ClickHouse Meets Iceberg

A practical look beyond the hype

Hello, Andi Pangeran

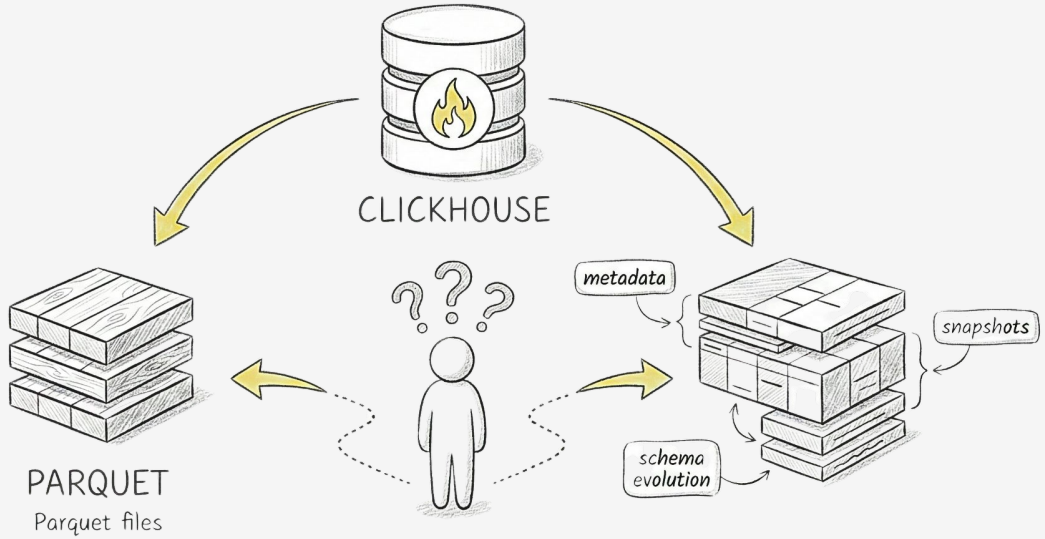
- Over *12+ years of experience* in the IT Industry
- Currently, *Principal Software Engineer* *at Electrum.id*

@apangeran @a_pangeran

@A_Pangeran

Parquet. Iceberg. DataLake. Lakehouse.

But... What Do They Actually Do?



So before choosing tools or architecture —
let's understand what's inside.

Agenda ?

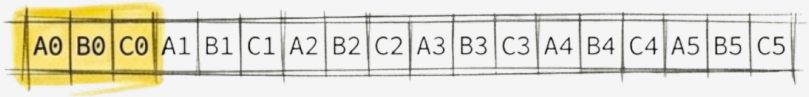
- **Parquet under the hood**
  What Parquet actually is — beyond "just files."
- **Why Iceberg exists**
  The limitations Parquet alone couldn't solve.
- **Iceberg internals & complexity**
  How it works — and what operational cost comes with it.
- **ClickHouse is all you need**
  Where ClickHouse fits as a fast, pragmatic engine.
- **Demo**
  Querying Parquet and Iceberg using ClickHouse.

Layout Models

A Logical table can have different physical reality

Row-based

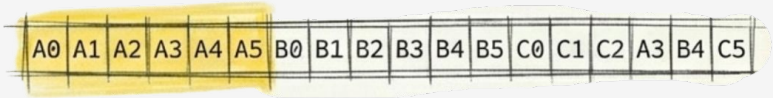
	Col A	Col B	Col C
Row 0	A0	B0	C0
Row 1	A1	B1	C1
Row 2	A2	B2	C2
Row 3	A3	B3	C3
Row 4	A4	B4	C4
Row 5	A5	B5	C5



- Great for **full-row reads, inserts,** and transactional workloads
- Weakness: **slow for analytics** — must read all columns

Columnar

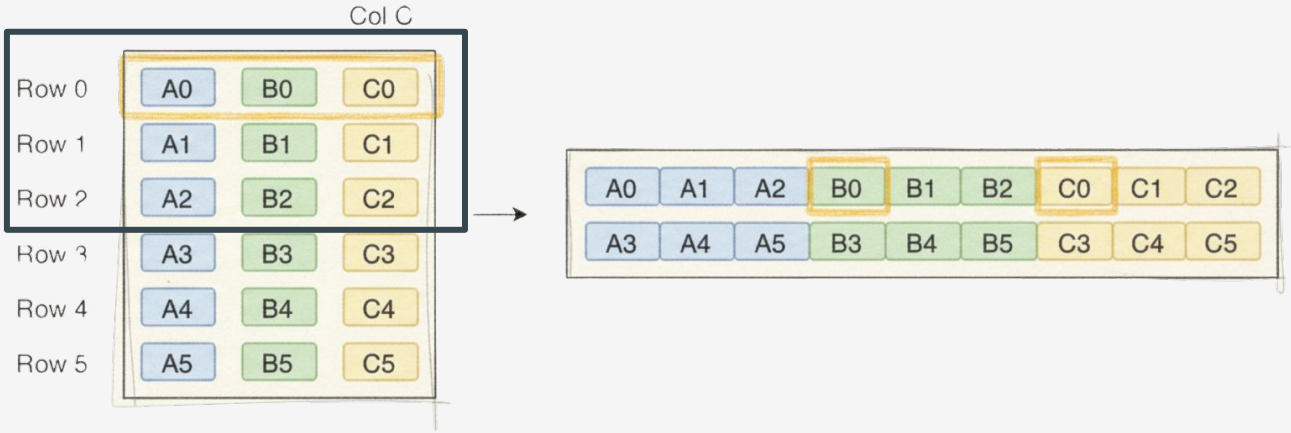
	Col A	Col B	Col C
Row 0	A0	B0	C0
Row 1	A1	B1	C1
Row 2	A2	B2	C2
Row 3	A3	B3	C3
Row 4	A4	B4	C4
Row 5	A5	B5	C5



- Ideal for **analytical scans, compression, filters, aggregates**
- Weakness: **slow for point lookups** and frequent updates

Layout Models

A Logical table can have different physical reality



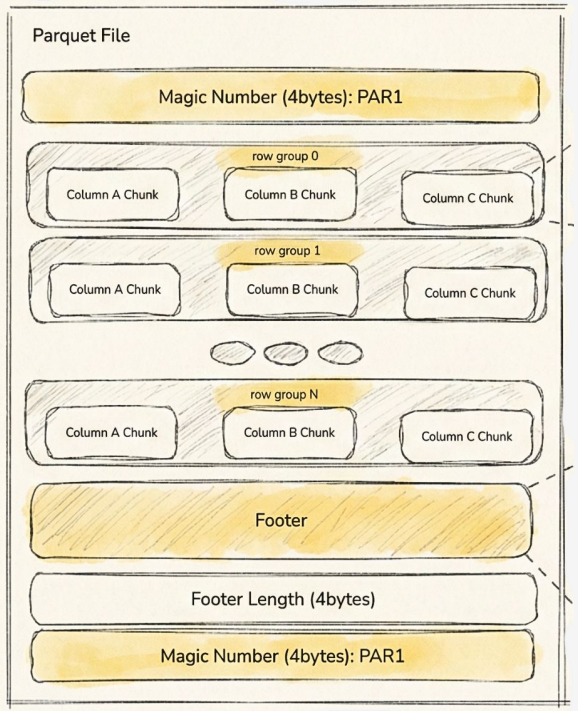
A hybrid layout stores data in **row-based blocks**, but inside each block the **attributes are stored column-wise**.

Parquet under the hood

What Parquet actually is — beyond "just files."

A Parquet File is Hierarchy and Metadata

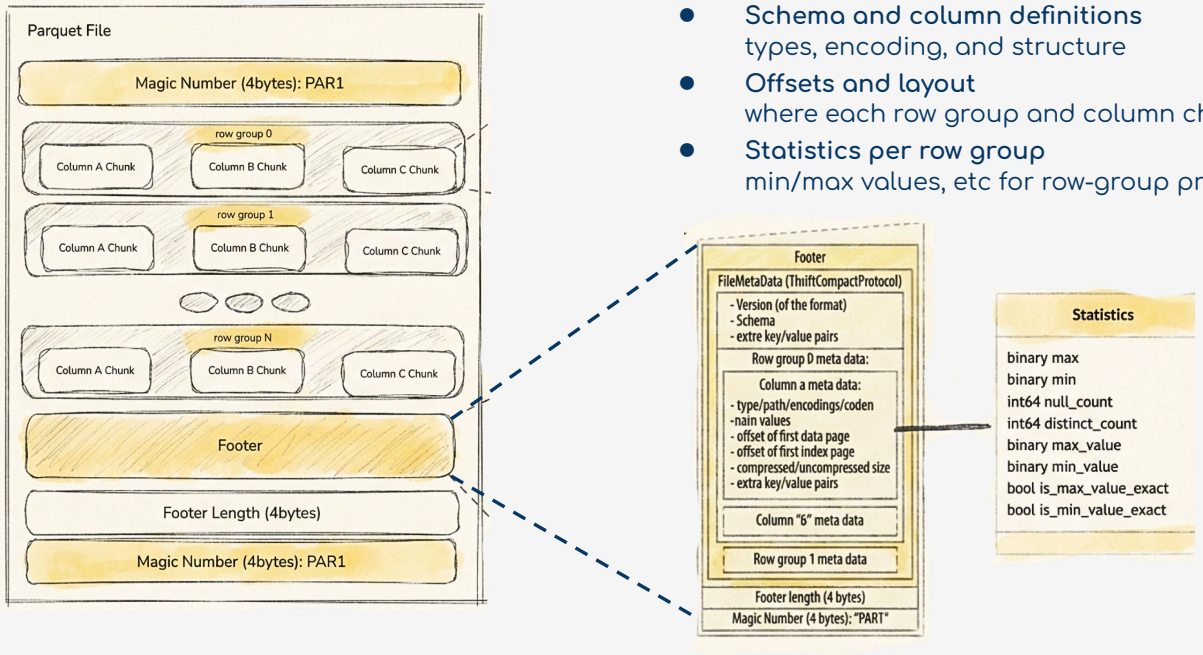
Parquet organizes data into structured layers rather than storing raw rows.



- Data is grouped into **row groups** — batches of rows stored together
- Each **row group** contains multiple **column chunks** — one chunk per column
- A **footer** stores **metadata** — schema, layout, and offsets describing the file

The Footer Holds the Blueprint of the File

Before reading any data, an engine learns the structure by reading the footer.

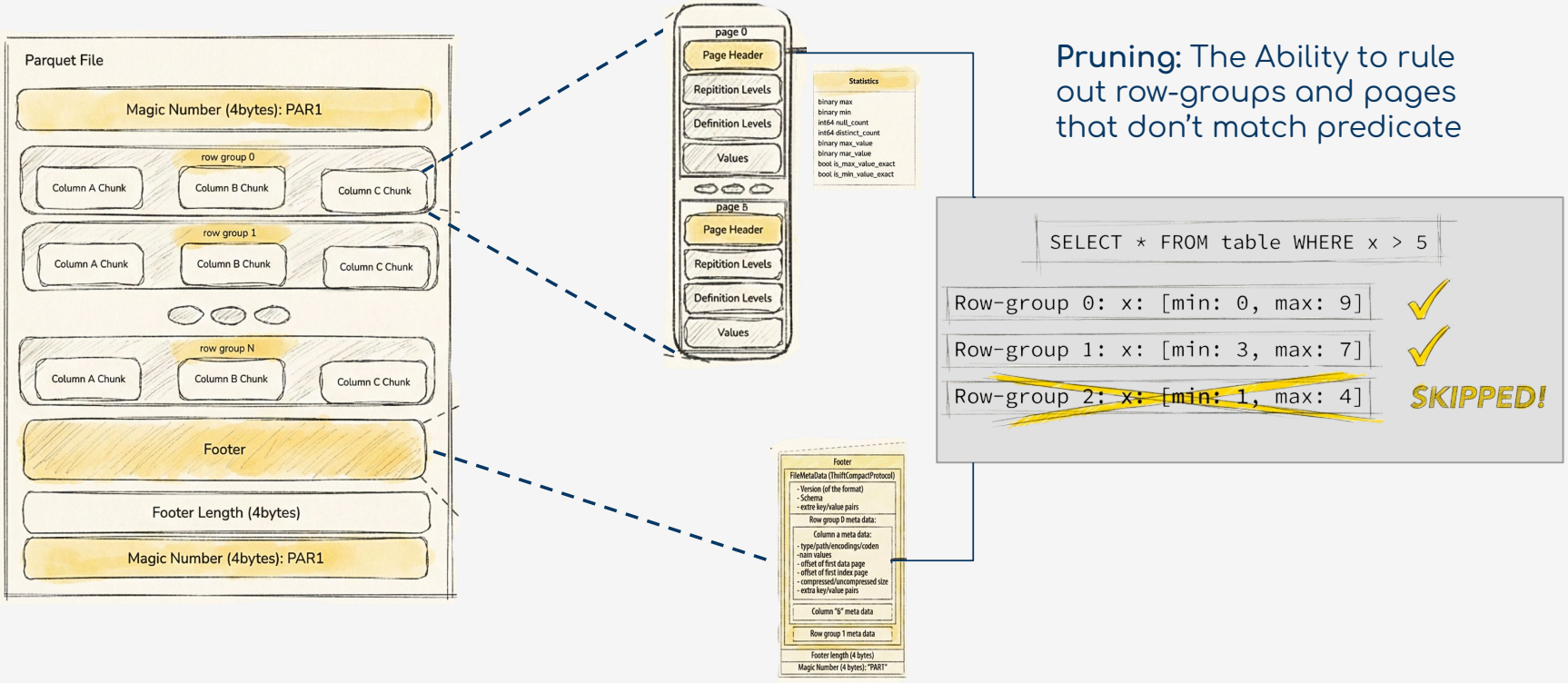


- **Schema and column definitions**
  types, encoding, and structure
- **Offsets and layout**
  where each row group and column chunk lives
- **Statistics per row group**
  min/max values, etc for row-group pruning

The footer gives enough information to navigate the file without loading everything.

Column Chunks Are Split Into Pages

Metadata doesn't stop at the footer — Parquet also stores details inside each column chunk.



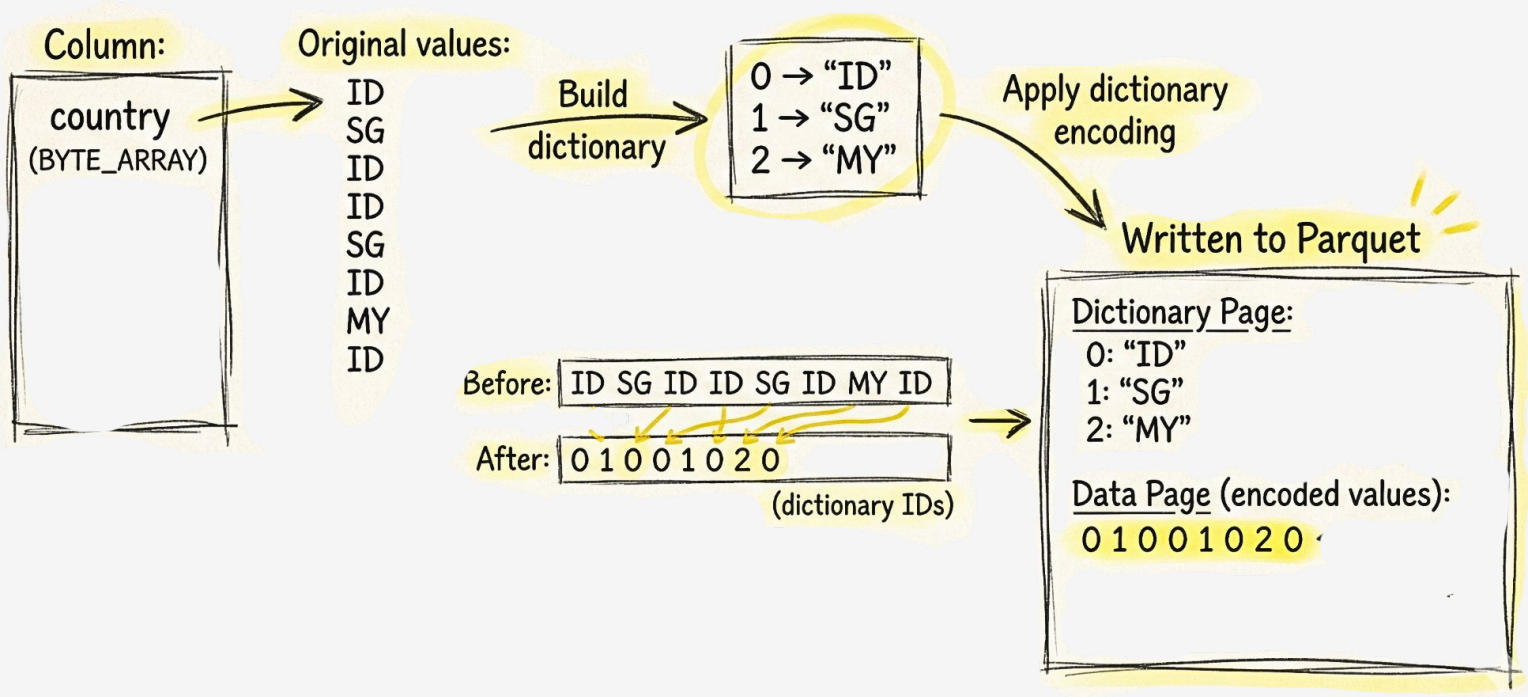
Pruning: The Ability to rule out row-groups and pages that don't match predicate

```
SELECT * FROM table WHERE x > 5
```

```
Row-group 0: x: [min: 0, max: 9]   ✓
Row-group 1: x: [min: 3, max: 7]   ✓
Row-group 2: x: [min: 1, max: 4]   SKIPPED!
```

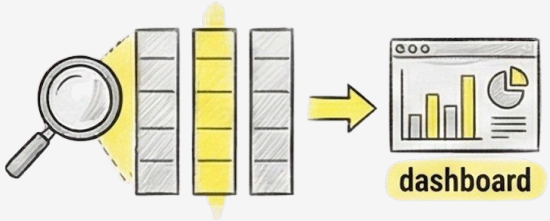
Data-aware encodings plus page-level compression (ZSTD/Snappy)

Encoding reduces file size and improves scan efficiency — dictionary encoding is just one example



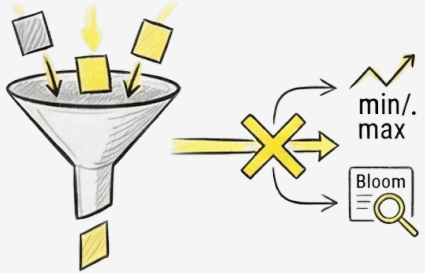
✅ What's Good About Parquet

1. Fast for analytical workloads



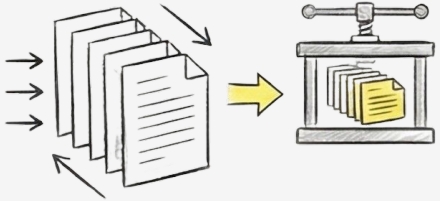
Parquet is columnar, engines only read the columns required by the query—**great for dashboards, analytics, and aggregations**.

2. Strong metadata for pruning



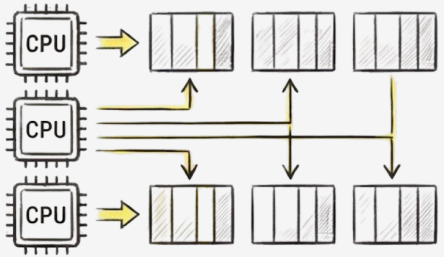
It stores min/max statistics, dictionary values, and even Bloom filters, enabling engines to **skip irrelevant chunks of data** before reading.

3. Efficient compression & encoding



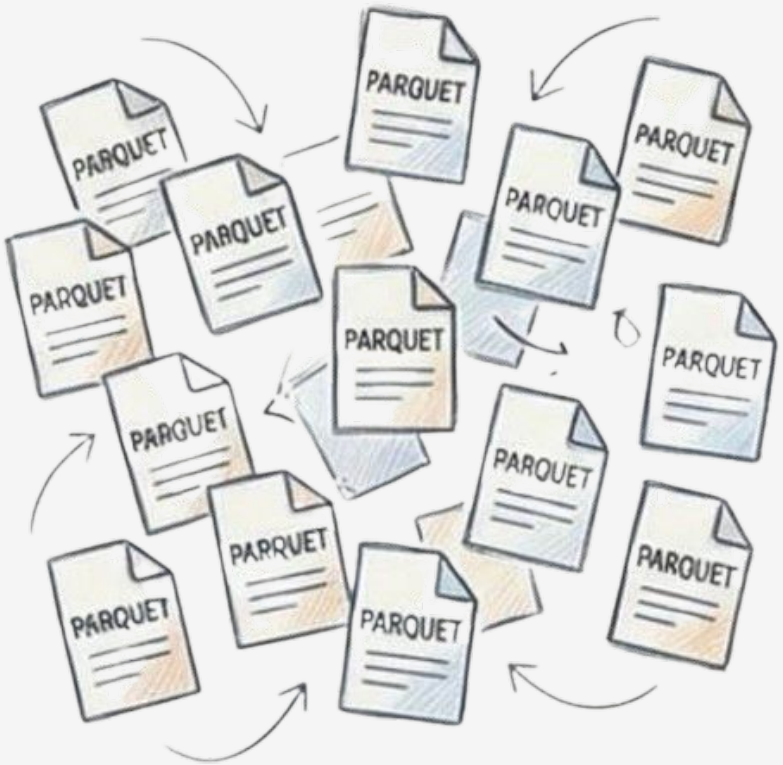
Data-aware encodings (dictionary, RLE, delta) plus page-level compression (ZSTD/Snappy), resulting in **smaller storage footprint and reduced I/O**.

4. Parallelizable



Files are divided into **row groups**, allowing engines to process them in **parallel**. Some engines even parallelize across **column chunks**.

⚠️ at scale, it creates fragmentation and inefficiency



🚫 Too Many Small Files

When the dataset fragments into thousands or millions of tiny Parquet files:

- **Excessive metadata reads** (file footers)
- **High storage request** overhead (S3/GCS GET calls)
- **Poor pruning** → queries degrade to scans

Result: fast schema, slow queries.

🚫 One Giant Parquet File

When the dataset grows into very large files :

- Slow to read and skip
- Hard to update (no rewriting single row)
- No concurrency control
- Writes require rewriting the entire file (expensive & slow)

Result: efficient scans, painful ingestion and mutation.

⚠️ bcoz

PARQUET It's a **file format**, not a table system

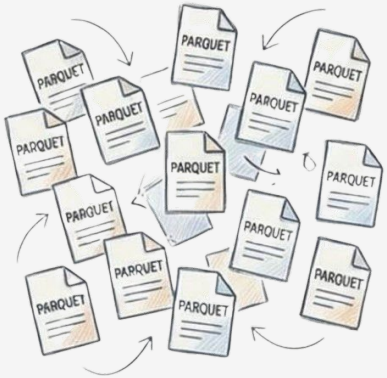
- ✗ Transaction
- ✗ Schema evolution
- ✗ Commit

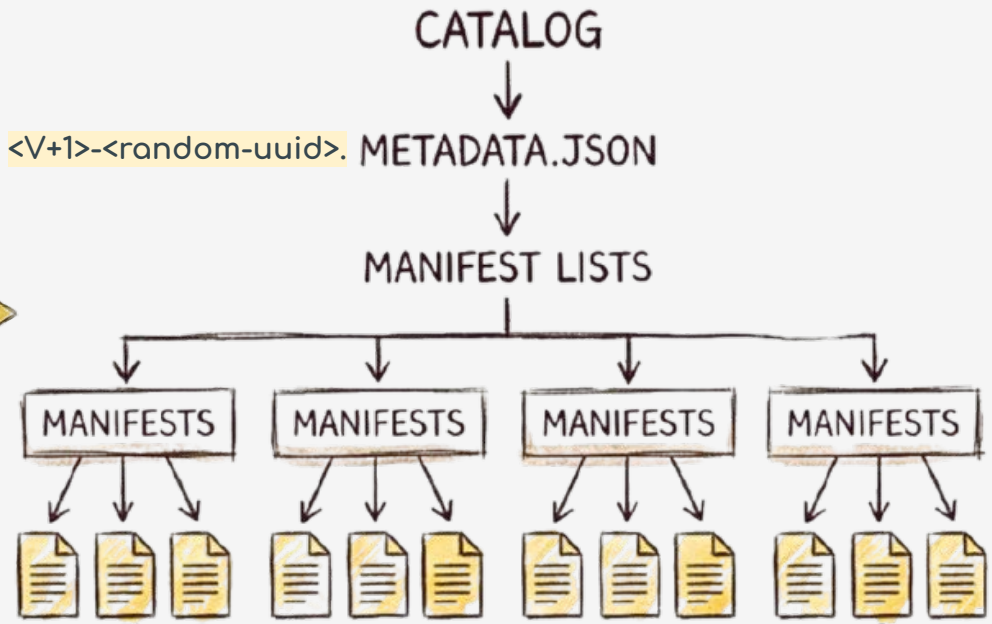
Parquet doesn't handle: transactions, schema evolution coordination,commits / snapshots, concurrency


ICEBERG

Iceberg Makes Files Behave Like a Table

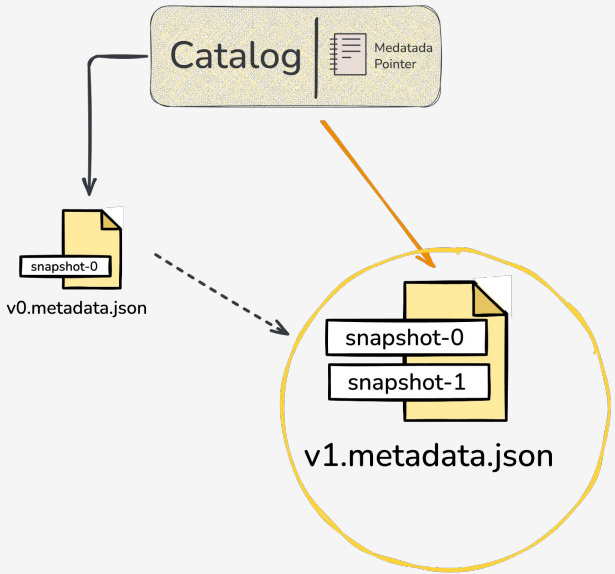
A table format that brings structure, governance, and transaction consistency to data stored in files.





Works with multiple file formats
Parquet, ORC, Avro (and extensible in future)

The Catalog is our single source of truth for table state



ATOMIC SWAP

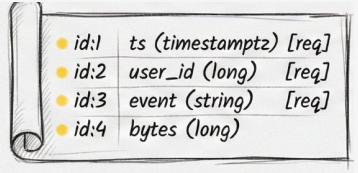
👉It tracks the current **metadata pointer** for every table

👉It ensures transactional integrity through an atomic compare-and-swap operation.

Metadata File the logical structure of the table and tracks all versions over time.

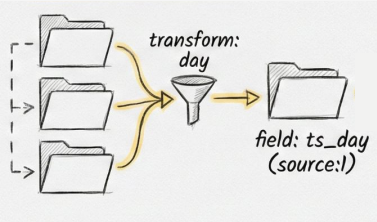
Schema

Defines the shape of the table (columns, types, identifiers). Ensures all readers/writers interpret data consistently, even across evolution.



Partition Specifications

Describes how data is logically grouped (e.g., day(ts) or bucket(user_id,16)). Enables efficient pruning without relying on folder names or physical layout.

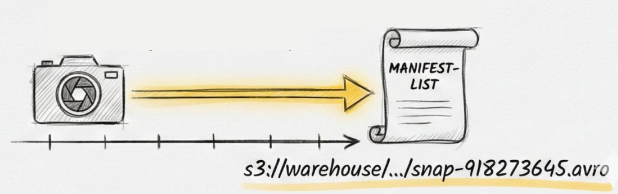


Sort Order

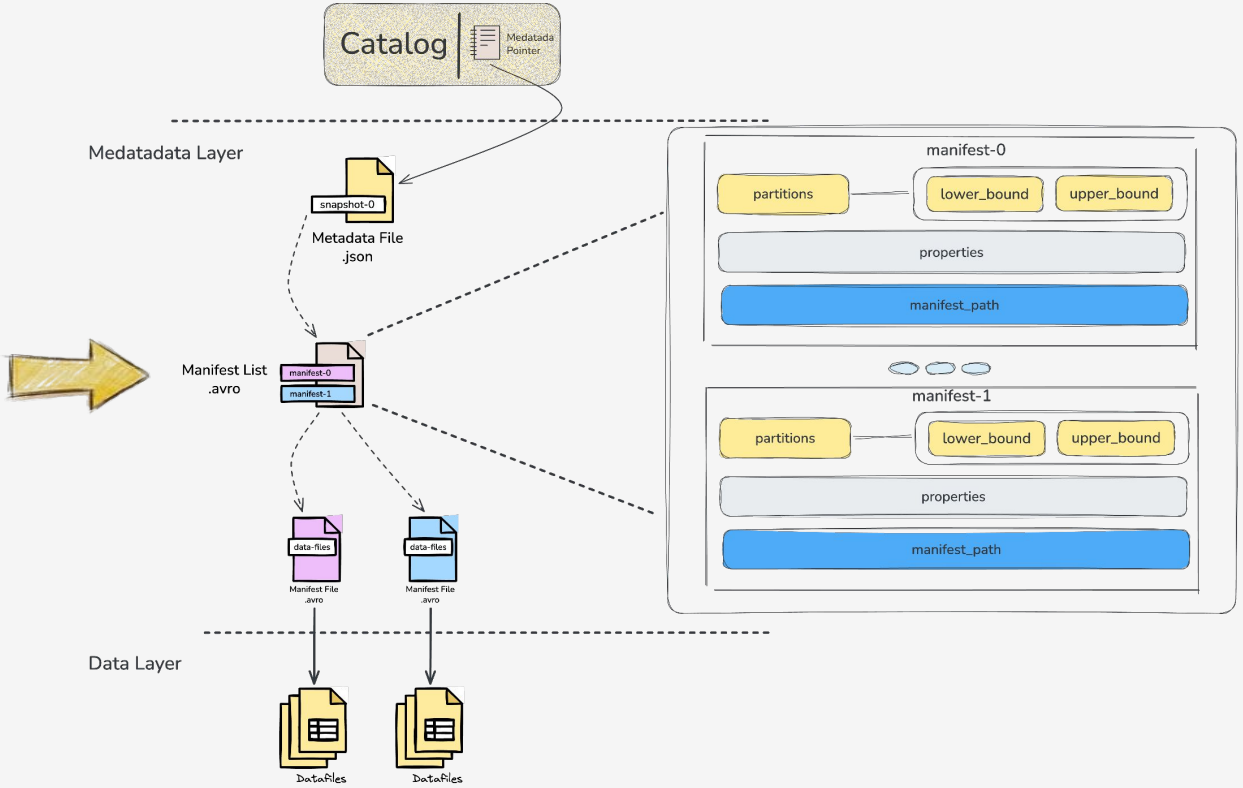
Defines how data is clustered inside files to improve scan locality and skip ranges efficiently.

Snapshots

Each snapshot represents a complete version of the table at a point in time. Supports **ACID transactions, rollback, branching, and time-travel queries.**



Manifest List enables the query engine to identify which manifests are relevant before reading any file metadata.



Manifest File Paths

A lightweight index pointing to one or more manifest files used by the snapshot.

Partition Boundaries (Min/Max)

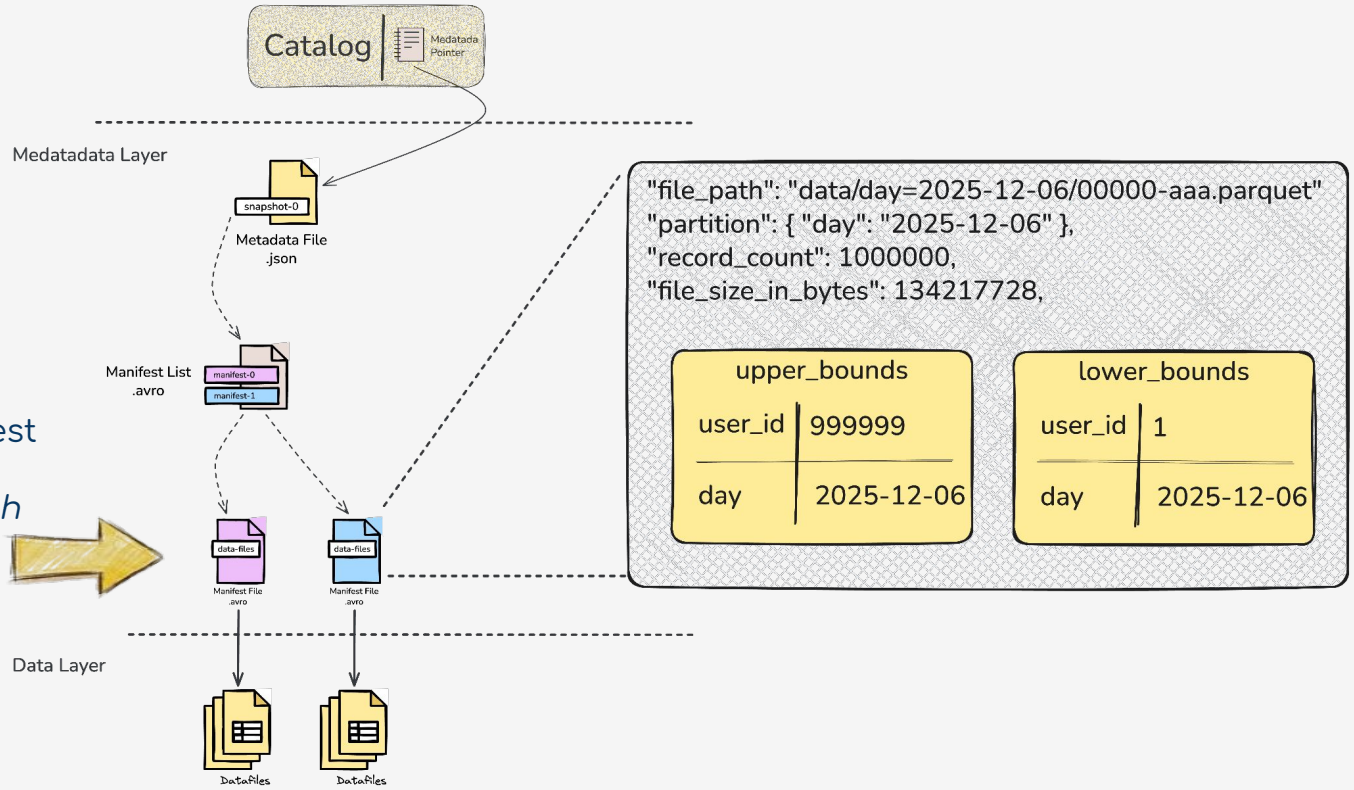
Lower/upper partition values allow the engine to skip entire manifests.
→ First-level pruning.

File Counts & Stats Summary

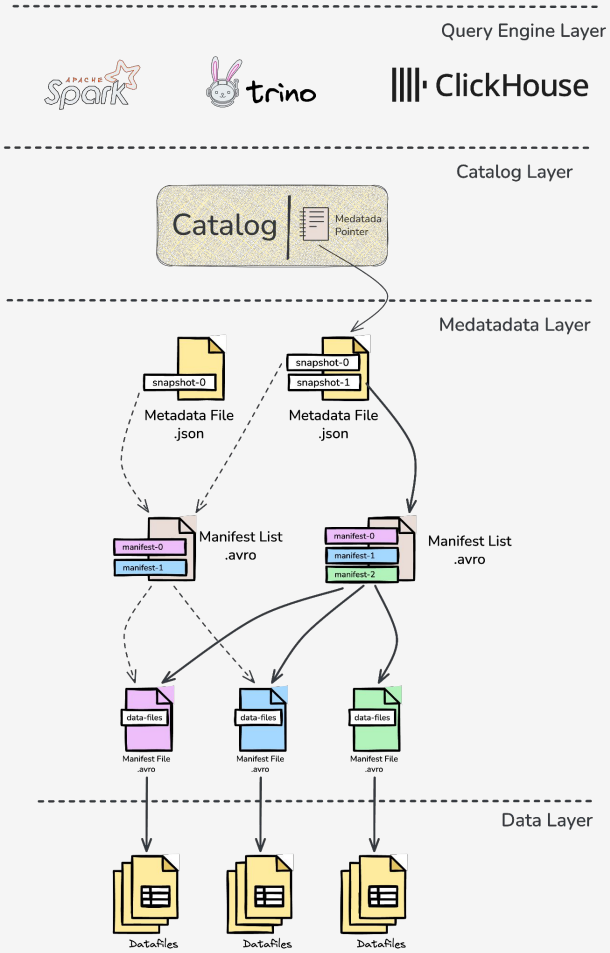
Total added/removed/existing files, row counts, and metadata state. Helps planning scan strategies without reading all manifests.

Manifest File provide the detailed metadata necessary for fine-grained pruning, so the engine reads only the files that contain relevant rows.

In short: the manifest file tells the engine *which files are worth reading*, and which can be skipped.

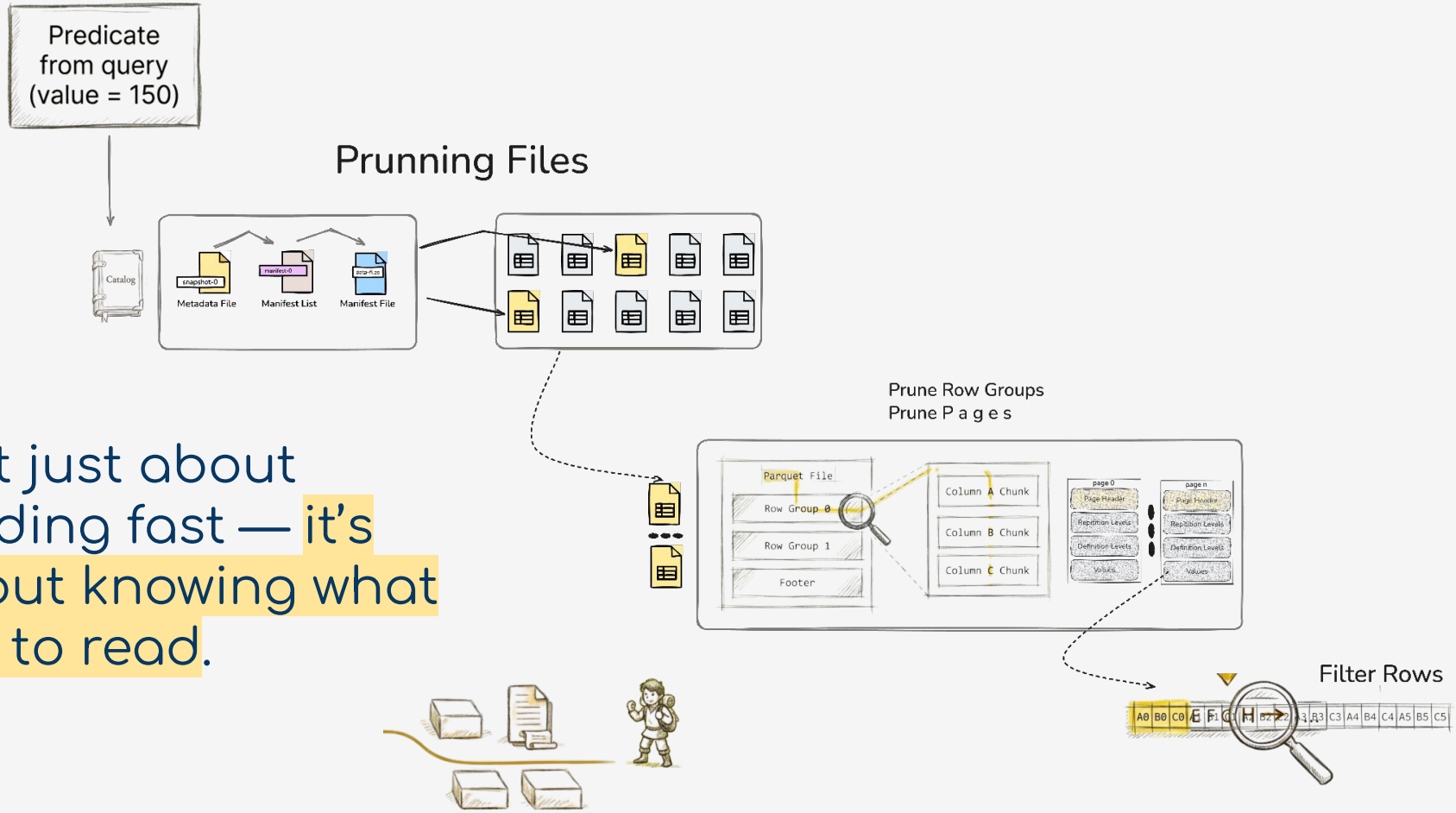


Apache Iceberg — Typical Architecture



Layer	Typical components / examples
Query engine layer	Trino, Spark, Flink, Dremio, Presto/Starburst, DuckDB, Clickhouse
REST Catalog	built-in Iceberg catalog/endpoint: • **Amazon S3 Tables,** • **Cloudflare R2 Data Catalog** Others: **Nessie**, Apache **Polaris**, ETC
Metadata layer	**Object storage:** S3 / R2 / GCS / MinIO (usually under .../table/metadata/)
Data layer	**Object storage:** S3 / R2 / GCS / MinIO (usually under .../table/data/)

The journey of a query engine



isn't just about reading fast — it's about knowing what not to read.

What's Good About Iceberg

Schema & partition evolution *(without folder hacks)*

Snapshots *(time travel / rollback / audit)*

Cross-engine interoperability *(shared table definition)*

Deletes / updates become possible (but not free)

The Hidden Cost

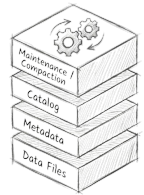
New questions: *Which catalog? how to compact? how to expire?*

More stacks to operate :

More components, more integration, more monitoring, more failure modes.

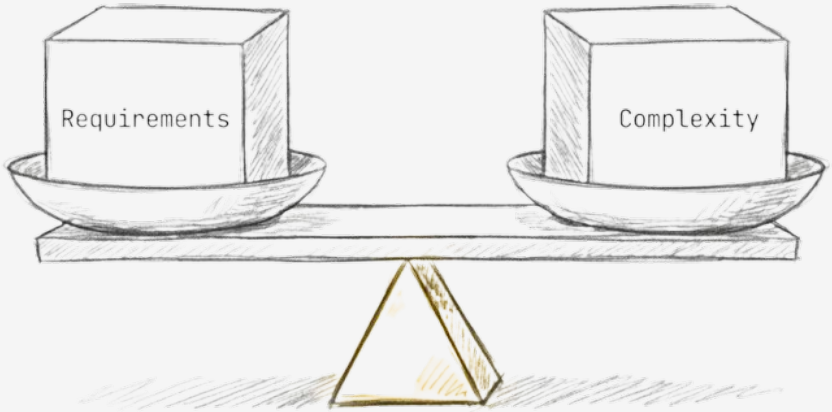
Operational Realities:

- **Catalog outages = all queries stop**, even if data files are healthy
- **Table Maintenance** Compaction, Expire Snapshots, Orphan Files



“Don’t over engineer”

Pick the simplest stack that meets your needs.



Every added layer introduces more failure modes and maintenance overhead.

If you need low-latency or realtime queries → store data in a fast analytics engine and colocate it with compute.

If you need cheap long-term retention → move cold data to object storage.

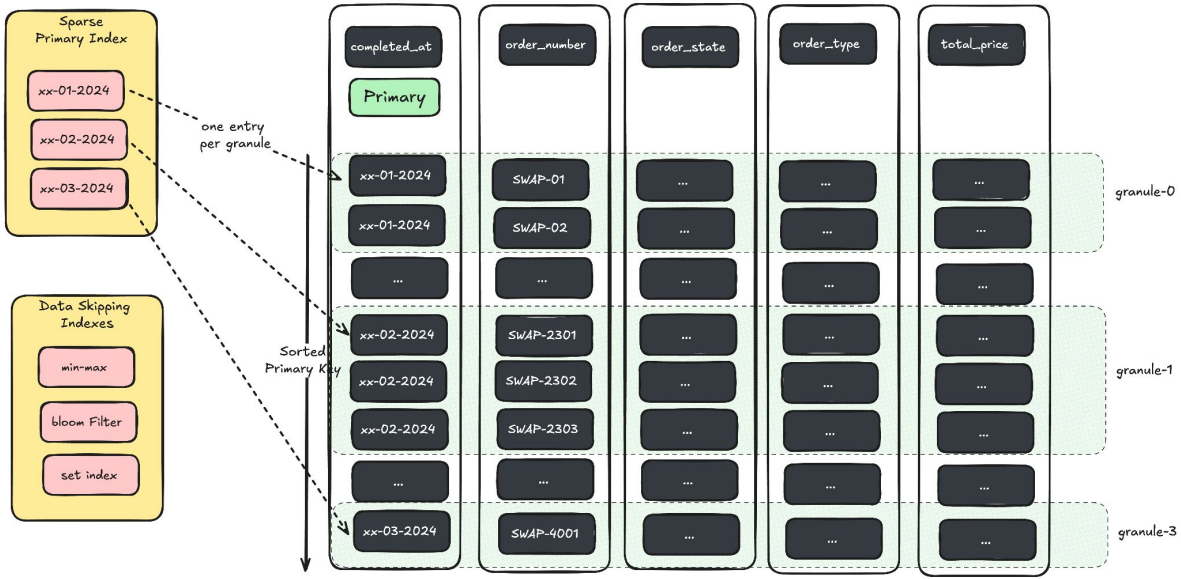
If you need interoperability across engines → use a table format.

Clickhouse is all you need

Fast queries, efficient storage, and streamlined data processing

Clickhouse delivers faster queries

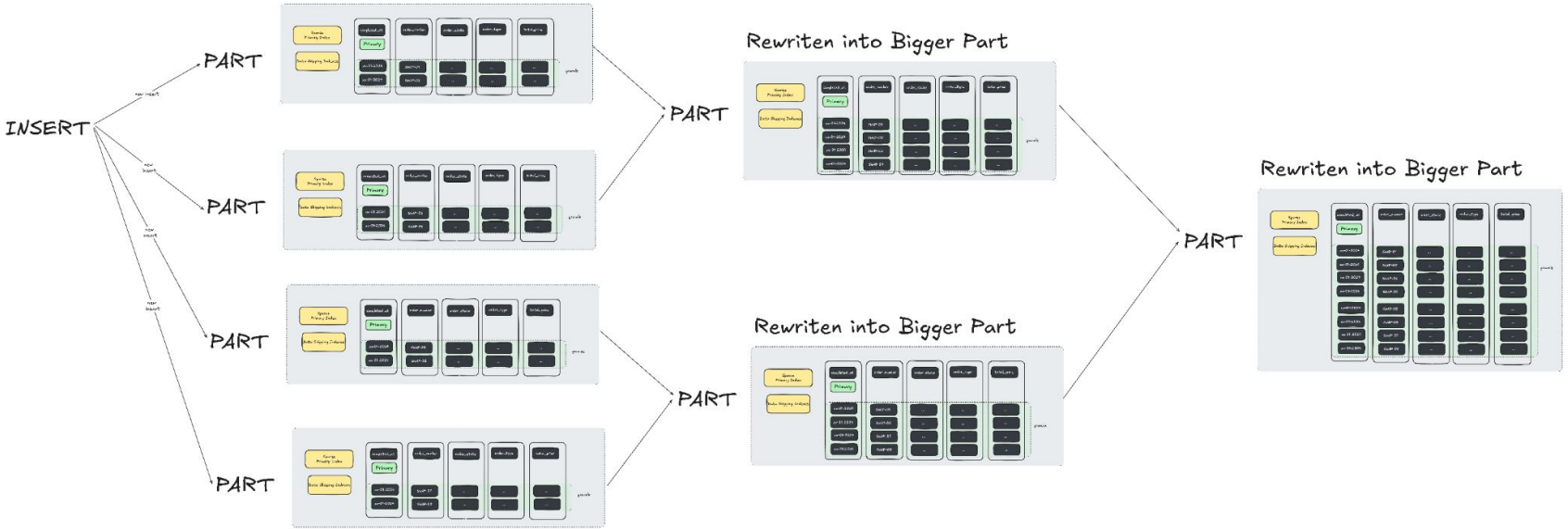
with unique **disk layout**, **Skip Indexes** optimized to reduce scan overhead



- Granules
- Sparse Primary Index
- Skip Indexes

Clickhouse no manual compaction ops

MergeTree maintains performance by controlling part fragmentation.



In the background, ClickHouse constantly **merges parts** maintain optimal query performance.

Clickhouse enables ELT natively

using Materialized Views, no additional systems required

⚙️ Real-time Materialized View

```
CREATE MATERIALIZED VIEW
IF NOT EXISTS mv_trx_agg_daily_orders

TO trx_agg_daily_orders
AS
      SELECT toDate(completed_at) AS order_date,
            order_type,
            countState()            AS total_orders,
            sumState(total_price)   AS total_price
      FROM trx_orders
      GROUP BY order_date, order_type;
```

Automatically processes new rows as they are inserted.

⏱️ Scheduled Materialized View

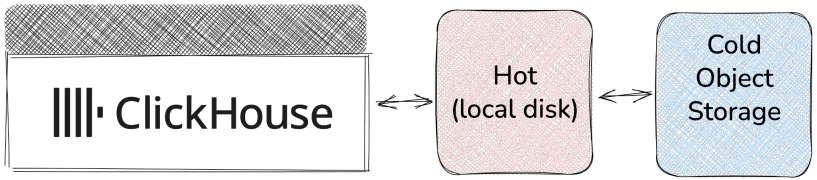
```
CREATE MATERIALIZED VIEW
IF NOT EXISTS mv_order_product_performance
REFRESH EVERY 1 DAY
DEPENDS ON trx_orders
APPEND TO order_product_performance
AS
      SELECT
      FROM trx_orders
      WHERE
      DATE(time_unix) = DATE(now() - INTERVAL 1 DAY)
      GROUP BY date_at, product_name
      ORDER BY date_at, product_name;
```

Runs on a defined cadence (daily, hourly, etc.) for batch-style transforms.

Clickhouse seamless tiered storage

Balance cost and performance through tiered storage policies.

```
<storage_configuration>
  <disks>
    <hot_ssd>...</hot_ssd>
    <cold_s3>
      <type>s3</type>
      ...
    </cold_s3>
  </disks>
  ...
  <policies>
    <local_disk_and_s3>
      <volumes>
        <hot>
          <disk>hot_ssd</disk>
        </hot>
        <cold>
          <disk>cold_s3</disk>
        </cold>
      </volumes>
      <move_factor>0.2</move_factor>
    </local_disk_and_s3>
  ....
</storage_configuration>
```



in this storage policy, two volumes are defined **hot** and **cold**.

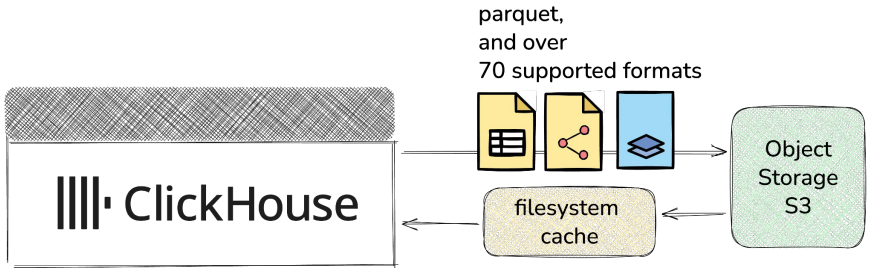
After the hot volume is filled with occupancy of **disk_size * move_factor**, the data is being moved to Object Storage (S3).

Clickhouse seamless s3 Table Engine/Function

```
INSERT INTO FUNCTION s3(
  …
   filename = '{_partition_id}.parquet',
   format = 'Parquet'
)
PARTITION BY concat('date=',
toString(toYYYYMMDD(completed_at)), '/', order_type)

SELECT *
FROM s3('http://xxx/*.parquet', 'xxx', 'xxxx', 'CSV')
SETTINGS filesystem_cache_name = 'cache_for_s3',
enable_filesystem_cache = 1;
```

```
<clickhouse>
    <filesystem_caches>
        <cache_for_s3>
            <path>path to cache directory</path>
            <max_size>10Gi</max_size>
        </cache_for_s3>
    </filesystem_caches>
</clickhouse>
```



Provides a **table-like interface** for selecting and inserting files stored in object storage.

The S3 table engine supports **more than 70 formats** (including parquet) [1].

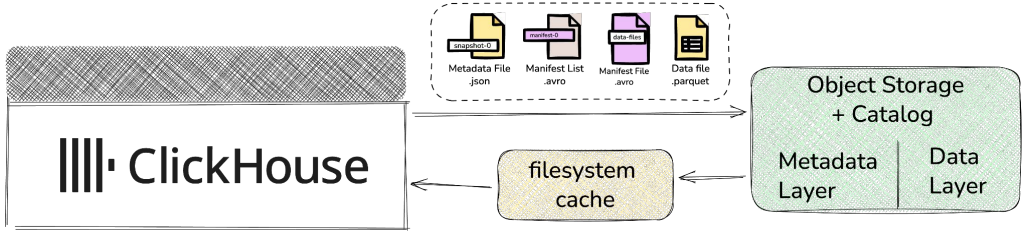
Includes optional **filesystem caching** to accelerate repeated reads and reduce cloud I/O.

1 https://clickhouse.com/docs/sql-reference/formats

Clickhouse seamless Iceberg Table Engine/Function

```
CREATE TABLE IF NOT EXISTS iot_data.battery_telemetry
(
    event_time      DateTime,
    battery_serial  String,
    state_of_charge Float32,
    state_of_health Float32,
    charge_cycles   UInt32,
    temperature     Float32,
    voltage         Float32
)
ENGINE = IcebergS3(
    'http://minio:9000/warehouse/iot_data/battery_telemetry/',
    'minio-root-user',
    'minio-root-password'
)
SETTINGS filesystem_cache_name = 'cache_for_s3',
enable_filesystem_cache = 1;
```

```
<clickhouse>
    <filesystem_caches>
        <cache_for_s3>
            <path>path to cache directory</path>
            <max_size>10Gi</max_size>
        </cache_for_s3>
    </filesystem_caches>
</clickhouse>
```



Provides a table-like interface to Apache Iceberg tables stored in Amazon S3, Azure, HDFS, or locally.

Support (Iceberg v1/v2), with optional **filesystem caching** to speed up repeated reads and reduce cloud I/O.

Starting from ClickHouse 25.7, experimental modifications are available (need explicitly enabled):
- SET allow_experimental_insert_into_iceberg = 1;
- SET allow_experimental_iceberg_compaction = 1;

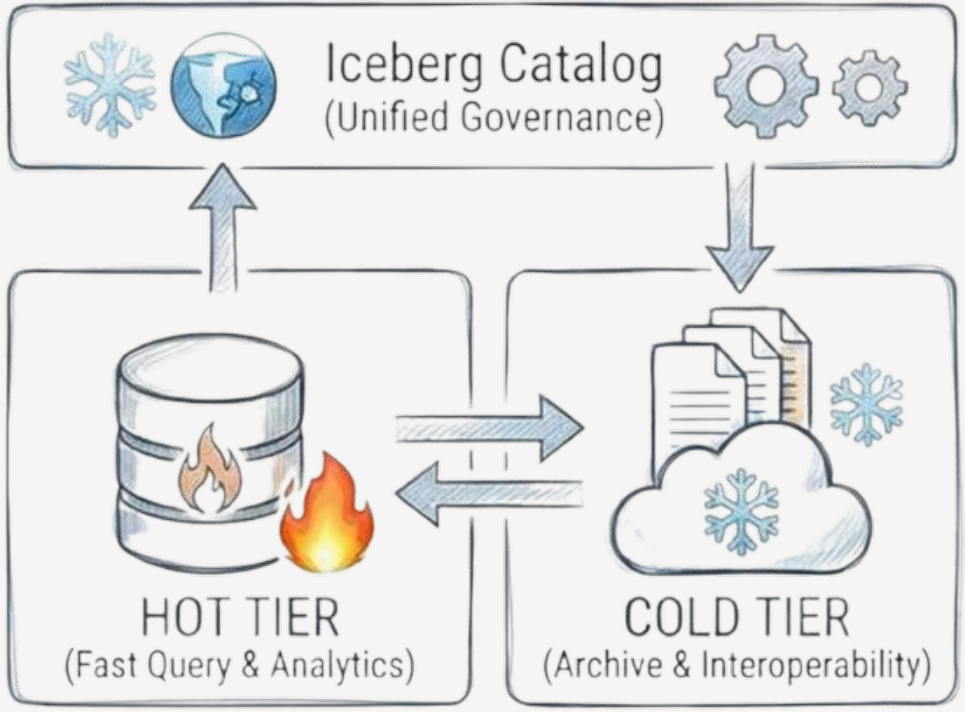
1 https://clickhouse.com/docs/sql-reference/table-functions/iceberg

DEMO

https://github.com/meong1234/clickhouse-iceberg

ClickHouse Meets Iceberg

- Understand the tools
- **Simplicity wins**, unless you truly need the complexity



References

An Extremely Technical Overview of How Apache Iceberg Planning Actually Works
YouTube. CMU Database Group.
https://www.youtube.com/watch?v=kJaD0WuQ1Bg&t=2147s

Are open-table-formats + lakehouses the future of observability?
ClickHouse Blog. (n.d.).

https://clickhouse.com/blog/lakehouses-path-to-low-cost-scalable-no-lockin-observability

AWS S3 Tables After the 10× Priceberg Plunge
Onehouse Blog. (n.d.).
https://www.onehouse.ai/blog/aws-s3-tables-after-the-10x-priceberg-plunge

Firebolt: Why Powering User Facing Applications on Iceberg is Hard
YouTube. CMU Database Group.
https://www.youtube.com/watch?v=Vf-N3JzWz0g

Accelerating Apache Parquet with metadata stores and specialized indexes using Apache DataFusion
YouTube. (Andrew Lamb).
https://www.youtube.com/watch?v=74YsJT1-Rdk

Querying Parquet with Millisecond Latency
Apache Arrow Blog. (2022, December 26).

https://arrow.apache.org/blog/2022/12/26/querying-parquet-with-millisecond-latency/

Parquet Viewer Tool
Xiangpeng Systems. (n.d.).
https://parquet-viewer.xiangpeng.systems/

Apache Parquet Data Page Encodings
Apache Parquet Documentation. (n.d.).
https://parquet.apache.org/docs/file-format/data-pages/encodings/

The Parquet Format and Performance Optimization Opportunities Boudewijn Braams
YouTube. (Databricks).
https://www.youtube.com/watch?v=1j8SdS7s_NY

Thanks...