Tabular Database Systems

⑤

Database-External Data in Parquet Files

Summer 2025

Torsten Grust
Universität Tübingen, Germany

1 ┆ Columnar Compressed Data Storage Outside the DBMS: Parquet

Fragility, space requirements, and parsing effort render CSV files problematic for applications that read/write GBs or TBs of data.

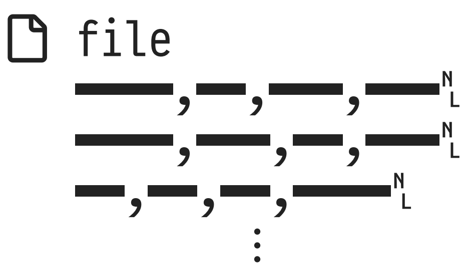
Alternative **database-external file formats** have been developed to address these issues. Among these, **Parquet**[1] is widely used.

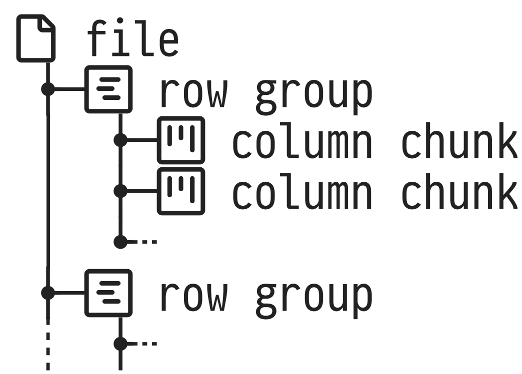
- Developed in the open since 2013 (initiated by Twitter *et al*).
- Stores **columns of typed data**.
- Built-in **compression** (on file and column levels).
- Incorporates **rich metadata** that supports projection and selection pushdown.
- Supported by libraries for a wide range of programming languages. Directly readable/writable by DuckDB.

[1] Apache Parquet (sponsored by the Apache Software Foundation): open source, column-oriented data file format designed for efficient data storage and retrieval.

CSV vs. Parquet

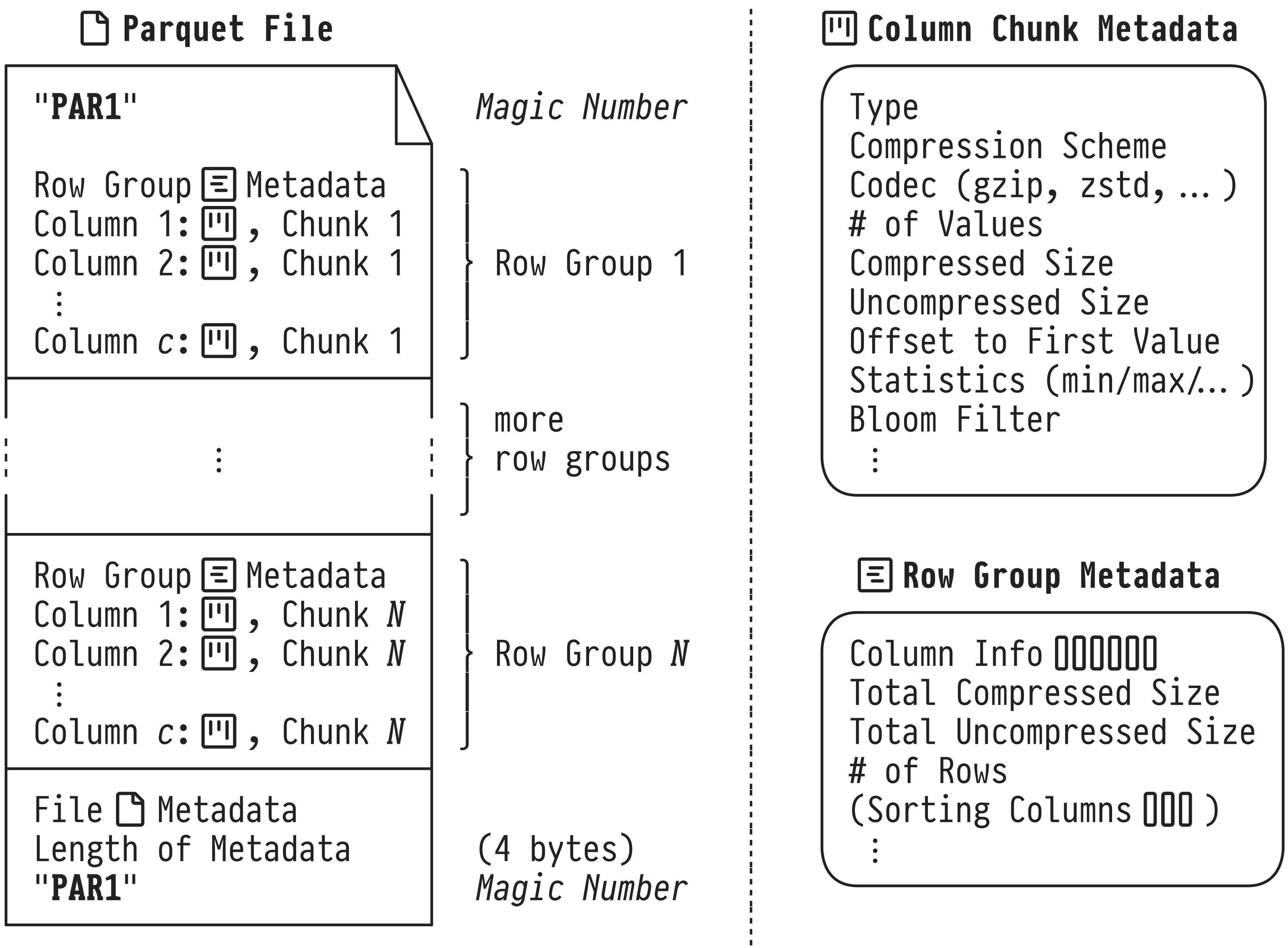
CSV	Parquet
monolithic	split into horizontal row groups
row-oriented (lines separated by ␤)	column-based (chunk-by-chunk)
plain text, uncompressed	supports compression: • file-level (gzip, zstd, …) • column-level (dictionary, RLE, …)
untyped (requires parsing)	typed: • scalar (INT32/64, FLOAT, BYTE_ARRAY, …) [• nested records]
no metadata (optional header row)	metadata for file/row groups/columns: • file format version • min/max/count statistics • cardinality, (un)compressed byte sizes





#024

A Sketch of Parquet's Storage Format

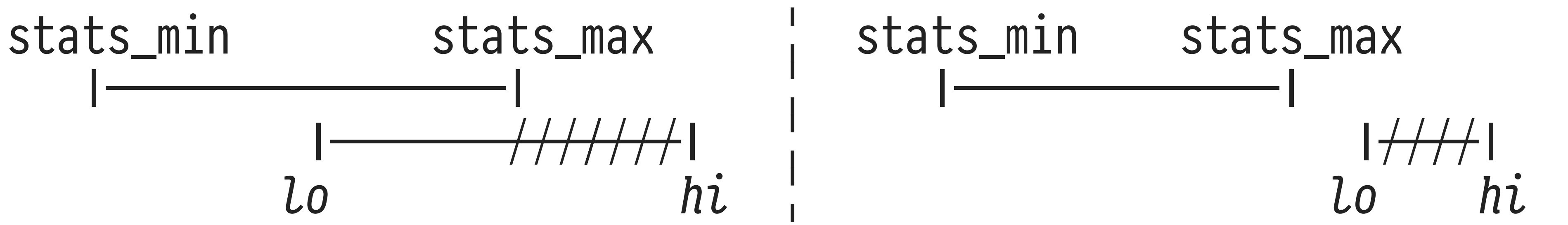


2 ┆ Pushing Projection and Filtering Down into Parquet Reading

Parquet's file structure + metadata enables readers (like DuckDB's `PARQUET_SCAN` operator) to **only access relevant data subsets.**

- **Projection pushdown:** Exploit **column-based layout.** In each row group, use `data_page_offset` to navigate the file to only read required column chunk(s).

- **Filter pushdown:** Exploit **statistics metadata.** Entirely skip row group if `stats_min`/`stats_max`[2] for column c indicate that filter predicates like $lo \leqslant c \leqslant hi$ will always fail (///).

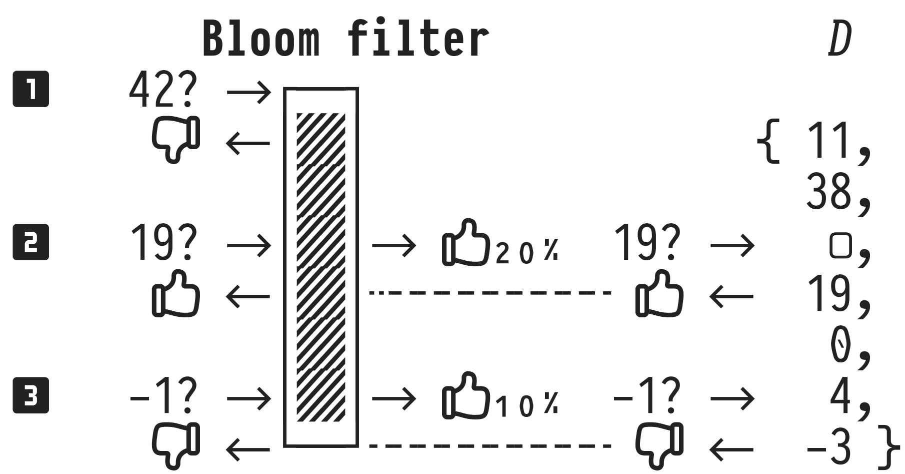


[2] In the DB research literature, the (`stats_min`,`stats_max`) pairs in all row groups are sometimes collectively referred to as **zone map** (for column c).

3 ¦ Optional in Parquet Files: Bloom Filters

If column values are randomly shuffled, all values may occur in all row groups and the effectiveness of zone maps is largely lost.

- **Bloom filters** are *compact* data structures that *over-approximate* the set D of distinct values[3] in a row group.
- Given the question $x \in D$?, Bloom filters either respond with
  - *definitely no* (👎) or
  - *probably yes* (👍$_{p\%}$, where p is a false positive rate).



#026

[3] Set D is the ***active* domain** of the row group. Example: if the values in a column have type int32 with domain $[-2^{31},2^{31})$, we typically have $D \Subset [-2^{31},2^{31})$ (read $\Subset$ as “significantly smaller”).

4 ¦ Partitioned Files and Filter Pushdown (Hive Partitioning)

If data volumes get truly large, it may make sense to **partition rows** into a family of files. Which partition will a row belong to?

```
🖿 nyc-yellow-cabs
┣━ 🖿 vendor_id=1
┃   ├── 🗀 payment_type=1
┃   │   └── ▦ data_0.parquet
┃   ┣━ 🖿 payment_type=3
┃   │   ┗━ ▦ data_2.parquet
┃   └── 🗀 payment_type=4
┃       └── ▦ data_3.parquet
┗━ 🖿 vendor_id=2
│   ├── 🗀 payment_type=1
│   │   └── ▦ data_4.parquet
│   ├── 🗀 payment_type=2
│   │   └── ▦ data_5.parquet
│   ┣━ 🖿 payment_type=3
│       ┗━ ▦ data_6.parquet
└── 🗀 vendor_id=4
    ├── 🗀 payment_type=1
    │   └── ▦ data_8.parquet
    └── 🗀 payment_type=2
        └── ▦ data_9.parquet
```

- Choose $n \geqslant 1$ columns $c_1,\ldots,c_n$ as partition criteria (left: n=2, vendor_id + payment_type).
- Arrange partitions in **tree hierarchy:**
  - Depth of tree: n+1.
  - Width of tree $\leqslant$ product of the size of active domains of columns c_i.
- Inner nodes: 🗀 OS directories, leaf nodes: ▦ data (Parquet or CSV).
- Filter query: traverse relevant subtrees/read relevant data files only (left: ┣━ 🖿 ≡ payment_type = 3).
- Particularly useful when I/O is slow.