CDC on Statement Based Replication (SBR)

Venkat Morampudi, Staff Software Engineer
Agenda

/ Introduction to CDC
/ MySQL Replication Streams
/ Implementing CDC on SBR
/ CDC at Box
Introduction to CDC
Change Data Capture (CDC)

Change Data Capture is a design pattern that enables capturing changes to data and notifying actors so they can react accordingly.
Anatomy of a CDC event

• CDC event is composed of
  • Pre-mutation state of the row (before)
  • Post-mutation state of the row (after)
  • Metadata
    • Table
    • Primary key
    • Mutation type

```json
{
    "metadata": {
        "primary_key": 2,
        "table": "fruit",
        "mutation_type": "update",
        "timestamp": 1611179777
    },
    "before": {
        "id": 2,
        "name": "banana",
        "quantity": 950,
        "expiry_date": "2020-01-02"
    },
    "after": {
        "id": 2,
        "name": "banana",
        "quantity": 720,
        "expiry_date": "2020-01-02"
    }
}
```
Typical CDC Pipeline Architecture for MySQL

- **Application** (R/W)
- **MySQL Primary**
- **MySQL Replica**
- **MySQL Replica**
- **MySQL Replica**
- **CDC Service**
- **Message Bus**

**Binlog format set to ROW**

**RBR** connections between MySQL Primary and replicas.
Different Types of MySQL Replication Streams

- Row-based Replication (RBR)
- Statement-based Replication (SBR)
CDC on Statement Based Replication

Row-based Replication (RBR)

- The binary log stores the record-level changes that occur to database tables.
- State of the row before and after can be extracted from the Binlog event.
- Binlog event doesn't contain the table metadata, i.e. it does not record the field names, only field number.

```
UPDATE `super_market`.`fruit`
SET quantity= quantity + 300
WHERE name = 'pear'
```

```
### UPDATE `super_market`.`fruit`
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2= 'pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=273 /* INT meta=0 nullable=0 is_null=0 */
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=573 /* INT meta=0 nullable=0 is_null=0 */
# at 569
```
Statement-based Replication (SBR)

- The binary log stores the SQL statements used to change databases
- Binlog event doesn’t contain pre-mutation state nor post-mutation state

```
UPDATE `super_market`.`fruit`
SET quantity= quantity + 300
WHERE name = 'pear'
```
## RBR vs SBR

### Summary

<table>
<thead>
<tr>
<th>Requirements for CDC event</th>
<th>RBR</th>
<th>SBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>primary key</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>mutation type</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>before</td>
<td>✅</td>
<td>✗</td>
</tr>
<tr>
<td>after</td>
<td>✅</td>
<td>✗</td>
</tr>
</tbody>
</table>
Implementing CDC on SBR
How do we get the pre-mutation state
Our Solution

SQL Comments
MySQL Query Comments

- MySQL supports placing comments within SQL statements
- MySQL ignores comments when parsing SQL statements
- Comments are preserved in the binlog with statement-based replication

```sql
select name, quantity from fruit where id in (123, 456) /*
trace_id=8826724f58f5586a.8826724f58f5586a<:8826724f58f5586a
&application=webapp
&user_id=12576
*/
```
Mechanics of Appending Pre-mutation State to Comments

1. Lock and load the row
2. Encode row into portable format
3. Append encoded row to query comment
4. Execute query
Requirements for Serializer

- Fully typed
  - Schema support
  - Schema evolution support
- Compact output
  - Encoded binary data should be very compact, so it takes up less storage space
  - Separate the schema from the encoded data
- Very fast
Avro for Serialization

<table>
<thead>
<tr>
<th>Requirements for Serializer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Typed with Schema</td>
</tr>
<tr>
<td>Schema Evolution</td>
</tr>
<tr>
<td>Compact</td>
</tr>
<tr>
<td>Separate Schema from Data</td>
</tr>
<tr>
<td>Pretty Fast</td>
</tr>
</tbody>
</table>
Schema Store

- Avro schema of encoded pre is stored in Confluence Schema Registry
- Every schema stored in Schema Registry has a numeric id associated with it
- Schema Id is appended to Query comments
Sample Query Comment

/*
trace_id=8826724f58f5586a.8826724f58f5586a:<8826724f58f5586a
&application=webapp
&pre_mutation_snapshot_schema_id=1091
&pre_mutation_snapshot_binary=FQQoYXQlZF9sxS5rXXISMTXSMQXzMDQSMDECssKQ1EoCFQESMDISMzX0MDXxXhQmdS5QLXRlc3QCZQQoYXQlZF9sxS5rX2Zvl9CX1ZvdmFfR2FsY2hlbmtvQ3QfZmlsZV8xMDXyMDMyMDMzXSMQXzMDQகGFzc3dvcmRfZm9yX0IgVm92YSBHYSxQxGVux28QcyBmxSxlIDESMDISMzXzMDXxXgXSc2hhcmVkJ2xpbmSXhXyMDXyMDQgK0So3USgISMTXSMQXzMDQSMDCEmZ1bmMtdGVzdXQmc2hhcmVkJ2xpbmXfZm9yX0QfVm92YV9HYSxQxGVux28Qc19mxSxlXzESMDISMzXzMDXyXmBSYXQzd29yZF9mb3QfQizBSb3ZhIEdhbGQoZS5rbydzIGZpbGUgMTXSMQXzMDMDMDCXX== */
Caveats with comments

Inclusion of pre-mutation state in comments results in query size explosion

- May need to increase `max_allowed_packet`
- Our existing `max_allowed_packet` value is big enough to support our needs
- p99 of query size increase is less than 100KB for us

Binlog files may get too big

- Might affect binlog file retention period on disk
- Not any worse than Row-Based Binlogs
- Additional 6-8MB/sec data added to our binlogs across all the shards

May need to impose restrictions on the mutation cardinality

- Protects from appending very large pre-mutation binaries to comments
Recap

<table>
<thead>
<tr>
<th>Requirements for CDC event</th>
<th>SBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>✓</td>
</tr>
<tr>
<td>primary key</td>
<td>✗</td>
</tr>
<tr>
<td>mutation type</td>
<td>✓</td>
</tr>
<tr>
<td>before</td>
<td>✗</td>
</tr>
<tr>
<td>after</td>
<td>✗</td>
</tr>
</tbody>
</table>
Computing post-mutation state & primary key
Credence

Distributed Data Access Service at Box

• Provides a uniform way to interact with relational data at Box
• Primarily responsible for protecting MySQL
• Provides strongly opinionated APIs supporting limited set of data access patterns
Credence Architecture

Applications -> Credence -> MySQL Primaries -> MySQL Replicas

Credence Architecture
Mutation Queries Supported By Credence

- Multi-row inserts with explicit column values
- Conditional updates only by primary key
  - New column values are explicit
- Delete only by primary key

<table>
<thead>
<tr>
<th>Type</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td><code>INSERT INTO &quot;super_market&quot;.&quot;fruit&quot; (id, name, quantity, expiry_date) VALUES (1,'apple',100,'2021-01-01'), (2,'banana',950,'2021-01-02')</code></td>
</tr>
<tr>
<td>Update</td>
<td><code>UPDATE &quot;super_market&quot;.&quot;fruit&quot; SET quantity = CASE WHEN id = 1 THEN 88 WHEN id = 2 THEN 950 ELSE quantity END WHERE id IN (1,2)</code></td>
</tr>
<tr>
<td>Delete</td>
<td><code>DELETE FROM &quot;super_market&quot;.&quot;fruit&quot; WHERE id IN (1,2)</code></td>
</tr>
</tbody>
</table>
Queries Not Supported by Credence

Mutation queries with MySQL computed column values

- ex: update fruit set quantity = quantity + 100 where id = 2
- Clients who need to execute this kind of query would read the locked row to compute final values before executing mutation queries in a transaction

Delete/Update queries with unbounded where clause

- ex: delete fruit where expiry_date < 1609509729
- Clients who need to execute this kind of query would load the primary keys first using the read apis and issue delete queries using primary keys in a transaction
How to Compute Post-mutation State & PK

- Parse mutation queries to extract columns modified and their new values (diff) as well primary keys
- Post can be computed by merging pre-mutation state with diff within CDC pipeline
  - post = pre + diff
- No additional changes to query size
Compute Post-mutation State & PK for Insert

```sql
INSERT INTO super_market. fruit (id, name, quantity, expiry_date) 
VALUES 
( 1, 'apple', 100, '2021-01-01' )
/*
trace_id=8826724f58f5586a.8826724f58f5586a<:8826724f58f5586a
&application=webapp
*/
```
Compute Post-mutation State & PK for Insert

Raw diff

(id, name, quantity, expiry_date)
(1, 'apple', 100, '2021-01-01')

<table>
<thead>
<tr>
<th>Columns changed</th>
<th>New values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>apple</td>
</tr>
<tr>
<td>quantity</td>
<td>100</td>
</tr>
<tr>
<td>expiry_date</td>
<td>2021-01-01</td>
</tr>
</tbody>
</table>
Compute Post-mutation State & PK For Insert

<table>
<thead>
<tr>
<th>Columns changed</th>
<th>New values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>apple</td>
</tr>
<tr>
<td>quantity</td>
<td>100</td>
</tr>
<tr>
<td>expiry_date</td>
<td>2021-01-01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Columns changed</th>
<th>New values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>apple</td>
</tr>
<tr>
<td>quantity</td>
<td>100</td>
</tr>
<tr>
<td>expiry_date</td>
<td>2021-01-01</td>
</tr>
</tbody>
</table>
Compute Post-mutation State & PK for Update

```
UPDATE `super_market`. fruit
SET quantity = CASE WHEN id = 1 THEN 88 ELSE quantity END
WHERE id IN (1)

trace_id=8826724f58f5586a.8826724f58f5586a<:8826724f58f5586a
&application=webapp
&pre_mutation_snapshot_schema_id=1091
&pre_mutation_snapshot_binary=FQQoYXQlZF9sxS5rXXISMTXSMQXzMDQSMDECssKQ1EoCFQESMDISMz==
```
Compute Post-mutation State & PK For Update

```
quantity = CASE WHEN id = 1 THEN 88 ELSE quantity END
```

```
pre_mutation_snapshot_schema_id=1091&
pre_mutation_snapshot_binary=FQQoYXQl
ZF9sxS5rXXISMTXSMQXzMDQSMDECss
KQ1EoCFQESMDISMzX0MDxXhQmdS5
QLXRlc3QCZQQoYXQlZF9sxS5rX2Zvc19
CX1ZvdmF==
```
Compute Post-mutation State & PK for Update

<table>
<thead>
<tr>
<th>Columns changed</th>
<th>New values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>apple</td>
</tr>
<tr>
<td>quantity</td>
<td>100</td>
</tr>
<tr>
<td>expiry_date</td>
<td>2021-01-01</td>
</tr>
</tbody>
</table>

Columns changed

<table>
<thead>
<tr>
<th>New values</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantity</td>
</tr>
</tbody>
</table>

diff

<table>
<thead>
<tr>
<th>Columns changed</th>
<th>New values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>1</td>
</tr>
<tr>
<td>name</td>
<td>apple</td>
</tr>
<tr>
<td>quantity</td>
<td>88</td>
</tr>
<tr>
<td>expiry_date</td>
<td>2021-01-01</td>
</tr>
</tbody>
</table>

post
Downside of Parsing Queries

- Query parsing can get complicated and often error prone
- Binlog tailer implementation is tightly coupled with data access service that creates and executes queries
Techniques to Avoid Query Parsing

// Append diff to query comments

// Append post-mutation state to query comments

Both options results in further explosion of query and binlog file size
Implementing CDC on SBR - Recap

<table>
<thead>
<tr>
<th>Requirements for CDC event</th>
<th>SBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>✔</td>
</tr>
<tr>
<td>primary key</td>
<td>✗</td>
</tr>
<tr>
<td>mutation type</td>
<td>✔</td>
</tr>
<tr>
<td>before</td>
<td>✗</td>
</tr>
<tr>
<td>after</td>
<td>✗</td>
</tr>
</tbody>
</table>
CDC at Box
CDC at Box

Architecture

- Credence
- MySQL Primary
- Schema Registry
- MySQL Replica
- CDC Binlog Tailer
- Kafka
- Checkpoint Store

- Binlog format set to STATEMENT
- PGTID
- SBR
- R/W
Scale of CDC at Box

100s
MySQL Shards

10,000s
CDC events per sec

10s
CDC Consumers