BLOG

Here you'll find everything you need to learn about digital software technology, development trends and beyond

INDUSTRIES

- ☐ Healthcare
- ☐ Financial Services
- ☐ Software Development
- ☐ Retail
- ☐ Energy, Oil & Gas
- ☐ Manufacturing
- ☐ Agriculture
- ☐ Other

SERVICES

- ☐ Security
- ☐ Data & Analytics
- ☐ IoT, XR, Robotics, AI & ML
- ☐ R&D
- ☐ Innovation Platform
- ☐ Quality Management

- [ ] AWS
- [ ] Google Cloud
- [ ] Salesforce

BY OLEKSANDR BERCHENKO

OCT 29, 2016

DATA & ANALYTICS SOFTWARE DEVELOPMENT

HOW TO PROCESS SLOWLY CHANGING DIMENSIONS IN HIVE

5 MIN READ

This article describes how to handle Slowly Changing Dimensions (SCD) in a data warehouse which uses Hive as a database.

Before reading on, you might want to refresh your knowledge of Slowly Changing Dimensions (SCD).

Let's imagine, we have a simple table in Hive:

```
CREATE TABLE dim_user (
  login VARCHAR(255), -- natural key
  premium_user BOOLEAN, -- SCD Type 2
  address VARCHAR(255), -- SCD Type 2
  phone VARCHAR(255), -- SCD Type 2, may be NULL
  name VARCHAR(255), -- SCD Type 1
  surname VARCHAR(255), -- SCD Type 1
  year_of_birth INT -- SCD Type 1, may be NULL
) STORED AS PARQUET;
```

Handling SCD Type 1 and SCD Type 2 may be trivial or at least well known in other

4. Anyway, UPDATE in ORC is too slow (update of each individual record requires its own MapReduce job).
5. There are only row level transactions (no BEGIN, COMMIT or ROLLB ACK statements).

Let's see how we can workaround all of them.

Suppose that "dim_user_production" is our existing table with current data. Its final schema (with surrogate keys and auxiliary fields) looks as follows:

```
CREATE TABLE dim_user_production (
 dim_user_id INT, -- surrogate key
  login VARCHAR(255), -- natural key
  premium_user BOOLEAN, -- SCD Type 2
  address VARCHAR(255), -- SCD Type 2
  phone VARCHAR(255), -- SCD Type 2, may be NULL
  name VARCHAR(255), -- SCD Type 1
  surname VARCHAR(255), -- SCD Type 1
  year_of_birth INT, -- SCD Type 1, may be NULL
  scd_version INT, -- historical version of the record (1 is the oldest)
  scd_start_date TIMESTAMP, -- start date and time
  scd_end_date TIMESTAMP, -- end date and time (9999-12-31 23:59:59 by d
  scd_active BOOLEAN, -- whether it's the latest version or not
) STORED AS PARQUET;
```

"dim_user_staging" is the table with new data to be processed. Its schema doesn't have surrogate keys or auxiliary fields and is identical to "dim_user" schema above.

1. Create a new table by copying the schema of the production table:
   ```
   DROP TABLE IF EXISTS dim_user_new;
   CREATE TABLE dim_user_new
   STORED AS PARQUET
   AS SELECT *
   FROM dim_user_production
   LIMIT 0;
   ```
2. Copy all the records from the production table that don't exist in the staging table:
   ```
   INSERT INTO TABLE dim_user_new
   ```

```
INSERT INTO TABLE dim_user_new
SELECT p .dim_user_id,
  p.login,
  p.premium_user,
  p.address,
  p.phone,
  s.name,
  s.surname,
  s.year_of_birth,
  p.scd_version,
  p.scd_start_date,
  p.scd_end_date,
  p.scd_active
FROM dim_user_production p
JOIN dim_user_staging s
ON p.login = s.login
AND p.scd_active = false;
```

4. Copy all the active records from the production table which don't have SCD Type 2 changes (apply SCD Type 1 changes if needed):

```
INSERT INTO TABLE dim_user_new
SELECT p.dim_user_id,
  p.login,
  p.premium_user,
  p.address,
  p.phone,
  s.name,
  s.surname,
  s.year_of_birth,
  p.scd_version,
  p.scd_start_date,
  p.scd_end_date,
  p.scd_active
FROM dim_user_production p
JOIN dim_user_staging s
ON p.login = s.login
```

```
INSERT INTO TABLE dim_user_new
SELECT p.dim_user_id,
  p.login,
  p.premium_user,
  p.address,
  p.phone,
  s.name,
  s.surname,
  s.year_of_birth,
  p.scd_version,
  p.scd_start_date,
  '2016-10-01 00:00:00', -- current timestamp for scd_end_date
  false -- false for scd_active
FROM dim_user_production p
JOIN dim_user_staging s
ON p.login = s.login
AND p.scd_active = true
WHERE p.premium_user != s.premium_user
OR p.address != s.address
OR COALESCE(p.phone, '') != COALESCE(s.phone, '');
```

6. Insert new active versions of records from the production table which have SCD Type 2 changes (apply SCD Type 1 changes if needed):

```
INSERT INTO TABLE dim_user_new
SELECT n.id + COALESCE(m.max_id, 0), -- new id for dim_user_id
  n.login,
  n.premium_user,
  n.address,
  n.phone,
  n.name,
  n.surname,
  n.year_of_birth,
  n.scd_version,
  '2016-10-01 00:00:00', -- current timestamp for scd_start_date
  '9999-12-31 23:59:59', -- default timestamp for scd_end_date
  true -- true for scd_active
```

```
      s.year_of_birth,
      p.scd_version + 1 AS scd_version
    FROM dim_user_production p
    JOIN dim_user_staging s
    ON p.login = s.login
    AND p.scd_active = true
    WHERE p.premium_user != s.premium_user
    OR p.address != s.address
    OR COALESCE(p.phone, '') != COALESCE(s.phone, '')
  ) n,
  (
    SELECT MAX(dim_user_id) AS max_id
    FROM dim_user_new
  ) m;
```

7. Copy all the records from the staging table which don't exist in the production table:

```
  INSERT INTO TABLE dim_user_new
  SELECT n.id + COALESCE(m.max_id, 0), -- new id for dim_user_id
    n.login,
    n.premium_user,
    n.address,
    n.phone,
    n.name,
    n.surname,
    n.year_of_birth,
    1, -- 1 for scd_version
    '2016-10-01 00:00:00', -- current timestamp for scd_start_date
    '9999-12-31 23:59:59', -- default timestamp for scd_end_date
    true -- true for scd_active
  FROM (
    SELECT row_number() OVER () AS id,
      s.login,
      s.premium_user,
      s.address,
      s.phone,
      s.name,
    SELECT MAX(dim_user_id) AS max_id
    FROM dim_user_new
  ) m;
```

8. Replace the content of the production table in a transactional mann er:
```
INSERT OVERWRITE TABLE dim_user_production
SELECT *
FROM dim_user_new;
```

Please take into account the way we handled fields of SCD Type 2 that may have NULL values (we don't need to compare fields of SCD Type 1):

```
COALESCE(p.phone, '') = COALESCE(s.phone, '')
COALESCE(p.phone, '') != COALESCE(s.phone, '')
```

Alternatively, you can use <=> operator (Hive 0.9.0 and higher):

```
p.phone <=> s.phone
NOT (p.phone <=> s.phone)
```

That's it. No magic here 😊