SQL KONFERENZ 2025



Parquet File Format

Under the Hood

Charley Hanania

MVP: Data Platform, MCT

CEO / Data Platform Specialist

QS2 AG – Quality Software Solutions

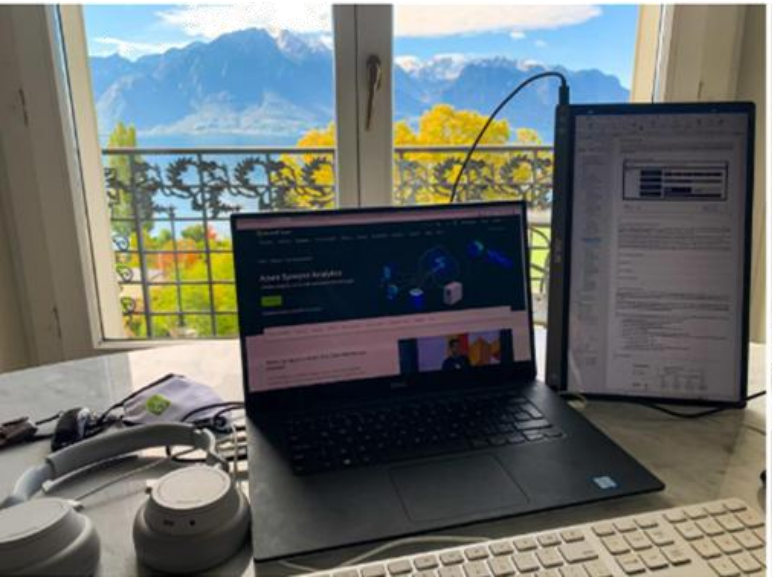


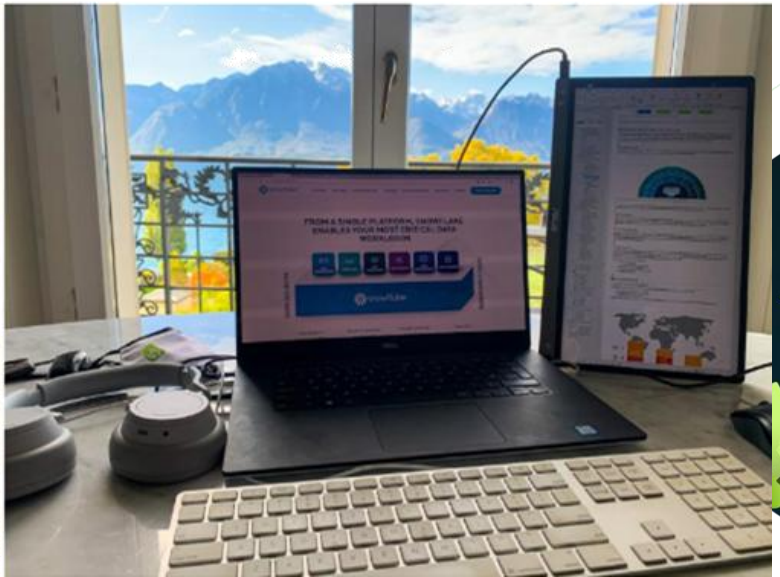
Charley Hanania • You

Data Platform Advisor / Principal Consultant / CEO - QS2 AG

4yr •

Finishing up a deep and difficult #Cloud Data Platform Strategy Evaluation between @SnowflakeDB and Microsoft #AzureSynapseAnalytics for a customer. Can you see how many pages so far? Friend over the weekend compared it to their PhD thesis.







SQL KONFERENZ 2025

Agenda

- Parquet Introduction
- Internals & Structures
- Key Benefits
- Tips on Optimisation Strategies
- Transactional Abstraction Layers
- Questions



SQL KONFERENZ 2025

SQL KONFERENZ 2025

Apache Parquet is a **columnar storage file format** designed for efficient data storage and retrieval.

It is optimized for big data processing and analytics environments. First introduced in 2013, it quickly gained traction due to its ability to improve the performance of read-heavy analytical queries.



Columnar Storage Unlike row-based formats (CSV, JSON), Parquet stores data in columns, allowing for efficient compression, encoding, and processing.
This reduces I/O operations by reading only necessary columns, leading to faster query execution.

Diving Into the Internals

Record Shredding, Nesting and Assembly:

Parquet uses an algorithm to accommodate complex data structures.

Data pages:

Store actual column data.

Dictionary pages:

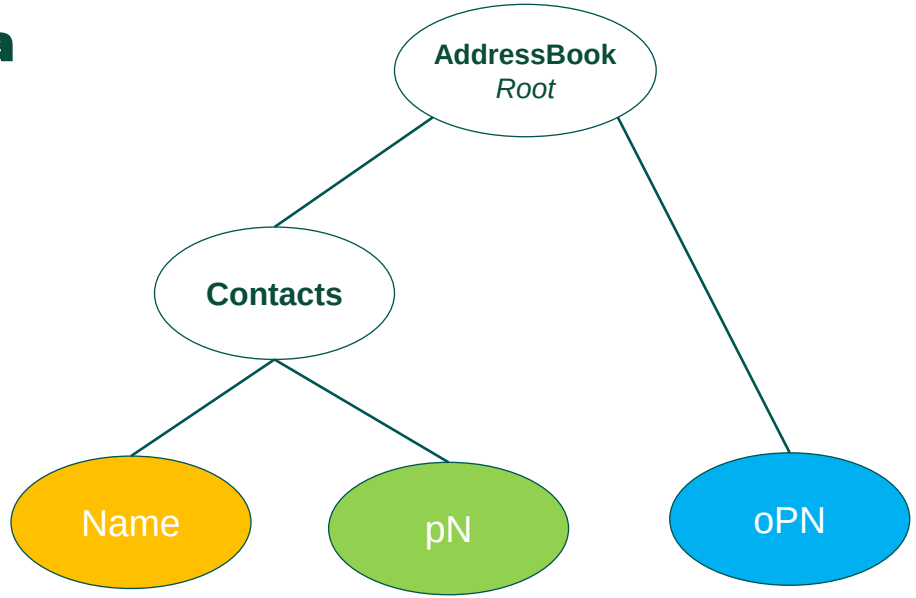
Store unique values for dictionary encoding.

Index pages:

Store index information for faster retrieval.



Nested Schema



Definition Level **0**

Definition Level **1**

Definition Level **2**
(Leaf)

Definition Level **1**
(Leaf)

Column	Type
owner	string
ownerPhoneNumbers	string
contacts.name	string
contacts.phoneNumber	string

AddressBook			
owner	ownerPhoneNumbers	contacts	
		name	phoneNumber
...	...	...	...
...	...	...	...
...	...	...	...

(1) Dremel made simple with parquet - https://blog.twitter.com/2013/dremel-made-simple-with-parquet
(2) Hadoop File Formats and Data Ingestion - Prasanth Kothuri

Columnar Format

Table Representation

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5

(1) Dremel made simple with parquet - https://blog.twitter.com/2013/dremel-made-simple-with-parquet
(2) Hadoop File Formats and Data Ingestion - Prasanth Kothuri

Columnar Format

Table Representation

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5

Row Format



(1) Dremel made simple with parquet - https://blog.twitter.com/2013/dremel-made-simple-with-parquet
(2) Hadoop File Formats and Data Ingestion - Prasanth Kothuri

Columnar Format

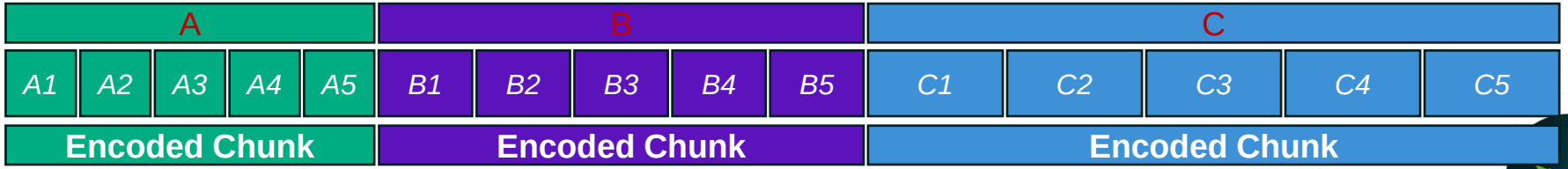
Table Representation

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5

Row Format



Columnar format



(1) Dremel made simple with parquet - https://blog.twitter.com/2013/dremel-made-simple-with-parquet
(2) Hadoop File Formats and Data Ingestion - Prasanth Kothuri

Inside the Apache Parquet format It's Open Source!

In traditional databases (Postgres, SQL Server*) and file types (CSV, JSON), data is stored as rows...

first_name	last_name	street	isPremiumCustomer	Next Month Buying Probability
Abigail	Hayes	4009 9th Street	1	1
Amanda	Clark	87 5th Street	0	2
Melissa	Cl			
Sara	Ben			
Isaac	Hend			
Emma	Ste			
Sean	Mc			
Jason	San			
Alexander	Wa			
Megan	Mc			

Customers.csv

```
first_name,last_name,street,isPremiumCustomer,nextmbuyprob
Abigail,Hayes,4009 9th Street,1,0.26340968441351104
Amanda,Clark,87 5th Street,0,0.17177893540625921
Melissa,Clark,12650 5th Street North,1,0.5438465585898219
Sara,Bennett,203 State Street,0,0.15856312140709694
Isaac,Henderson,732 Magnolia Drive,1,0.19389081059172666
Emma,Stewart,47 Route 32,1,0.8833057932776623
Sean,Morris,254 Lincoln Avenue,0,0.4156333382834478
Jason,Sanchez,26050 Madison Avenue,0,0.5862717219123177
Alexander,Walker,54051 Valley Road,1,0.5941814808482648
Megan,Morris,31 Delaware Avenue,1,0.6289626353900487
```

...which is not optimal for analytical needs:

- 👎 For some file types (like CSV), there is no embedded schema
- 👎 Compression is not good (you're kinda limited to text compression)
- 👎 You need to scan(read) all the rows

*Unless you're using columnstore tables :)

In Parquet, data is stored as columns

Customers.parquet

first_name Abigail Amanda Melissa Sara Isaac Emma Sean Jason

last_name Hayes Clark Clark Bennett Henderson Stewart Morris Sanchez

isPremiumCustomer 1 0 1 0 1 x2 0 x2

Category 0.263409684 0.171778935 0.543846559 0.158563121 0.193890811 0.883305793 0.415633338 0.586271722

Col 1 STRING **Col 2 STRING** **Col 3 BOOLEAN** **Col 4 UINT_32**

File structure is simplified, omitting details like row groups.

- 🔍 Parquet embeds the data Schema
- 🧮 Want the number of premium customers? You can compute that aggregate without reading any other unnecessary data!
- 🗜️ We can efficiently compress repetitive data like this one

With Fabric, you can easily read parquet files stored in OneLake or in other clouds

@cmaneu

Parquet File Structure

Parquet files are organized in a columnar storage format.

File Metadata: High-level information about the Parquet file.

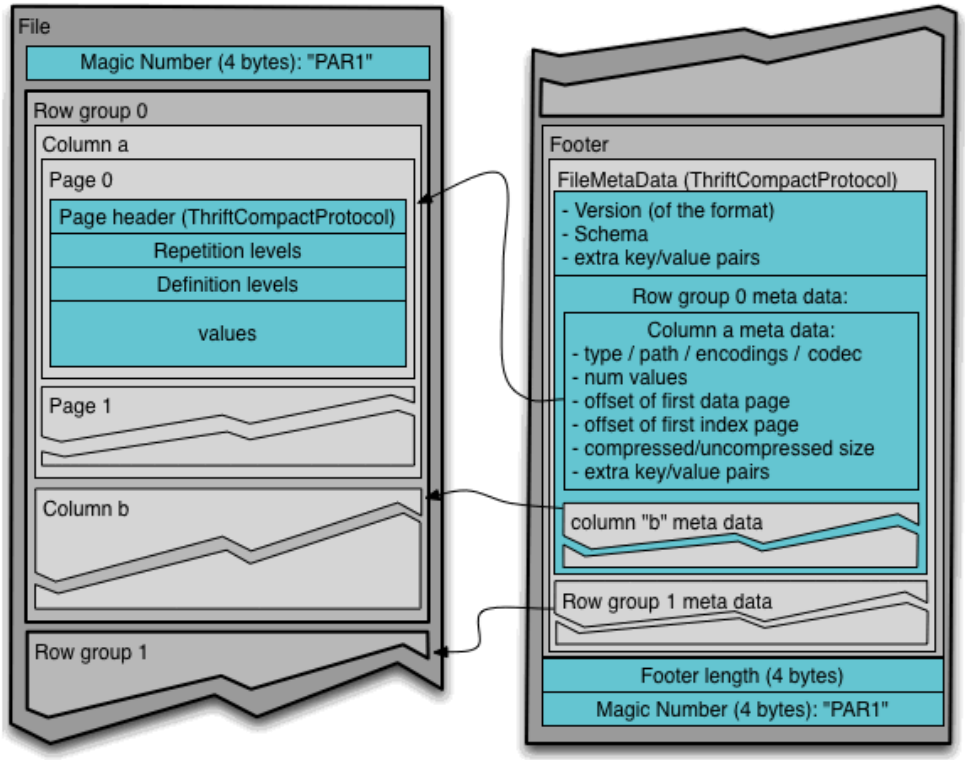
Row Groups: Data is partitioned into row groups.

Column Chunks: Row groups consist of column chunks corresponding to columns in the dataset.

Pages: Column chunks are divided into pages, which are units of compression and encoding



Parquet File Structure & Configuration



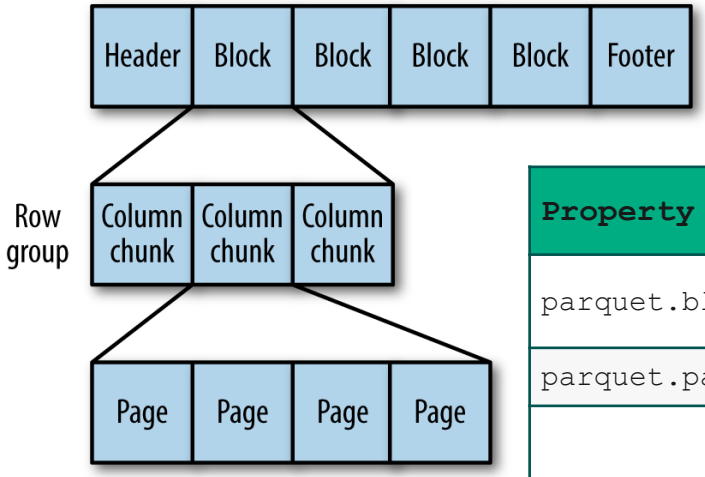
Self-describing

- File Metadata is written after the data to allow for single pass writing.
- Readers are expected to first read the file metadata to find all the column chunks they are interested in.
- The columns chunks should then be read sequentially.



https://parquet.apache.org/

Parquet File Structure & Configuration



Configurable parquet parameters

Property name	Default value	Description
parquet.block.size	128 MB	The size in bytes of a block (row group).
parquet.page.size	1MB	The size in bytes of a page.
parquet.dictionary.page.size	1MB	The maximum allowed size in bytes of a dictionary before falling back to plain encoding for a page.
parquet.enable.dictionary	true	Whether to use dictionary encoding.
parquet.compression	UNCOMPRESSED	The type of compression: UNCOMPRESSED, SNAPPY, GZIP & LZO

https://parquet.apache.org/

Types of Metadata

File-level metadata: Schema, version information.

Row group-level metadata: Statistics about row subsets (row count, column sizes, compression).

Column-level metadata: Statistics about individual columns (min/max values, null counts, data types)



SQL KONFERENZ 2025

Key Benefits

Predicate Pushdown: Filters are pushed down to the Parquet format, skipping irrelevant data and improving performance.

Compression and Encoding: Parquet uses compression algorithms and encoding techniques specific to each column, optimizing storage and data transfer.

Schema Evolution: Adapts to changing data requirements without significant data transformation.

Interoperability: Compatible with various big data processing tools.

Metadata: Includes schema and encoding information for query optimization



Column Pruning and Predicate Pushdown

Table Representation

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5



Column Pruning and Predicate Pushdown

Column Pruning

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5



Column Pruning and Predicate Pushdown

Column Pruning

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5

Predicate Pushdown

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5



Column Pruning and Predicate Pushdown

Column Pruning

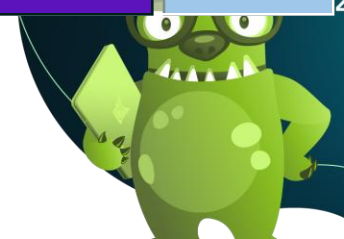
A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5

Predicate Pushdown

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5

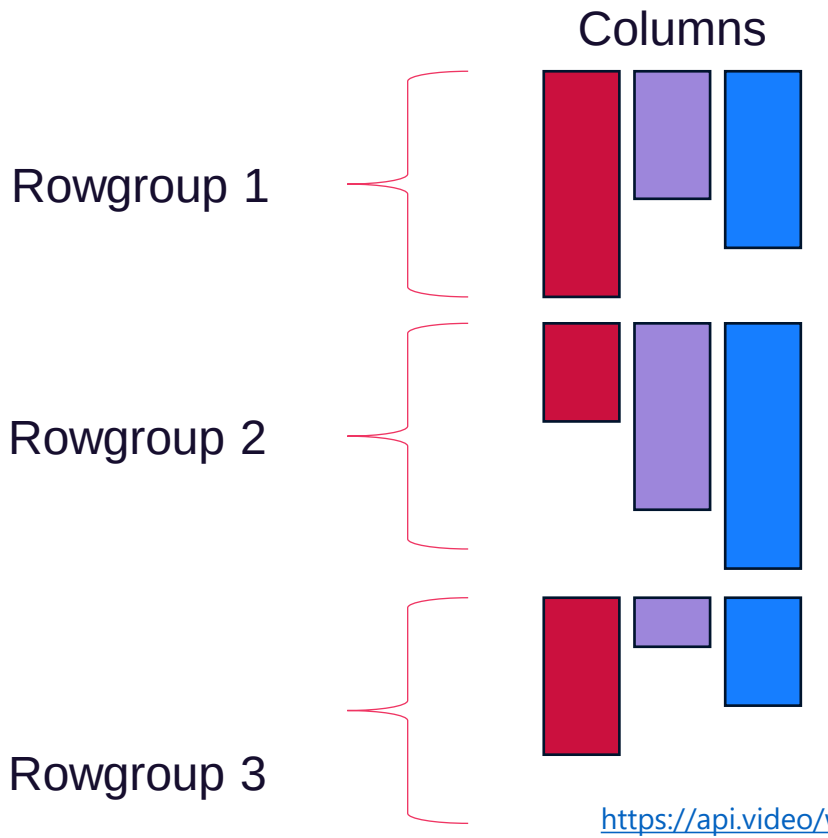
Read needed data only

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C4
A5	B5	C5



SQL KONFERENZ 2025

Strange name - “Parquet”





Compression using

- **Dictionary encoding**
- **Bit packing**
- **Run length encoding (RLE)**

https://api.video/what-is/run-length-encoding



SQL
KONFERENZ
2025



Tips on Optimisation Strategies

Remember:

Performance Tuning & Optimisation is a Journey, not a Destination

SQL KONFERENZ 2025

Optimisation Strategies

- **Partitioning**
  - Mostly by Date (YYYYMMDD)
  - Similar to having a clustered index in a database (Physical layout)
  - One partitioning scheme per dataset
- **Tip**
  - Know the pattern by which the dataset will be queried
  - Determine the Partitioning at the Medallion Layer



SQL KONFERENZ 2025

Optimisation Strategies

- **Batching**
  - Affects the size of data in the Row Groups
- **Tip**
  - Consider larger row groups to improve compression efficiency with more data to compress at once.
  - Larger leads to more efficient sequential reads (faster than random reads), but needs more memory to process
  - Adjust to balance the trade-offs between I/O efficiency, memory usage, and compression



SQL KONFERENZ 2025

Optimisation Strategies

- **Sorting**
  - Data loaded is sorted at ingestion
  - Affects the Compression savings
  - Run-Length Encoding (RLE) is applied at this level
- **Tip**
  - Use sorting to reduce the size of your data
  - Sort by the lowest cardinality column first
    - This leads to the highest reduction in data size.



SQL KONFERENZ 2025

Optimisation Strategies

- **Abstraction layers**
  - Examples: Apache Iceberg, Apache Hudi and Databricks Delta Lake
  - Gives a Relational/Operational Table concept over the Parquet files
- **Tip**
  - Use Abstractions that work for your use case
  - Think Ingestion, Analytics or Transactional Streaming
  - Interoperability is another factor



SQL KONFERENZ 2025

Optimisation Strategies

- **Data Types**
  - Remember that encoding happens when you write and read the data
- **Tip**
  - Ensure that you're aligned beforehand so that additional compute isn't needed



SQL KONFERENZ 2025

Optimisation Strategies

- **External Compression**
  - After the columnar compression is applied, this is applied at the file level



- **Tip**
  - Snappy: Known for a good balance of speed and compression ratio
  - Gzip: For less frequently accessed data, Gzip might be used if storage space is critical and read times are less important.

SQL
KONFERENZ
2025



Transactional Abstraction Layers

Apache Iceberg, Apache Hudi and Databricks Delta Lake







What's ACID?

Atomicity

- A series of database operations either **all** occur, or **nothing** occurs. A guarantee of atomicity prevents updates to the database occurring only partially.

Consistency

- Any changes to values in an instance are consistent with changes to other values in the same scope.

Isolation

- Other operations cannot access data that has been modified during a transaction that has not yet completed.
- The question of isolation occurs in case of concurrent transactions, and within the same database.

Durability

- The ability to recover the committed transaction updates against any kind of system failure



Why the need for ACID?

Batches

- Traditionally Data warehouses receive changes and additions in **timed batches**.
- Batches would be **processed in offline hours** when reporting needs were low.

Transformations (ETL/ELT/ELTL)

- Data moving through the transformation process may need to be **changed over time** after data has arrived, affecting reporting consistency over time. eg. GDPR
- **Errors** may occur during loading requiring cleanup and retries from commit points

Conflicting Analytical Needs

- Many implementations saw the need of both reporting on **historical** data trends as well as **(near) real-time** analysis on the current operational workloads. This was generally achieved through Lambda & Kappa Architectures

Standardization and Validation

- The flexibility of the bigdata distributed filesystems needs additional **validations and cleanup** for it to be usable.



Apache Iceberg

- Can be layered on top of multiple filetypes, eg Parquet & Avro
- Uses catalogs to manage table metadata
  - The “metadata tree” tracks a table’s partitioning, sorting, and schema over time
  - These point to the current state of the table and list all the Parquet data files that make up the table at a given point in time
  - The metadata allows you to query the data as it existed at a specific point in time, allowing time travel
- Uses snapshots to manage changes to the table which represent the state of the table at a specific point in time
- When writing new data it creates new Parquet files and updates the metadata to point to these new files
  - The metadata management adds some overhead to write operations
- Uses column IDs to track columns, so it can identify columns even if they are renamed
- Supports other abstractions like Delta Lake



Apache Hudi



- Enhances Parquet with extra metadata/indexing capabilities to enable updates and deletes
- Maintains an index that maps each record to the Parquet file it belongs to
  - Stores metadata about each write operation, eg the commit time and user who changed
- Provides two main ways to handle write operations
  - Copy-on-Write: rewrites the entire Parquet file containing that record
  - Merge-on-Read: Updates are written to separate "delta" files, & later merged during reads
- Metadata allows querying the data as it existed at a specific point in time (Time Travel)
- Setting up/managing Hudi can be more complex than simply writing Parquet files



Delta Lake



- Written as standard Parquet files with additional metadata
- Adds a transaction log in the _delta_log directory to the Parquet files (aka DeltaLog)
  - Each file represents a commit and contains pointers to Parquet files & metadata
  - Checkpoint files generated every 10 commits which saves the entire state of the table at a given point in time
- Implements serializable isolation for concurrent operations
- Uses a metadata layer for metadata tracking like schema shift
- Performs schema validation on write
  - If compatible, it will pass and the write will succeed
  - If incompatible, it will cancel the transaction and no data is written



Thanks!

Questions?



SQL KONFERENZ 2025

- **Contact**: Charley.Hanania@qs2.ch
- **Twitter/X**: https://x.com/CharleyHanania
- **LinkedIn**: https://www.linkedin.com/in/charleyhanania/

Session Feedback

SQL KONFERENZ 2025

