.id AS __id FROM _ql_committed_example where data.pkid=1;
<Ledger:demo>
INFO:
{
    __id: "GZKrFFAAEWf9DER6GG51zU"
}

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.select.html
https://docs.aws.amazon.com/qldb/latest/developerguide/working.unique-id.html
Select history(<table>, [<from>, <to>])

qldbshell> SELECT * FROM history(example) AS h
     WHERE h.metadata.id = 'GZKrFFAAEWf9DER6GG51zU'

{ blockAddress: {
    strandId: "HwVFarvqn1uKvQAhkkWcBk",
    sequenceNo: 7
},
hash: {{JXJ1RW2B2SAuYOk0wSay1cMWUY5cZHT6irdEziNMri=}},
data: {
    pkid: 1,
    name: "Mickey Mouse",
    address: {
        line1: "123 Shoe Rd",
        city: "Mouseville",
        state_province: "ACME",
        country: "US"
    }
},
metadata: {
    id: "GZKrFFAAEWf9DER6GG51zU",
    version: 0,
    txTime: 2021-05-06T16:03:44.740Z,
    txId: "Dp04XzWevtB1IUOCJnXMcv"
}
,...

https://docs.aws.amazon.com/qldb/latest/developerguide/working.history.html
Select history(<table>, [<from>, <to>])

```sql
qldbshell >
SELECT COUNT(*) AS cnt,
       MIN(metadata.version) AS min_version,
       MAX(metadata.version) as max_version
FROM history(example) h
WHERE h.metadata.id = 'GZKrFFAAEWf9DER6GG51zU'
INFO:
{
  cnt: 7,
  min_version: 0,
  max_version: 6
}
qldbshell > SELECT COUNT(*) AS cnt, MIN(metadata.version) AS min_version, MAX(metadata.version) as max_version, min(metadata.txTime) FROM history(example) h
WARNING: Error while executing query: An error occurred (BadRequestException) when calling the SendCommand operation: <UNKNOWN>: at line <UNKNOWN>, column <UNKNOWN>: <UNKNOWN>; Expected number: 2021-05-06T16:03:44.740Z
```

Ledger: demo
History Transaction Id

qldbshell(tx: 7ISCTGjk3L0FpHzKkMzk3y) > COMMIT
INFO: Transaction with transaction id 7ISCTGjk3L0FpHzKkMzk3y committed

qldbshell > select * from history(example) where metadata.txId=7ISCTGjk3L0FpHzKkMzk3y;

{  
  blockAddress: {
    strandId: "HwVFarvqn1uKvQAhkkWcBk",
    sequenceNo: 43
  },
  hash: {{sSU12hM1Bq5acEvQ8SQeTNFXcvaUUbj7bCNVgIFrcEo=}},
  data: {
    pkid: 1,
    ...
  }
  metadata: {
    id: "GZKrFFAAEWF9DER6GG51zU",
    version: 2,
    txTime: 2021-05-06T17:37:20.028Z,
    txId: "7ISCTGjk3L0FpHzKkMzk3y"
  }
}
Demonstrating History change

```sql
qldbshell > UPDATE example
    SET city='Mouseville', state_province = 'ACME Inc'
    WHERE pkid=1;
{
    documentId: "GZKrFFAAEWF9DER6GG51zU"
},
{
    documentId: "GZKrFFAAEWF9DER6GG51zU"
}
```

- No schema validation to indicate wrong columns (e.g. bad code)
History change (not expected results)

```sql
qldbshell > select data.address.city, data.address.state_province from history(example) where data.pkid=1;
{
   city: "Mouseville",
   state_province: "ACME"
},
{
   city: "ACME Inc",
   state_province: "ACME"
},
{
   city: "ACME Inc",
   state_province: "ACME"
},
{
   city: "ACME Inc",
   state_province: "ACME"
},
{
   city: "Mouseville",
   state_province: "ACME"
}
```

Change was not reflected???
Observation - Lack of column validation

- **Originally**
  - address.city, address.state_province

- **Updated**
  - city, state_province

- **Subsequently**
  - NULL values or remove?

```sql
UPDATE example SET city=NULL, state_province = NULL, address.city = 'Mouseville',
address.state_province = 'ACME Inc' WHERE pkid=1;
```

```sql
FROM example AS e WHERE e.pkid=1 REMOVE e.city
FROM example AS e WHERE e.pkid=1 REMOVE e.state_province;
```
History change (which column name?)

```sql
qlbshell > select data.address.city, data.address.state_province, data.city, data.state_province from history(example) where data.pkid=1;
...
{
  city: "ACME Inc",
  city: "Mouseville",
  state_province: "ACME",
  state_province: "ACME Inc"
},
{
  city: "Mouseville",
  state_province: "ACME"
}
```

Result of Bad SQL

```sql
select data.address.city, data.address.state_province, data.city AS bad_city,
h.data.state_province AS bad_state from history(example) h where h.data.pkid=1;
{
  city: "ACME Inc",
  state_province: "ACME"
},
{
  city: "ACME Inc",
  state_province: "ACME",
  bad_city: "Mouseville",
  bad_state: "ACME Inc"
},
{
  city: "Mouseville",
  state_province: "ACME Inc",
  bad_city: null,
  bad_state: null
},
```
QLDB Audit Summary

- SELECT <tablename>
- SELECT _ql_committed_<tablename>
- SELECT BY
- SELECT history(<tablename>[<from>,<to>])
Syntax Observations
(a.k.a. Migration Complexities)
Observations - Syntax Checking

qldbshell > select pkid, notacolumn from example
where pkid=1;
INFO:
{
  pkid: 1
}
INFO: (0.2160s)

mysql> SELECT  pkid, noacolumn from example where pkid=1;
ERROR 1054 (42S22): Unknown column 'noacolumn' in 'field list'
Observations - Implied Type Conversion

```
qldbshell > select pkid, notacolumn from example
where pkid='1';
INFO: (0.2879s)
```

```
mysql> SELECT  pkid from example where pkid=1;
+------+
| pkid |
+------+
|    1 |
+------+
1 row in set (0.00 sec)
```

```
qldbshell > select pkid, notacolumn from example
where pkid='1';
```
Observations - Bulk Delete

qldbshell > delete from example;                         <Ledger:demo>
WARNING: Error while executing query: An error occurred
(BadRequestException) when calling the SendCommand operation: Exceeded
document transaction limit: More than 40 documents were modified within
transaction
Observations - Quotes

qldbshell > select address from example where name = 'Mickey Mouse';
{
  address: {
    line1: "123 Shoe Rd",
    city: "Mouseville",
    state_province: "ACME",
    country: "US"
  }
}
INFO: (0.3062s)
qldbshell > select address from example where name = "Mickey Mouse";
INFO: (0.2086s)
Observations - Column Aliasing but Reserved Word?

qldbshell > SELECT COUNT(*) FROM history(example) where data.name = 'Mickey Mouse';
{
  _1: 1
}
qldbshell > SELECT COUNT(*) AS rows FROM history(example) where data.name = 'Mickey Mouse';
WARNING: Error while executing query: An error occurred (BadRequestException) when calling the SendCommand operation: Parser Error: at line 1, column 20: expected identifier for alias, KEYWORD : rows; Expected identifier for AS-alias
INFO: (0.2664s)
qldbshell > SELECT COUNT(*) AS cnt FROM history(example) where data.name = 'Mickey Mouse';
{
  cnt: 2
}

https://docs.aws.amazon.com/qldb/latest/developerguide/ql-reference.reserved.html
Observations

- No Case sensitive objects - *Good*
- No implied type conversion supported - *Good*
- No syntax on column names - *Not so Good*
- Column aliases but unknown reserved words? - *Awkward*
- Unable to bulk delete - Requires programming loop - *Not Good*
- Shell/Syntax, e.g. semicolon ;-) *Annoying - Maturity*
- Quotes, single (') v double (") - *Good, but code changes*
  - Annoying to cut/paste output in double quotes to value in single quotes
- Query performance (lack of indexes) e.g. using history()
- There is no "who" made a change?
- Error messages can be frustrating
  - Misspelled table is - No such variable named 'exmaple3'
POC Migration of an existing Application
POC Criteria

- Identify a system/sub-system that wasn't 100s/1000s of tables
- Identify a system that would benefit from auditing
  - E.g. not high write once type data (e.g. login records)
- Determine how to perform data ingestion/migration
- What is involved for the application to use QLDB?
- Identify feature gaps
- Determine if feasible to continue to phase 2
RDBMS Example

- Primary Table
- Audit table (general purpose, i.e. supports multiple tables)
- Table for INSERT state of primary table
- Table for UPDATE state of primary table
- Table for DELETE state of primary table
- Secondary table & intersection table
  - Tables of INSERT/UPDATE/DELETE state of secondary table
Example Code

```
MYSQL_ROOT_PASSWD=$(date | md5sum | cut -c1-20)"#"; echo ${MYSQL_ROOT_PASSWD}

docker run --name mysql-qldb-migration -e MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWD} -d mysql/mysql-server:latest

docker logs mysql-qldb-migration

docker exec -i mysql-qldb-migration mysql -uroot -p${MYSQL_ROOT_PASSWD} < schema.sql

docker exec -i mysql-qldb-migration mysql -uroot -p${MYSQL_ROOT_PASSWD} demo < qldb-members-example.sql

docker exec -it mysql-qldb-migration mysql -uroot -p${MYSQL_ROOT_PASSWD} demo

docker exec -i mysql-qldb-migration mysql -uroot -p${MYSQL_ROOT_PASSWD} < migration-user.sql

docker exec -it mysql-qldb-migration /bin/bash

echo "SELECT * FROM members ORDER BY id" | mysqlsh --sql --result-format=json/array --uri=migration:qldb@localhost/demo > members.json

cat members.json
```

https://gist.github.com/ronaldbradford/c3ecbc720b63bd1f42b55055251f856
QLDB Optimizations

- Move "who" from the audit table to Primary Table
- Eliminated Audit Table (who, when, why)
  - Why requires programming? E.g. Insert, Update, Delete
    - Version: 0 - Insert
    - Version with no data element - Delete
    - Remaining Versions - Update
- Eliminated 3 DML state tables for primary table (now history)
- Move 'n' primary address columns and 'n' postal address to a single column
- Added child table (n rows) as nested column
  - Got all auditing for free
Data Mockup

- Python code to create tables/indexes
- Python code to insert data
- Python pyqldb not compatible with qldbshell
  - Multiple virtual environments necessary
- Bulk loading data
  - You can't TRUNCATE TABLE
  - Deleting large sets fails with single statement
  - There is no limit option

See Revision 1
https://gist.github.com/ronaldbradford/c3ecbcd720b63bd1f42b55055251f856
Data Migration

- QLDB data is easiest in native JSON format
  - Imported 5.7 data into MySQL 8
- Anonymize PII
- Docker mysql doesn't have mysqlsh needed to use mysql/mysql-server
- JSON dump via mysqlsh
  - Took a few goes for best result-format
  - Took a few goes for automated execution
  - user@localhost doesn't work, needed user@127.0.0.1
- Need a shared volume in container - Load and dump
Specific Command Syntax examples

Docker mysql/mysql-server not mysql

```
$ docker run --name mysql-qldb-migration -e MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWD} -d mysql/mysql-server:latest

$ docker exec -it mysql-qldb-migration /bin/bash

# The mysqlsh syntax of choice.

$ echo "SELECT * FROM members ORDER BY id" | mysqlsh --sql --result-format=json/array --uri=migration:qldb@localhost/demo > members.json
```
Trials of a first time user

$ virtualenv qldb
$ source qldb/bin/activate
$ pip install pyqldb==3.1.0

$ python qldb-poc-v1.py

# Can't create again, objects exist, DROP too complicated
$ python qldb-poc-v2.py #commented out creates

# Reran demo load uncovered there is no unique constraint, that is going to
# TRUNCATE TABLE is not an option either
$ python qldb-poc-v3.py # different table with .json file

Later reference
How do I load historical activity?

- You cannot inject history
- You cannot simulate a replay of history
  - txTime is actual time at insert
Load Testing

- Not Performed
- Retries & Optimization
  - [https://docs.aws.amazon.com/qldb/latest/developerguide/driver.best-practices.html](https://docs.aws.amazon.com/qldb/latest/developerguide/driver.best-practices.html)
- Some limits are fixed quotas
  - [https://docs.aws.amazon.com/qldb/latest/developerguide/limits.html#limits.fixed](https://docs.aws.amazon.com/qldb/latest/developerguide/limits.html#limits.fixed)
  - [https://docs.aws.amazon.com/qldb/latest/developerguide/driver-errors.html](https://docs.aws.amazon.com/qldb/latest/developerguide/driver-errors.html)
- QLDB Lab (code is TODO & and no contact details)
Future Architecture

- QLDB Streams
  - Push data back to a MySQL
  - Push data to other data stores

https://docs.aws.amazon.com/qldb/latest/developerguide/streams.html
Conclusion
Conclusion

- Immutable & verifiable auditing does exactly what you want
- Lift & shift application unlikely
  - SQL Syntax - Observations
  - Data Migration Complexities
- Lack of unique constraints a big factor
- Must be source of truth
  - Can stream out (e.g. Kinesis -> MySQL legacy)
  - No incoming
- No performance/volume testing evaluation
Thank You

https://perconalive.com