databricks

DELTA LAKE

Authors: Amine El Helou, Cici Xue

Last updated 2023-02-22

Agenda

Overview

- Data Lakes evolution
- Delta Lake Architecture
- Getting Started with Delta

(Deep) Dive

- **Delta features (MERGE INTO, Time Travel, Schema Evo…)**
- **Optimize, Vacuum**
- Change Data Feed
- Photon

Closing off

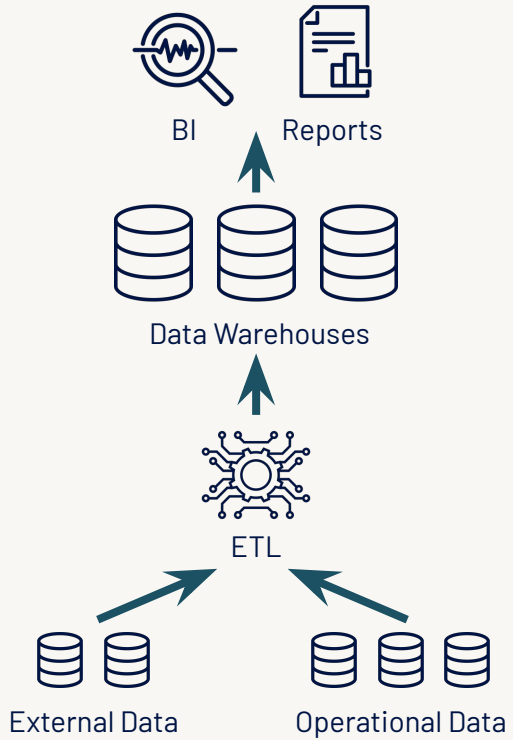
- Summary
- Questions
- Additional resources

Data Lakes Evolution

Data Warehouses were purpose-built for BI and reporting, however...

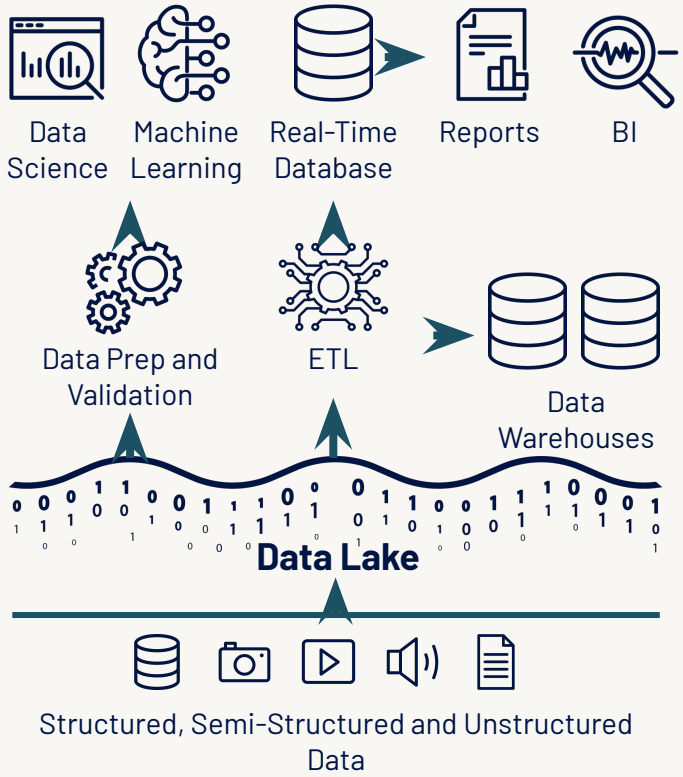
- No support for data science, ML
- Limited support for streaming
- Closed & proprietary formats

Therefore, most data is stored in data lakes & blob stores



Data Lakes
could handle all your data for data science and ML, however...

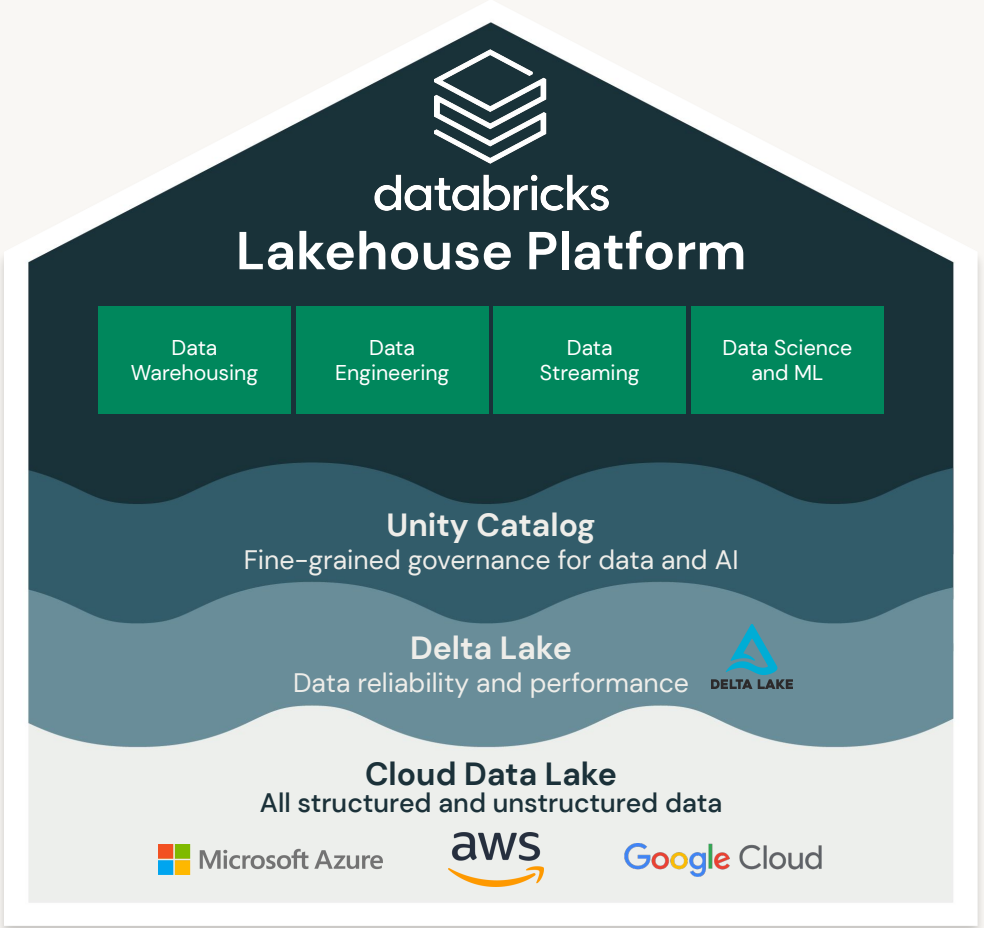
- Poor BI support
- Complex to set up
- Poor performance
- Unreliable data swamps



Delta Lake

An open, reliable, performant, and secure data storage and management layer for your data

- Open sourced by Databricks -- delta.io
- Fresh & reliable data with a single source of truth
- Data warehouse performance with data lake economics
- Advanced security and standards to meet compliance needs
- Builds on an open format (Parquet) and adds an open source transaction log



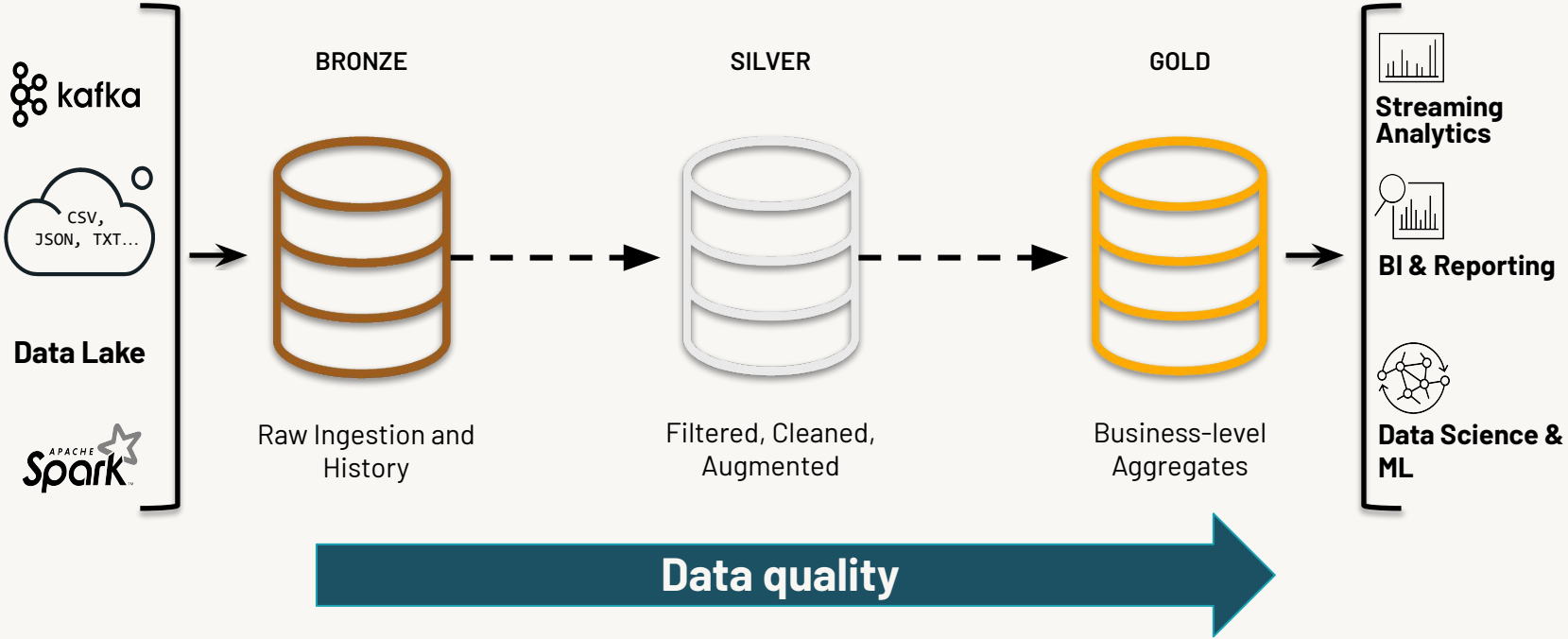
Open and agile

Leverage vast open-source ecosystem and **avoid vendor lock-in** with Delta Lake.

Standardize big data storage in your own ecosystem, in open Parquet format, that can be consumed by other systems outside of Databricks.

+

The Delta Lake – Multihop medallion architecture



Switching to Delta

Get Started with Delta using Spark APIs

Instead of **parquet**...

```
spark
    .read
    .format("parquet")
    .load("/data")
```

... simply say **delta**

```
spark
    .read
    .format("delta")
    .load("/data")
```

Using Delta with your Existing Parquet Tables

Step 1: Convert Parquet to Delta Tables

```
CONVERT TO DELTA parquet.`path/to/table` [NO STATISTICS]
[PARTITIONED BY (col_name1 col_type1, col_name2 col_type2, ...)]
```

Step 2: Optimize Layout for Fast Queries

```
OPTIMIZE events
WHERE date >= current_timestamp() - INTERVAL 1 day
ZORDER BY (eventType)
```

Get Started with Delta using Spark APIs

Instead of **parquet**...

```
CREATE TABLE ...
USING parquet
...

dataframe
    .write
    .format("parquet")
    .save("/data")
```

... simply say **delta**

```
CREATE TABLE ...
USING delta
…

dataframe
    .write
    .format("delta")
    .save("/data")
```


Get Started with Delta using DeltaTable APIs

Use DeltaTableBuilder API to create table through python

```
DeltaTable.create(spark)
  .tableName("<TableName>")
  .addColumn("<ColName>", "<DataType>")
  . . .
  .partitionedBy("<ColName>")
  .execute()
```

Get Started with Delta and Spark Streaming

Delta as **source** …

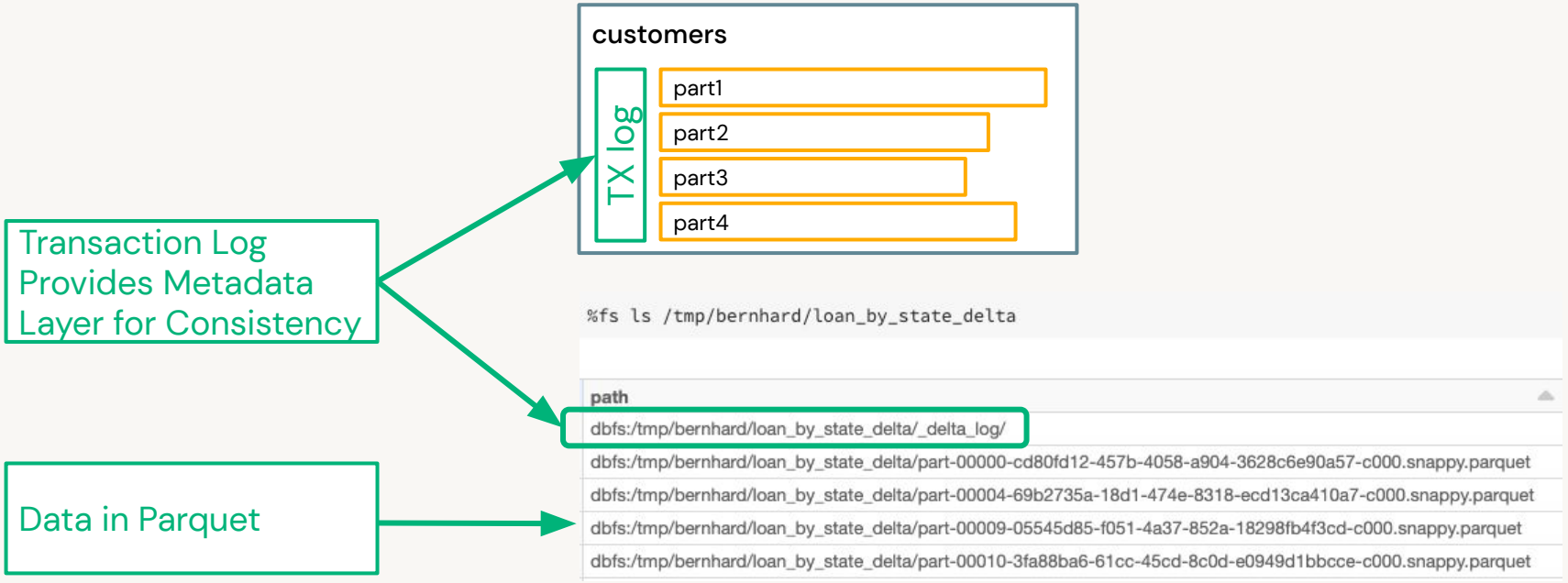
```
spark.readStream
     .format("delta")
     .option(...)
     .load("/data")
```

… and **sink**

```
events.writeStream
     .format("delta")
     .outputMode(...)
     .option(...)
     .start("/data")
```


Delta Lake Components

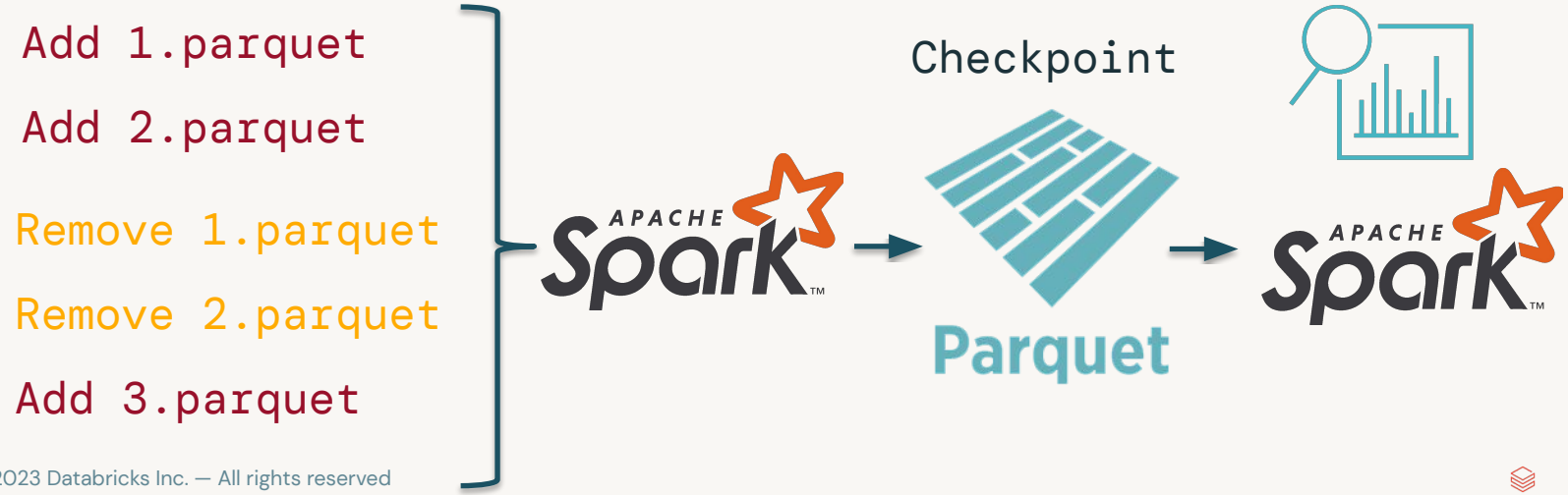
Delta Tables



https://databricks.com/blog/2019/08/21/diving-into-delta-lake-unpacking-the-transaction-log.html

Checkpoints as data

- Transaction log considered data
- JSON files will be checkpointed into parquet snapshots
- Transaction log is kept on cluster as cached RDD



Transaction Log / Metadata

Paquet Checkpoint

JSON Transaction

```
%fs ls /mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log
```

	path	name	size
3	dbfs:/mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log/.s3-optimization-2	.s3-optimization-2	0
4	dbfs:/mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log/00000000000000000000.checkpoint.parquet	00000000000000000000.checkpoint.parquet	28009
5	dbfs:/mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log/00000000000000000000.json	00000000000000000000.json	46221
6	dbfs:/mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log/00000000000000000001.crc	00000000000000000001.crc	94
7	dbfs:/mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log/00000000000000000001.json	00000000000000000001.json	5522
8	dbfs:/mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log/_last_checkpoint	_last_checkpoint	24

```
dfParquetTransaction = spark.read.parquet('/mnt/datalake/delta_workshop_data/lending_club_accepted_2007-2018/_delta_log/00000000000000000000.checkpoint.parquet')
display(dfParquetTransaction)
```

▸ (2) Spark Jobs
▸ dfParquetTransaction: pyspark.sql.dataframe.DataFrame = [txn: struct, add: struct ... 4 more fields]

	txn	add	remove	metaData	protocol	commitInfo
1	null	null	null	null	▸ {"minReaderVersion": 1, "minWriterVersion": 2}	null
	null	null	null	▸ {"id": "2739cf3a-a95b-4a56-9bd3-4dc7b7f19a84", "name": null, "description": null, "format": {"provider": "parquet", "options": {}}, "schemaString": "{\"type\":\"struct\",\"fields\":[{\"name\":\"id\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"member_id\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"loan_amnt\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},{\"name\":\"funded_amnt\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},{\"name\":\"funded_amnt_inv\",\"type\":\"double\",\"nullable\":true,\"metadata\":{}},{\"name\":\"term\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},	null	null

Scalable Metadata

Delta Features

How MERGE works

What feature are we talking about?

```
MERGE INTO customers     -- Delta table
USING source             -- Incremental dataset
ON customers.customerId = source.customerId
WHEN MATCHED AND customers.customerId < 100 THEN
    DELETE *
WHEN MATCHED THEN
    UPDATE SET address = source.address
WHEN NOT MATCHED
    THEN INSERT (customerId, address) VALUES
(source.customerId, source.address)
```


MERGE Basics

```
MERGE INTO my_table d USING my_view s ON condition
WHEN MATCHED AND d.status = 'CLOSED' THEN DELETE
WHEN MATCHED THEN UPDATE SET d.status = 'CLOSED'
WHEN NOT MATCHED THEN INSERT *
```

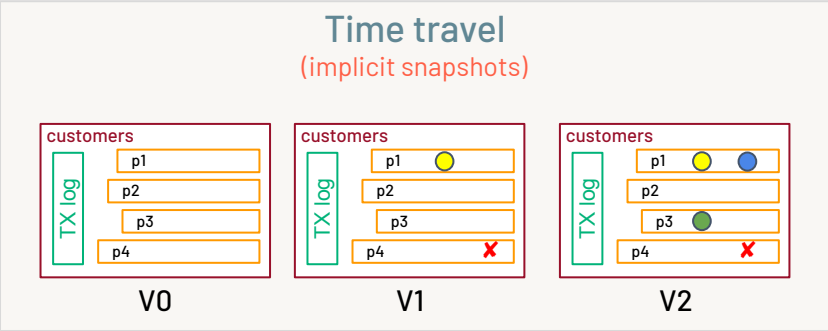
- Leverage data skipping (partition + stats) and figure out which files may match *predicate*
- Perform inner join and collect input_file_name for all rows that match *condition.* Ensure 1-1 mapping
- Rewrite files found above by updating the rows that match the *conditions* using a full outer join

Time Travel

Time Travel

Key Features

- Perform “point in time” queries on your data
- See how it has evolved over time



Syntax

```
SELECT *
FROM customers -- Delta table
TIMESTAMP AS OF
[timestamp_expression]
```

```
SELECT *
FROM customers -- Delta
table
VERSION AS OF [version]
```


AWS || Azure || GCP

RESTORE a Delta table

```
RESTORE TABLE delta.`path/to/source_table`
    TO TIMESTAMP AS OF <timestamp>
```

```
DeltaTable
    .forName("spark", "customers")
    .restoreToVersion(123)
```

Key Features

- Restores a Delta table to an earlier state.
- Restoring to an earlier version number or a timestamp is supported.
- Restoring a table to an older version where the data files were deleted manually or by vacuum will fail.

Time Travel use cases

Reproduce experiments & reports

```
SELECT count(*) FROM events
TIMESTAMP AS OF timestamp;

SELECT count(*) FROM events
VERSION AS OF version;
```

```
spark.read.format("delta").option("timestampAsOf",
timestamp_string).load("/events/")
```

Rollback accidental bad writes

```
RESTORE TABLE events
    TO TIMESTAMP AS OF 2020-01-18;

RESTORE TABLE events
    TO VERSION AS OF 123;
```


Delta Rapidstart workshop

Schema Enforcement and Evolution (drift) and Validation 101

Schema Enforcement and Evolution

- **Schema enforcement**
  - Default

- **Schema evolution (drift) -- multiple flavors**
  - Append a DataFrame with an evolving schema to a Delta table
  - Merge a DataFrame with an evolving schema to a Delta table
  - Overwrite a Delta table with data with an evolving schema

Schema evolution

Append a DataFrame with evolving schema to a Delta table

- **Simple use case for illustration**
  - There is an existing Delta table with 2 columns - id and name.
  - The latest DataFrame has 3 columns - id, name and year.
    - Change: a new column has been added to the dataset: year.
  - The requirement is to ***append*** the new data to the existing Delta table and also retain the new column in the Delta table.

```
df.write
   .format("delta")
   .mode("append")
   .option("mergeSchema", "true")
   .saveAsTable("my_delta_table")
```

Schema evolution

Merge a DataFrame with evolving schema to a Delta table

- **Simple use case for illustration**
  - There is an existing Delta table with 2 columns - id and name.
  - The latest DataFrame has 3 columns - id, name and year.
    - Change: a new column has been added to the dataset: year.
  - The requirement is to ***merge*** the new data to the existing Delta table and also retain the new column in the Delta table.

```
spark.conf.set("spark.databricks.delta.schema.autoMerge.enabled", "true")

MERGE INTO merge_target_table target    -- Delta table
USING source_table_for_merge source
ON target.id = source.id
WHEN MATCHED
     THEN UPDATE SET *
WHEN NOT MATCHED
     THEN INSERT *
```

Schema evolution

Overwrite a Delta table with a new DataFrame with evolving schema

- **Simple use case for illustration**
  - There is an existing Delta table with 3 columns - id, name and year.
  - The latest DataFrame has 2 columns - id and tech.
    - A column has been renamed: name to tech.
    - Another column has been dropped: year
  - The requirement is to write this data to the same table and ***overwrite the data and schema*** of the existing Delta table.

```
df.write
  .format("delta")
  .mode("overwrite")
  .option("overwriteSchema", "true")
  .saveAsTable("my_delta_table")
```

Schema Evolution workshop

Constraints

NOT NULL Constraint

```
CREATE TABLE events (
    id LONG NOT NULL,
    date STRING NOT NULL,
    location STRING,
    description STRING);


ALTER TABLE events_1 CHANGE COLUMN date DROP NOT NULL;
ALTER TABLE events_1 CHANGE COLUMN id   SET  NOT NULL;
```

Key Features

- Constraints are set at table schema level
- You can create or drop NOT NULL constraints using the ALTER TABLE CHANGE COLUMN command

CHECK Constraint

```
CREATE TABLE events (
    id LONG NOT NULL,
    date STRING NOT NULL,
    location STRING,
    description STRING);
```

```
ALTER TABLE events_2 ADD CONSTRAINT  dateWithinRange CHECK date > '1900-01-01';
ALTER TABLE events_2 DROP CONSTRAINT dateWithinRange;
```

Key Features

- ALTER TABLE ADD CONSTRAINT verifies that all existing rows satisfy the constraint before adding it to the table

OPTIMIZE, ZORDER & VACUUM

Partitioning Delta Tables

- Partition data to improve query performance
- Partition a column which has lower cardinality (For Higher Cardinationlity, use Z-Order instead)
- Each partition size to be at least 1 GB
- Don’t partition if total size is less than 1 TB

```
CREATE TABLE Table(ColA STRING, ColB STRING, ColC Date)
Using Delta
PARTITIONED BY(ColC)
Location <>
```

OPTIMIZE, ZORDER & VACUUM

- OPTIMIZE (with and without ZORDER) performs compaction, where possible to 1 GB file each and improves data skipping.

- Z-Ordering is a technique to colocate related information in the same set of files. Further improves data skipping.
  - Allows for sequentially efficient reads from cloud storage, decreasing total time to read data from storage into memory.

- VACUUM cleans up the untracked and redundant files to limit storage costs.

Compaction

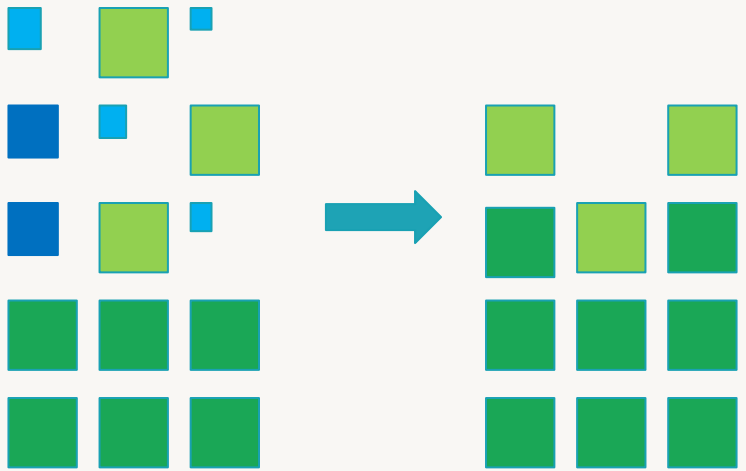
```
OPTIMIZE events -- Delta table
    ZORDER BY (eventType)
```

Key Features

- Solves the “small files problem”
- Improves data skipping
- Automatic (with conf setting)

How OPTIMIZE Works

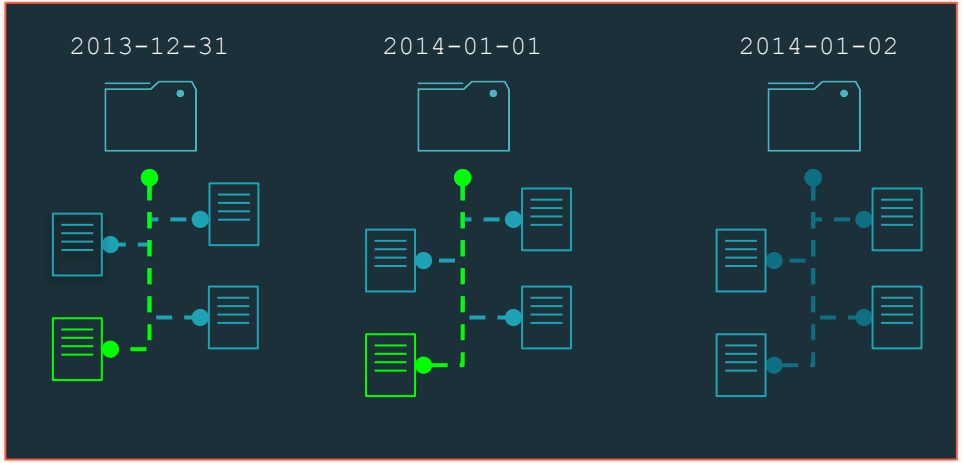
OPTIMIZE my_table



Z-Ordering

Z-Ordering is a multidimensional clustering technique, whereby statistically similar data is *co-located* in order to maximise the efficiency of data skipping, using Hibert Curves

```
SELECT ... FROM events
WHERE date_registered <=
2014-01-01
```



Key Features

- Safely skip more data
- Much faster queries
- Range partitions data

AWS || Azure || GCP

Z-Ordering

```
optimize table zorder by col
```

Old Layout

file_name	col_min	col_max
1.parquet	6	8
2.parquet	3	10
3.parquet	1	4

New Layout

file_name	col_min	col_max
1.parquet	1	3
2.parquet	4	7
3.parquet	8	10

Z-Ordering

```
select * from table where col = 7
```

Old Layout

file_name	col_min	col_max
1.parquet	6	8
2.parquet	3	10
3.parquet	1	4

New Layout

file_name	col_min	col_max
1.parquet	1	3
2.parquet	4	7
3.parquet	8	10

Z-Ordering Info and Tips

- Z-Ordering is pretty effective on up to 3 columns
- Z-Ordering creates 256MB - 1 GB files (by default, 32-64mb is usually better)
- Z-Order on columns commonly used in filters/where predicates

Table Size	File Size	Approx files in table
10GB	256MB	40
5TB	512MB	18000
10TB	1GB	25000

Databricks Delta Lake and Stats

- Databricks Delta Lake collects stats about the first N columns
  - `dataSkippingNumIndexedCols = 32`
- These stats are used in queries
  - Metadata only queries: `select max(col) from table`
    - Queries just the Delta Log, doesn't need to look at the files if `col` has stats
  - Allows us to skip files
    - Partition FIlters, **Data Filters**, Pushed Filters apply in that order
  - TimeStamp and String types aren't always very useful
    - Precision/Truncation prevent exact matches, have to fall back to files sometimes
- Avoid collecting stats on long strings
  - Put them outside first 32 columns or collect stats on fewer columns
    - `alter table change column col after col32`
    - `set spark.databricks.delta.properties.defaults.dataSkippingNumIndexedCols = 3`

Scheduling OPTIMIZE

- OPTIMIZE (with and without ZORDER) merges small files into larger files. It makes heavy use of shuffling under the hood and creates big in-memory partitions
- Recommended cluster configuration: compute optimized as it does large amounts of Parquet decoding and encoding; autoscaling to adapt on spikes in data volume day over day
- Recommended time schedule: Align with frequency of change, this is a cost/performance trade-off. Typically at least once per day
- Use a separate cluster (it can be quite intensive and you do not want to influence other jobs).
  - More so, if the jobs have strict SLAs and there are downstream pipelines waiting on the completion of these jobs.

Scheduling VACUUM

- VACUUM is not very intensive on the cluster. The main task is file listing (this is done recursively and in parallel on the workers). Needs to be triggered explicitly.
- Recommended cluster configuration: cheap auto scaling cluster 0-4 worker nodes. Most of the time is spent on waiting for the cloud storage to return which files exist.
- Recommended time schedule: Depends on how far you want to go back in time with time travel (30 days is the maximum by default).
  - This is a storage cost/convenience trade off. It is recommended to keep files at least for a week to guarantee snapshot isolation.
- Use a separate cluster to not influence other jobs

OPTIMIZE vs. ZORDER vs. VACUUM

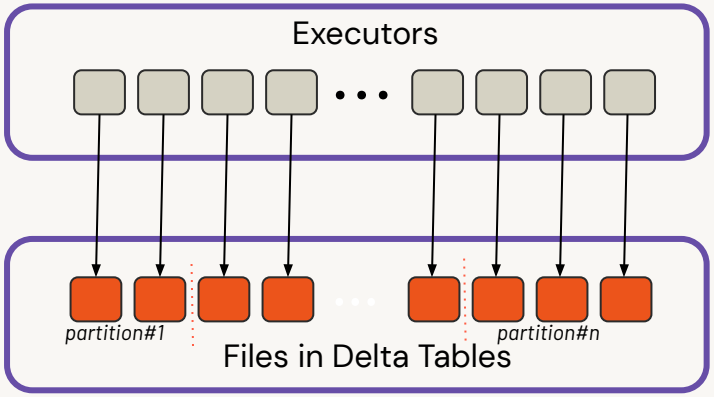
	OPTIMIZE	ZORDER	VACUUM
Also known as	Bin packing or compaction	Multi-dimensional clustering	Clean up untracked files
Goal	Evenly-balanced data files with respect to their size on disk (not number of tuples per file)	Evenly-balanced data files with respect to the number of tuples (not data size on disk)	Delete old/redundant untracked files of Delta table
Idempotency (run multiple times)	Idempotent: if run twice, latter has no effect	Not Idempotent: incremental operation (e.g. for new partitions)	Not Idempotent: if run multiple times consecutive days. Cleans up untracked files based on the retention specified
Makes changes to data in the Delta Table	No	No	No
Makes changes to the data layout in the Delta Table	No	Yes	No
Affects downstream readers	No	No	Yes: if retention specified is too low else No
Concurrency issues	Can conflict in Merge, Update, Delete, OPTIMIZE	Can conflict in Merge, Update, Delete, OPTIMIZE	Cannot conflict unless the retention specified is too low

Auto Optimize

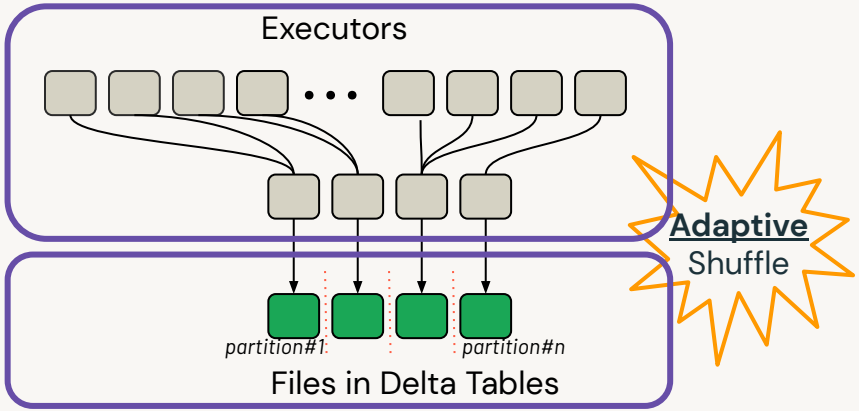
Auto Optimize

Automatically compact small files during individual writes to a Delta table

Traditional Writes



Optimized Writes



***NOTE**: Auto Optimize performs compaction only on small files; doesn't Z-Order the files.*

Auto Optimize – How it works

- **For existing tables:**

```
%sql
ALTER TABLE [table_name | delta.`<table-path>`]
SET TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true,
                   delta.autoOptimize.autoCompact = true);
```

- **To ensure all new Delta tables have these features enabled:**

```
%sql
SET spark.databricks.delta.properties.defaults.autoOptimize.optimizeWrite = true;
SET spark.databricks.delta.properties.defaults.autoOptimize.autoCompact = true;
```

- **This feature can be enabled / disabled for a SparkSession with config:**

```
spark.databricks.delta.optimizeWrite.enabled
spark.databricks.delta.autoCompact.enabled
```

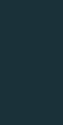

AWS || Azure || GCP

OPTIMIZE vs. Auto Optimize

	OPTIMIZE	Auto Optimize
Default file size	1 GB	128 MB
Performed on	Data on disk	Data in flight (during individual DF writes)
Z-ORDERing	Yes	No
Mode of invocation	Explicit (could be scheduled)	Implicit (automatic once enabled)

OPTIMIZE/ZORDER/Auto-Optimize workshop

Change Data Feed



How Does Delta Change Data Feed Work?

Original Table (v1)

PK	B
A1	B1
A2	B2
A3	B3

Change data (Merged as v2)

PK	B
A2	Z2
A3	B3
A4	B4

Change Data Feed Output

PK	B	Change Type	Time	Version
A2	B2	Preimage	12:00:00	2
A2	Z2	Postimage	12:00:00	2
A3	B3	Delete	12:00:00	2
A4	B4	Insert	12:00:00	2

A1 record did not receive an update or delete.
So it will not be output by CDF

Typical use cases

Silver & Gold Tables

Improve Delta performance by processing only changes following initial MERGE comparison to accelerate and simplify ETL/ELT operations

Materialized Views

Create up-to-date, aggregated views of information for use in BI and analytics without having to reprocess the full underlying tables, instead updating only where changes have come through

Transmit Changes

Send Change Data Feed to downstream systems such as Kafka or RDBMS that can use it to incrementally process in later stages of data pipelines

Audit Trail Table

Capturing Change Data Feed outputs as a Delta table provides perpetual storage and efficient query capability to see all changes over time, including when deletes occur and what updates were made

Getting started with Delta Change Data Feed

Enable CDF on TABLE…

```
ALTER TABLE ...
SET TBLPROPERTIES
(delta.enableChangeDataFeed =
true);
```

in SQL, Python, or Scala

… or Establish CDF on CLUSTER

```
spark.conf.set("spark.databric
ks.delta.properties.defaults.e
nableChangeDataFeed", "true")
```

in Python, or Scala

Using Delta Change Data Feed

Query changes…

```
SELECT … FROM
table_changes('tableName',
    startingVersion
    [,endingVersion])
or
SELECT … FROM
table_changes('tableName',
    'startingTimestamp'
    [,'endingTimestamp'])
```

… and store them

```
INSERT INTO TABLE ...
USING delta ...
as
SELECT … FROM
table_changes(...)
```

in SQL, Python, or Scala

Getting started with Delta Change Data Feed

```
1  %sql
2  SELECT * FROM table_changes('silverTable', 2, 4) order by _commit_timestamp
```

▸ (1) Spark Jobs

	Country	NumVaccinated	AvailableDoses	_change_type	_commit_version	_commit_timestamp
1	Australia	100	3000	insert	2	2021-04-12T20:48:05.000+0000
2	USA	10000	20000	update_preimage	3	2021-04-12T20:48:08.000+0000
3	USA	11000	20000	update_postimage	3	2021-04-12T20:48:08.000+0000
4	UK	7000	10000	delete	4	2021-04-12T20:48:11.000+0000

Showing all 4 rows.

When to Use Delta Change Data Feed

- Delta changes include updates and/or deletes
- Small fraction of records updated in each batch
- Data received from external sources is in CDC format
- Send data changes to downstream application

- Delta changes are append only
- Most records in the table updated in each batch
- Recreation or regeneration of past partitions
- Data received comprises destructive loads
- Find and ingest data outside of the Lakehouse

Photon

Photon is Databricks' next generation query engine, built to be faster than anyone else

100% APACHE SPARK COMPATIBLE query engine

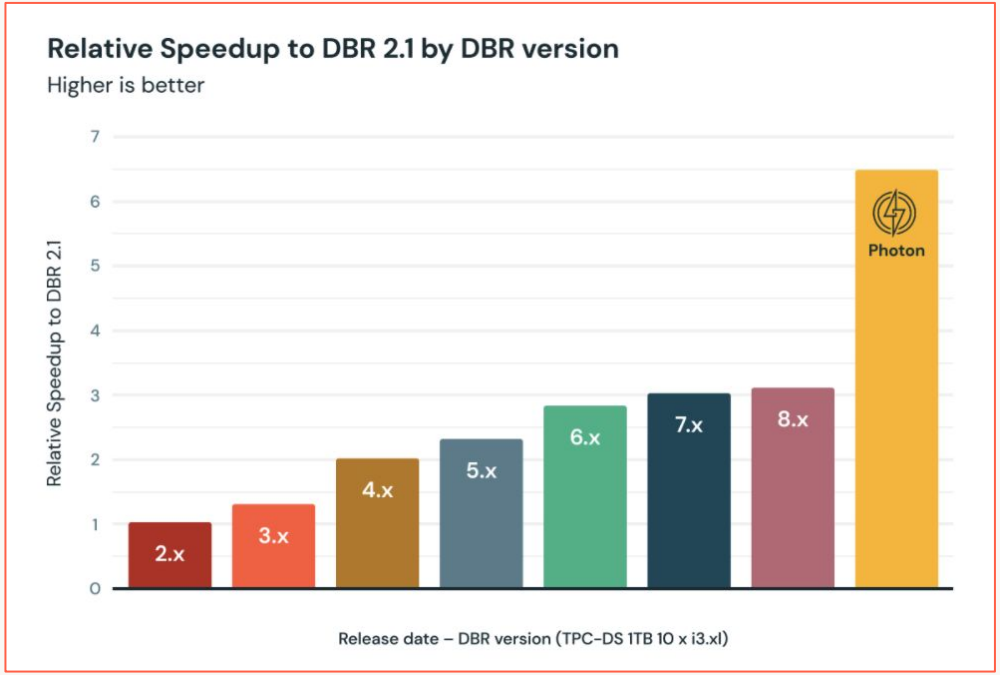
Built from the ground up to deliver the FASTEST PERFORMANCE

FOR ALL DATA USE CASES
across data engineering, data science,
machine learning, and data analytics.

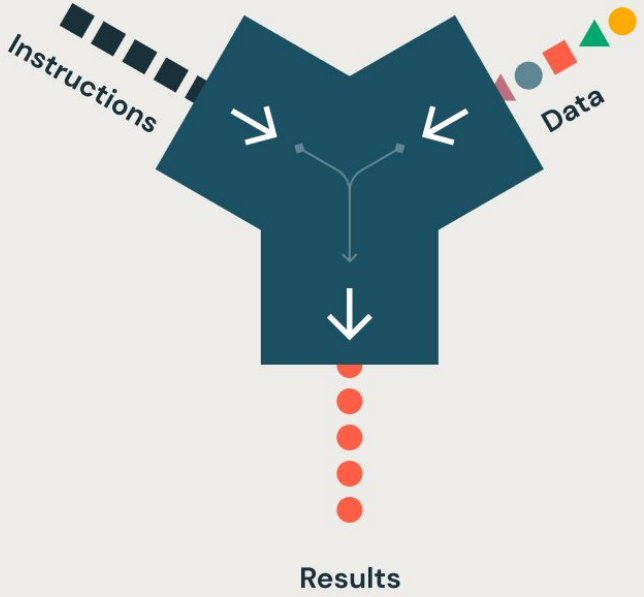
Photon

Dramatically faster query engine with zero tuning or setup - only on Databricks

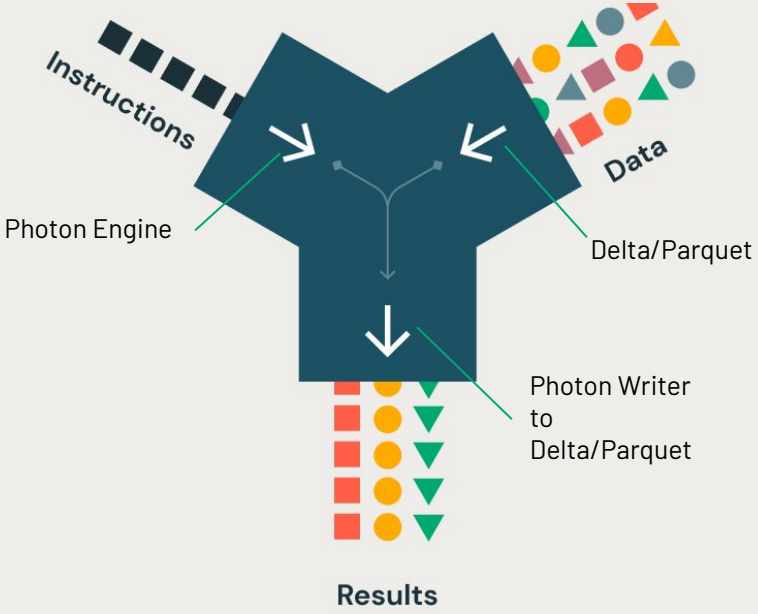
- Fast query performance
  - Built for modern hardware with up to 6x better price/perf compared to other cloud data warehouses
- No code changes
  - Apache Spark compatible APIs mean existing code works as-is
- Broad language support
  - Support for languages beyond SQL including Python, Scala, R, Java



Spark Instructions



Photon Instructions

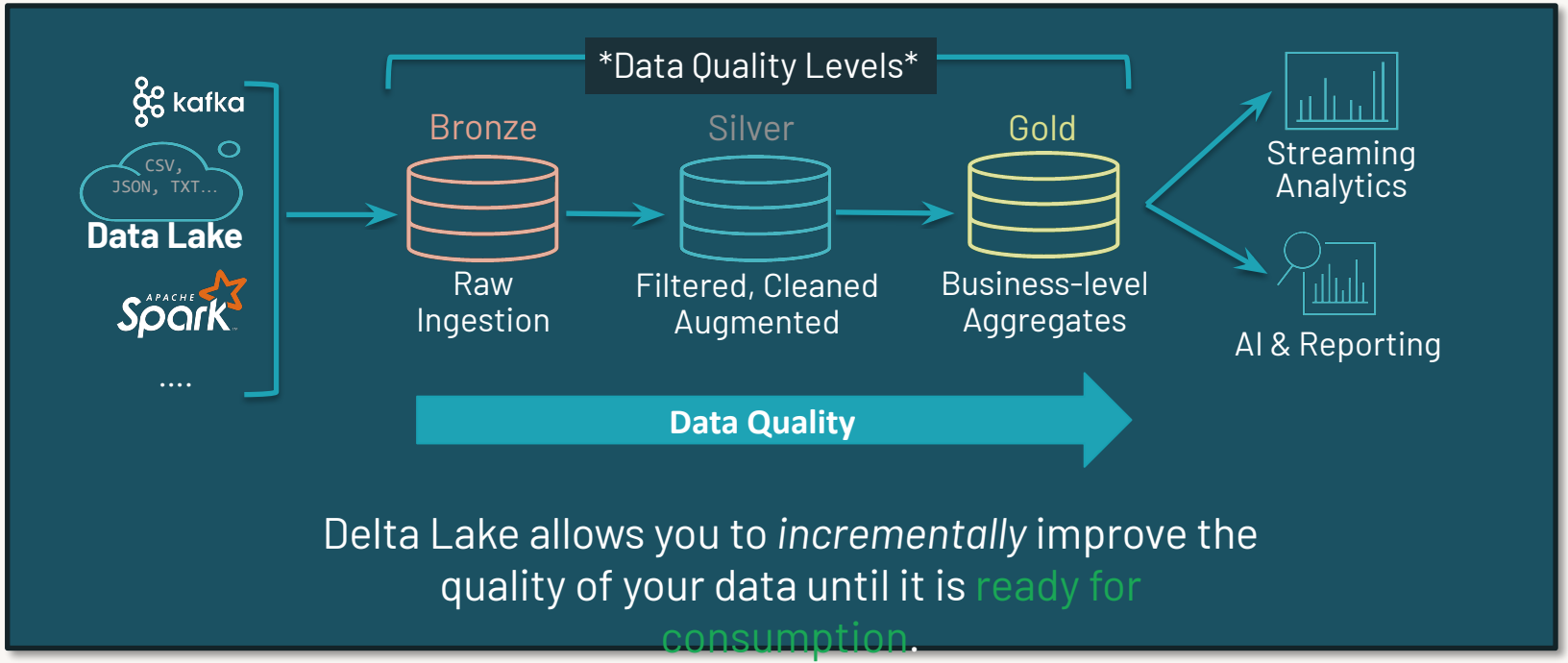


Delta is better with Photon

Use case	Without Photon	With Photon
Tiny files	112 minutes	16 minutes
Join	43 minutes	18 minutes
Wide Table (>100 columns)	120 minutes	7 minutes

- All the above benchmarks were done with the same cluster size for each use case
- All the above benchmarks were done based on reading and writing data in delta format
- Window functions and UDFs are not currently supported by Photon ⇒ fall back to standard Spark

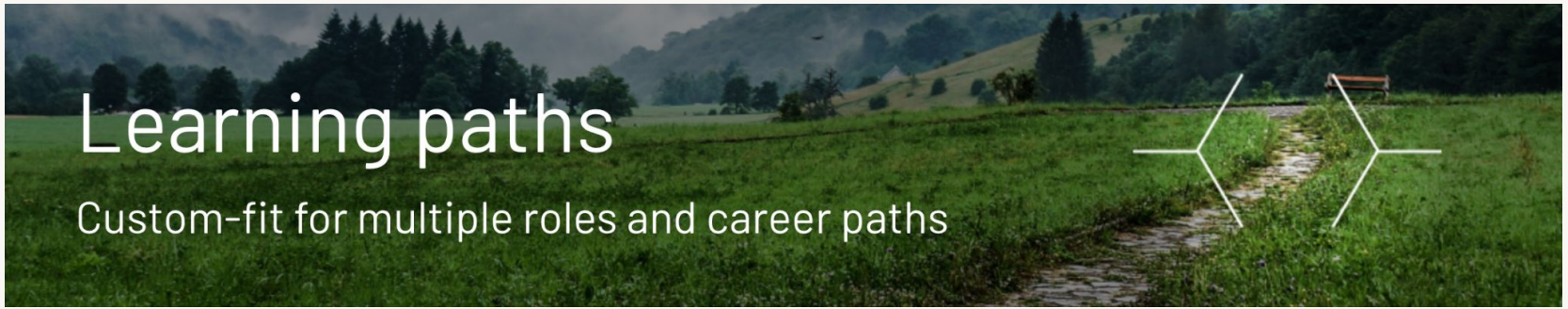
Summary, Questions, Discussion…



DELTA LAKE

Additional resources

Self-paced Courses

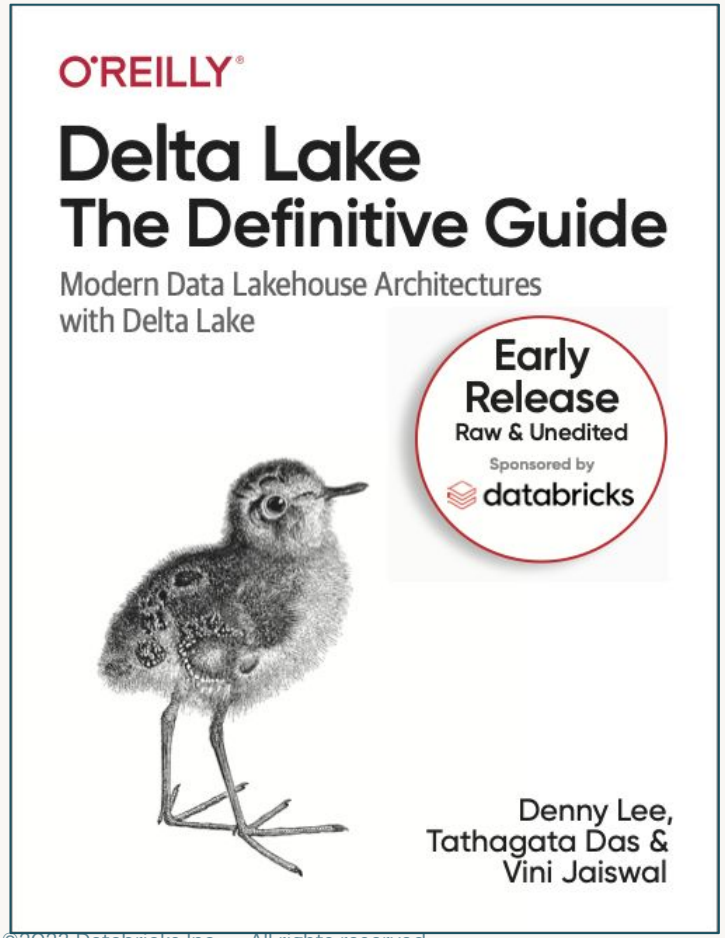


https://databricks.com/learn/training/learning-paths
https://databricks.com/learn/certification

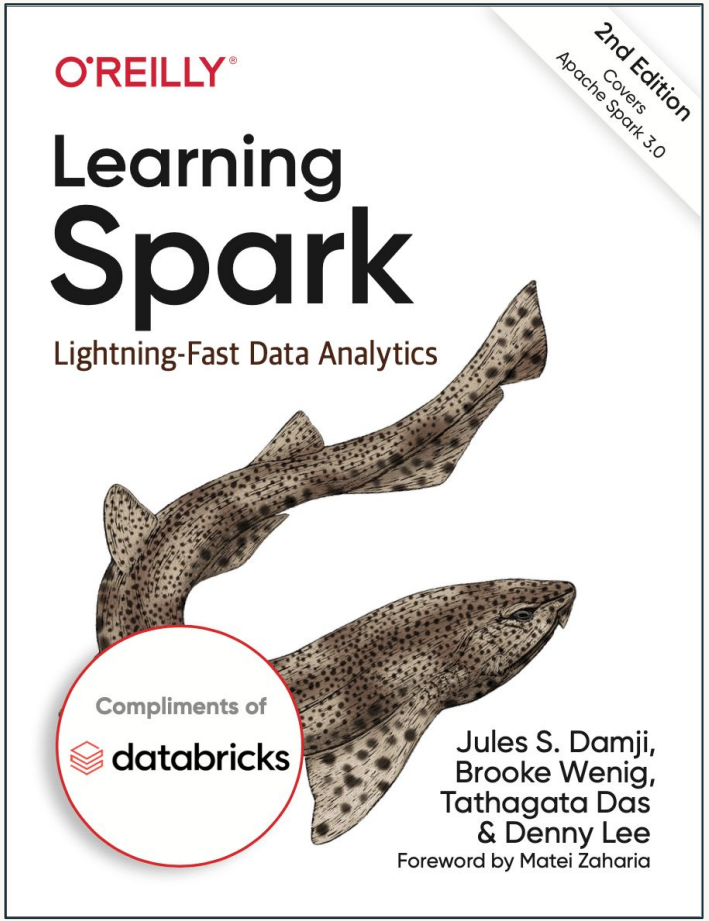
Free for all Databricks customers

Learning Paths for All Roles

http://dbricks.co/dldg



https://dbricks.co/ebLearningSpark



Getting Started: Top 10 Delta Lake assets

Asset	Type	Audience	Teaser
Top 5 Reasons to Convert Your Cloud Data Lake to Delta Lake	Blog	Practitioner	Check out this blog to learn about the top reasons customers convert to Delta Lake: Prevent Data Corruption, Faster Queries, Increase Data Freshness, Reproduce ML Models, Achieve Compliance
Delta Lake Explainer Video	Video	DM	Watch this short video to learn what Delta Lake is and how it can help simplify your architecture.
Spark + AI Summit Keynote 2020	Virtual Event	DM	Hear how Delta Lake is the best foundation for your lakehouse from Databricks CEO, Ali Ghodsi in the 2020 DAIS keynote.
Demystifying Delta Lake	Podcast	DM	Listen to this episode in the Data Brew Podcast featuring an interview with Michael Armbrust, who leads the team at Databricks that designed and built Structured Streaming and Delta Lake.
How Apache Spark 3.0 and Delta Lake Enhance Data Lake Reliability	Webinar	Practitioner	Watch this webinar to hear how Delta Lake boosts performance and reliability to ensure successful data analytics and machine learning projects for downstream data teams.
Delta Lake Demo	Demo	Practitioner	Check out this demo of the main features of Delta Lake, including unified batch and streaming data processing, schema enforcement and evolution, time travel, and support for UPDATEs/MERGEs/DELETEs, as well as some of the performance enhancements available with Delta Lake on Databricks.
The Delta Lake Series	eBook Series	Practitioner/ DM	Check out this ebook series, where we explore how Delta Lake brings quality, reliability, security, and performance to your data lake to enable a lakehouse architecture in a series of 5 ebooks.
O'Reilly Survey: Nine Out of Ten Enterprises Want Simpler Data Platform Architectures	3rd Party Asset	DM	Read the survey conducted by O'Reilly that uncovers the current challenges of cloud data platforms, the benefits and drawbacks of certain architectures, and the key things to consider when evaluating a solution.
Building Reliable Data Lakes with Delta Lake	Virtual Hands-On Lab	Practitioner	Check out this Delta Lake Virtual Hands-on-Lab that gives a deeper dive into what Delta Lake is, shares how to build highly scalable and reliable data pipelines with Delta Lake, and has a hands-on walkthrough.
Fundamentals of Delta Lake (paid)	Databricks Academy	Practitioner	Consider taking this Databricks Academy course that explores the fundamental concepts behind Delta Lake.

Delta Lake ecosystem

- Delta Standalone Readers
  - GitHub Repo and Blogpost
- Databricks SQL
  - Keynote and Blogpost
- Delta Sharing
  - Keynote and Blogpost
- Delta Cheatsheet

Docs and Talks

Documentation and best practices references

- Delta lake Guide (Azure, AWS, GCP)
- Delta Open Source Project
- Various optimizations in Delta Lake (Azure, AWS, GCP)
- Delta Lake Concurrency Control (Azure, AWS, GCP)
- Z-order curve
- Delta Streaming reads and writes(Azure, AWS, GCP)
- Delta Lake MERGE(Azure, AWS, GCP)
- Auto Loader (Azure, AWS, GCP)

Talks

- Diving into Delta Lake 2.0
- Delta Lake, the Foundation of Your Lakehouse
- Ensuring Correct Distributed Writes to Delta Lake in Rust with Formal Verification
- A Modern Approach to Big Data for Finance
- Designing and Building Next Generation Data Pipelines at Scale with Structured Streaming
- Apache Spark Core – Practical Optimization
- Tech Talk series: Diving into Delta Lake

Blogs

- Guide to Delta Lake Sessions at Data + AI Summit 2022
- Auto Loader
- 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
Tech Talk | Diving into Delta Lake Part 1: Unpacking the Transaction Log
Tech Talk | Diving into Delta Lake Part 2: Enforcing and Evolving the Schema
Tech Talk | Diving into Delta Lake Part 3: How do DELETE, UPDATE, and MERGE work
Tech Talk | Top Tuning Tips for Spark 3.0 and Delta Lake on Databricks
Tech Chat | Slowly Changing Dimensions (SCD) Type 2

Thank you