Antra

People · Process · Performance

Agenda

- Intro to Delta Lake

Intro to Delta Lake

Antra

What are traditional Data Lakes?

- They are storage repositories that stores a large amount of raw data(current and historical) in native formats
- May also contain relational databases with live transactional data.
- Versatile, scalable, cheap.

Problems with traditional Data Lakes

- Not ready for direct data analysis and ML
- Hard to enforce schema for data, prone to inconsistent and low quality data.
- Difficulty sorting data by an index if data is spread across many files and partitioned.
- Failed jobs might leave data in corrupt states
- Too much overhead opening & reading when working with large amount of small files
- Partitioning is only “poor man’s indexing”
- No caching, low throughput

What is Delta Lake?

- **Data Lake + Delta file format = Delta Lake**
- A file format that's designed specifically to work with Spark and DBFS
- Has both open-source and managed offerings
- Data is stored as **Parquet** files, and a **transaction log** is maintained to track changes to the table.

Data Lake vs Delta Lake

	Data Lake	Delta Lake
Transaction & Isolation	Multiple data pipelines reading and writing concurrently, great for scalability, but bad for ensuring data integrity	Enables ACID transactions, even serializable isolation level.
Metadata handling	No distributed processing of metadata	Treats metadata like regular data, utilizing distributed processing to handle metadata. Great for data that's truly big(PB scale, billions of files)
Version control	No version control	Provides data versioning
Storage Format	Different file formats, easy to dump data into, but hard to use	All data stored in Delta format
Streaming Data	Poor support for streaming data	Out-of-the-box support for streaming data
Schema	Hard to specify/enforce/change schema	Easy to specify/enforce/change schema due to unified file format

Working with Delta

- **Tables** are equivalent to **dataframes** except
  - Table are defined at the workspace level, persists between notebooks
  - Data frames are defined at the notebook level
- Tables in Delta are generally classified in three levels
  - Raw
  - Bronze -> single source of truth
    - Raw, unprocessed data,with some metadata added
  - Silver
    - Cleaned, preprocessed data that is directly queryable
  - Gold
    - Highly refined views of data
      - Aggregated data
      - Feature tables

Working with Delta

- Delta lakes are easy to create
  - When writing, simply specify “delta” instead of “parquet” or ”csv”

```
data.write.format("delta").mode("overwrite").save("/tmp/delta-table")
```

- We can use Spark SQL directly on a directory of Delta data by specifying the directory path

```
SELECT * FROM delta.`/path/to/delta_directory`
```

- We can also create a table using Spark SQL

```
spark.sql("""DROP TABLE IF EXISTS customer_data_delta""")
spark.sql("""
  CREATE TABLE customer_data_delta
  USING DELTA
  LOCATION '{}'
""".format(DataPath))
```

* Since schema is already stored in metadata, we don’t need to specify it when creating table

Working with Delta - Append

- To append to delta lake, all we need to do is to change the mode of the previous write query

```
(newDataDF
   .write
   .format("delta")
   .partitionBy("Country")
   .mode("append")
   .save(DataPath)
)
```

- Changes in the data file will be reflected in the tables based on the file immediately

Working with Delta - Upsert

- Upsert is used to simultaneously update and insert data

```
%sql
MERGE INTO customer_data_delta
USING upsert_data
ON customer_data_delta.InvoiceNo = upsert_data.InvoiceNo
  AND customer_data_delta.StockCode = upsert_data.StockCode
WHEN MATCHED THEN
  UPDATE SET *
WHEN NOT MATCHED
  THEN INSERT *
```

Optimizations in Delta Lake

- Optimize
  - Performs **file compaction**
  - Small files are compacted together into new larger files **up to 1GB**
    - The 1GB size was determined by the Databricks optimization team as a trade-off between query speed and run-time performance when running Optimize.
  - Recommend to run Optimize on a daily basis, during off-hours, and increase/decrease frequency **based on business needs.**

Optimizations in Delta Lake

- Partitioning
  - Used to speed up queries with **WHERE clause**
  - Relies on data being correctly partitioned
  - Will skip partitions that doesn't satisfy the WHERE condition
- ZOrdering
  - Another way to try to reduce the number of files searched for a query
  - Colocate related information in the same set of files.
  - Effectiveness goes down as more columns are considered.
- You can not use partitioning and ZOrdering on the same column

```
%sql
OPTIMIZE delta_data_source
ZORDER BY COL1,COL2...
```

Optimizations in Delta Lake

- Vacuum
  - Used to save some storage space
  - Cleans up invalid data files
    - Created by optimize/updates/upserts/deletions

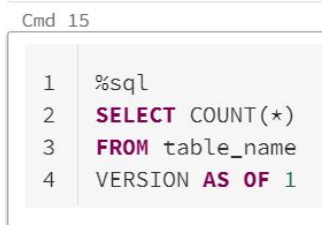
```
VACUUM name-of-table RETAIN number-of HOURS;
```

    - It's recommended to **NOT** clean things younger than 7 days
      - old snapshots and uncommitted files can still be in use by concurrent readers or writers to the table.

Using Time Travel in Delta Lake

- Viewing the history of a Delta table
  - This will give some metadata of all versions of this table
  - Result is in a tabular format with “version” (integer) being one of the columns

- To query an older version of a Delta table, use **AS OF**
  - The AS OF clause should directly follow FROM, before WHERE conditions



Antra