

Implementation of Materialized Views in Cassandra

Alex Correia-Santos

Keywords:

Materialized Views Cassandra Key-Value Stores

1. Types of Views

1.1. Join View

```
CREATE JOIN <AB> OF <CF_A>.<COL> AND <CF_B>
```

Creates a Column Family (named <AB>) which stores the keys of the column family <CF_A> as columns for the key <COL>. Everytime a insertion is made into the base table <CF_A>, or a <COL> is updated, the view will be automatically updated to match the changes.

1.2. Index View

```
CREATE INDEX VIEW <A_v> OF <CF_A>.<COL_v>
```

Creates a copy of the column family <CF_A> with the values of the column <COL_v> acting as the key instead. As <COL_v> might have repeated values, the columns will be stored in a set-like fashion.

1.3. Non-key Join View

Such view can be obtained by creating an `Index View` then a `Join View`.

2. Views Properties

The following properties face different advantages and drawbacks, therefore they should be tested and changed in order to get the most efficient solutions.

- **Read-only**

Views cannot be changed directly by the user, the only functions that can be executed over a view are **GET** and **DROP**. (**pros:** guaranteed integrity of the view, **cons:** extra “check” to every function)

- **Data Duplication**

All the columns inserted into the base table will be automatically inserted into the related views as **index:column_name**. (**pros:** faster information retrieval, **cons:** slower data insertion and lower consistency)

- **Cascade Drop**

Whenever a base table is dropped, every view related to such base table shall be dropped aswell (**pros:** less space taken by (what might be) useless data, **cons:** slower drops)

3. Consistency Issues

- **Insert into base view ->Insert into materialized view**

Guarantees that materialized view will only be updated if base view is successfully updated. Can not guarantee that both views are going to be in the same state in case of a write fail in the materialized view.

4. Consistency Issues Solutions

- **Full Lock**

Only one write a time, guarantees a full consistency between views but will decrease the performance drastically and affect directly the goal of a key-value stores.

- **Lock field**

If possible, locking only a field can guarantee that both views are in the same state before allowing any other operation to be made on that field. It might decrease the database's performance.

- **Check** *timestamp* \rightarrow *Rollback*

In case that after writing into the materialized view, the base view fails to update the same information, the server may check the timestamp of the last update and rollback to the previous value if it was never updated after the failed update. (It might require a **Lock** when it fails)