3.3. Incrementally Updating a Hive Table Using Sqoop and an External Table

It is common to perform a one-time ingestion of data from an operational database to Hive and then require incremental updates periodically. Currently, Hive does not support SQL Merge for bulk merges from operational systems. Instead, you must perform periodic updates as described in this section.

Note

This procedure requires change data capture from the operational database that has a primary key and modified date field where you pulled the records from since the last update.

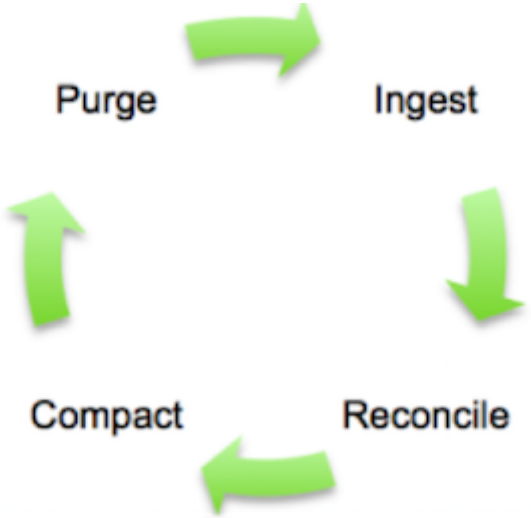
Overview

This procedure combines the techniques that are described in the sections "Moving Data from HDFS to Hive Using an External Table" and "Using Sqoop to Move Data into Hive."

Use the following steps to incrementally update Hive tables from operational database systems:

1. **Ingest:** Complete data movement from the operational database (base_table) followed by change or update of changed records only (incremental_table).
2. **Reconcile:** Create a single view of the base table and change records (reconcile_view) to reflect the latest record set.
3. **Compact:** Create a reporting table (reporting_table) from the reconciled view.
4. **Purge:** Replace the base table with the reporting table contents and delete any previously processed change records before the next data ingestion cycle, which is shown in the following diagram.

Figure 1.3. Data Ingestion Lifecycle



The base table is a Hive internal table, which was created during the first data ingestion. The incremental table is a Hive external table, which likely is created from .CSV data in HDFS. This external table contains the changes (INSERTs and UPDATEs) from the operational database since the last data ingestion.

Note

Generally, the table is partitioned and only the latest partition is updated, making this process more efficient.

Incrementally Updating Data in Hive

1. **Ingest the data.**

   a. Store the base table in the ORC format in Hive.

   The first time that data is ingested, you must import the entire table from the source database. You can use Sqoop. The following example shows importing data from Teradata:

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail
--connection-manager org.apache.sqoop.teradata.TeradataConnManager --username dbc
--password dbc --table SOURCE_TBL --target-dir /user/hive/base_table -m 1
```

   b. Store this data into an ORC-formatted table using the Steps 2 - 5 shown in "Moving Data from HDFS to Hive Using an External Table."

   The base table definition after moving it from the external table to a Hive-managed table looks like the below example:

```
CREATE TABLE base_table (
        id STRING,
        field1 STRING,
        modified_date DATE)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS ORC;
```

c. Store the incremental table as an external table in Hive.

It is more common to be importing incremental changes since the last time data was updated and then merging it. See the section "Using Sqoop to Move Data into Hive" for examples of importing data with Sqoop.

In the following example, `--check-column` is used to fetch records newer than `last_import_date`, which is the date of the last incremental data update:

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail
--connection-manager org.apache.sqoop.teradata.TeradataConnManager
--username dbc --password dbc --table SOURCE_TBL --target-dir /user/hive/incremental_table -m 1
--check-column modified_date --incremental lastmodified --last-value {last_import_date}
```

You can also use `--query` to perform the same operation:

```
sqoop import --connect jdbc:teradata://{host name}/Database=retail
--connection-manager org.apache.sqoop.teradata.TeradataConnManager --username dbc
--password dbc --target-dir /user/hive/incremental_table -m 1
--query 'select * from SOURCE_TBL where modified_date > {last_import_date} AND $CONDITIONS'
```

d. After the incremental table data is moved into HDFS using Sqoop, you can define an external Hive table over it with the following command:

```
CREATE EXTERNAL TABLE incremental_table (
        id STRING,
        field1 STRING,
        modified_date DATE)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    location '/user/hive/incremental_table';
```

2. **Reconcile or merge the data.**

Create a view that uses UNION ALL to merge the data and reconcile the base table records with the new records:

```
CREATE VIEW reconcile_view AS
SELECT t1.* FROM
    (SELECT * FROM base_table
     UNION ALL
     SELECT * from incremental_table) t1
JOIN
    (SELECT id, max(modified_date) max_modified FROM
        (SELECT * FROM base_table
         UNION ALL
         SELECT * from incremental_table)
     GROUP BY id) t2
ON t1.id = t2.id AND t1.modified_date = t2.max_modified;
```

EXAMPLES:

- **Figure 1.4. Dataset after the UNION ALL Command Is Run**

UNION: (SELECT * FROM base_table UNION SELECT * from incremental_table)

id,	*field1,*	*field2,*	*field3,*	*field4,*	*field5,*	*modified_date*	
1,	abcd,	efgh,	4,	7,	10.2,	2014-02-01 09:22:52	base_table
2,	abcde,	efgh,	5,	7,	10.2,	2014-02-01 09:22:52	
3,	abcde,	efgh,	6,	7,	10.2,	2014-02-01 09:22:52	
1,	abcdef,	efgh,	7,	7,	10.2,	2014-03-01 09:22:52	incremental_table
2,	abc,	efgh,	8,	7,	10.2,	2014-03-01 09:22:52	

- **Figure 1.5. Dataset in the View**

VIEW: reconcile_view

id,	*field1,*	*field2,*	*field3,*	*field4,*	*field5,*	*modified_date*
1,	abcdef,	efgh,	7,	7,	10.2,	2014-03-01 09:22:52
2,	abc,	efgh,	8,	7,	10.2,	2014-03-01 09:22:52
3,	abcde,	efgh,	6,	7,	10.2,	2014-02-01 09:22:52

Note

In the `reconcile_view` only one record exists per primary key, which is shown in the `id` column. The values displayed in the `id` column correspond to the latest modification date that is displayed in the `modified_date` column.

3. **Compact the data.**

   The view changes as soon as new data is introduced into the incremental table in HDFS (`/user/hive/incremental_table`, so create and store a copy of the view as a snapshot in time:

```
DROP TABLE reporting_table;
CREATE TABLE reporting_table AS
SELECT * FROM reconcile_view;
```

4. **Purge data.**

   a. After you have created a reporting table, clean up the incremental updates to ensure that the same data is not read twice:

```
hadoop fs -rm -r /user/hive/incremental_table/*
```

   b. Move the data into the ORC format as the base table. Frequently, this involves a partition rather than the entire table:

```
DROP TABLE base_table;
CREATE TABLE base_table (
        id STRING,
        field1 STRING,
        modified_date DATE)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
```

```
    STORED AS ORC;

INSERT OVERWRITE TABLE base_table SELECT * FROM reporting_table;
```

Handling Deletes

Deletes can be handled by adding a DELETE_DATE field in the tables:

```
CREATE VIEW reconcile_view AS
    SELECT t1.* FROM
        (SELECT * FROM base_table
         UNION
         SELECT * FROM incremental_table) t1
     JOIN
         (SELECT id, max(modified_date) max_modified FROM
             (SELECT * FROM base_table
              UNION
              SELECT * FROM incremental_table)
      GROUP BY id) t2
   ON t1.id = t2.id AND t1.modified_date = t2.max_modified
   AND t1.delete_date IS NULL;
```

Tip

You can automate the steps to incrementally update data in Hive by using Oozie. See "Using HDP for Workflow and Scheduling (Oozie)."