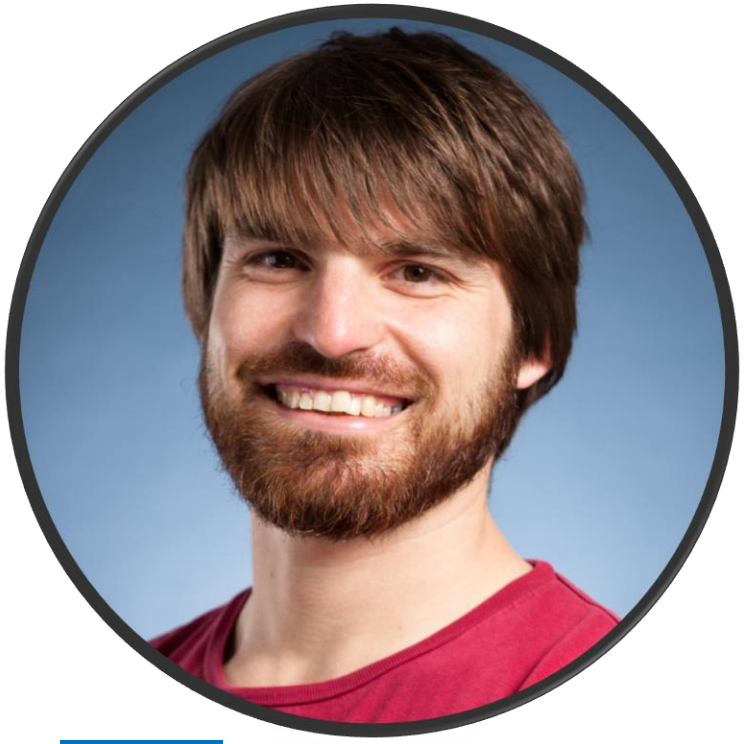
DATA+AI SUMMIT 2022

Deep-Dive into Delta Lake

Gerhard Brueckl

Cloud Data Architect @ paiqo GmbH

ORGANIZED BY databricks



Microsoft Most Valuable Professional

@gbrueckl

blog.gbrueckl.at

gerhard@gbrueckl.at

https://github.com/gbrueckl

DatabricksPS

Databricks VSCode

PowerBI Connector

DELTA LAKE

paiqo

www.paiqo.com

SQL saturday VIENNA

AUSTRIA PASS

Agenda

- What is Delta Lake?
- What is the Delta Log?
- How does it Work?
- File & Storage Management
- Streaming
- Properties
- Conclusion & Lessions Learned

What is Delta Lake?

What is Delta Lake?

https://delta.io

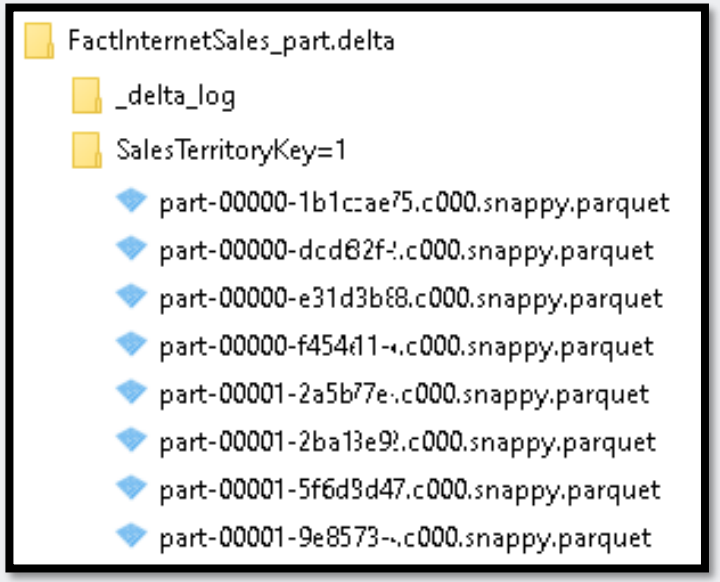
Delta Lake is an open-source storage framework that enables building a Lakehouse architecture with compute engines including Spark, PrestoDB, Flink, Trino, and Hive and APIs for Scala, Java, Rust, Ruby, and Python.

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

What is Delta Lake?

https://delta.io

- Everything is stored in one folder
  - Meta-data
  - Transaction log / **Delta Log**
  - Data
- Could basically Copy & Paste whole Delta table
- Supports any storage sub-system
- Consumer only needs location

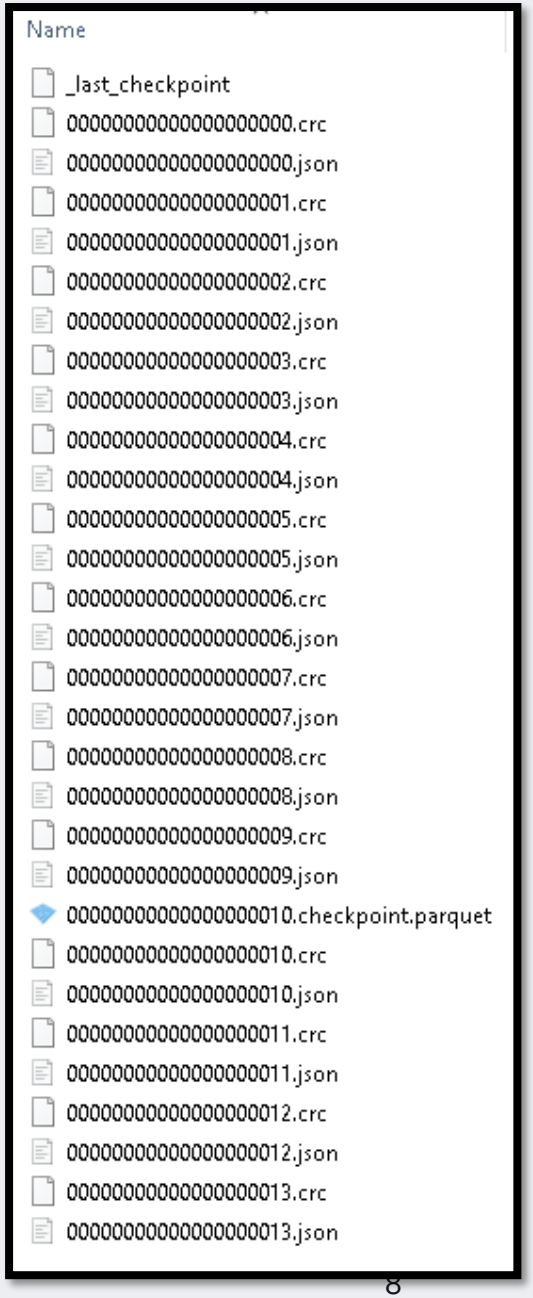


What is the Delta Log?

What is the Delta Log?

The Transactional Layer

- Contains
  - Table schema + changes
  - References to files
  - Metadata and metrics
- Stored as JSON and Parquet
- One file/version per transaction
- Allows [optimistic] concurrency control
- Used for time-travel, streaming, …



What is the Delta Log?

DESCRIBE HISTORY

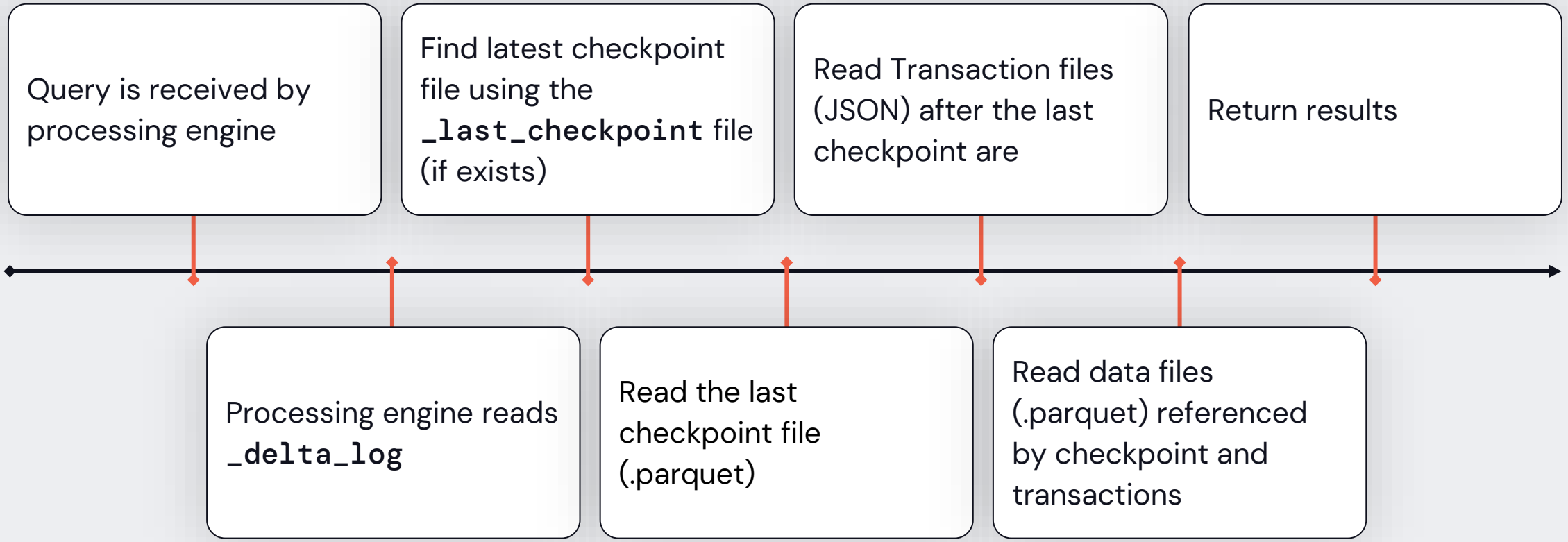
```
DESCRIBE HISTORY gold.my_big_table
```

▸ (1) Spark Jobs

Table Data Profile

	version	timestamp	userId	userName	operation	operationParameters
1	1185	2022-06-13T16:45:39.000+0000			OPTIMIZE	{"predicate": "[]", "zOrderBy": "[]", "batchId": "0", "auto": "false"}
2	1184	2022-06-13T16:18:24.000+0000			VACUUM END	{"status": "COMPLETED"}
3	1183	2022-06-13T16:18:19.000+0000			VACUUM START	{"retentionCheckEnabled": "false", "defaultRetentionMillis": "259200000"}
4	1182	2022-06-13T13:59:19.000+0000			MERGE	{"predicate": "((target.purchase_id = updates.purchase_id) AND (target.iteration "[{\"predicate\":\"(((NOT (updates.store_fee_rate = target.store_fee_rate)) OR (NO target.store_fee_description))) OR ((NOT (updates.store_fee_absolute = target.sto (updates.net_sales_after_fees = target.net_sales_after_fees)))))\",\"actionType\":\"up [{\"actionType\":\"insert\"}]"}
5	1181	2022-06-13T13:56:44.000+0000			MERGE	{"predicate": "(target.purchase_id = updates.purchase_id)", "matchedPredicates "notMatchedPredicates": "[]"}

Processing of a simple Query



How does Delta Lake work?

DML Operations – UPDATE

User

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

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

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

_delta_log

0000.json

"add": { "part-01.parquet", … }

0001.json

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

Storage

part-01
(3 rows)

part-01
(3 rows)

part-02
(3 rows)

DML Operations – DELETE

User

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

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

Product	Price
Notebook	900 €
Tablet	500 €

_delta_log

0000.json

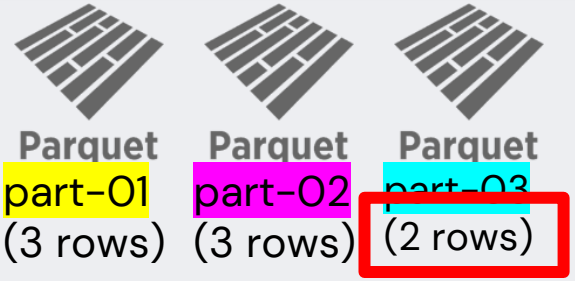
0001.json

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

0002.json

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

Storage



DML Operations – INSERT

User

Product	Price
Notebook	900 €
Tablet	500 €

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

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

_delta_log

0000.json
0001.json

0002.json

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

0003.json

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

Storage

DML Operations

- Operations are logged in **`_delta_log`**
  - Old files are **logically(!)** removed
  - New files are added

- Most operations create new files! Even a **`DELETE` `can`**!

Can create A LOT of files!

File & Storage Management

Data Management – VACUUM

User

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

```
VACUUM DimProduct
```

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

_delta_log

0000.json
...
0003.json

0004.json

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

0005.json

{"VACUUM START “, ... "numFilesToDelete”: 2, ... }

0006.json

{"VACUUM END “, ... “numDeletedFiles”: 2, ... }

Storage

Data Management – OPTIMIZE

User

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

```
OPTIMIZE DimProduct
```

Product	Price
Notebook	900 €
Tablet	500 €
Monitor	200 €

_delta_log

0000.json
...
0005.json

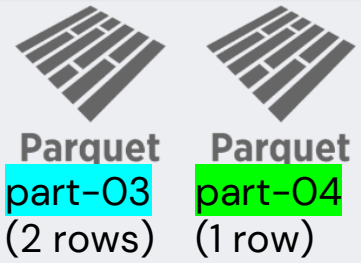
0006.json

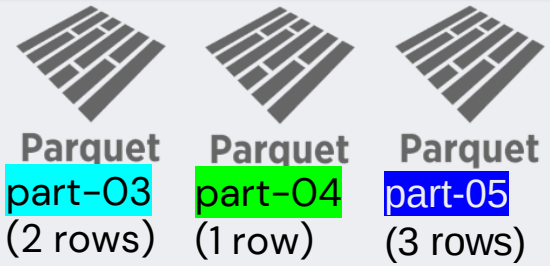
{"VACUUM END “, … “numDeletedFiles”: 2, … }

0007.json

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

Storage





Data Management

VACUUM

- Physically removes unreferenced files older than X days
- Never touches files of latest version of Delta table!

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

OPTIMIZE

- Collapse small files into bigger files
- Clustering / Ordering
- Improve query performance

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

Data Management

VACUUM and OPTIMIZE

- VACUUM DRY RUN
  - Only shows first 1000 files to be deleted
  - Use SCALA to get the actual number of files to be removed!
- Can take a long time!
- OPTIMIZE
  - works per partition level
- Duplicates data!

Data Management

RESTORE

- Restores a previous state of the Delta table
- At `version` or `timestamp`
- Meta-data only operation
- Creates a new version

```
RESTORE events
TO TIMESTAMP AS OF
'2022-05-03'
```

CLONE

- `SHALLOW` or `DEEP`
- Forks Delta Log
  - `DEEP`: copies data files
  - `SHALLOW`: references data files
- Ideal for testing

```
CREATE TABLE
events_clone
SHALLOW CLONE events;
```

Data Management

RESTORE and CLONE

- You can RESTORE as often as you want
  - To rollback another RESTORE
- RESTORE does not create any new [data] files

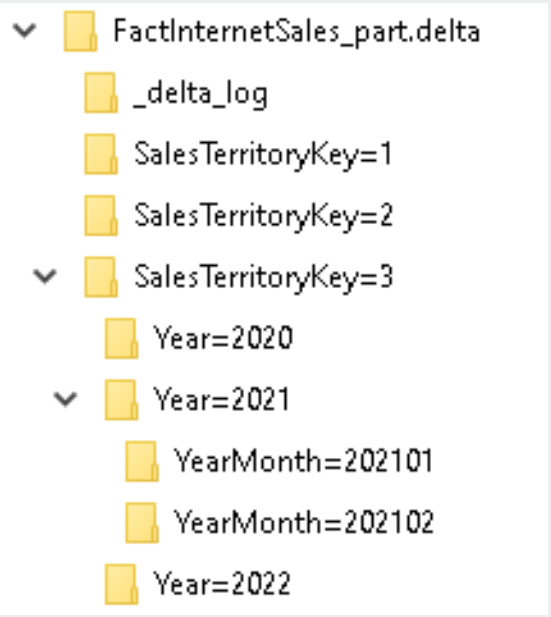
- DEEP Clones are incremental and can be used for Backup

Partitioning

Partitioning

Basics

- Delta Tables can be partitioned
  - For ETL performance (usually on Bronze, Silver)
  - For query performance (usually on Gold)
- **Goal:** touch as few partitions as possible/necessary
  - ETL and Query performance can conflict
  - Explicitly specify Partitioning columns



- Partition by Time [and ?]

Partitioning

Advanced

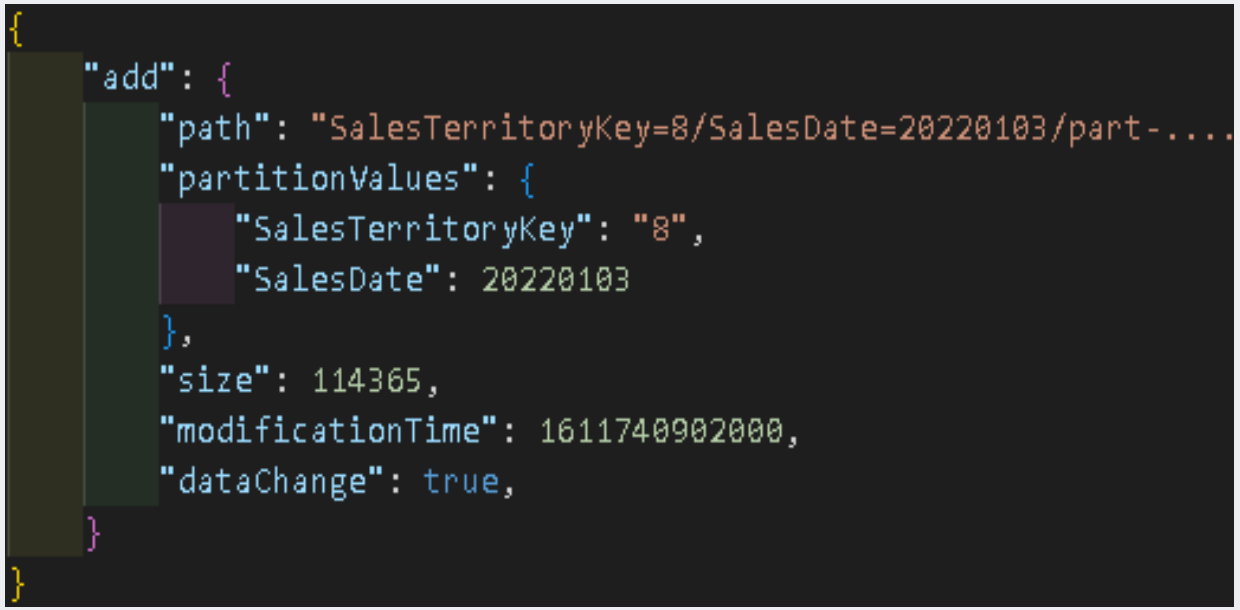
- Avoid over-partitioning!
  - < few 1000s partitions
  - Single partition should be > 1 GB
- Use generated columns
  - **`EventTimestamp`** -> partition by **`CAST(EventTimestamp AS DATE)`**
  - Delta engine will [try to] push filters on **`EventTimestamp`** down to partition
- Used to separate transactions and processing jobs
  - Explicitly specify partitions you touch (e.g. **`MERGE`** target)!
  - Check Delta Log history for query predicates!

Partitioning

Advanced

- Physical .parquet file does not contain the partitioning-columns!

_delta_log Entry



- `path` could point anywhere!

- You do not need to specify all partitioning columns sequentially!

Streaming

Streaming

Basics

- Delta Lake can be used as source and target for streaming
- It's technically still [micro-]batches
  - As is Spark Streaming
- Streaming works on a file-level
- Files are processed in order of
  - Version/Transaction number
  - File index (`part-XXXXX…snappy.parquet`)

Streaming

Advanced

- Checkpoints
  - Track state of what has already been processed from source
  - One checkpoint per source
  - Could stream from same source multiple times using different checkpoints
- `MERGE` only with `foreachBatch()`
- Control the Trigger/Batch size!
- Avoid `Trigger.Once`
- Can stop/resume stream at any time

Delta Lake
Table Properties

Delta Lake Table Properties

- Can be defined on different levels
- Table Properties
  - `delta.autoOptimize.optimizeWrite`
  - `spark.databricks.delta.properties.defaults.optimizeWrite` (default for new tables)
- Configured Settings during Execution
  - `spark.databricks.delta.optimizeWrite.enabled`

- Execution settings have priority over table properties!

Delta Lake Table Properties

Important Table Properties to know

- delta.appendOnly
- delta.autoOptimize.autoCompact
- delta.autoOptimize.optimizeWrite
- delta.deletedFileRetentionDuration
- delta.logRetentionDuration
- delta.dataSkippingNumIndexedCol

Delta Lake Table Properties

- Use defaults for commands
- Define exceptions on table level

→ No need to use individual commands per table

- Changing table properties are also a Delta transaction

Conclusion
&
Lessons Learned

Conclusion

Take Aways

- Delta Lake can solve a lot of problems for you
- File management is crucial
- Data maintenance jobs are mandatory
- Use table properties

Conclusion

References

- The internals of Delta Lake by Jacek Laskowski https://books.japila.pl/delta-lake-internals/
- Delta Transaction Log Protocol https://github.com/delta-io/delta/blob/master/PROTOCOL.md

DATA+AI
SUMMIT 2022

Thank you

Gerhard Brueckl

Cloud Data Architect @ paiqo.com