SQL Day

15 edycja konferencji SQLDay

8-10 maja 2023, WROCŁAW + ONLINE

Data Community

partner złoty

Future Processing

partner srebrny

Objectivity

summ-it

kursySQL.pl

VOLVO

partner brązowy

devart

Kamil Nowinski

Delta Lake Tables 101

Kamil Nowiński

Microsoft Data Platform **MVP**

Speaker, blogger, data enthusiast

Group Manager at Avanade UK&I (www.avanade.com)

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

Member of the Data Community PL

Founder of blog SQLPlayer (www.AzurePlayer.net)

GitHub: #adftools, SCD Merge Wizard and more…

SQL Server Certificates:

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

MCSE Data Management & Analytics, DevOps Expert

Moreover: Bicycle, Running, Digital photography

@NowinskiK, @Azure_Player

SQLPlayer is renaming

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

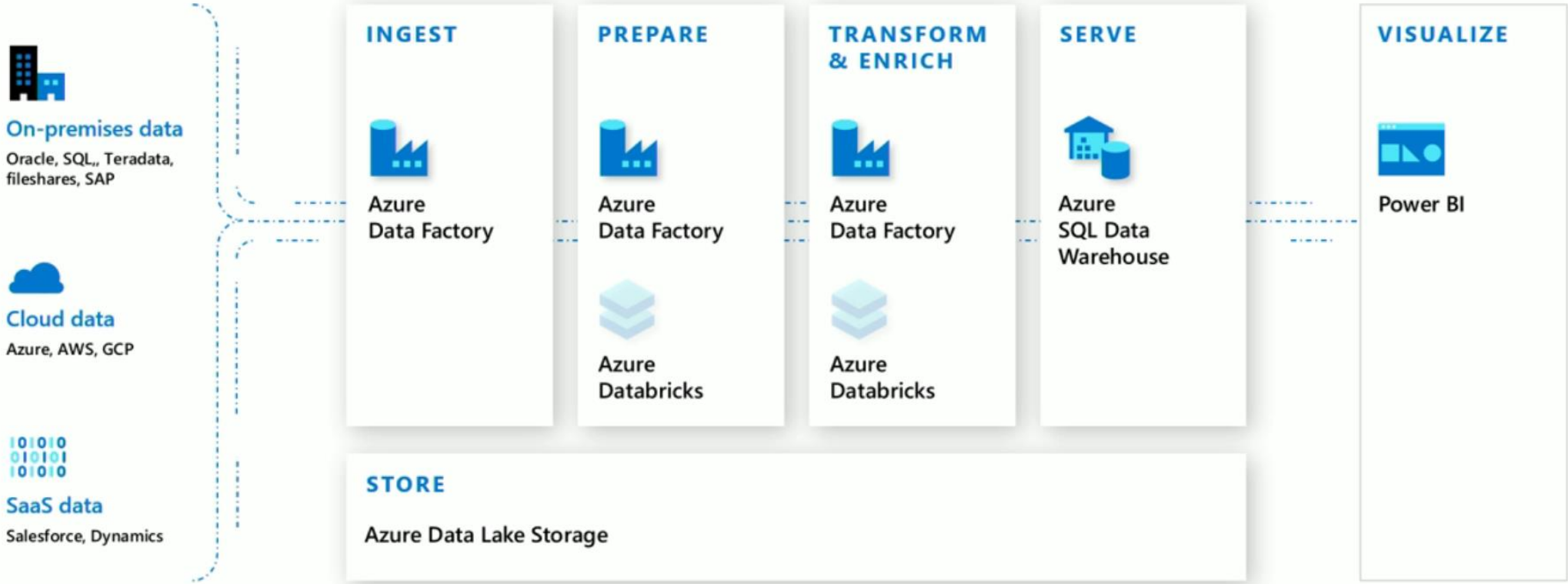
Slides available:

https://azureplayer.net/slides

AGENDA

- A bit of theory & history…
- Let's practice: DEMO time
    - Delta Lake Tables in Lake Databases (Synapse)

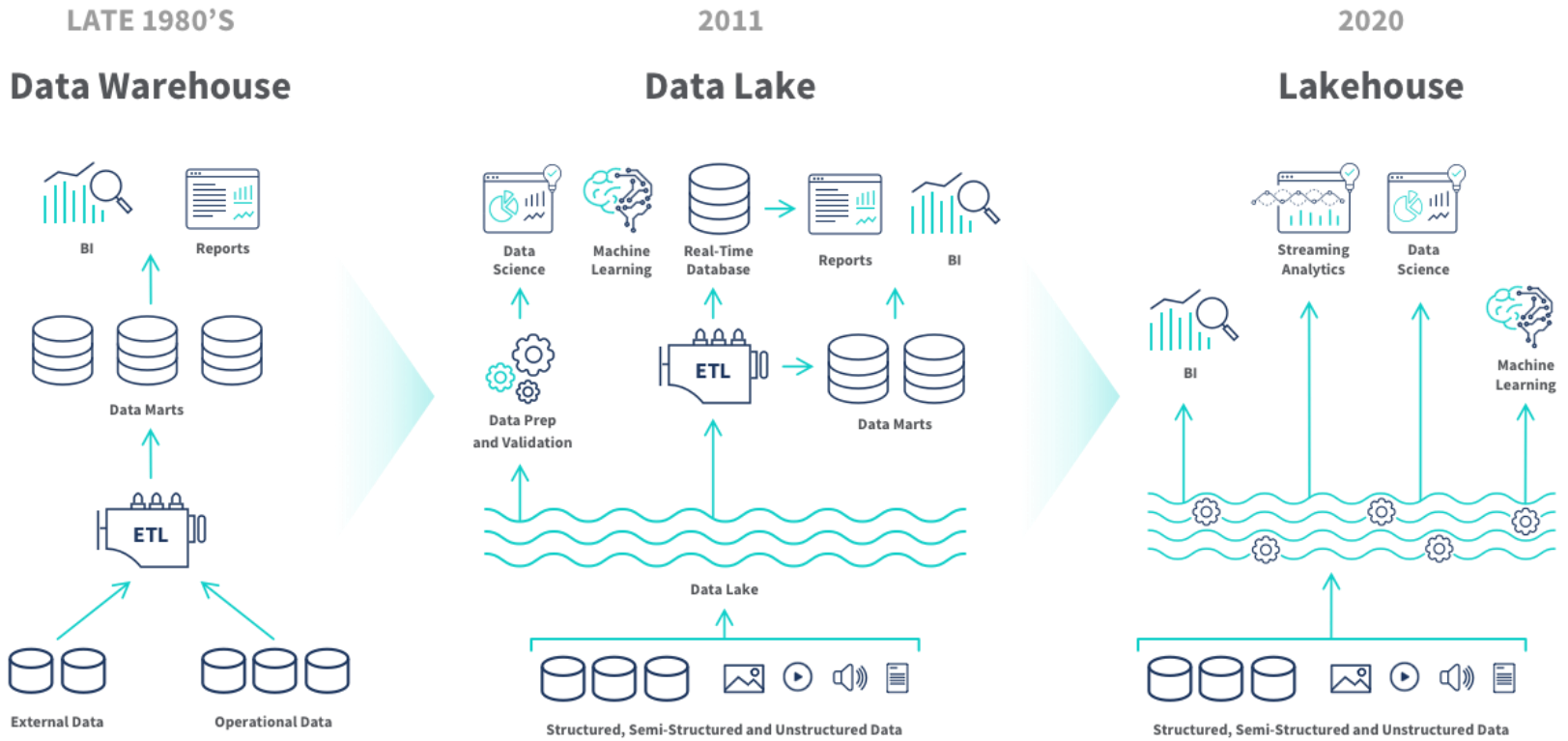
Modern Data Warehouse



Data Lake + Warehouse = Lakehouse

[> What Is a Lakehouse?]

Data Warehouse evolution

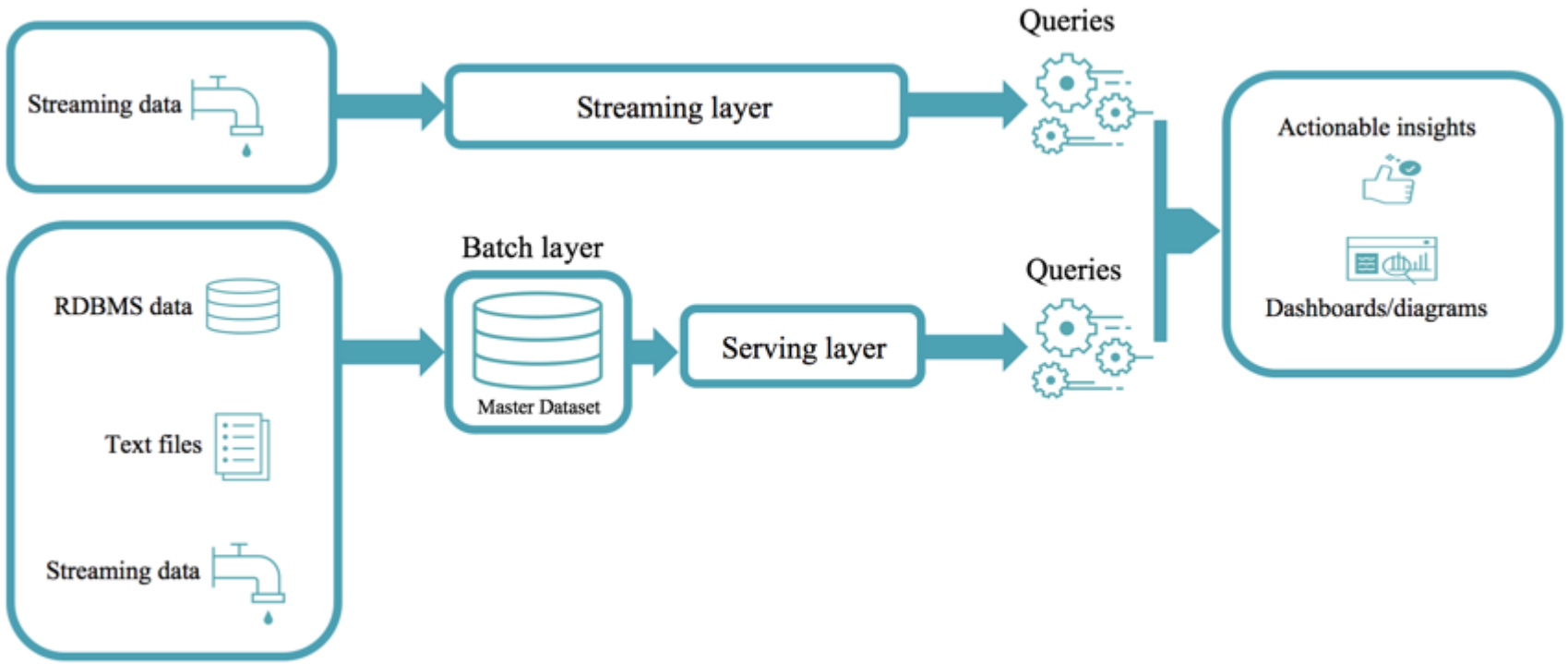


https://www.qlik.com/us/data-lake/data-lakehouse

Introduction

- Lambda Architecture
- Streaming & batch
- Time travel
- Upserts & deletes

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.

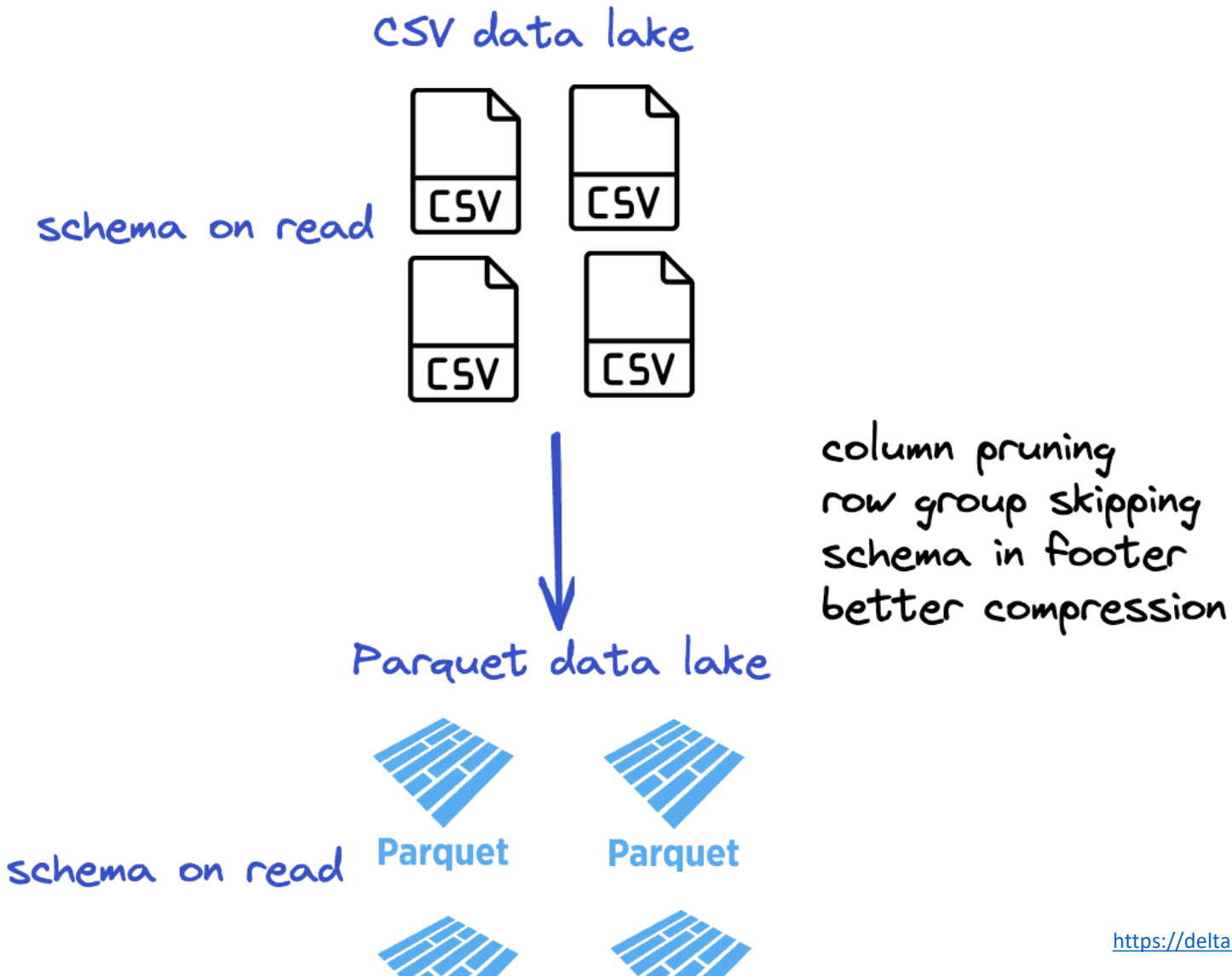
Characteristics of Parquet:

- 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

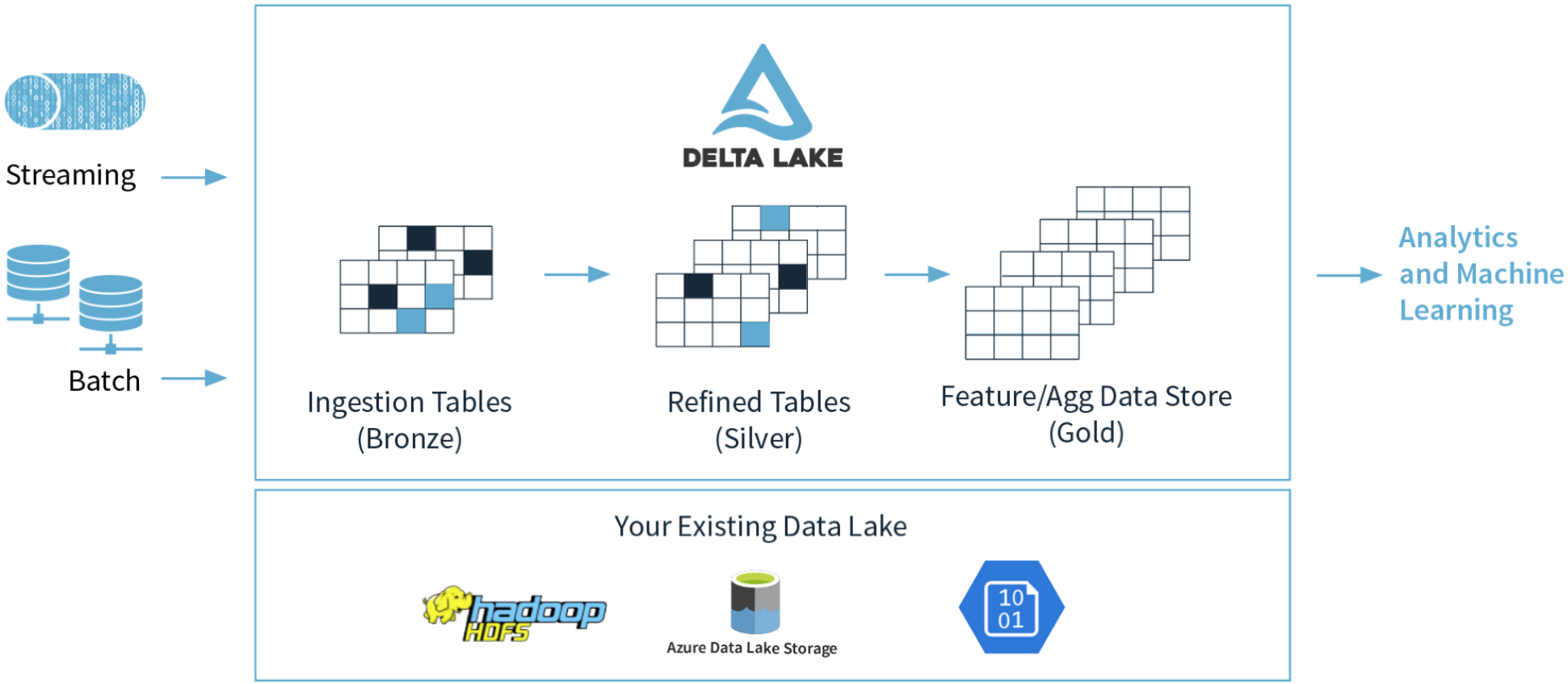
- No schema enforcement
- Updates – rewrite entire file
- Transactions inconsistency
- No interoperability between batch & streaming workloads
- No versioning

Advantages of Delta Lake over CSV



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

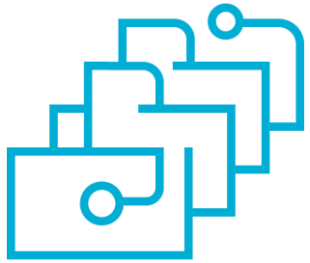
Medal architecture



Delta – Key Features

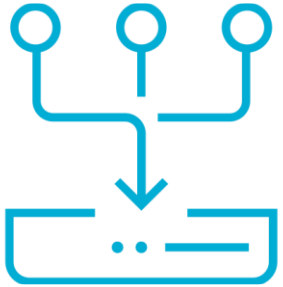
ACID Transactions

Scalable Metadata

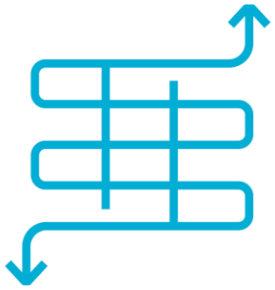


Time Travel

Open Source

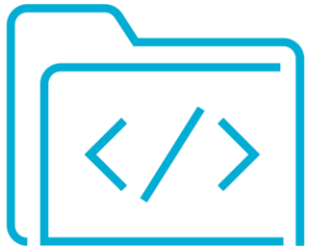


Unified Batch/Streaming



Schema Evolution / Enforcement

Audit History



DML Operations

DEMO

Delta Lake Tables support

- Databricks (of course!)
- Delta format in Azure Data Factory & Synapse
  - Mapping Data Flow
  - Copy Activity (only with Databricks connector)
- Synapse notebook
- Synapse SQL Serverless (read – tables, views)
- Power BI

Schema Evolution

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!

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

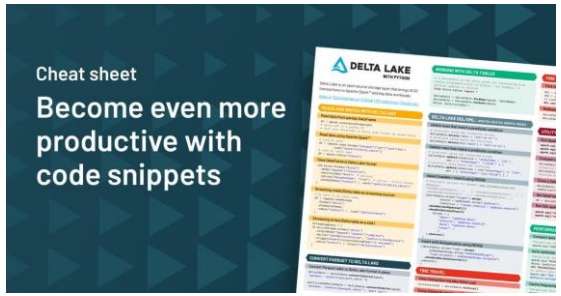
```
/**
  VACUUM operations and file metrics will now be recorded
  in a table's history.
*/
VACUUM default.taxi;

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

Summary

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)



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

SQL Day

15 edycja konferencji SQLDay

8-10 maja 2023, WROCŁAW + ONLINE

Data Community

partner złoty

Future Processing

partner srebrny

Objectivity

summ-it

kursySQL.pl

VOLVO

partner brązowy

devart