



# databricks

The Data and AI Company



Berlin  
Buzzwords  
2021

Tue 15th Jun,  
2021

# Simplifying upserts and deletes on



**DELTA LAKE**

**Prashanth Babu**

EMEA RSA Practice Lead

# Agenda

- 1 Intro
- 2 Challenges with Data Lakes
- 3 Features of Delta Lake
- 4 Update, Delete & Upsert on a Delta Lake table
- 5 Optimize & Vacuum of a Delta Lake table
- 6 Demos
- 7 Outro and further references

# About Me

## **Prashanth Babu**

Practice Lead - Resident Solutions Architect

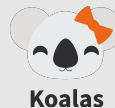
@ Databricks, London — since 2018

<https://www.linkedin.com/in/P7h>





## ORIGINAL CREATORS



## CUSTOMERS

**5000+**

Across the globe



# Lakehouse

One simple platform to unify all of  
your data, analytics, and AI workloads

# databricks **Lakehouse Platform**

Open | Simple | Collaborative

Data Science &  
Engineering

BI &  
SQL Analytics

Machine Learning

Real-time Data  
Applications

Data Management & Governance

Open Data Storage



Structured



Semi-structured



Unstructured



Streaming



# We're working with enterprises in every industry

## Healthcare & Life Sciences



## Manufacturing & Automotive



## Media & Entertainment



## Financial Services



## Public Sector



## Retail & CPG



## Energy & Utilities



## Digital Native







# Challenges with Data Lakes



# The future is here, it's just not evenly distributed

**83%** CEOs say AI is a strategic priority

**MIT Sloan**  
Management Review

**\$3.9T** Business value created by AI in 2022

**Gartner**

**85%** Of big data projects fail

**Gartner**

**87%** Of data science projects never make it into production

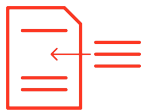
**VB**

# Challenges with Data Lakes



## 1. Hard to append data

Adding newly arrived data leads to incorrect reads



## 2. Modification of existing data is difficult

GDPR/CCPA requires making fine grained changes to existing data lake



## 3. Jobs failing mid way

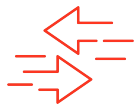
Half of the data appears in the data lake, the rest is missing

# Challenges with Data Lakes



## **4. Real-time operations**

Mixing streaming and batch leads to inconsistency



## **5. Costly to keep historical versions of the data**

Regulated environments require reproducibility, auditing, governance



## **6. Difficult to handle large metadata**

For large data lakes the metadata itself becomes difficult to manage

# Challenges with Data Lakes



## 7. “Too many files” problems

Data lakes are not great at handling millions of small files



## 8. Hard to get great performance

Partitioning the data for performance is error-prone and difficult to change



## 9. Data quality issues

It's a constant headache to ensure that all the data is correct and high quality



**DELTA LAKE**

# A new standard for building data lakes

## **An opinionated approach to building robust Data Lakes**

- Adds reliability, quality, performance to Data Lakes
- Brings the best of data warehousing and data lakes
- Builds on an open format (parquet) and adds an open source transaction log





# Delta Lake is comprised of:

- Delta tables
- The Delta optimization engine
- The Delta Lake storage layer

# Delta Lake offers:

- ACID transactions on Spark
- Scalable metadata handling
- Streaming and batch unification
- Schema enforcement
- Time travel
- Upserts and deletes
- Fully configurable/optimizable
- Structured Streaming support



# Delta Lake tackling the challenges with Data Lakes

1. Hard to append data
2. Modification of existing data difficult
3. Jobs failing mid way
4. Real-time operations hard
5. Costly to keep historical data versions
6. Difficult to handle large metadata
7. "Too many files" problems
8. Poor performance
9. Data quality issues

## ACID Transactions

### Make every operation transactional

- It either fully succeeds - or it is fully aborted for later retries



```
/path/to/table/_delta_log  
- 0000.json  
- 0001.json  
- 0002.json  
- ...  
- 0010.parquet
```

# Delta Lake tackling the challenges with Data Lakes

1. Hard to append data
2. Modification of existing data difficult
3. Jobs failing mid way
4. Real-time operations hard
5. Costly to keep historical data versions
6. Difficult to handle large metadata
7. "Too many files" problems
8. Poor performance
9. Data quality issues

## ACID Transactions

### **Make every operation transactional**

- It either fully succeeds - or it is fully aborted for later retries

### **Review past transactions**

- All transactions are recorded and you can go back in time to review previous versions of the data (i.e. *time travel*)

```
SELECT * FROM events  
TIMESTAMP AS OF ...
```

```
SELECT * FROM events  
VERSION AS OF ...
```

# Delta Lake tackling the challenges with Data Lakes

1. Hard to append data
2. Modification of existing data difficult
3. Jobs failing mid way
4. Real-time operations hard
5. Costly to keep historical data versions
6. Difficult to handle large metadata
7. "Too many files" problems
8. Poor performance
9. Data quality issues

## Spark under the hood

- Spark is built for handling large amounts of data
- All Delta Lake metadata stored in open Parquet format
- Portions of it cached and optimized for fast access
- Data and its metadata always co-exist. No need to keep catalog<>data in sync

# Delta Lake tackling the challenges with Data Lakes

1. Hard to append data
2. Modification of existing data difficult
3. Jobs failing mid way
4. Real-time operations hard
5. Costly to keep historical data versions
6. Difficult to handle large metadata
7. "Too many files" problems
8. Poor performance
9. Data quality issues

## Indexing

### **Automatically optimize a layout that enables fast access**

- Partitioning: layout for typical queries
- Data skipping: prune files based on statistics on numericals
- Z-ordering: layout to optimize multiple columns
- Auto-optimize – periodically re-layout data in the background

**OPTIMIZE** events  
**ZORDER BY** (eventType)



# Delta Lake tackling the challenges with Data Lakes

1. Hard to append data
2. Modification of existing data difficult
3. Jobs failing mid way
4. Real-time operations hard
5. Costly to keep historical data versions
6. Difficult to handle large metadata
7. "Too many files" problems
8. Poor performance
9. Data quality issues

## Schema validation, evolution

### Schema validation and evolution

- All data in Delta Tables have to adhere to a strict schema validation
- But also includes schema evolution in merge operations

```
MERGE INTO events
USING changes
ON events.id = changes.id
WHEN MATCHED THEN
    UPDATE SET *
WHEN NOT MATCHED THEN
    INSERT *
```

# Updates, Deletes and Upserts on a Delta Lake Table



# In this talk, we will focus on the following

1. Hard to append data
2. Modification of existing data difficult

Updates, Deletes and  
Upserts

3. Jobs failing mid way
4. Real-time operations hard
5. Costly to keep historical data versions
6. Difficult to handle large metadata

7. "Too many files" problems
8. Poor performance
9. Data quality issues

Optimize and Vacuum

# Updates, Deletes and Upserts

## Sample Use cases

- GDPR compliance
- CDC from traditional dbs
- Sessionization
- Deduplication

## Few of the Challenges

- Inefficient
- Possibly incorrect
- Hard to maintain
- Unreliable

# UPDATE

## Key Features

- Updates the column values for the rows that match a predicate
- When no predicate is provided, updates the column values for all rows
- Supports subqueries in the WHERE predicate, including IN, NOT IN, EXISTS, NOT EXISTS, and scalar subqueries

## Syntax

```
UPDATE languages
```

```
SET name = 'python3'
```

```
WHERE name = 'python'
```

# DELETE

## Key Features

- Deletes the rows that match a predicate
- When no predicate is provided, deletes all rows
- Supports subqueries in the **WHERE** predicate, including **IN**, **NOT IN**, **EXISTS**, **NOT EXISTS**, and scalar subqueries

## Syntax

```
DELETE FROM customers  
WHERE id = 219897
```

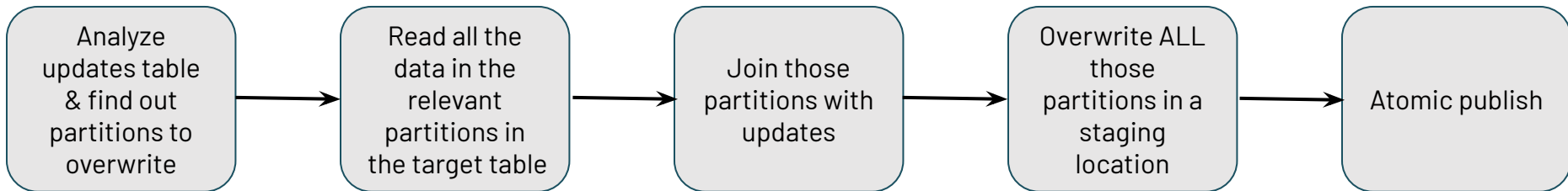


# MERGE (UPSERT)

## Merge without Delta Lake – approach#1

- Select all of the data from your table not including the rows you want to delete
- Create a new table based on the previous query
- Delete the original table
- Rename the new table to the original table name for downstream dependencies

## Merge without Delta Lake – approach#2



# MERGE (UPSER) – Delta Lake

```
MERGE INTO customers      -- Delta Lake table
USING updates
ON customers.customerId = source.customerId
WHEN MATCHED THEN
    UPDATE SET address = updates.address
WHEN NOT MATCHED
    THEN INSERT (customerId, address) VALUES
    (updates.customerId, updates.address)
```

# Steps of a MERGE (roughly speaking)

update table and target table

1. Inner-join between update and target using the condition in the ON clause → Returns to the driver: list of files in **target** that contain matching rows
2. Short circuiting happens:
  - If there were NO matching rows in **target** in step 1, use an append-only write to append **update** to **target**
  - Else: Use a full-outer-join to consolidate all the changes between **update** and **target**
3. Atomically commit to the Delta transaction log, removing the matching files of step 1 and adding the new files from step 2

# Optimize and Vacuum a Delta Lake table



# OPTIMIZE and VACUUM

- OPTIMIZE (with and without ZORDER) to do compaction and improve data skipping.

## **OPTIMIZE** events

```
WHERE date ≥ current_timestamp() - INTERVAL 1 day  
ZORDER BY (eventType)
```

- VACUUM to clean up old (untracked) files to limit storage costs.

```
VACUUM events;
```

# Demos



# Further study and references

O'REILLY®

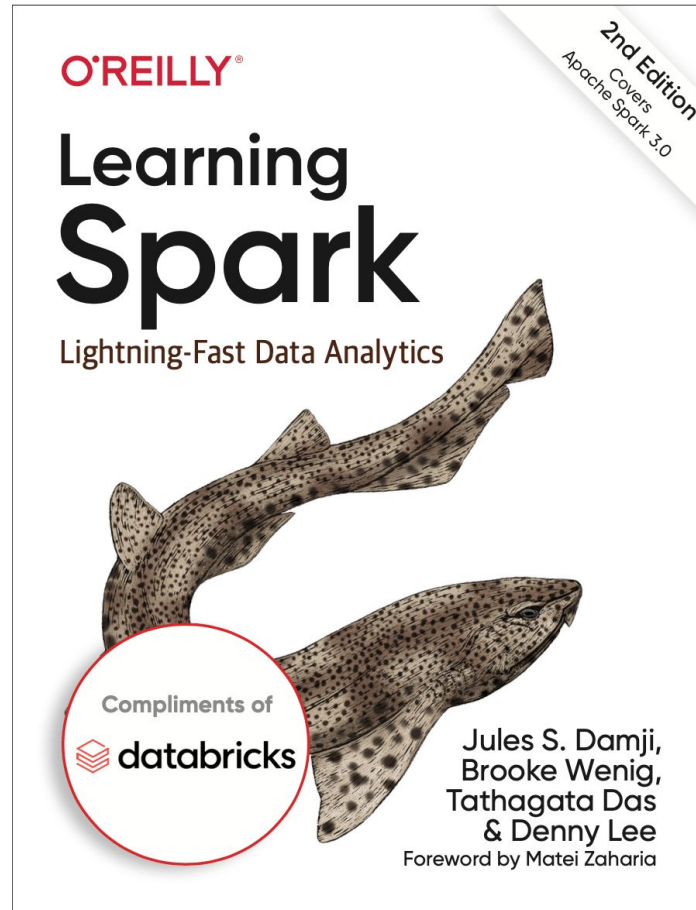
# Delta Lake The Definitive Guide

Modern Data Lakehouse Architectures  
with Delta Lake



Denny Lee,  
Tathagata Das &  
Vini Jaiswal

<https://dbricks.co/ebLearningSpark>



# Docs and Talks

## Documentation and best practices references

- [Delta Lake Guide](#)
- [Delta Open Source Project](#)
- [Various optimizations in Delta Lake](#)
- [Delta Lake MERGE](#)
- [Delta Lake Concurrency Control](#)
- [Z-order curve](#) and [Hilbert curve](#)
- [Delta Streaming reads and writes](#)
- [Delta Lake: The Definitive Guide -- O'Reilly publishers](#)
- [Learning Spark -- O'Reilly publishers](#)

## Talks:

- [Threat Detection and Response at Scale](#)
- [Winning the Audience with AI: How Comcast Built An Agile Data And Ai Platform At Scale \(Comcast\)](#)
- [Building Robust Production Data Pipelines with Databricks Delta](#)
- [Designing and Building Next Generation Data Pipelines at Scale with Structured Streaming](#)
- [Tech Talk series: Diving into Delta Lake](#)

# Blogs

- [Efficient Upserts into Data Lakes using Databricks Delta](#)
- [New Databricks Delta Features Simplify Data Pipelines](#)
- [Introducing Delta Time Travel for Large Scale Data Lakes](#)
- [Simplifying Change Data Capture with Databricks Delta](#)
- [Building a Real-Time Attribution Pipeline with Databricks Delta](#)
- [Processing Petabytes of Data in Seconds with Databricks Delta](#)
- [Simplifying Streaming Stock Data Analysis Using Databricks Delta](#)
- [Make Your Oil and Gas Assets Smarter by Implementing Predictive Maintenance with Databricks](#)
- [Build a Mobile Gaming Events Data Pipeline with Databricks Delta](#)
- [Delta Lake tagged blogs](#)
- [Diving Into Delta Lake: Unpacking The Transaction Log](#)

# Webinars - YouTube playlist

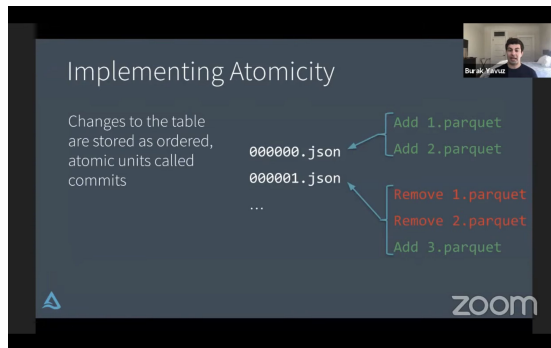
Tech Talk | The Genesis of Delta Lake - An Interview with Burak Yavuz

<https://youtu.be/F-5t30CI96g>



Tech Talk | Diving into Delta Lake Part 1: Unpacking the Transaction

<https://youtu.be/F91G4RoA8is>




# Webinars - YouTube playlist

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

<https://youtu.be/tjb10n5wVs8>

What is Schema Evolution?

- Schema evolution allows users to easily change a table's current schema to accommodate data that is changing over time.
- Most commonly used operations for
  - append
  - overwrite
- Use `.option('mergeSchema', 'true')` to your `.write` or `.writeStream` Spark command.
- Also can use `spark.databricks.delta.schema.autoMerge = True` to Spark configuration.

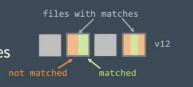


Tech Talk | Diving into Delta Lake Part 3: How do  
DELETE, UPDATE, and MERGE work


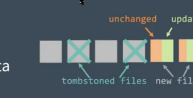
<https://youtu.be/7ewmcdrylsA>

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



# Thank you