databricks

Streaming Data and CDF

LECTURE

Deleting Data in Databricks

In this lecture, we will explore data deletion in Databricks, its tracking, propagation with CDF, and related limitations and compliance use cases.

Data Deletion in Databricks

Data deletion needs special attention!

- Companies need to handle data deletion requests carefully to main compliance with **privacy regulations** such as GDPR and CCPA.
- PII for users need to be effectively and efficiently handled in Databricks, including deletion.
- These operations usually handled in pipelines separate from ETL pipelines.
- CDF data can be used to propagate deletion action to downstream table.

In order to comply with privacy regulations like GDPR and CCPA, PII needs to be effectively and efficiently handled in Databricks, and deleting PII in particular requires special attention. Deletion is typically handled in pipelines that are separate from ETL pipelines.

CDF data can be used to propagate deletion actions to downstream tables. This is similar to filtering multiplex bronze tables to implement different downstream pipelines for performing ETL, where we're ingesting data into a CDC feed.

With user data, actions can be filtered out in the change data feed. So for any additions like new records, new user records, and updated user records, we would continue inserting the data into our silver pipelines. For deletion events, we can handle this in a separate pipeline that addresses the specific privacy requirements for deleting data.

Recording Important Data Changes

Using commit messages

- Delta Lake supports **arbitrary commit messages** that will be recorded to the Delta transaction log and viewable in the table history. This can help with later **auditing**.
- Commit messages can be;
  - Set at global level
  - Can be specified as part of write operation. For example, data insertion can be labeled based on processing type; manual, automated.

Delta Lake supports arbitrary commit messages that are recorded in the Delta transaction log and viewable in the table history. This can help with later auditing. Commit messages can be set at a global level and can be specified as part of a write operation. For example, data insertion can be labeled based on processing types, such as manual or automated.

Propagating Data Deletion with CDF

How can CDF be used for propagating deletes?

- Data deletion requests can be streamlined with automated triggers using Structured Streaming.
- CDF can be separately leveraged to **identify records** need to be deleted or modified in downstream tables.

Note: When Delta Lake's history and CDF features are implemented, deleted PII values are still present in older versions of the data.

- Using **Vacuum** command will physically delete PIIs
- **Deleting at a partition boundary** will make the whole process more efficient

Data deletion requests can be streamlined with automated triggers using structured streaming. CDF can be separately leveraged to identify records needed to be deleted or modified in downstream tables like we mentioned earlier. Note that when Delta Lake's history and CDF features are implemented, deleted values are still present in older versions of the data. We can solve this by deleting at a partition boundary.

CDF Retention Policy

Important notes for CDF configuration

- File deletion will not actually occur until we VACUUM our table!
- CDF records follow the same retention policy of the table. VACUUM command deletes CDF data.
- By default, the Delta engine will prevent VACUUM operations with less than 7 days of retention. To manually run VACUUM for these files;
  - Disable Spark's retention duration check (retentionDurationCheck.enabled)
  - Run VACUUM with DRY RUN to preview files before permanently removing them
  - Run VACUUM with RETAIN 0 HOURS!

CDF retention policy file deletion won't occur until the table is vacuumed. CDF records follow the same retention policy of the Delta table, so the VACUUM command deletes CDF data, and by default, the Delta engine will prevent automatic vacuum operations with less than seven days of retention.

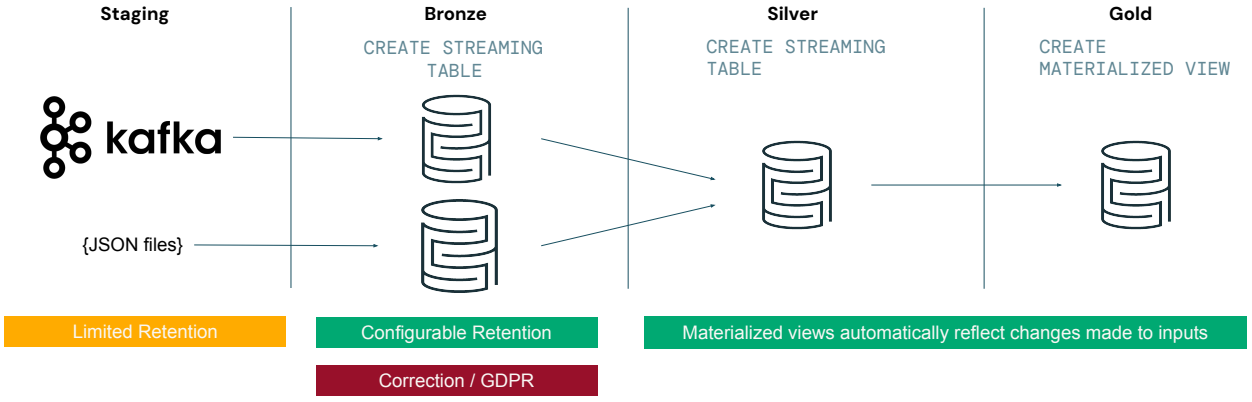
For these files, you need to disable Spark's retention duration check. Run vacuum with dry run to preview files before permanently removing them and run vacuum with zero hours of retention. Note that there are the steps that you would take for Delta tables when not using Lakeflow Spark Declarative Pipelines.

Can I perform DML on a streaming Table? (i.e. GDPR)

The next question is, "Can I perform Data Manipulation actions such as Inserts, Updates, Deletes, and Merges Into a streaming table?".

Example GDPR use case

Using streaming tables for ingestion and materialized views after



Let's review a GDPR use case with streaming tables and materialized views.

1. In staging, information is ingested from Kafka, where we have limited retention. In this example, we ingest JSON files and stream them into our bronze table. Append only is used because we're working with a streaming table. From here, APPLY CHANGES INTO is used on our silver table to process the data.
2. In Lakeflow Spark Declarative Pipelines, we have configurable retention, where you can set the pipeline reset permission to prevent the table from refreshing. This can be useful when you need to delete a user permanently.
3. For downstream operations, you can do full refreshes on the silver tables and gold tables with Materialized Views to re-compute these values based on this. However, this can be expensive for larger tables.

DML works on streaming tables only

Updates, deletes, inserts and merges on streaming tables.

- **Ensure compliance** for retention periods on a table.

```
DELETE FROM my_live_tables.users
WHERE updated < current_time() - INTERVAL 3 years;
```

- **Scrub PII** from data in the lake.

```
UPDATE my_live_tables.users
SET email = hash(email, salt)
WHERE id = 2;
```

User

id	email	updated
~~1~~	~~****@gmail.com~~	~~01/01/2019~~
2	****@company..	02/01/2024
3	****@hotmail...	02/01/2025

- **Append new data**.

```
INSERT INTO my_live_tables.users
VALUES (3, hash(email, salt), current_time());
```


You can also fix your data whenever needed for cases like:

- You can delete your data using date intervals to ensure compliance for retention periods.
- Scrub your PII on a specific column; or merge it with another table.
- Or just continue appending new data.

Materialized Views don't support DML

MVs retain data's state from last refresh

- **No direct updates allowed** Insert, Update, Delete, Merge Into.
- **Precomputed results** for faster data retrieval.

```
CREATE OR REFRESH MATERIALIZED VIEW users_by_date AS
  SELECT count(*), updated
  FROM my_live_tables.users
  GROUP BY (updated);
```

- **Refresh mechanism** rather than updating with every query.
  - Manually or in Lakeflow Spark Declarative Pipelines execution

```
REFRESH MATERIALIZED VIEW users_by_date FULL;
```

With Materialized Views, you cannot directly perform INSERT, UPDATE, or DELETE operations due to their nature of storing the query results from the last refresh. This allows for faster data retrieval for the gold layer tables.
Instead, materialized views are updated through refresh operations, which are executed during the Lakeflow Spark Declarative Pipelines. If an MV is manually created, it can be manually refreshed by using the "Refresh" command.

Thank you for completing this lesson and continuing your journey to develop your skills with us.