DML Internals: How do Delete, Update, Merge work

Diving into Delta Lake Series

Tathagata "TD" Das, Denny Lee

Who are we?

Tathagata *"TD"* Das

Staff Software Engineer – Databricks

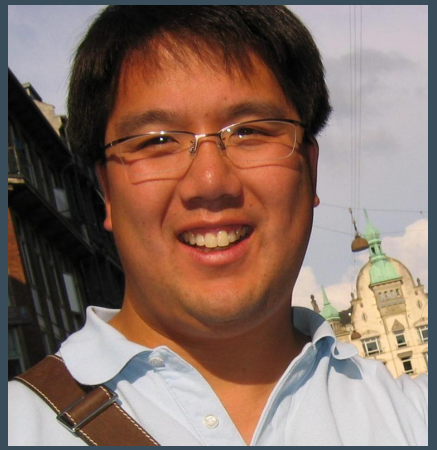
Developing Apache Spark™ since 2011

Started *Spark Streaming* project in AMPLab, UC Berkeley

Core developer of *Structured Streaming* and *Delta Lake*

Who are we?

Denny Lee



Staff Developer Advocate – Databricks

Working with Apache Spark™ since v0.6

Former Senior Director Data Science Engineering at Concur

Former Microsoftie - Cosmos DB, HDInsight (Isotope)

Masters Biomedical Informatics - OHSU

BS in Physiology - McGill

Delta Lake – 1 slide intro

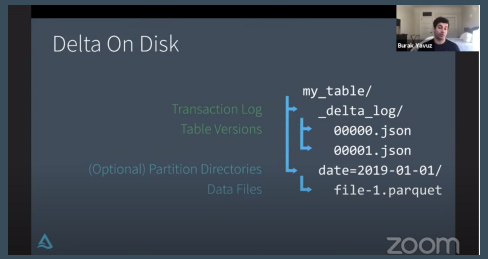
Delta Lake is an open-source storage layer that brings ACID transactions to Spark workloads

- Open format
- Scalable metadata
- Time travel
- Schema enforcement
- Audit history

Previous webinars in the series

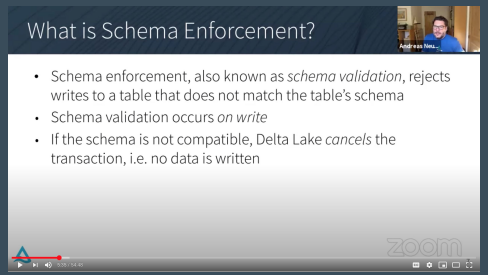
Diving into Delta Lake Part 1: Unpacking the Transaction Log

https://www.youtube.com/watch?v=F91G4RoA8is



Diving into Delta Lake Part 2: Enforcing and Evolving the Schema

https://www.youtube.com/watch?v=tjb10n5wVs8



Outline

Data Manipulation Language (DML) Operations

- Update, Delete, Merge
- Internals
- Performance tuning

Common design patterns using DML Operations

DML Operations

Update

```
SQL*: UPDATE table SET x=y WHERE <predicate>

Scala:
    val dt = DeltaTable.forPath(path)
    dt.updateExpr("predicate", Map("x" -> "y"))


Python:
    dt = DeltaTable.forPath(path)
    dt.update("predicate", { "x": "y" })
```

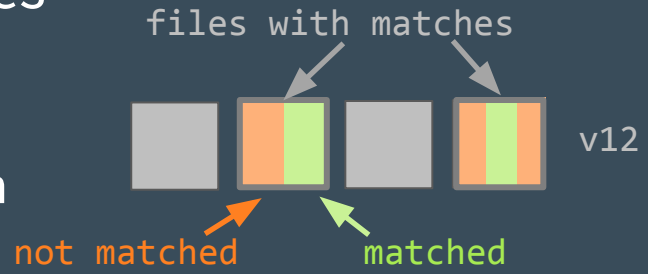
*Databricks Delta Lake, or in future, OSS Delta Lake on Spark 3.0

Update – Under the hood

```
SQL:  UPDATE my_table SET x=y WHERE predicate
```

Updates data at the granularity of files

- Uses two scans on the relevant data
- Scan 1: Find and select which files contain data matching predicate
  - Uses the predicate to skip data when it can

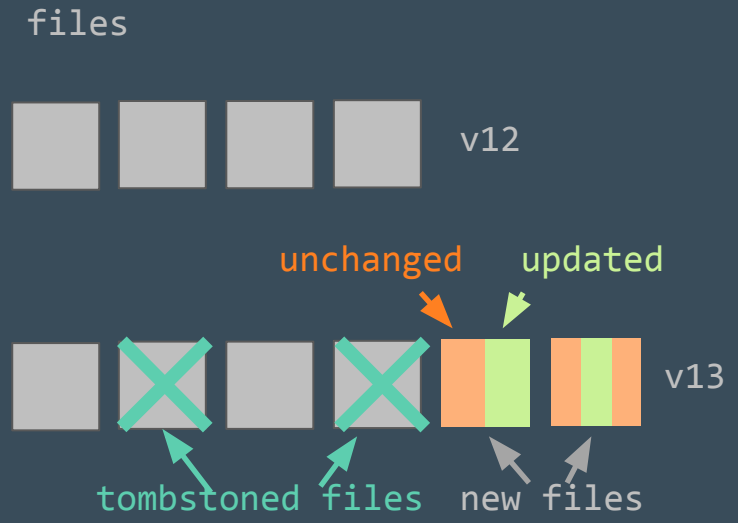


Update – Under the hood

```
SQL:  UPDATE my_table SET x=y WHERE predicate
```

Updates data at the granularity of files

- Uses two scans on the relevant data
- Scan 1: Find and select which files contain data matching predicate
- Scan 2: Read selected files and rewrite them as new files with unchanged + updated data



Over to Denny!

Update + Time travel – Easy debugging

```
SQL:  UPDATE my_table SET x=y WHERE predicate
```

Earlier data can be still queried with Time Travel

```
spark.read
  .option("versionAsOf", "12")
  .load(path)
```

Update – Improving Performance

Add more predicates to narrow down the search space

Databricks Delta Lake optimizations – better data skipping

- Z-order Optimize
  - Like multi-column sorting, but better
  - improves efficacy of column stats
- Bloom filters

Delete

SQL*: DELETE FROM table WHERE *<predicate>*

Scala:

```
val dt = DeltaTable.forPath(path)
dt.delete("predicate")
```

Python:

```
dt = DeltaTable.forPath(path)
dt.delete("predicate")
```

*Databricks Delta Lake, or in future, OSS Delta Lake on Spark 3.0

Delete + Vacuum

```
SQL:  DELETE FROM table WHERE <predicate>
```

Data not deleted on disk until older versions are vacuumed

```
VACUUM table [RETAIN x HOURS]
```

Deletes all old and unnecessary files not need by versions earlier than x hours

Default vacuum retention interval is 7 days

Vacuum with retention 0 to remove all versions except the latest

CAUTION: Do not run vacuum 0 when other writes are in progress

Merge

Standard SQL syntax

```
MERGE INTO target t USING source s
ON t.key = s.key
WHEN MATCHED THEN UPDATE SET x = y
WHEN NOT MATCHED THEN INSERT (x) VALUES y
```

Merge – Extended syntax

Clause conditions

```
MERGE INTO target t USING source s
ON t.key = s.key
WHEN MATCHED AND <clause_condition1> THEN UPDATE
SET x = y
WHEN NOT MATCHED AND <clause_condition2> THEN
INSERT (x) VALUES y
```

Merge – Extended syntax

Multiple matched clauses and delete

```
MERGE INTO target d USING source s
ON t.key = s.key
WHEN MATCHED AND <clause_condition1> THEN UPDATE
SET x = y
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED AND <clause_condition2> THEN
INSERT (x) VALUES y
```

Merge – Extended syntax

Star to auto-expand to target table columns

```
MERGE INTO target t USING source s
ON t.key = s.key
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *
```

UPDATE SET * equivalent to for every column in target Delta table
UPDATE SET col1 = s.col1, col2 = s.col2, …

Source must have the same columns

Merge – Programmatic APIs

Scala:

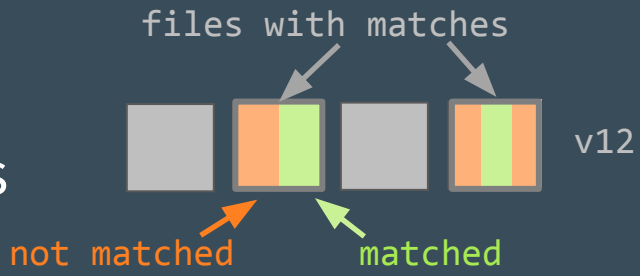
```
deltaTable.alias("t")
  .merge(
    sourceDF.alias("s"),
    "t.key = s.key")
 .whenMatched().updateAll()
 .whenNotMatched().insertAll()
 .execute()
```

Python:

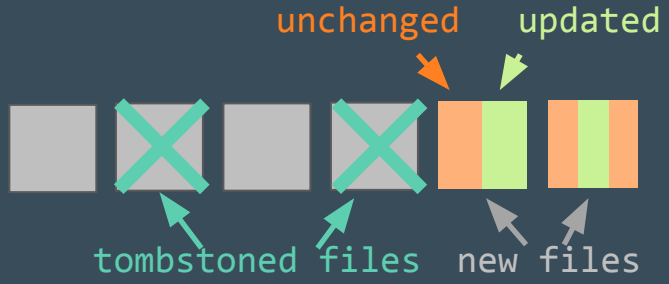
```
deltaTable.alias("t")
  .merge(
    sourceDF.alias("s"),
    "t.key = s.key")
 .whenMatchedUpdateAll()
 .whenNotMatchedInsertAll()
 .execute()
```

Merge – Under the hood

Scan 1: Inner join between target and source to select files that have matches



Scan 2: Outer join between the selected files in target and source and write the update/deleted/inserted data



Over to Denny!

Merge – Improving Performance

Scan 1: Inner join slow?

Add more predicates to narrow down the search space for matches

Adjust shuffle partitions

Adjust broadcast thresholds

If too many small files in the table, compact them, but don't make very large files as more data will need to unnecessarily copied when file rewritten

Databricks Delta Lake: Use Z-order optimize to exploit locality of updates

Merge – Improving Performance

Scan 2: outer join slow?

- Adjust shuffle partitions
  - Can generate too many small files for partitioned tables
  - Reduce files by enabling automatic repartitioning before writes (with Delta Lake 0.6.0, or with *Optimized Writes* in *Databricks Delta Lake*)

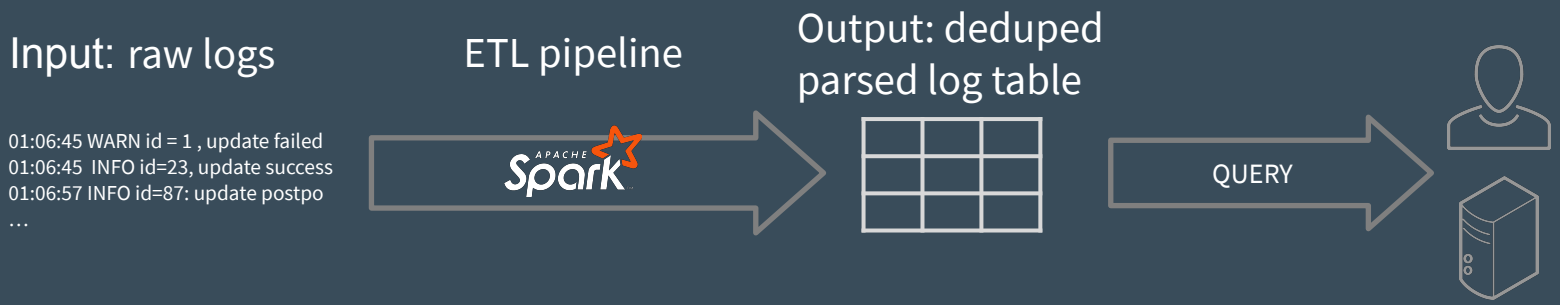
- If full outer join, Spark cannot do broadcast join therefore adjust broadcast thresholds; if right outer join, Spark can and adjust broadcast thresholds

- Cache the source table/DataFrame, can speed up the scan 2
  - *CAUTION: don't cache the target table, can lead to cache coherency issues*

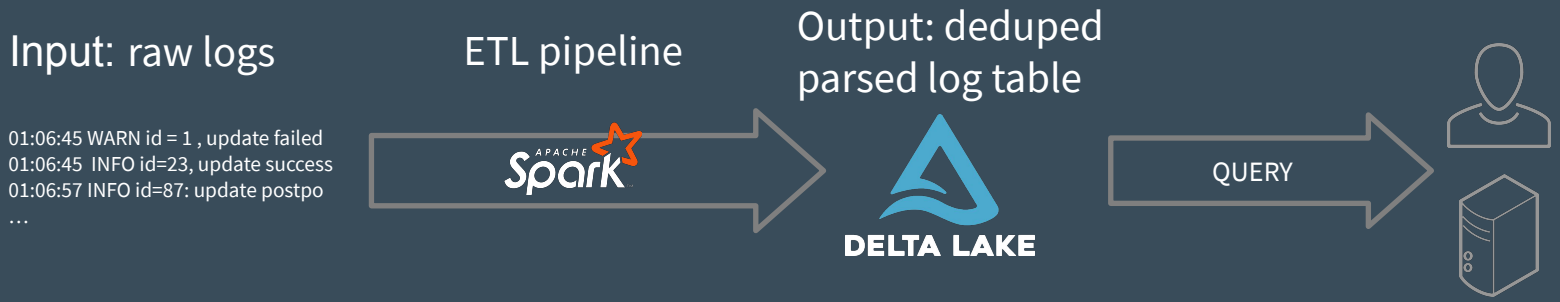
Common design patterns

with DML Ops

Pattern 1: Deduplication during ETL



Pattern 1: Deduplication during ETL



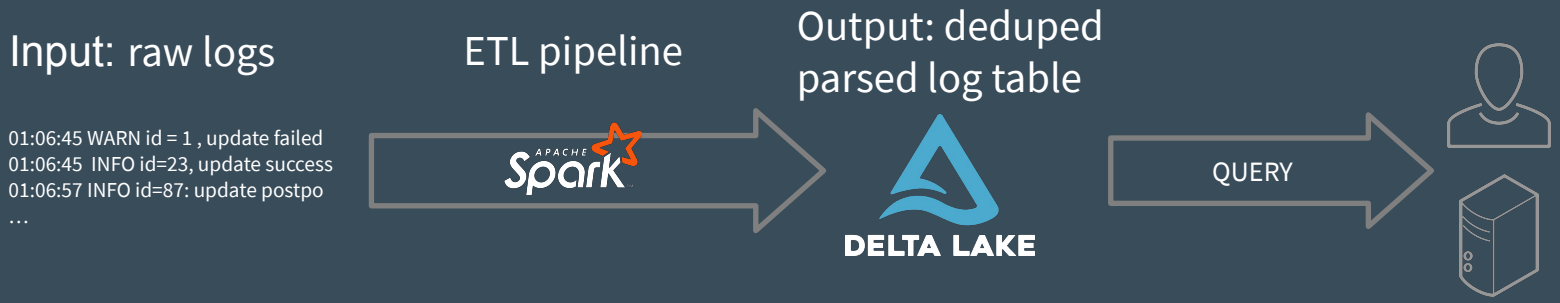
Parse the raw logs

Use merge to insert parsed log only when the unique id does not match

```
MERGE INTO logTable
USING parsedLogs
ON logTable.uniqueId = parsedLogs.uniqueId
WHEN NOT MATCHED THEN INSERT *
```

Problem: tries to match with the whole table

Pattern 1.1: Deduplication during ETL

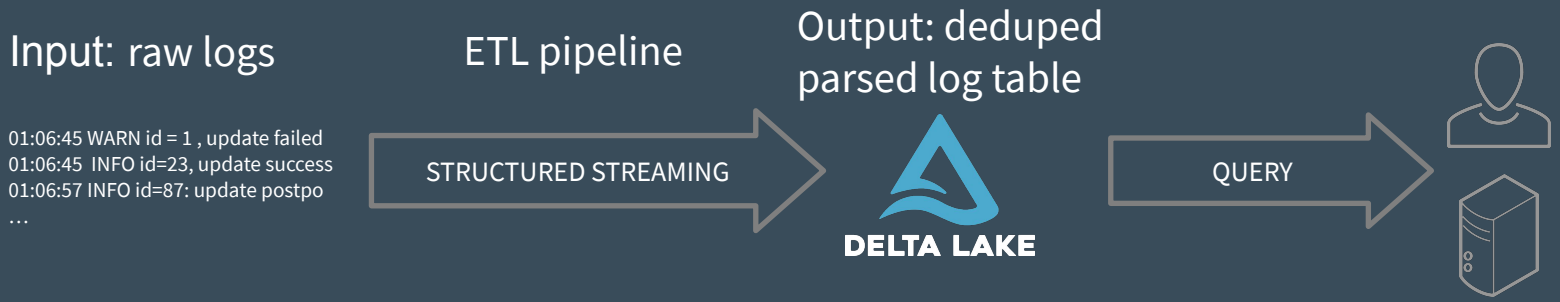


Better solution:

If dupes are expected for max 7 days, use that as a predicate in merge

```
MERGE INTO logTable
USING parsedLogs
ON logTable.uniqueId = parsedLogs.uniqueId AND
    logTable.date > current_date() - INTERVAL 7 DAYS
WHEN NOT MATCHED AND
    updates.date > current_date() - INTERVAL 7 DAYS
THEN INSERT *
```

Pattern 1.1: Deduplication during ETL



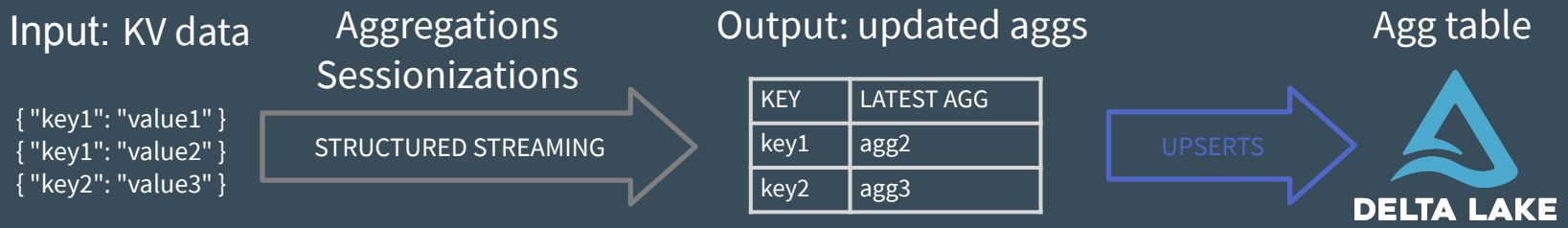
Dedup in streaming using foreachBatch!

```
streamingParsedLogs.writeStream.foreachBatch {
    (parsedLogDF: DataFrame, batchId: Long) =>

        parsedLogDF.createOrReplaceTempView("parsedLogs")

        parsedLogDF.sparkSession.sql(
            "MERGE INTO logTable USING parsedLogs …")
}
```

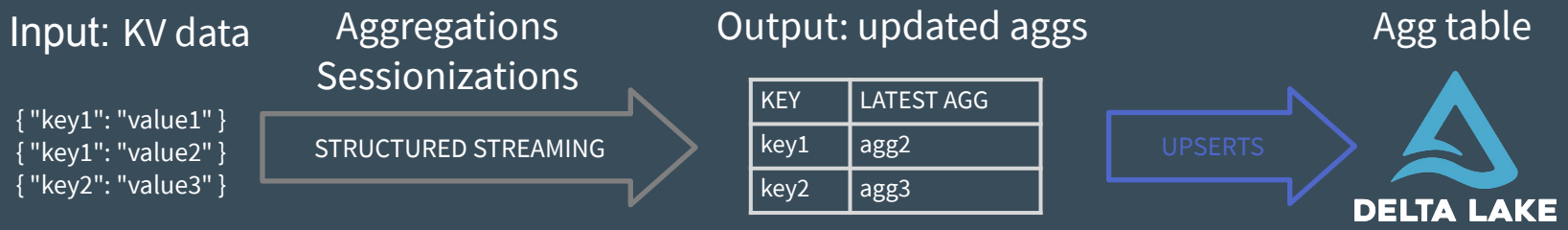
Pattern 2: Streaming aggregates



Use Structured Streaming to compute aggregates

Write to Delta for downstream analytics

Pattern 2: Streaming aggregates




```
streamingParsedLogs.writeStream.foreachBatch {
    (aggsDF: DataFrame, batchId: Long) =>

  deltaTable.alias("t")
    .merge(aggsDF.alias("s"), "t.key = s.key")
    .whenMatched().updateAll()
    .whenNotMatched().insertAll()
    .execute()
}
```

Pattern 2.1: Change log from agg table

Get the change log of key-value updates from the aggregation Delta table for syncing to other databases

This is hard because Delta tracks changes in files, not rows

Pattern 2.1: Change log from agg table

Append the updated key-values into a staging *changelog* table

Stream from changelog table to

- Upsert from staged table into the final agg table
- Sync changes to other databases, etc.

Pattern 3: GDPR – Simple way

Delete *user* from *userinfo* table

- Removes user from latest version but not from disk
- Past versions will still have the user data

Vacuum table to delete history of the table

- If vacuumed with zero retention, then past versions with user data will be removed

Problem: all user's history get deleted

userinfo table

USER	ADDRESS
user1	currentAddr1
user2	currentAddr2

Pattern 3.1: GDPR with all user history

Explicitly maintain user history in the latest version of the table

- Multiple rows per user
- Current info clearly marked
- Use SCD Type 2 operations with Merge to update

Delete all rows for a user, then vacuum

user_info_history table

USER	ADDRESS	IS_CURRENT
user1	oldAddr1	false
user1	oldAddr2	false
user1	currentAddr1	true
user2	oldAddr5	false
user2	currentAddr7	true

See Merge in Delta docs for full example of SCD Type 2

Pattern 4: Apply change data with deletes

Captured DB changes

```
INSERT a, 1
INSERT b, 2
UPDATE a, 3
DELETE b
INSERT b, 4
```

Apply changes to Delta table


```
MERGE INTO target t USING (
  -- find last change for each key
  SELECT key, last.newVal AS newVal, last.deleted AS deleted FROM (
    SELECT key, MAX(struct(time, newValue, deleted)) AS last
    FROM changes GROUP BY key
  )
) s
ON s.key = t.key
WHEN MATCHED AND s.deleted = true
    THEN DELETE
WHEN MATCHED
    THEN UPDATE SET key = s.key, value = s.newVal
WHEN NOT MATCHED AND s.deleted = false
    THEN INSERT (key, value) VALUES (key, newVal)
```

Delete clause in merge makes this possible

Run this extended merge query in foreachBatch

See Merge examples in Delta docs

Community and Ecosystem

Delta Lake Releases and Roadmap

0.5.0 (Dec 2019)

Support for Presto and other processing engines using manifest files

Improved concurrency

Improved merge perf when only insert clause

Improved support for file compaction

0.6.0 (Apr 2020)

Schema evolution in merge

Improved merge perf with repartitioning

Improved merge perf when no insert clause

Operation metrics in table history

0.7.0 (~June 2020)

Support for Apache Spark 3.0

Support for tables defined in Hive metastore, SQL DDLs *(CREATE/ALTER TABLE)*, SQL DMLs *(UPDATE/DELETE/MERGE)*

Delta Lake Connectors

Standardize your big data storage with an open format accessible from various tools

Delta Lake Partners and Providers

More and more partners and providers are working with Delta Lake

Google Dataproc

Privacera

Informatica

Streamsets

WANDisco

Qlik

Users of Delta Lake

Tencent 腾讯

Alibaba Group

COMCAST

viacom

ciena

JAM CITY

edmunds

databricks

healthdirect Australia

Booz | Allen | Hamilton®

McAfee Together is power.

CONDÉ NAST

TILTING POINT

Mc Graw Hill Education

acorns

EVERQUOTE

Namely

usermind

ZEISS

split

upwork

DOLLAR SHAVE CLUB

wehkamp

Thank You

“Do you have any questions for my prepared answers?”

– Henry Kissinger