Databricks for the SQL Developer

Gerhard Brueckl

Cloud Data Architect @ paiqo GmbH

Microsoft Most Valuable Professional

SSAS MAESTRO by Microsoft

@gbrueckl

blog.gbrueckl.at

gerhard@gbrueckl.at

https://github.com/gbrueckl

DatabricksPS

Databricks VSCode

PowerBI Connector

paiqo

www.paiqo.com

SQL saturday VIENNA

PASS AUSTRIA

Agenda

What is Databricks / Spark?

How is Databricks / Spark different to traditional RDBMS?

SQL with Databricks

Delta Lake

Advanced SQL techniques

What is Databricks / Spark?

What is Spark ?

- Open-source Cluster computing framework
  - Massive Parallel Processing with linear scale

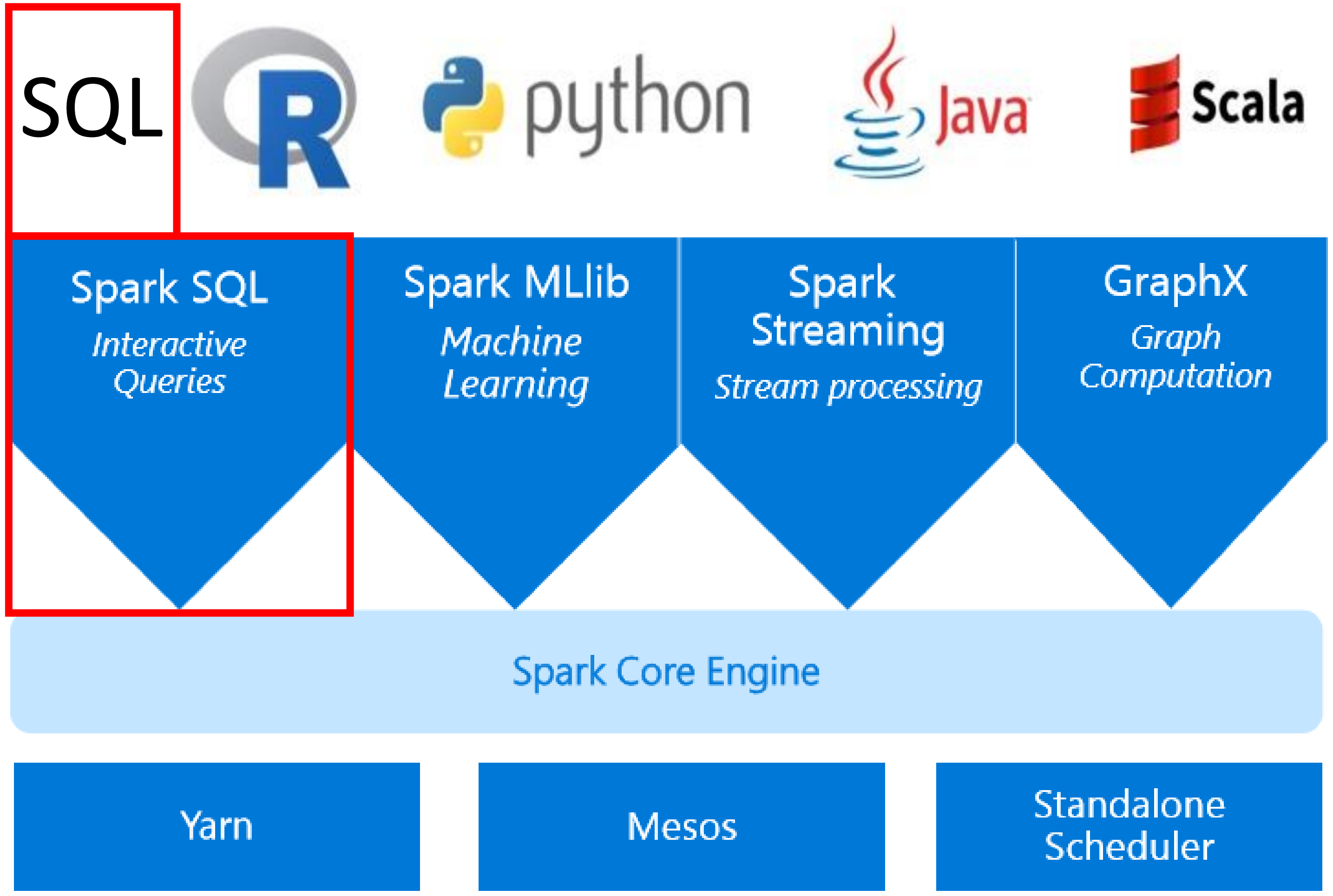
- Built for: Speed/Scalability, Ease-of-Use, Extensibility

- Support for multiple languages
  - Java, Scala, Python, R, **SQL**

What is Spark ?

Spark unifies

- Batch Processing
- Interactive SQL
- Real-time processing
- Machine Learning
- Deep Learning
- Graph Processing



What is Databricks ?

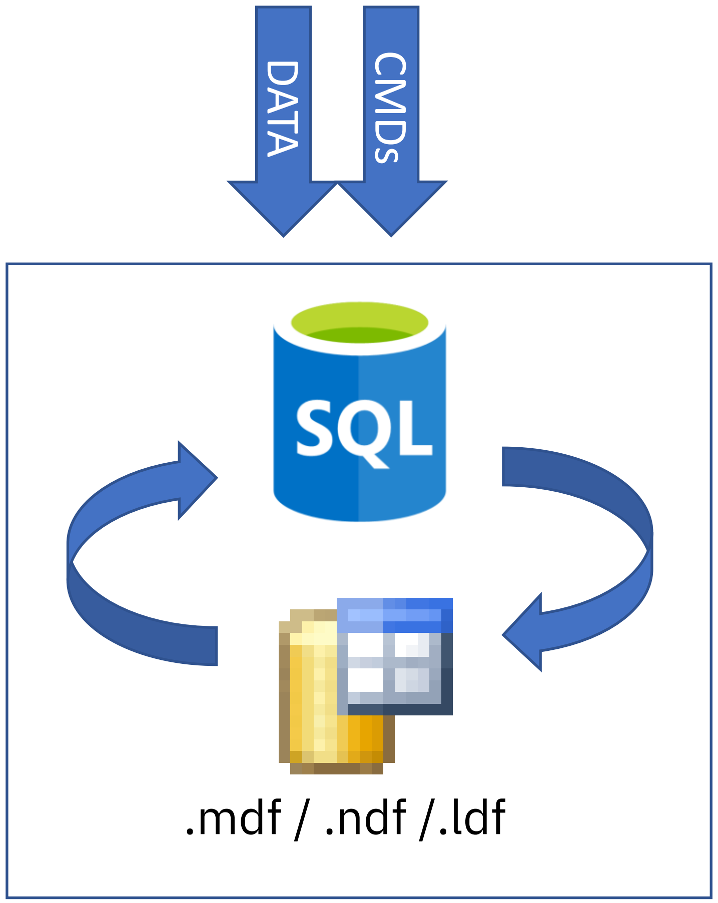
- Company that provides a Big Data processing solutions in the Cloud using Apache Spark
  - Databricks on AWS
  - Azure Databricks
  - Databricks on Google Cloud
  - **No on-prem solution!**
- Creators of Apache® Spark™

How is Big Data / Spark different to traditional RDBMS?

Classic RDBMS vs. Big Data/Spark

Classic RDBMS

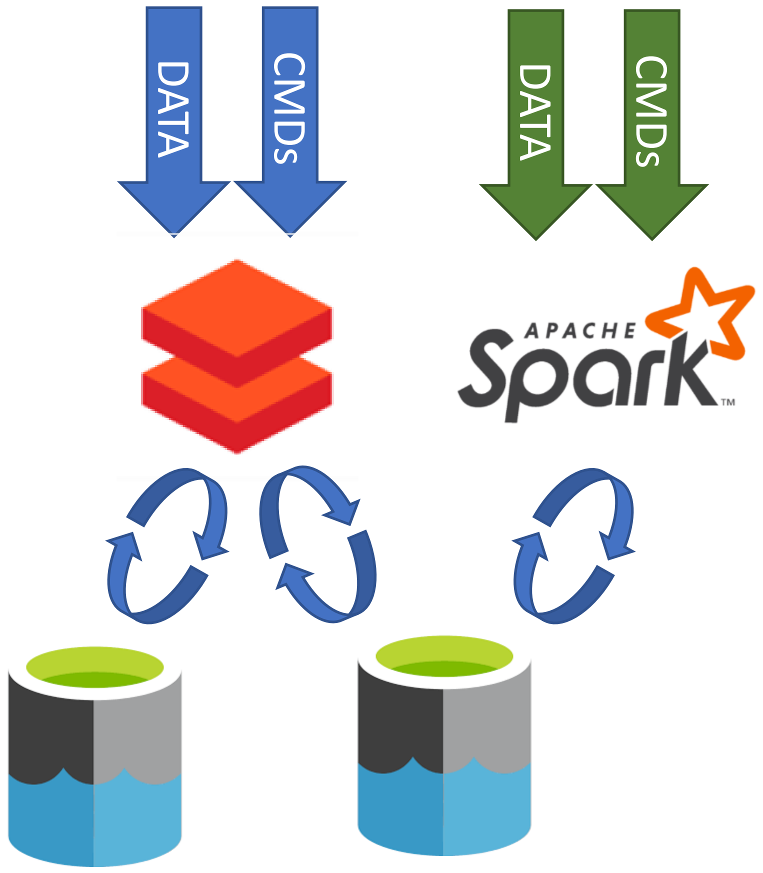
- Single point of access to your data
- Limited resources / no scale-out
- All process use the same resources
  - ETL vs. user queries
- Storage is managed internally



Classic RDBMS vs. Big Data/Spark

Big Data processing with Spark

- Separation of storage and compute (!)
- Only spin up compute when necessary
- Can use multiple compute engines
- Cheap storage
- Can attach any storage



Classic RDBMS vs. Big Data/Spark

RDBMS	**Big Data / Spark**
Server	Cluster(s) + Metadata/Metastore
Database (e.g. AdventureWorks)	Metastore
Schema (e.g. dbo)	Database
Table (e.g. DimProduct)	Files/Location + Metadata
Index	---
Stored Procedure	Notebook
UDF	UDF
View	View

SQL with Databricks

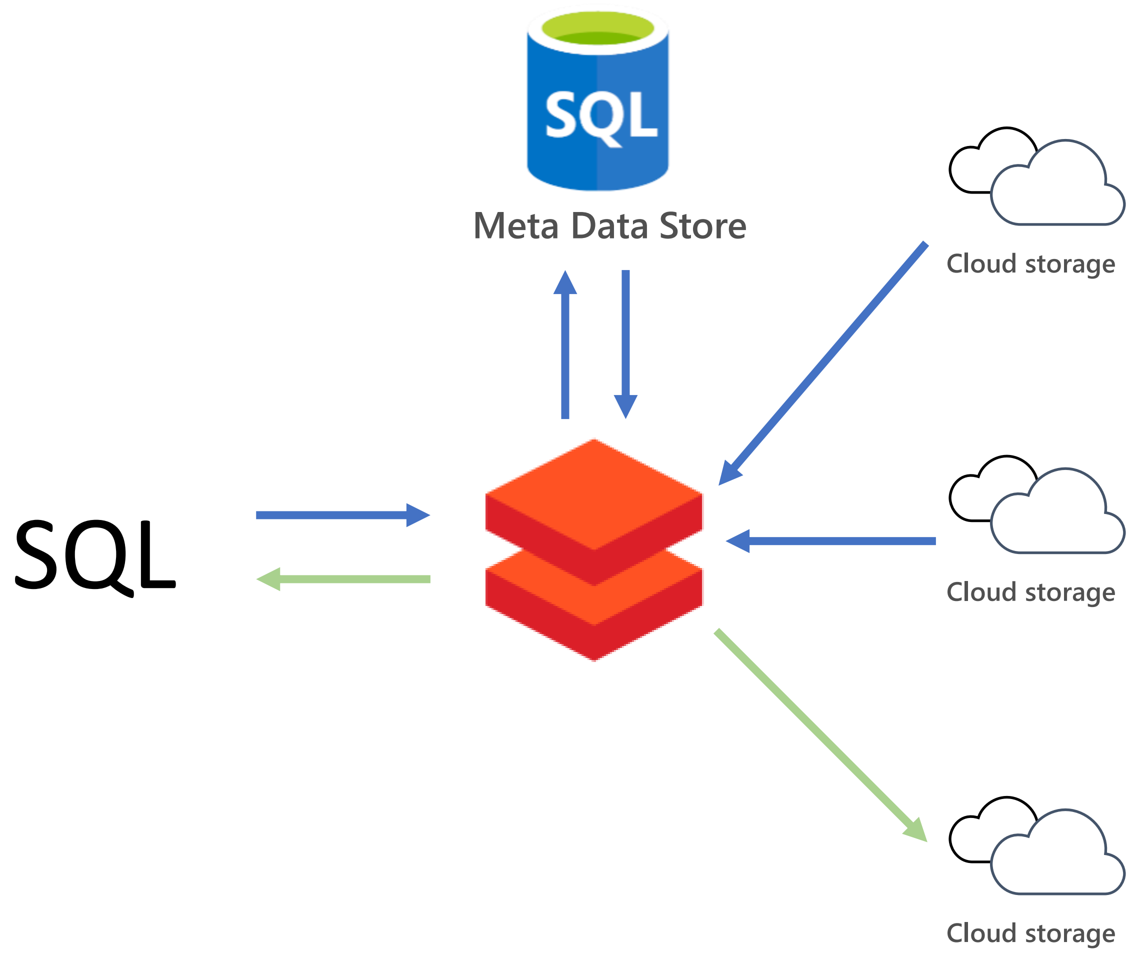
When to use Databricks (over RDBMS)

- Scalability
- Flexibility
  - Structured data
  - Unstructured data
- Open Standard
  - Apache Spark

- Single tool for all workloads
  - ETL
  - Data Scientists
  - BI / Reporting
- Cloud solution

Batch processing only – no OLTP!!!

Processing of a (SQL) query



1. Client submits SQL Query
2. Databricks queries Meta Data Catalog
   - Checks syntax/columns
   - Returns storage locations
3. Databricks queries storage for raw data
4. Data is loaded into memory of nodes
5. Data is processed on nodes using Spark
6. "Result"
   - Data is written directly to storage services
   - Data is returned to client

The HIVE Meta-Store

- Contains meta-data of all SQL objects
  - Tables (managed vs. external)
  - Views
  - Functions
- Location of the data
- Structure
- Format
- …

Tables are just references, the data resides on the storage!

SELECT and INSERT

SELECT

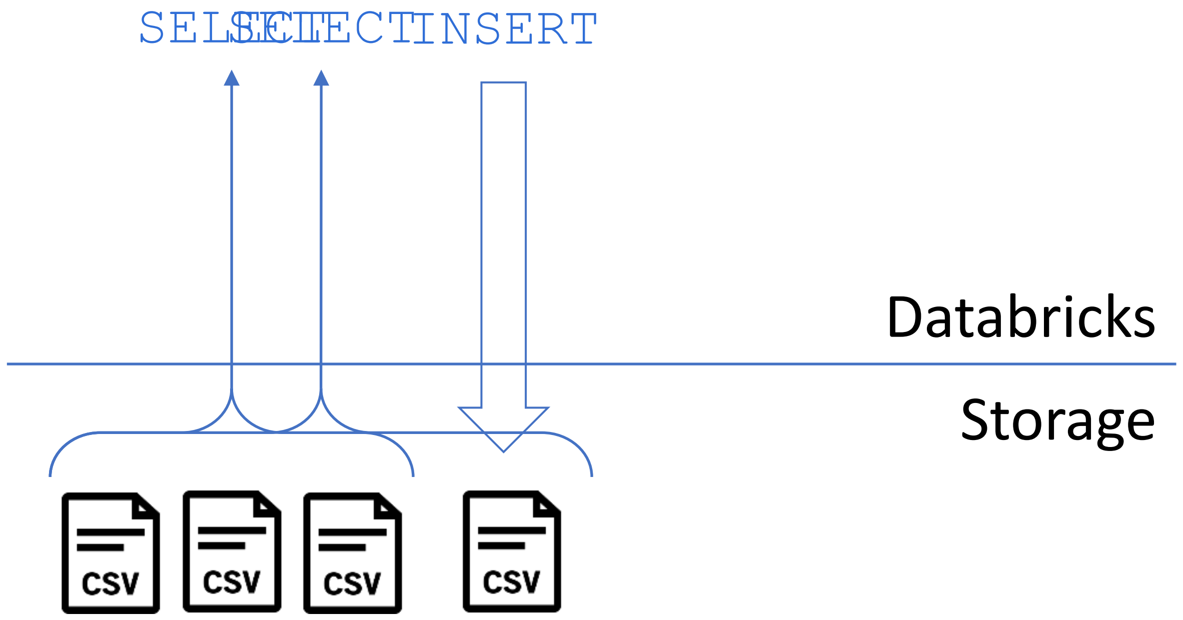
- Reads content of files

INSERT

- Creates new file on the storage

UPDATE / DELETE / MERGE

- Not natively support
- → Delta Lake



Supported SQL features

- ANSI SQL
  - Joins
  - Groupings/Aggregations
  - Rollup/Cube/GroupingSets
  - Subselects/CTEs
  - Window Functions
  - Built-in Functions (date, text, …)
  - …

- Additional features
  - Complex datatypes
    - struct / map / array
  - Custom functions
  - Custom aggregators
  - Caching
  - Easy transition to other languages
    - Python, R, Scala, Java, …

DEMO

Delta Lake

Delta Lake – delta.io

Delta Lake is an open-source storage layer that brings ACID transactions to Apache Spark™ and big data workloads.

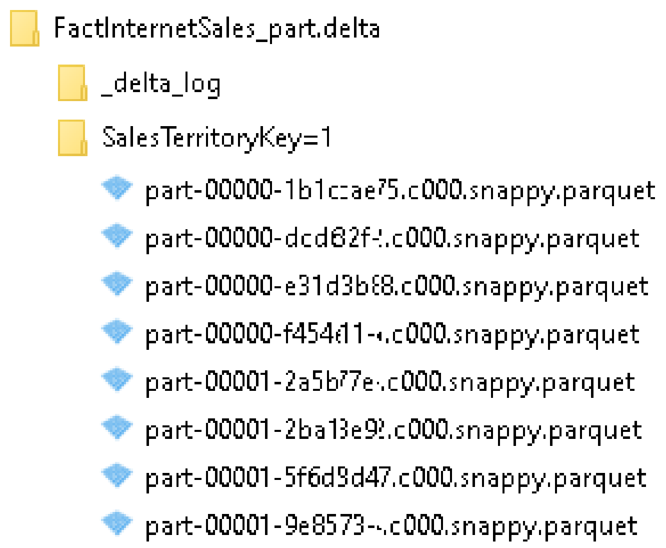
- ACID compliant transactions
  - Optimistic Concurrency Control
- Support for UPDATE / MERGE
- Time-Travel
- Schema enforcement and evolution
  - Across multiple files/folders
- Batch & Streaming
- 100% compatible with Apache Spark

Delta Lake – delta.io

- Everything is stored in one folder
  - Data
  - Meta-data
  - Transaction log / _delta_log

- Could basically Copy & Paste whole Delta table

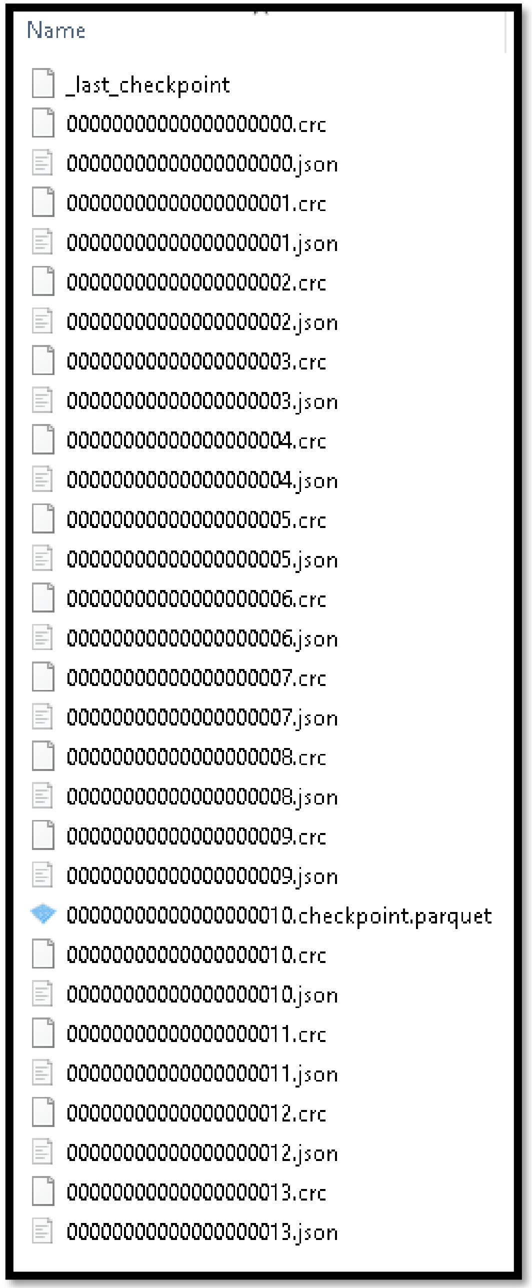
- Hive meta-store only needs location



```
FactInternetSales_part.delta
    _delta_log
    SalesTerritoryKey=1
        part-00000-1b1czae75.c000.snappy.parquet
        part-00000-dcd82f-.c000.snappy.parquet
        part-00000-e31d3b88.c000.snappy.parquet
        part-00000-f454d11-.c000.snappy.parquet
        part-00001-2a5b77e-.c000.snappy.parquet
        part-00001-2ba13e9.c000.snappy.parquet
        part-00001-5f6d9d47.c000.snappy.parquet
        part-00001-9e8573-.c000.snappy.parquet
```

Delta Log

- Contains all meta-data
  - Table schema
  - References to files
- Stored as JSON and Parquet
- Stores all transactions
  - Files added/deleted
  - Changes in meta-data
  - Transaction metric
- Allows concurrency control
- Used for time-travel



```
Name
_last_checkpoint
00000000000000000000.crc
00000000000000000000.json
00000000000000000001.crc
00000000000000000001.json
00000000000000000002.crc
00000000000000000002.json
00000000000000000003.crc
00000000000000000003.json
00000000000000000004.crc
00000000000000000004.json
00000000000000000005.crc
00000000000000000005.json
00000000000000000006.crc
00000000000000000006.json
00000000000000000007.crc
00000000000000000007.json
00000000000000000008.crc
00000000000000000008.json
00000000000000000009.crc
00000000000000000009.json
00000000000000000010.checkpoint.parquet
00000000000000000010.crc
00000000000000000010.json
00000000000000000011.crc
00000000000000000011.json
00000000000000000012.crc
00000000000000000012.json
00000000000000000013.crc
00000000000000000013.json
```

CREATE new Delta Table in Hive meta-store

```
CREATE TABLE IF NOT EXISTS MyTable
(id INT, name STRING, region INT)
USING DELTA
LOCATION '/mnt/adls/tables/DimProductDelta'
PARTITIONED BY (region)
TBLPROPERTIES ('myKey' = 'myValue')
```

Use existing Delta Table in Hive meta-store

```
CREATE TABLE IF NOT EXISTS MyTable
~~(id INT, name STRING, region INT)~~
USING DELTA
LOCATION '/mnt/adls/tables/MyTable'
~~PARTITIONED BY (region)~~
~~TBLPROPERTIES ('myKey' = 'myValue')~~
```

Omit everything except

- **USING**
- **LOCATION**

DML Operations – Delta Lake - UPDATE

User

Product	Price
Notebook	900 €
PC	1,500 €
Tablet	500 €

```
UPDATE TABLE DimProduct
SET Price = 1300
WHERE Product = 'PC'
```

Product	Price
Notebook	900 €
PC	1,300 €
Tablet	500 €

_delta_log

000000000.json

```
"add": {
        "path": "part-01.parquet",
        ...
}
```

000000001.json

```
"remove": { "path": "part-01.parquet", ... },
"add": { "path": "part-02.parquet", ... }
```

Storage

Parquet
part-01
(3 rows)

Parquet
part-01
(3 rows)

Parquet
part-02
(3 rows)

DML Operations - Delta Lake – DELETE

User

Product	Price
Notebook	900 €
PC	1,300 €
Tablet	500 €

```sql
DELETE FROM DimProduct
WHERE Product = 'PC'
```

Product	Price
Notebook	900 €
Tablet	500 €

_delta_log

000000000.json

```
"add": {
     "path": "part-01.parquet",
... }
```

000000001.json

```
"remove": {
   "path": "part-01.parquet", ... },
"add": {
   "path": "part-02.parquet", ... }
```

000000002.json

```
"remove": { "path": "part-02.parquet", ... },
"add": { "path": "part-03.parquet", ... }
```

Storage

Parquet	Parquet	Parquet	Parquet	Parquet
part-01	part-02	part-01	part-02	part-03
(3 rows)	(3 rows)	(3 rows)	(3 rows)	(2 rows)

DML Operations - Delta Lake – INSERT

User

Product	Price
Notebook	900 €
Tablet	500 €

```sql
INSERT INTO DimProduct
VALUES ('Monitor', 200)
```

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

_delta_log

000000000.json

000000001.json

"remove": {
  "path": "part-01.parquet", … },
"add": {
  "path": "part-02.parquet", … }

000000002.json

"remove": { "path": "part-02.parquet", … },
"add": { "path": "part-03.parquet", … }

000000003.json

"add": { "path": "part-04.parquet", … }

Storage

Parquet
part-01
(3 rows)

Parquet
part-02
(3 rows)

Parquet
part-03
(2 rows)

Parquet
part-01
(3 rows)

Parquet
part-02
(3 rows)

Parquet
part-03
(2 rows)

Parquet
part-04
(1 row)

DML Operations - Delta Lake – VACUUM

User

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

```
VACUUM DimProduct
```

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

_delta_log

000000000.json
000000001.json

000000002.json

```
"remove": {
   "path": "part-02.parquet", … },
"add": {
   "path": "part-03.parquet", … }
```

000000003.json

```
"add": {
   "path": "part-04.parquet",
… }
```

Storage

part-01
(3 rows)

part-02
(3 rows)

part-03
(2 rows)

part-04
(1 row)

part-03
(2 rows)

part-04
(1 row)

DML Operations - Delta Lake – OPTIMIZE

User

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

```
OPTIMIZE DimProduct
```

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

_delta_log

000000000.json
000000001.json

000000002.json
000000002.json

```
"remove": {
   "path": "part-02.parquet", … },
"add": {
   "path": "part-03.parquet", … }
```

000000003.json

```
"add": {
   "path": "part-04.parquet",
… }
```

```
"remove": {
   "path": "part-03.parquet“, "part-04.parquet" },
"add": {
   "path": "part-05.parquet", … }
```

Storage

Parquet
part-03
(2 rows)

Parquet
part-04
(1 row)

Parquet
part-03
(2 rows)

Parquet
part-04
(1 row)

Parquet
part-05
(3 rows)

UPDATE / DELETE / MERGE

- Always results in new files! Even a DELETE!

- Operations are logged in _delta_log
  - Old files are invalidated
  - New files are added/referenced

- Conflicts have to be handled by the User!

Can create A LOT of files!

OPTIMIZE / VACUUM
To manage files

OPTIMIZE

- Collapse small files into bigger files
- Bin-Packing / Ordering
- Improve query performance
- Creates another copy of the data!

```
OPTIMIZE events
WHERE date = 20200101
ZORDER BY (eventType)
```

VACUUM

- Removes unreferenced files older than X days
- Never touches latest version of files

```
VACUUM events
[RETAIN num HOURS]
[DRY RUN]
```

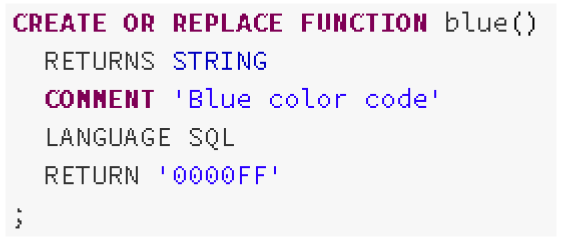
DEMO

Advanced SQL techniques

Advanced SQL techniques
User Defined Functions (UDFs)

UDFs in SQL

- Scalar valued
- Table valued
- Temporary or persisted
- Can be nested



```sql
CREATE OR REPLACE FUNCTION blue()
  RETURNS STRING
  COMMENT 'Blue color code'
  LANGUAGE SQL
  RETURN '0000FF'
;
```

Advanced SQL techniques
User Defined Functions (UDFs)

UDFs from Code

- Python/R/Scala/JAVA
- Scalar valued only (but complex types)
- Must be registered to be used in SQL

Session-Level only

- Can be nested

User Defined Aggregates (UDAs)

- SCALA and JAVA only

Advanced SQL techniques
External JDBC sources

- Connect to any JDBC source
- Exposed as regular SQL table

```
CREATE TABLE myJdbcTable
USING org.apache.spark.sql.jdbc
OPTIONS (
  url "jdbc:<databaseServerType>://<jdbcHostname>:<jdbcPort>",
  table "<jdbcDatabase>.myTable",
  user "<jdbcUsername>",
  password "<jdbcPassword>"
)
```