Data Saturday Slovenia 2023

Sponsors

Kamil Nowinski

Delta Lake Tables 101

Kamil Nowiński

Microsoft Data Platform **MVP**

Analytics Architect, Speaker, blogger, data enthusiast

Group Manager at Avanade UK&I

>20 yrs of experience as DEV/BI/(DBA)

Founder of blog AzurePlayer

GitHub: #adftools, SCD Merge Wizard and more…

Member of the Data Community PL

Former co-organizer of SQLDay (PL), volunteer at SQLBits

SQL Server Certificates:

MCITP, MCP, MCTS, MCSA, MCSE Data Platform,

MCSE Data Management & Analytics, DevOps Expert

Moreover: Bicycle, Running, Digital photography

@NowinskiK, @Azure_Player

Blog

Azure Player

Play with data & have fun!

- Technical posts
- Various skill level
- Cheet sheets
- Recommended books
- Many useful other links
- Interviews (Podcast)
- YouTube Channel: www.AzurePlayer.net/YouTube

www.AzurePlayer.net

Scan me

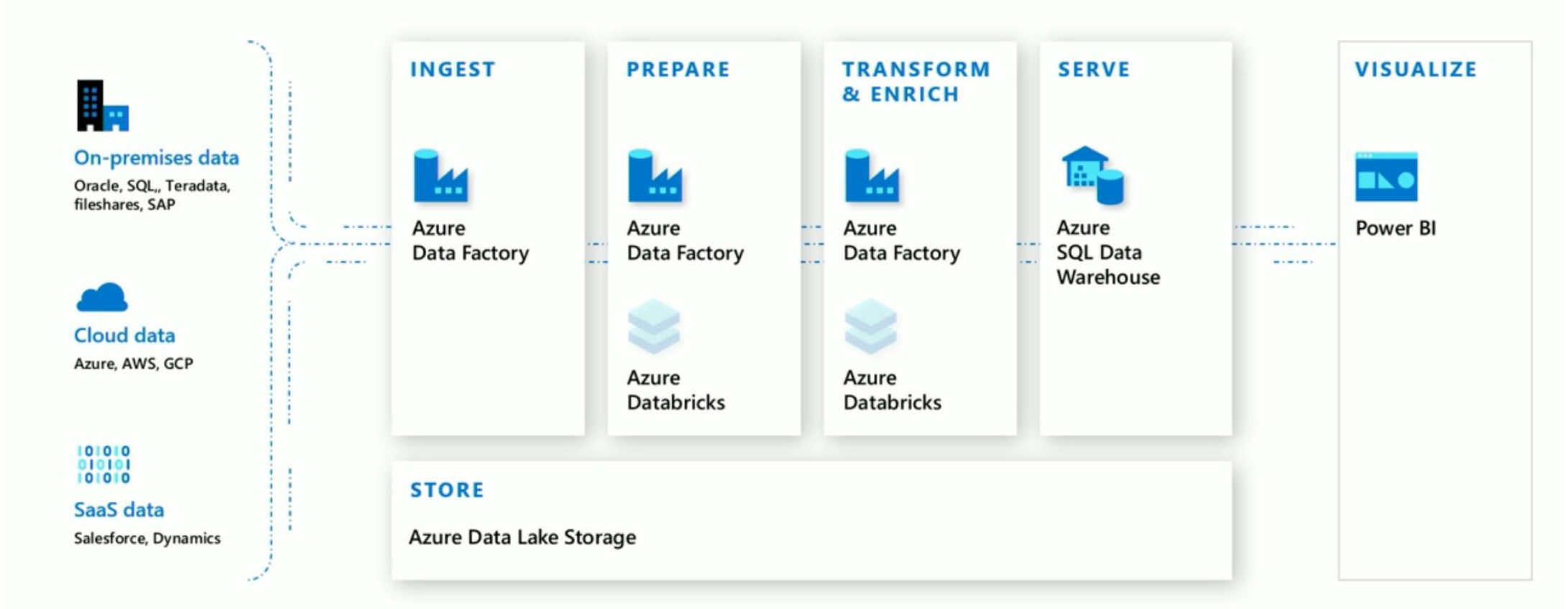
Slides available:

https://azureplayer.net/slides

AGENDA

- A bit of theory…
- … and history
- Key features of Delta Lake Tables
- Let's practice: DEMO time
    - Delta Lake Tables in Lake Databases (Databricks)
- Key takeaways

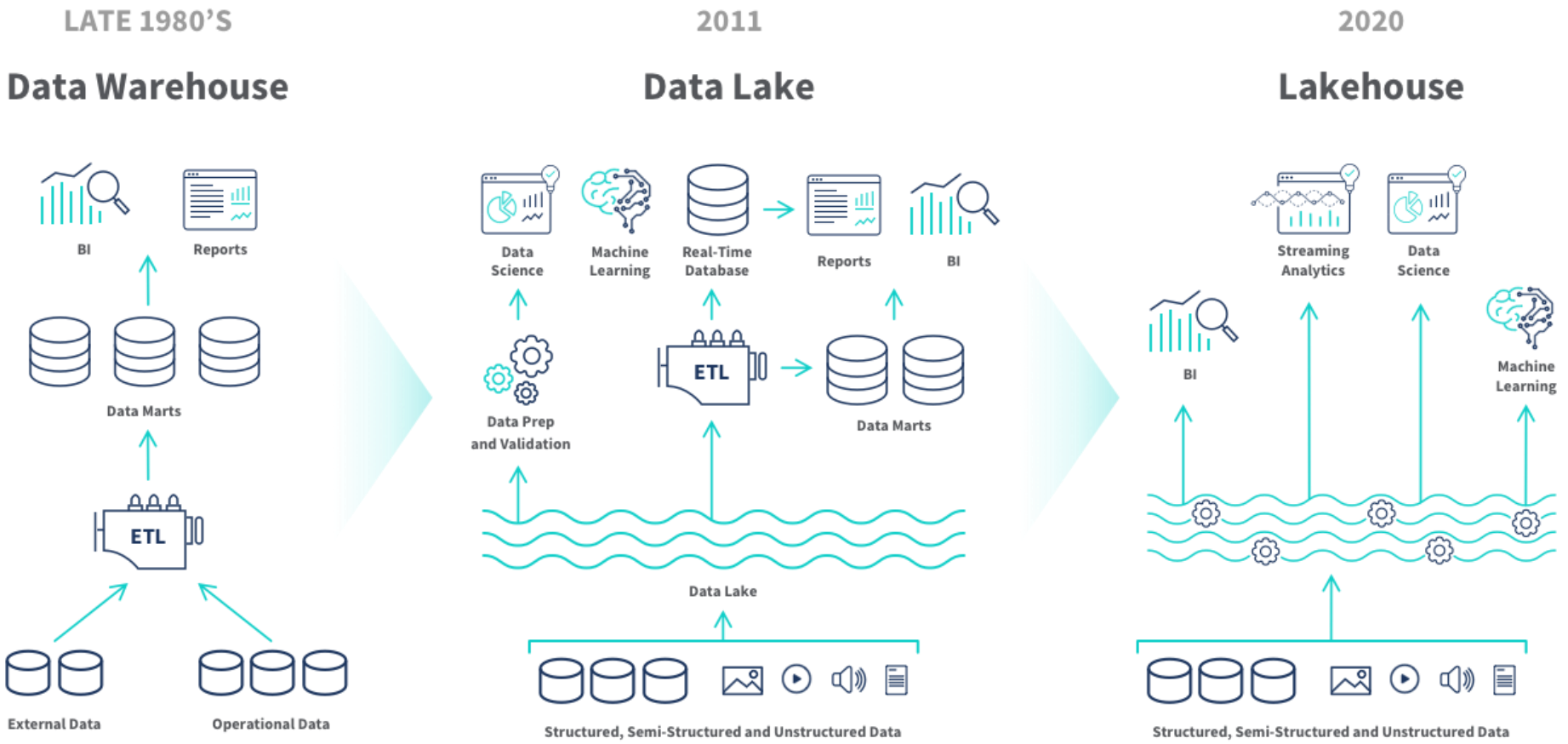
Modern Data Warehouse



Data Lake + Warehouse = Lakehouse

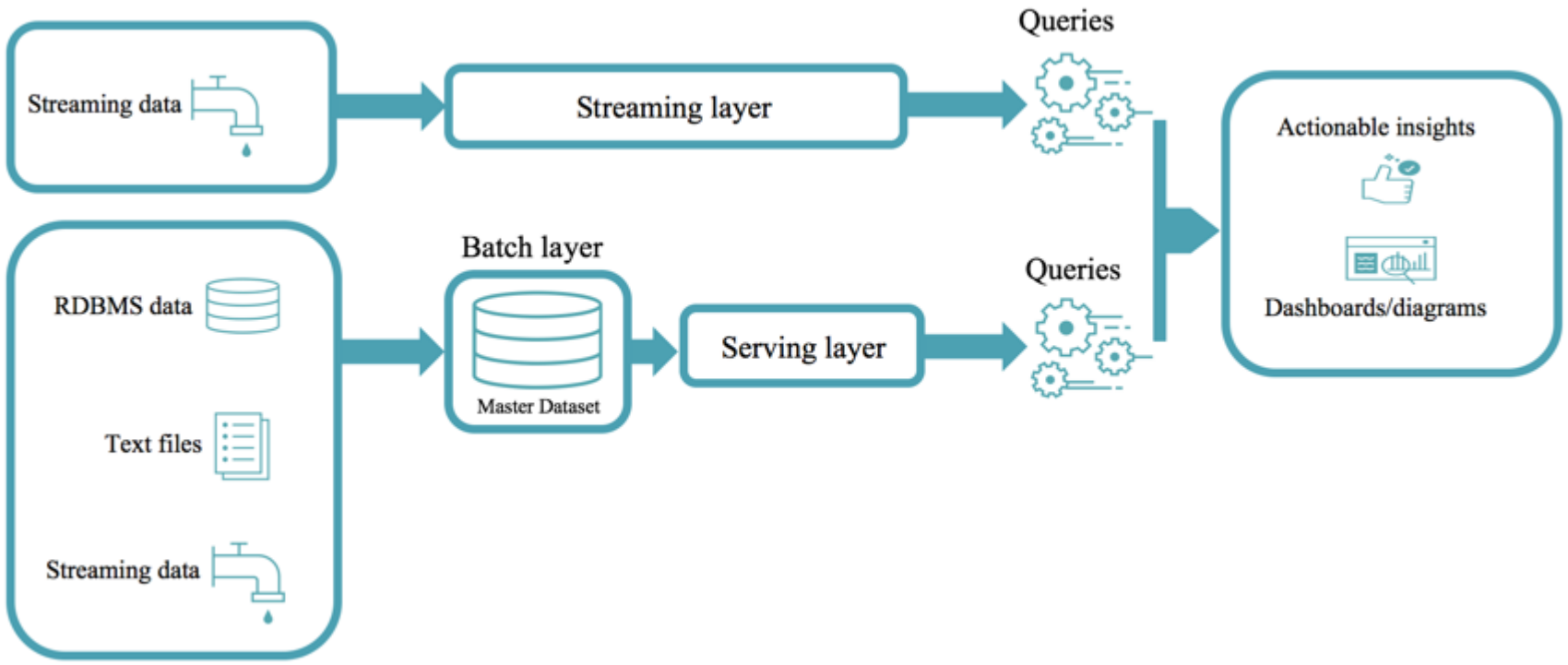
[> What Is a Lakehouse?]

Data Warehouse evolution



https://www.qlik.com/us/data-lake/data-lakehouse

Lambda Architecture



Parquet format

What is Parquet?

Apache Parquet is an open source, column-oriented data file format designed for efficient data storage and retrieval. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk. Apache Parquet is designed to be a common interchange format for both batch and interactive workloads.

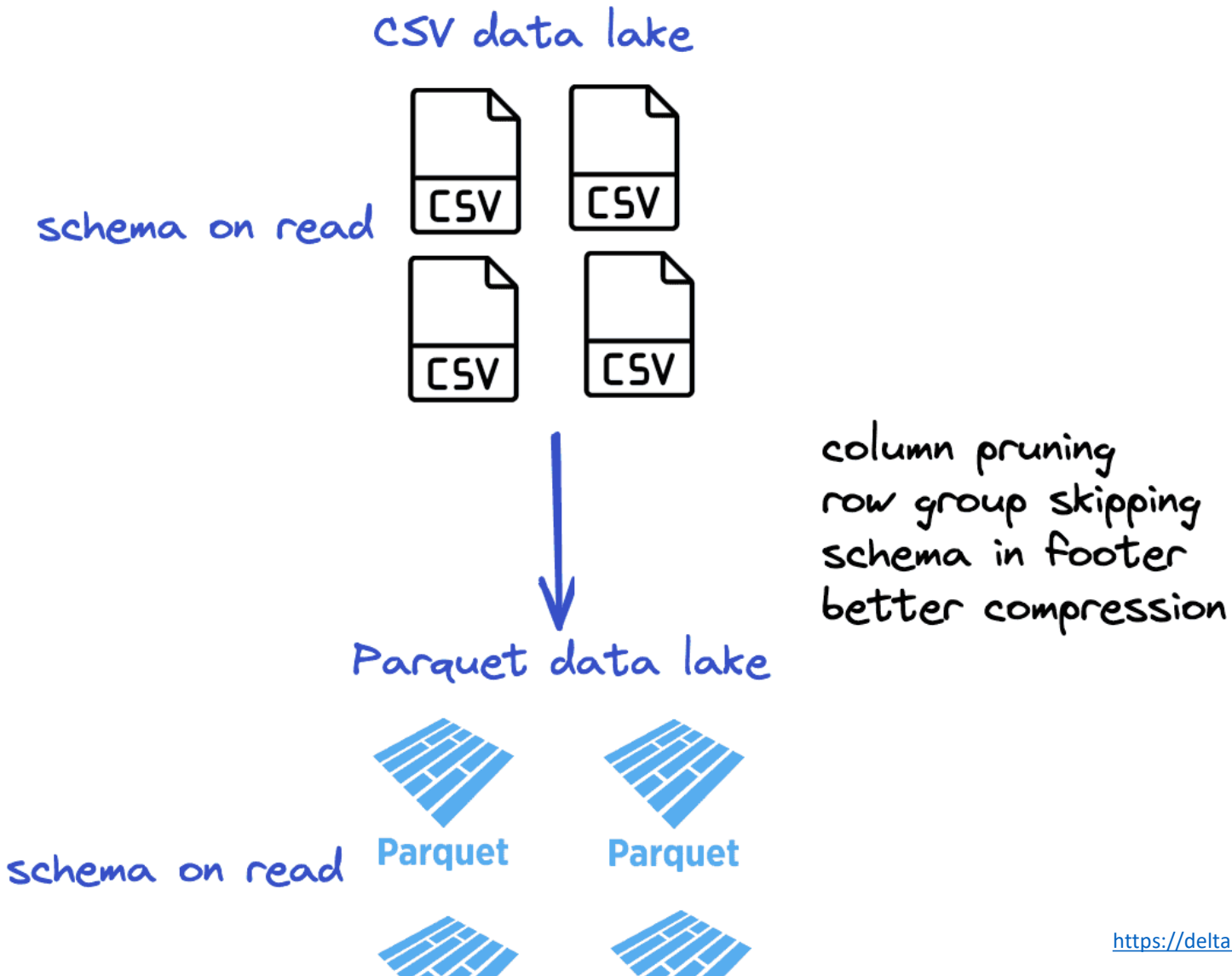
Characteristics of Parquet:

- **Free and open-source** file format
- Language **agnostic**
- Column-based format
- Used for analytics (**OLAP**) use cases
- **Highly efficient** data compression and decompression
- **Supports complex data types** and advanced nested data structures

Problems with Parquet format

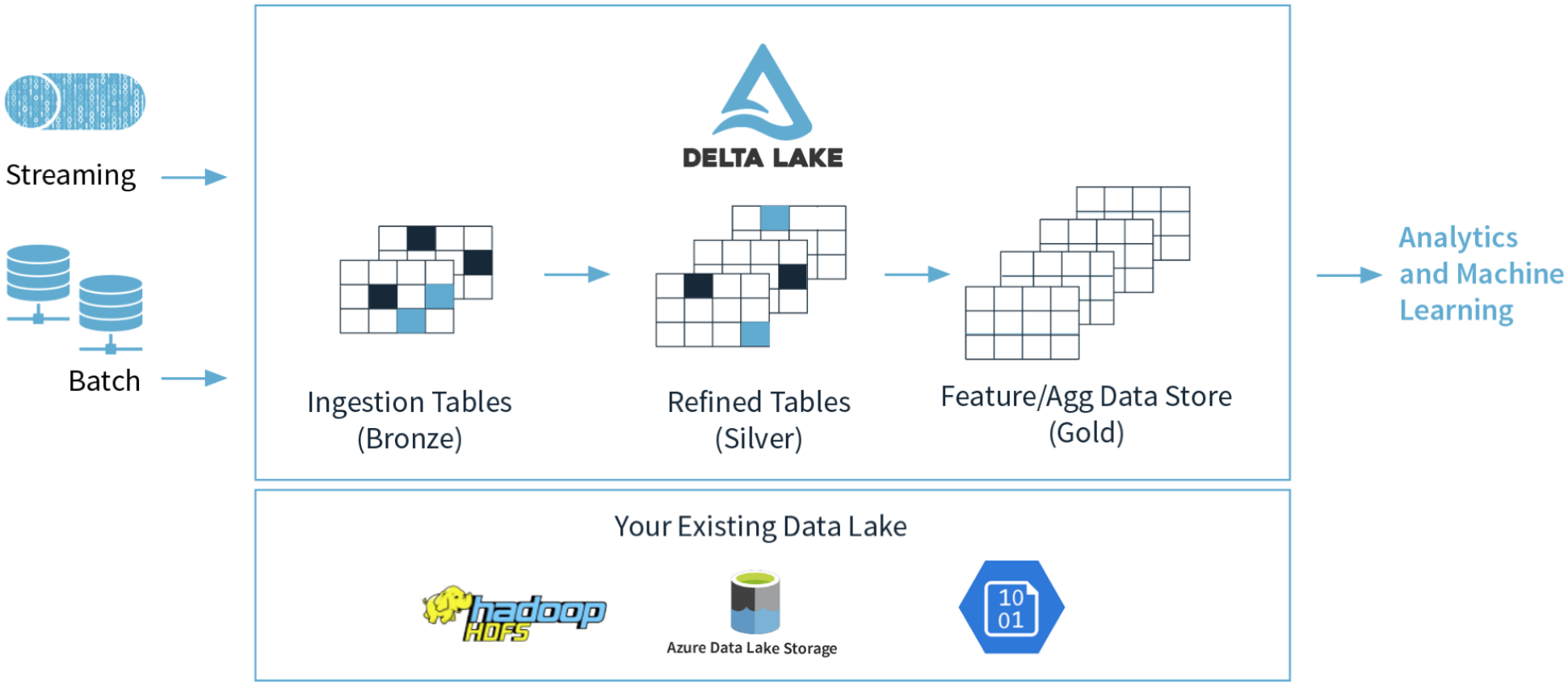
- No schema enforcement
- Updates – rewrite entire file/all files
- Transactions inconsistency
- No interoperability between batch & streaming workloads
- No versioning

Advantages of Delta Lake over CSV



https://delta.io/blog/2023-03-22-convert-csv-to-delta-lake/

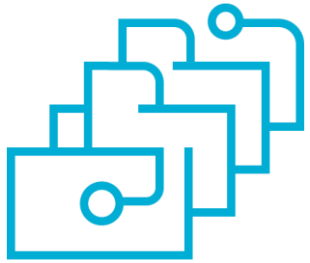
Medallion Architecture



Delta Lake – Key Features

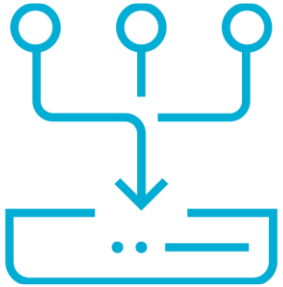
ACID Transactions

Scalable Metadata

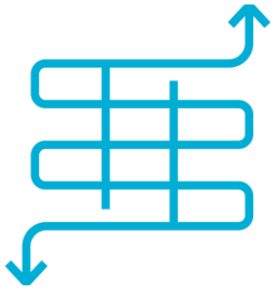


Time Travel

Open Source

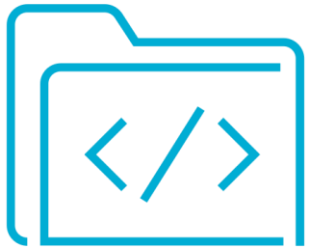


Unified Batch/Streaming



Schema Evolution / Enforcement

Audit History

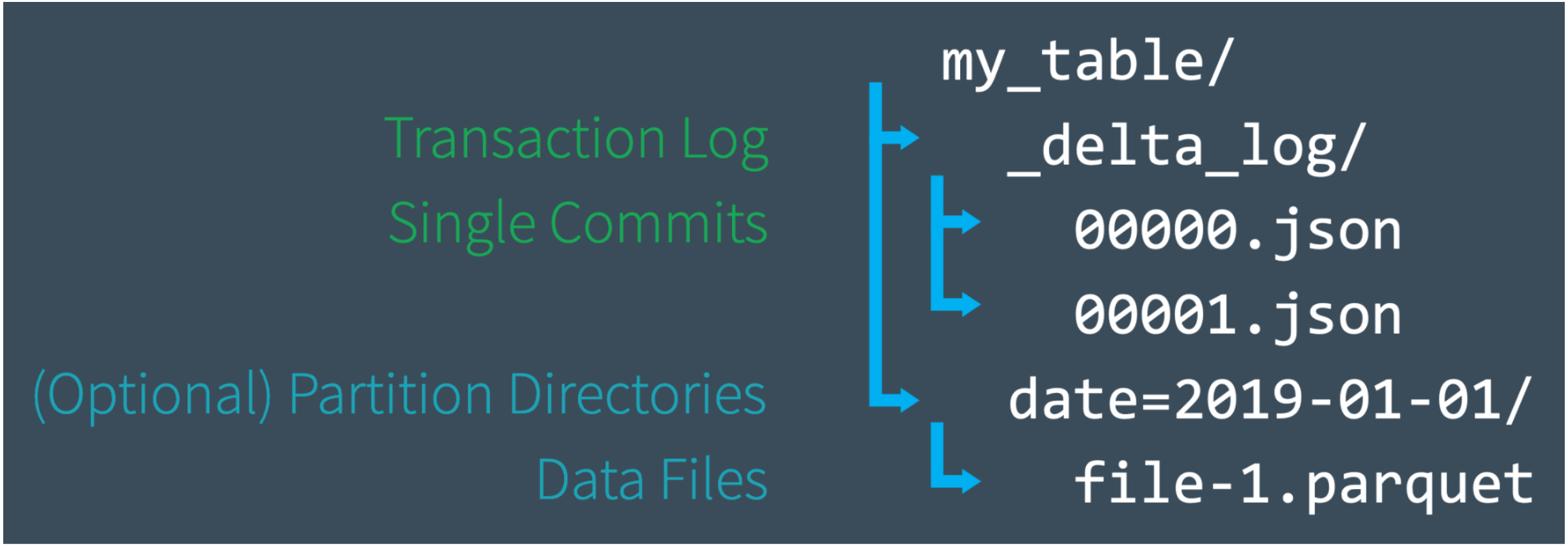


DML Operations

CONVERT

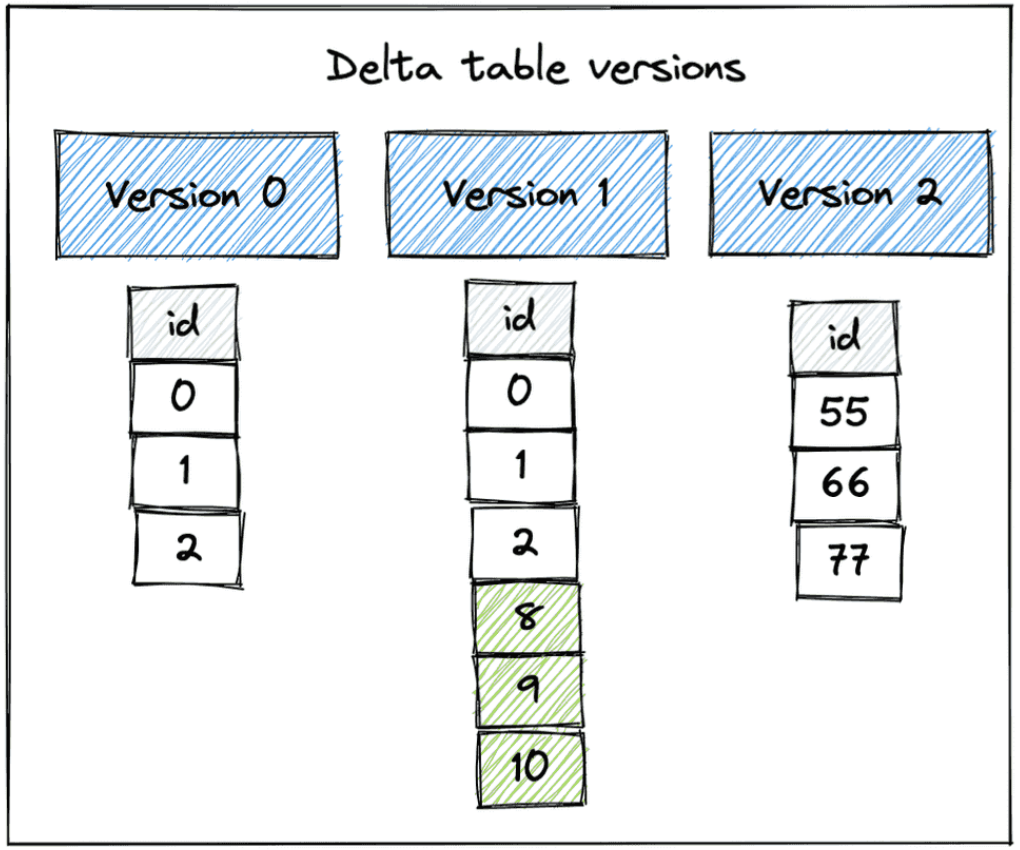

```
CONVERT TO DELTA database_name.table_name; -- only for Parquet tables

CONVERT TO DELTA parquet.`s3://my-bucket/path/to/table`
  PARTITIONED BY (date DATE); -- if the table is partitioned

-- Uses Iceberg manifest for metadata
CONVERT TO DELTA iceberg.`s3://my-bucket/path/to/table`;
```

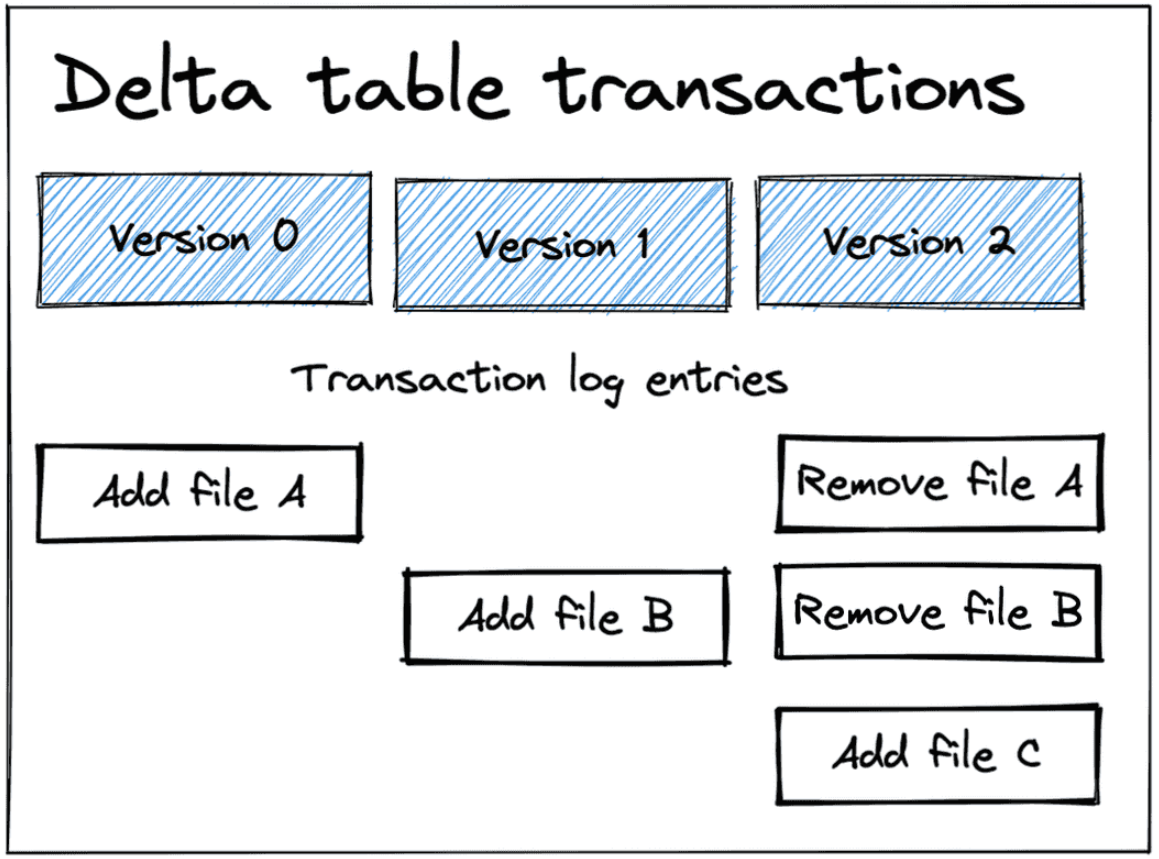
Transaction Log



Transaction Log

Time Travel





Metadata Tools

Command	Result
`DESCRIBE TABLE`	Schema of a table
`DESCRIBE EXTENDED`	Schema of a table + table details
`DESCRIBE HISTORY`	Table's changes & versions

DESCRIBE TABLE

- `DESCRIBE TABLE Lakedb.deltaUsers`

	col_name	data_type	comment
1	UserId	int	null
2	DisplayName	string	null
3	LastAccessDate	date	null
4	Reputation	int	null

DESCRIBE TABLE EXTENDED

- `DESCRIBE EXTENDED Lakedb.deltaUsers`

	col_name	data_type	comment
1	UserId	int	null
2	DisplayName	string	null
3	LastAccessDate	date	null
4	Reputation	int	null
5			
6	# Detailed Table Information		
7	Catalog	spark_catalog	
8	Database	Lakedb	
9	Table	deltaUsers	
10	Type	EXTERNAL	
11	Location	dbfs:/datarelay2023/delta/deltausers	
12	Provider	delta	
13	Owner	root	
14	Table Properties	[delta.minReaderVersion=1,delta.minWriterVersion=2]	

View Table Details

- `DESCRIBE DETAIL '/delta/deltausers/'`
- `DESCRIBE DETAIL Lakedb.deltaUsers`

Column	Type	Description
format	string	Format of the table, that is, `delta`.
id	string	Unique ID of the table.
name	string	Name of the table as defined in the metastore.
description	string	Description of the table.
location	string	Location of the table.
createdAt	timestamp	When the table was created.
lastModified	timestamp	When the table was last modified.
partitionColumns	array of strings	Names of the partition columns if the table is partitioned.
numFiles	long	Number of the files in the latest version of the table.
sizeInBytes	int	The size of the latest snapshot of the table in bytes.

format	id	name	description	location	createdAt	lastModified	partitionColumns
delta	4a67f614-32f2-465f-85fa-5e466983393c	spark_catalog.lakedb.deltausers	null	dbfs:/datarelay2023/delta/deltausers	2023-09-30T01:20:30.202+0000	2023-09-30T01:21:46.000+0000	[]

numFiles	sizeInBytes	properties	minReaderVersion	minWriterVersion	tableFeatures	statistics
2	2409	{}	1	2	["appendOnly",	{}

minReaderVersion	int
	Minimum version of readers (according to the log protocol) that can d the table.
	imum version of writers (according to the log protocol) that can te to the table.

History

- `DESCRIBE HISTORY Lakedb.deltaUsers`
- `DESCRIBE HISTORY delta.`/delta/users``

	version	timestamp	userId	userName	operation	operationParameters	job
1	2	2023-09-30T01:21:46.000+0000	4809069939003851	kamil@nowinski.net	MERGE	{"predicate": "[\"(UserId#14978 = UserId#14982)\"]", "matchedPredicates": "[{\"actionType\":\"update\"}]", "notMatchedPredicates": "[{\"actionType\":\"insert\"}]", "notMatchedBySourcePredicates": "[]"}	null
2	1	2023-09-30T01:21:10.000+0000	4809069939003851	kamil@nowinski.net	MERGE	{"predicate": "[\"(UserId#13638 = UserId#13646)\"]", "matchedPredicates": "[{\"actionType\":\"update\"}]", "notMatchedPredicates": "[{\"actionType\":\"insert\"}]", "notMatchedBySourcePredicates": "[]"}	null

notebook	clusterId	readVersion	isolationLevel	isBlindAppend	operationMetrics	userMetadata
{"notebookId": "2353687493752337"}	0926-221804-4448n59m	1	WriteSerializable	false	{"numTargetRowsCopied": "2", "numTargetRowsDeleted": "0", "numTargetFilesAdded": "2", "numTargetBytesAdded": "2409", "numTargetBytesRemoved": "1247", "numTargetDeletionVectorsAdded": "0", "numTargetRowsMatchedUpdated": "1", "executionTimeMs": "8325", "numTargetRowsInserted": "0", "numTargetRowsMatchedDeleted": "0", "scanTimeMs": "1742", "numTargetRowsUpdated": "1", "numOutputRows": "3", "numTargetDeletionVectorsRemoved": "0", "numTargetRowsNotMatchedBySourceUpdated": "0", "numTargetChangeFilesAdded": "0", "numSourceRows": "1", "numTargetFilesRemoved": "1", "numTargetRowsNotMatchedBySourceDeleted": "0", "rewriteTimeMs": "6485"}	null
{"notebookId": "2353687493752337"}	0926-221804-4448n59m	0	WriteSerializable	false	{"numTargetRowsCopied": "0", "numTargetRowsDeleted": "0", "numTargetFilesAdded": "1", "numTargetBytesAdded": "1247", "numTargetBytesRemoved": "0", "numTargetDeletionVectorsAdded": "0", "numTargetRowsMatchedUpdated": "0", "executionTimeMs": "3056", "numTargetRowsInserted": "3", "numTargetRowsMatchedDeleted": "0", "scanTimeMs": "1979", "numTargetRowsUpdated": "0", "numOutputRows": "3", "numTargetDeletionVectorsRemoved": "0", "numTargetRowsNotMatchedBySourceUpdated": "0", "numTargetChangeFilesAdded": "0", "numSourceRows": "3", "numTargetFilesRemoved": "0", "numTargetRowsNotMatchedBySourceDeleted": "0", "rewriteTimeMs": "970"}	null
{"notebookId": "2353687493752337"}	0926-221804-4448n59m	null	WriteSerializable	true	{}	null

Schema Validation

- All DataFrame **columns** must exist in the target table
- DataFrame column **data types** must match
- DataFrame **column names** cannot differ **only by case**

Schema Evolution

Azure Player
Play with data & have fun!

Delta Lake allows for schema evolution

1: Suppose you have this Delta table

```
+----------+---+
|first_name|age|
+----------+---+
|       bob| 47|
|        li| 23|
|   leonard| 51|
+----------+---+
```

2: Data to append

```
+----------+---+-------+
|first_name|age|country|
+----------+---+-------+
|     frank| 68|    usa|
|   jordana| 26| brasil|
+----------+---+-------+
```

Extra column of data

3: Schema enforcement will prevent the mismatched append

```
df.write.format("delta").mode("append").save("tmp/fun_people")
```

👮🛑 Exception: no data appended!

4: Use mergeSchema to enable schema evolution 🥰

```
df.write.option("mergeSchema", "true").mode("append").format("delta").save(
    "tmp/fun_people"
)
```

```
spark.read.format("delta").load("tmp/fun_people").show()
```

```
+----------+---+-------+
|first_name|age|country|
+----------+---+-------+
|   jordana| 26| brasil|
|     frank| 68|    usa|
|   leonard| 51|   null|
|       bob| 47|   null|
|        li| 23|   null|
+----------+---+-------+
```

So amazingly cool!

Schema Evolution - Automatic

Columns that are present in the DataFrame but missing from the table are automatically added as part of a write transaction when:

- write or writeStream have .option("**mergeSchema**", "true")
- spark.databricks.delta.**schema.autoMerge.enabled** is true

Update Table Schema

Azure Player
Play with data & have fun!

Columns	Query (in SQL)	Behaviour without schema evolution (default)	Behaviour with schema evolution

Target columns: key, value
 	
Source columns: key, value, new_value
	`MERGE INTO target_table t`
	`USING source_table s`
	`ON t.key = s.key`
	`WHEN MATCHED`
	`THEN UPDATE SET *`
	`WHEN NOT MATCHED`
	`THEN INSERT *`
		The table schema remains unchanged; only columns key, value are updated/inserted.	The table schema is changed to (key, value, new_value). Existing records with matches are updated with the value and new_value in the source. New rows are inserted with the schema (key, value, new_value).

Target columns: key, old_value
 	
Source columns: key, new_value
	`MERGE INTO target_table t`
	`USING source_table s`
	`ON t.key = s.key`
	`WHEN MATCHED`
	`THEN UPDATE SET *`
	`WHEN NOT MATCHED`
	`THEN INSERT *`
		UPDATE and INSERT actions throw an error because the target column old_value is not in the source.	The table schema is changed to (key, old_value, new_value). Existing records with matches are updated with the new_value in the source leaving old_value unchanged. New records are inserted with the specified key, new_value, and NULL for the old_value.

Target columns: key, old_value
 	
Source columns: key, new_value
	`MERGE INTO target_table t`
	`USING source_table s`
	`ON t.key = s.key`
	`WHEN MATCHED`
	`THEN UPDATE`
	`  SET new_value = s.new_value`
		UPDATE throws an error because column new_value does not exist in the target table.	The table schema is changed to (key, old_value, new_value). Existing records with matches are updated with the new_value in the source leaving old_value unchanged, and unmatched records have NULL entered for new_value. See note (1).

Target columns: key, old_value
 	
Source columns: key, new_value
	`MERGE INTO target_table t`
	`USING source_table s`
	`ON t.key = s.key`
	`WHEN NOT MATCHED`
	`THEN INSERT (key, new_value)`
	`  VALUES (s.key, s.new_value)`
		INSERT throws an error because column new_value does not exist in the target table.	The table schema is changed to (key, old_value, new_value). New records are inserted with the specified key, new_value, and NULL for the old_value. Existing records have NULL entered for new_value leaving old_value unchanged. See note (1).

DEMO

DEMO Agenda

- Convert CSV to Parquet and to Delta
- Read, insert & update data (DML)
- Transaction Log & DESCRIBE command
- Time Travel
- Schema validation & enforcement
- Compact files / partition

DELTA LAKE

WHAT'S NEW? - RELEASE 2.3.0

pip install delta-spark

TABLE CLONE COMMANDS

```
-- Clone table definition (metadata)
CREATE TABLE taxi_clone LIKE taxi;

-- Clone table definition & data
CREATE TABLE taxi_clone
  SHALLOW CLONE taxi;
```

RECORD VACUUM OPERATIONS

```
-- Now recorded in the table history
VACUUM default.taxi;
DESCRIBE HISTORY default.taxi;
```

MERGE INTO COMMAND

```
/**
 Support when not
 matched by source
 clause
*/
MERGE INTO target AS t
USING source AS s
 ON t.id = s.id
WHEN MATCHED THEN
 UPDATE SET *
WHEN NOT MATCHED BY
SOURCE THEN DELETE;
```

READ TABLE CHANGES (CDF)

```
-- View changes between snapshot interval
SELECT * FROM table_changes('taxi', 3, 7);

-- Not using a Catalog? No problem.
SELECT * FROM table_changes_by_path(
  '/tmp/taxi', '2023-04-09', '2023-04-10');
```

CONVERT ICEBERG TABLES

```
-- Will not copy data (metadata op)
CONVERT TO DELTA
iceberg.`/tmp/iceberg/taxi`;
```

#1 - Zero-copy CONVERT TO DELTA

```
/**
  Generates a Delta table in the same location.
  Will not copy data (metadata operation).
*/
CONVERT TO DELTA iceberg.`/tmp/iceberg/taxi`;
```

#2 - SHALLOW CLONE Command

```
/**
  Clone a Delta, Parquet, or Iceberg table.
  Copies the table definition and data files.
  Note: creates a *new* transaction log.
*/
CREATE TABLE taxi_clone SHALLOW CLONE taxi;
```

#3 - Idempotent DML Operations

```
/**
  Support idempotent writes for *all* DML operations
  (INSERT/DELETE/UPDATE/MERGE).
  `txnAppId` - a unique string that identifies the app
  `txnVersion` - a monotonically increasing number
*/
SET spark.databricks.delta.write.txnAppId='DeltaLakeDemo';
SET spark.databricks.delta.write.txnVersion=9446;
UPDATE taxi
   SET passenger_count=4
 WHERE pride_id=1098;
```

#4 - WHEN NOT MATCHED BY SOURCE

```
/**
  SQL support coming in Spark 3.4 (17 days)!
*/
MERGE INTO default.taxi AS target
USING new_taxi_data AS source
 ON source.ride_id=target.ride_id
WHEN NOT MATCHED BY SOURCE THEN DELETE;


# Use Python API in the meantime
targetDF.merge(sourceDF, "source.ride_id=target.ride_id") \
 .whenNotMatchedBySourceDelete().execute()
```

#5 - CREATE TABLE LIKE Command

```
/**
  Clone an existing Delta table.
  Note:Copies the metadata only and no data files.
       Will create an empty table.
*/
CREATE TABLE taxi_clone LIKE taxi;
```

#6 - Read Table Changes (CDF)

```
-- Read table changes that between an interval.
SELECT * FROM table_changes('taxi_clone', 3, 5);

-- Timestamps work too! (If you don't prefer versions)
SELECT * FROM table_changes('taxi_clone', '2023-04-09', '2023-04-10);

-- Don't have a Catalog? No problem!
SELECT * FROM table_changes_by_path('tmp/taxi_clone', '2023-04-09');
```

#7 - CDF for Column-mapped Tables

```
/**
  Previously, tables with column-mapping did *not* support
  change data feed reads after this operation.
*/
ALTER TABLE taxi RENAME COLUMN ride_id TO id; -- version 3

/**
  This release supports batch CDF reads for tables
  that have used column-mapping to DROP or RENAME a column.
*/
SELECT * FROM table_changes('taxi', 1, 3) -- this works now!
```

#8 - Faster S3 Reads & Writes

```
// Improves file listing efficiency during snapshot calc.
// Note: this features works on S3A filesystems *only*.
bin/spark-shell \
  --packages io.delta:delta-core_2.12:2.3.0, \
             org.apache.hadoop:hadoop-aws:3.3.1 \
  --conf "spark.hadoop.delta.enableFastS3AListFrom=true"


# Can be set the configuration on the Spark context too
sc.hadoopConfiguration.set(
  'spark.hadoop.delta.enableFastS3AListFrom', 'true')
```

#9 - Record VACUUM operations

```
/**
  VACUUM operations and file metrics will now be recorded
  in a table's history.
*/
VACUUM default.taxi;

-- Start time, end time, and operation metrics will now appear!
DESCRIBE HISTORY default.taxi;
```

Release 2.4

- https://github.com/delta-io/delta/releases/tag/v2.4.0
- Documentation: https://docs.delta.io/2.4.0/
- Support for Apache Spark 3.4.

Release 3.0 (preview)

- https://github.com/delta-io/delta/releases/
- Documentation: https://docs.delta.io/3.0.0rc1/
- Built on top of Apache Spark 3.4.

Databricks Runtimes

Standard

14.0	Scala 2.12, Spark 3.5.0
13.3 LTS	Scala 2.12, Spark 3.4.1
13.2	Scala 2.12, Spark 3.4.0
13.1	Scala 2.12, Spark 3.4.0
13.0	Scala 2.12, Spark 3.4.0
12.2 LTS	Scala 2.12, Spark 3.3.2
11.3 LTS	Scala 2.12, Spark 3.3.0
10.4 LTS	Scala 2.12, Spark 3.2.1
9.1 LTS	Scala 2.12, Spark 3.1.2
7.3 LTS	Scala 2.12, Spark 3.0.1

ML

14.0 ML	GPU, Scala 2.12, Spark 3.5.0
14.0 ML	Scala 2.12, Spark 3.5.0
13.3 LTS ML	GPU, Scala 2.12, Spark 3.4.1
13.3 LTS ML	Scala 2.12, Spark 3.4.1
13.2 ML	GPU, Scala 2.12, Spark 3.4.0
13.2 ML	Scala 2.12, Spark 3.4.0
13.1 ML	GPU, Scala 2.12, Spark 3.4.0
13.1 ML	Scala 2.12, Spark 3.4.0
13.0 ML	GPU, Scala 2.12, Spark 3.4.0
13.0 ML	Scala 2.12, Spark 3.4.0
12.2 LTS ML	GPU, Scala 2.12, Spark 3.3.2
12.2 LTS ML	Scala 2.12, Spark 3.3.2

Uncategorised

Runtime 10.2	Scala 2.12, Spark 3.2
Runtime 10.2	Scala 2.12, Spark 3.2
Runtime 3.0	Scala 2.10, Spark 2.2
Runtime 3.0	Scala 2.10, Spark 2.2
Spark 2.0	Scala 2.10, Spark 2.0
Spark 1.6.2	Scala 2.10, Spark 1.6.2
Spark 1.5.2	Scala 2.10, Spark 1.5.2

Synapse Runtimes

Apache Spark *	3.3
Python	3.10
Scala	2.12.15
Java	1.8.0_282
Delta Lake	2.2
R	4.2.2

Apache Spark *	3.2
Python	3.8
Scala	2.12.15
Java	1.8.0_282
.NET Core	3.1
.NET for Apache Spark	2.0
Delta Lake	1.2

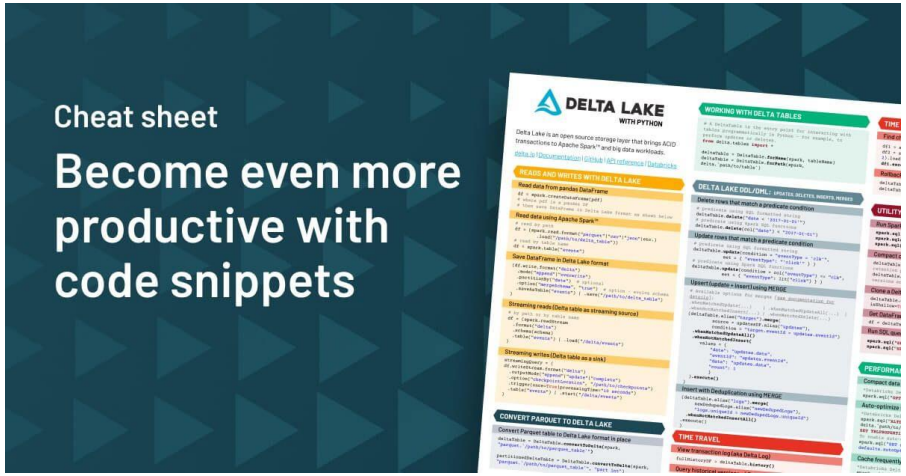
Summary

Summary

- Delta Lake is optimized storage layer
- Delta Lake operates on Parquet files underneath
- File-based transaction log for ACID transactions
- Open Source
- Supported by Azure Databricks, Synapse & ADF
- Default storage format for all operations on Azure Databricks

Resources

- https://delta.io/
  - https://docs.delta.io/latest/releases.html
  - https://github.com/delta-io/delta/
  - https://github.com/delta-io/delta/releases/
  - https://github.com/delta-io/delta-examples/
  - Table batch reads and writes
- Lambda Architecture
- Delta Lake Blogs
- What is Delta Lake? (Microsoft)
- Getting Started with Delta Lake (Databricks)
- Follow on LinkedIn: Delta Lake
- Unpacking The Transaction Log
- Cheat sheets on Learn with AzurePlayer (free)



Resources


DELTA LAKE
DELTA LAKE 2.3.0
Support reading Change Data Feed (CDF) in SQL queries
Improved read and write performance on S3
Record VACUUM operations in the transaction log
Support reading Delta tables with deletion vectors
DELTA LAKE
Release Notes:
https://github.com/delta-io/delta/releases/tag/v2.3.0
@deltalakeoss
go.delta.io/slack
delta.io

Thank you!


kamil@azureplayer.net


@NowinskiK @Azure_Player


AzurePlayer.net


https://AzurePlayer.net/slides

Kamil Nowinski
Microsoft Data Platform MVP
Analytics Architect, Azure DevOps Engineer Expert