Building a Data Lake Solution Using DuckDB

Subash Roul
Engineering Manager

Fivetran
Agenda

- About Fivetran
- Data Lake as a Destination
- Data Lake Writer Architecture
- Leveraging DuckDB for Merging Data
- DuckDB Enhancements
- Data Lake Future Explorations
About Fivetran
400+ Connectors

14+ Destinations
Our mission

Make access to data as simple and reliable as electricity
Data Lake as a Destination
Fivetran managed load
Ease of use
Readily-consumable data
Read data using different Lakehouse query engines
Automated table maintenance
Data Lake Writer Architecture
Diagram showing the integration of Fivetran with AWS S3 and Delta Lake. The process includes:

- **Writer (S3 + Iceberg)**: Ingress through Nginx, merge with the Data Lake Writer (DLW) Cluster.
Key Highlights

- Distributed execution of merges
  - Parallel processing for speed
  - Scalable in terms of workload sizes
- Extensible to other table formats.
Leveraging DuckDB for Merging Data
COPY (  
  SELECT "id", "value",  
  FROM read_parquet('data_file_1.parquet') AS existing  
  WHERE NOT EXISTS (  
    SELECT TRUE  
    FROM read_parquet(['delete.parquet']) AS staging  
    WHERE "existing"."id" = "staging"."id"  
  )  
) TO 'data_file_3.parquet' (  
  FORMAT 'parquet',  
  FIELD_IDS { "id" : 1, "value" : 2},  
  ROW_GROUP_SIZE_BYTES '512mb'  
)
COPY (  
    SELECT "id", "value",  
        FROM read_parquet('data_file_1.parquet') AS existing  
    WHERE NOT EXISTS ( 
        SELECT TRUE  
            FROM read_parquet(['delete.parquet']) AS staging  
            WHERE "existing"."id" = "staging"."id"  
    )  
) TO 'data_file_3.parquet' (  
    FORMAT 'parquet',  
    FIELD_IDS { "id" :1, "value" :2},  
    ROW_GROUP_SIZE_BYTES '512mb'  
)
Update

**Incoming**
- `update.parquet` (id: 5)

**Existing**
- `data_file_1.parquet` (id 1 - 10)
- `data_file_2.parquet` (id 11 - 20)

**Contains the record to update**

**After Merge**
- `data_file_1.parquet` (id 1 - 10)
- `data_file_2.parquet` (id 11 - 20)
- `data_file_3.parquet` (id 1 - 10)

Unlinked from the latest snapshot

Has the same records as `data_file_1.parquet`, with updated values for record with id 5

COPY (  
SELECT existing."id", existing."value"  
FROM read_parquet('data_file_1.parquet') AS existing  
    LEFT JOIN read_parquet('update.parquet') AS staging  
    ON "existing"."id" = "staging"."id"  
) TO 'data_file_3.parquet' (  
    FORMAT 'parquet',  
    FIELD_IDS { "id" :1, "value" :2},  
    ROW_GROUP_SIZE_BYTES '512mb'
)
It’s going well!

- No more custom code for rewriting files.
  - Simpler code
  - Less maintenance
Why We Chose DuckDB?

- Strong support for parquet read/write/cast
- Vectorized engine
- Excellent performance for file rewrites
- Simple to use – simple code and no other external dependencies
Insights from Recent Experiments

Merge Queries

- Performance improves as DuckDB threads increase (up to a certain level)
- Increasing DuckDB threads helps if there are sufficient row groups to process
- Slower full file downloads compared to AWS SDK – we are collaborating with DuckDB on this
- Expectedly, our merge queries that import from and export to disks perform much better with local SSDs
- Decreasing row group sizes, reduces DuckDB’s memory consumption
- Performance of DuckDB’s selective column retrieval from remote S3 buckets, was satisfactory
DuckDB Enhancements for Fivetran
Past Enhancements for Fivetran

- Support field ID for columns in parquet writing
- Reduction in memory consumption
  - Static linking to Jemalloc
  - Swapping to disk in case row group memory usage exceeds
  - Configurable row group size (`ROW_GROUP_SIZE_BYTES`)
- Outputting multiple files in single query
- Performance improvements for large number of columns (2000+)
  - Writing 8 columns at a time rather than 1 column, improving performance
Ongoing/Upcoming Enhancements for Fivetran

- File encryption support + performance improvement for encryption
- Output file rotate based on size threshold
- Import/export performance improvements
Data Lake Future Explorations
• Fully pipelined rewrite using remote-only DuckDB queries
• Support Merge-on-Read
• Data sharing between Data Lake and warehouses
• Auto-partitioning
Thank you!
Questions?