Changing Data in BigQuery

04

Google Cloud

Topics

01	Managing Change in Data Warehouses
02	Handling Slowly Changing Dimensions
03	DML Statements in BigQuery
04	DML: Best Practices and Common Issues

01

Managing Change in Data Warehouses

BigQuery manages service availability, but you control the duration and timeliness of your datasets

- All table modifications are ACID-compliant.
- Timeliness might be affected by:
  - Periodic loads versus streaming ingest
  - Priority of load jobs versus analytics jobs

No need to delete older data

In BigQuery, you can afford to retain older data:

- Use sliding windows based on partition field.
- Benefit from BigQuery long-term storage pricing.

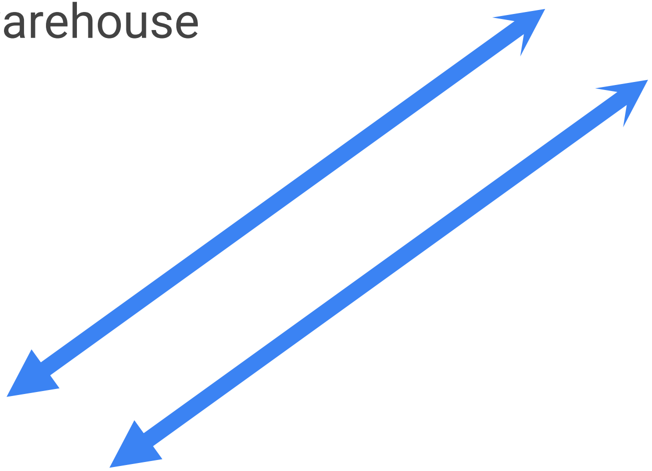
If deleting older data is necessary, set up automatic table/partition expiry.

Make changes without affecting users

- Data warehouses are constantly evolving as users want new and different functionality and data.
- As a result, you must be able to change both schemas and data without affecting users.

Schema changes are usually scheduled as version upgrades

Design, develop, and test the upgrade in parallel while the previous version of the data warehouse is serving analysis workloads.



What about data changes?

Typically ***facts*** don't change, but ***dimensional*** attributes might.

Fact tables versus dimension tables

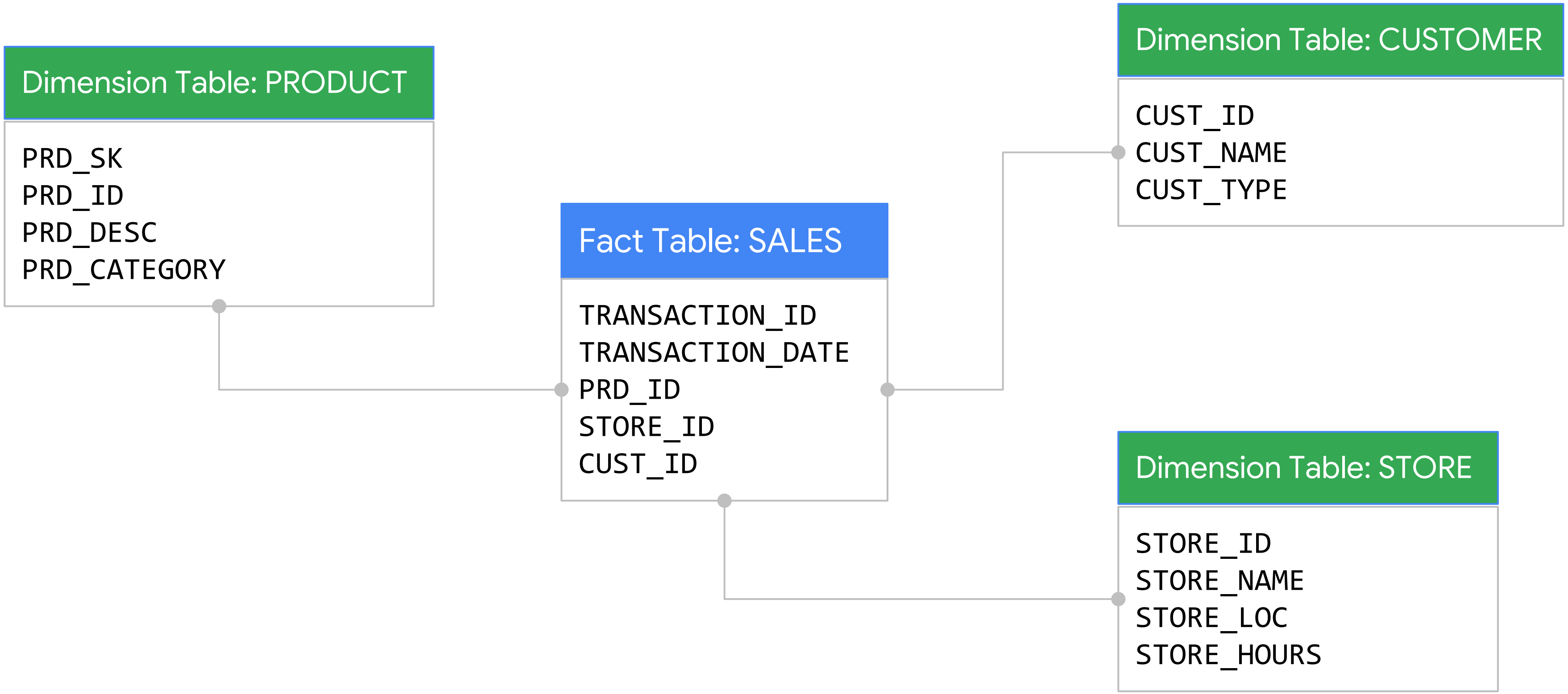
Fact table
holds measurements, metrics, or facts about a business process.

- Examples include sales records.
- Has two types of columns: those that contain facts and those that are a foreign key to a dimension table.

Dimension table
contains descriptive attributes to be used as query constraining.

- Examples include customer, employee, product data.
- Relatively static data **that can change slowly** but unpredictably, rather than according to a regular schedule.

Star schema example with fact and dimension tables



Slowly changing dimensions

In data warehousing,
Slowly Changing Dimensions (SCD)
is an important concept
that is used to enable the
historic aspect of data
in an analytical system.

02

Handling Slowly Changing Dimensions

SCD Type 1: Overwrite attribute value

Before:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	health and beauty

After:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	~~health and beauty~~ cosmetics

SCD Type 2: Change attribute value and maintain history

Before:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	START_DATE	END_DATE
123	ABC	ace moisturizer cream - 100 oz	health and beauty	31-Jan-2009	NULL

After:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	START_DATE	END_DATE
123	ABC	ace moisturizer cream - 100 oz	health and beauty	31-Jan-2009	18-JUL-2017
124	ABC	ace moisturizer cream - 100 oz	cosmetics	19-JUL-2017	NULL

Create view to use in analytics queries

```
CREATE VIEW products_current as (
     SELECT PRD_SK, PRD_ID, PRD_DESC, PRD_CATEGORY, PRD_START_DATE
     FROM dimension_table
     WHERE END_DATE IS NULL
);
```

In a denormalized schema, no changes may be needed to previous fact table rows

TRANSACTION_DATE	PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	UNITS	AMOUNT
18-JUL-2017	123	ABC	ace moisturizer cream - 100 oz	health and beauty	2	25.16
19-JUL-2017	124	ABC	ace moisturizer cream - 100 oz	cosmetics	1	13.50

SCD Type 2: Other options

Using version number to maintain history:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	VERSION
123	ABC	ace moisturizer cream - 100 oz	health and beauty	0
124	ABC	ace moisturizer cream - 100 oz	cosmetics	1

Using effective date and current flag:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	EFFECTIVE_DATE	CURRENT_FLAG
123	ABC	ace moisturizer cream - 100 oz	health and beauty	31-Jan-2009	N
124	ABC	ace moisturizer cream - 100 oz	cosmetics	19-JUL-2017	Y

SCD Type 3: Maintain history by adding columns

Base table:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY		
123	ABC	ace moisturizer cream - 100 oz	CATEGORY_NAME	START_DATE	END_DATE
			health and beauty	31-Jan-2009	18-JUL-2017
			cosmetics	18-JUL-2017	NULL

Create view to use in analytics queries

```
CREATE VIEW products_current as (
    SELECT PRD_SK, PRD_ID, PRD_DESC,
        PRD_CATEGORY.ordinal[array_length(PRD_CATEGORY)] as PRD_CAT
    FROM dimension_table
);
```

View:

PRD_SK	PRD_ID	PRD_DESC	PRD_CAT
124	ABC	ace moisturizer cream - 100 oz	cosmetics

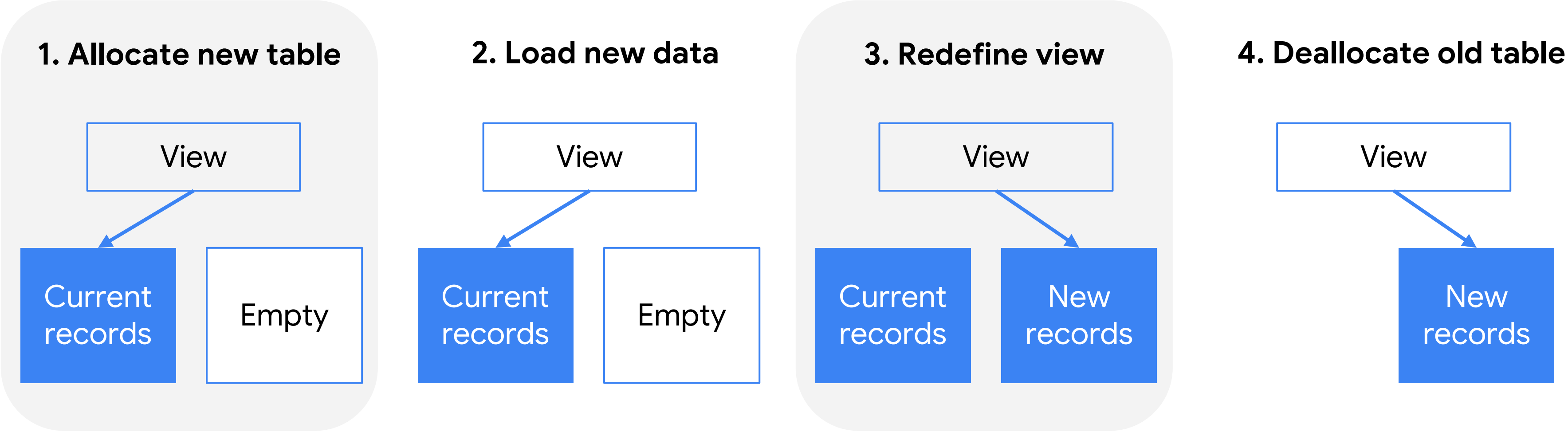
For SCD, there's no one-size-fits-all solution

Changes may be handled differently due to the performance implications of DML:

- In a denormalized table:
  - View switching
  - In-place partition loading

- Isolate changes by using normalized data schema and isolate change in small dimension tables.
  - Update data masking.

Option 1: View switching



View switching considerations

- Zero down-time
- DML on main view might be problematic
- Apply view-switching to custom partitions

Option 2: In-place partition loading

To replace data in a target partition with data from a query of another table:

```
bq query --use_legacy_sql=false --replace \
         --destination_table 'flight_data.fact_flights_part$20140910' \
         'select * REPLACE(...) from `ods.load_flights_20140910`
```

To replace data in a target partition by loading from Cloud Storage:

```
bq load  --replace \
         --source_format=NEWLINE_DELIMITED_JSON
         'flight_data.fact_flights_part$20140910' \
         gs://{bucket}/load_flights_20140910.json
```

Option 3: Update data masking

For tables with data that can change frequently, even within the course of a day, you can implement a conditional join through a view.

```
SELECT f.order_id as order_id, f.customer_id as customer_id,
    IFNULL(u.customer_first_name, f.customer_first_name) as customer_first_name,
    IFNULL(u.customer_last_name, f.customer_last_name) as customer_last_name
FROM fact_table f
LEFT OUTER JOIN pending_customer_updates u
ON f.customer_id = u.customer_id
```

03

DML Statements in BigQuery

Overwrite with UPDATE DML statement for SCD Type 1

```
UPDATE dimension_table
SET PRD_CATEGORY="cosmetics"
WHERE PRD_SK="123"
```

Dimension table:

PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY
123	ABC	awesome moisturizer cream - 100 oz	~~health and beauty~~ cosmetics

Maintain row history using INSERT for SCD Type 2

Insert new record into dimension table :

```
INSERT dimension_table
(PRD_SK, PRD_ID, PRD_DESC, PRD_CATEGORY, VERSION)
VALUES (124,'ABC','ace moisture cream - 100 oz','cosmetics',1)
```

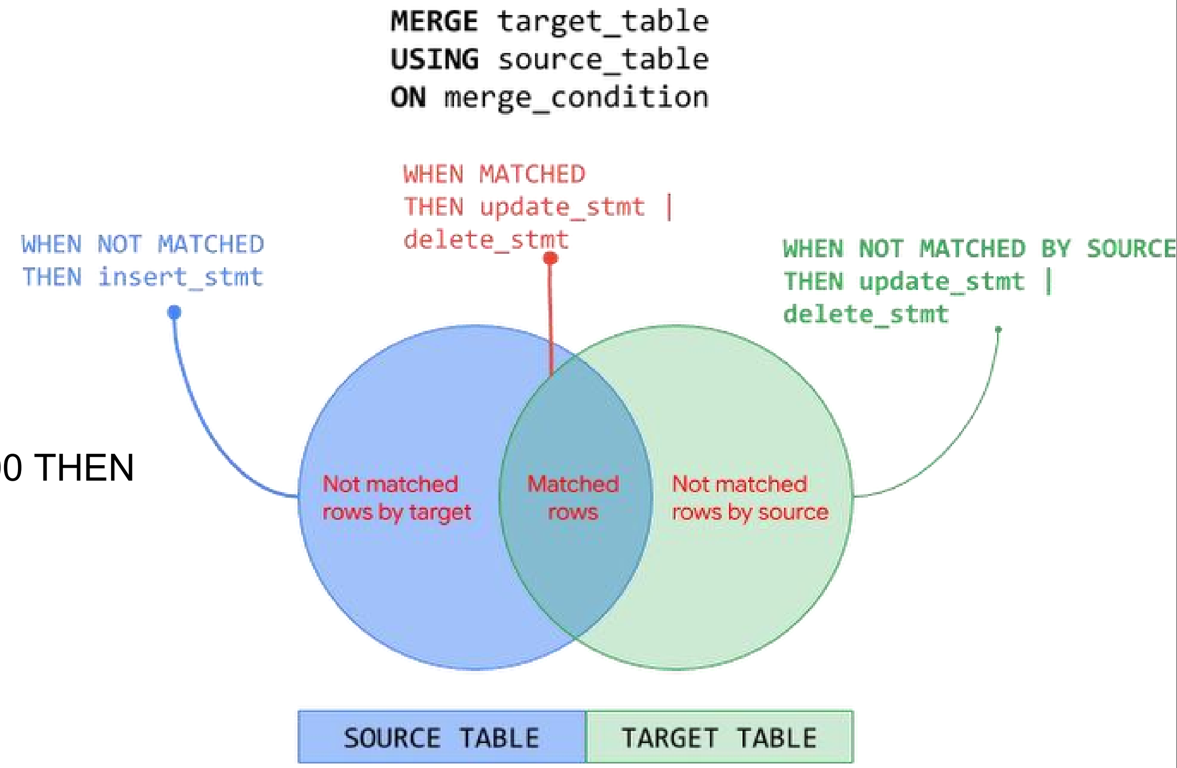
PRD_SK	PRD_ID	PRD_DESC	PRD_CATEGORY	VERSION
123	ABC	ace moisturizer cream - 100 oz	health and beauty	0
124	ABC	ace moisturizer cream - 100 oz	cosmetics	1

UPDATE = DELETE + INSERT

- UPDATE is implemented as DELETE old row + INSERT updated row.
- For SCD Type 2, where you change the attribute value to maintain history, you do an UPDATE and an INSERT.
  - For example, you will *update* the end_date or current_flag attribute of the existing row, and *insert* a new row with a null end_date or Y flag.

MERGE is INSERT, UPDATE, DELETE over a single table in one statement

```
MERGE target_table T
USING source_table S
ON T.field2 = S.field2
WHEN MATCHED AND T.field2 > 100 THEN
  DELETE
WHEN NOT MATCHED THEN
  INSERT(field2) VALUES(field2)
```



Update dimension table using MERGE UPDATE for SCD Type 1

```
MERGE dimension_table as MAIN using
temporary_table as TEMP
on MAIN.PRD_SK = TEMP.PRD_SK
when matched then
UPDATE SET
MAIN.PRD_CATEGORY = TEMP.PRD_CATEGORY
when not matched then
INSERT VALUES(TEMP.PRD_SK, TEMP. PRD_ID, TEMP. PRD_SK, TEMP.
PRD_CATEGORY)
```

DML concurrency

- During any 24-hour period, the first 1500 statements that INSERT into a table run concurrently.
- The UPDATE, DELETE, and MERGE DML statements are called *mutating DML statements*.
    - BigQuery runs up to 2 of them concurrently, after which up to 20 are queued as PENDING.

DML conflicts

- Mutating DML statements that run concurrently on a table cause DML statement conflicts when the statements try to mutate the same partition.
- The statements succeed only if they don't modify the same partition. BigQuery tries to rerun failed statements up to three times.

04

DML: Best Practices and Common Issues

DML best practices

- Use partitioned tables if updates or deletions generally happen on older data or on data in a date-localized manner.
- Use updates over clustered tables where there is clustering locality on the modified rows; they will generally perform better.
- Group DML operations together into larger ones to amortize the cost of running smaller operations.
- Avoid partitioning tables if the amount of data in each partition is small and each update modifies a large fraction of partitions in the table.

DML considerations

- Maximum of 2 concurrent non-insert DML jobs per table. This helps prevent conflicts.
    - After which, BigQuery queues up to 20 non-insert DML jobs per table.
    - Additional statements past the maximum queue length for each table fail.
- Metadata update rate limit is applied at job insert time.
- Commit latency time after the job is done can be very long sometimes.
    - This gets worse with a large number of partitions.
- Single row inserts or single row updates are generally an **undesirable pattern**.

DML common performance bottlenecks

- Updating rows with no locality causes rewrite of entire table, even with partitions.
  - For example, when you update few rows from all files.
  - Consider using clustering.
- Commit time sometimes runs longer than actual execution time.
  - “Byte Counting Query” is required for Billing Quota Check
- Narrow mutations (or deletions) with locality not as fast as they should be because the underlying files are large.

Questions?

Google Cloud