databricks

Designing the Foundation

LECTURE

Data Skipping and Liquid Clustering

In this lecture, we will explore data skipping, Z-ordering, partitioning challenges, and how Liquid Clustering with predictive optimization improves query performance in Databricks Delta Lake.

Data Skipping

Simple, well-known I/O pruning technique

- Track file-level stats like min & max
- Leverage them to avoid scanning irrelevant files

```
SELECT input_file_name() as
“file_name”,
       min(col) AS “col_min”,
       max(col) AS “col_max”
FROM table
GROUP BY input_file_name()
```

file_name	col_min	col_max
1. parquet	6	8
2. parquet	3	10
3. parquet	1	4

With data skipping, students would first check the labels on candy bags before opening them. If a bag doesn’t contain the specific candies needed, it’s skipped. This avoids unnecessary effort and speeds up the entire counting process, similar to how Spark skips files that aren’t relevant to a query.

Z-Ordering

```
Optimize Table Z-Order by Column
```

Old Layout

file_name	col_min	col_max
1. parquet	6	8
2. parquet	3	10
3. parquet	1	4

New Layout

file_name	col_min	col_max
1. parquet	1	3
2. parquet	4	7
3. parquet	8	10

Z-ordering is a way to organize data within a table based on a specific column so that similar values are stored together in the same files. Although Databricks currently recommends liquid clustering over Z-ordering or partitioning for new tables, Z-ordering is still sometimes useful. With Z-ordering, data is physically organized by the chosen column, and after this process, each data file will record the minimum and maximum values for that column in its metadata (for example, in the footer of a Parquet file). This setup allows Spark, when running a query that filters on the Z-ordered column, to check these min and max values and skip any files that cannot possibly contain the required data, without even opening them. This reduces the number of files that must be read and improves overall query performance, especially for queries filtering on columns with many distinct values.

Z-Ordering

```
Select * from Table Where Col = 7
```

Old Layout

file_name	col_min	col_max
1. parquet	6	8
2. parquet	3	10
3. parquet	1	4

New Layout

file_name	col_min	col_max
1. parquet	1	3
2. parquet	4	7
3. parquet	8	10

With Z-ordering applied, when a query searches for a specific value (for example, selecting rows where a column equals seven), the benefit becomes clear. Before Z-ordering, Spark may have to open multiple files to check if they contain the value seven, since there's no efficient way to know which files are relevant. After Z-ordering, the data is organized so that the value seven, if present, will be contained within a specific file, as indicated by the min and max statistics stored in each file's metadata. Spark can use these statistics to immediately identify and open only the file that could contain the target value, skipping all other files that do not contain seven. This targeted access avoids unnecessary file reads and improves the speed and efficiency of query execution.

Databricks Delta Lake and Stats

- Databricks Delta Lake collects stats about the first N columns
  - dataSkippingNumIndexedCols = 32
- These stats are used in queries:
  - Metadata only queries: select max(col) from table
    - Queries just the Delta Log, doesn't need to look at the files if col has stats
  - Allows us to skip files
    - Partition FIlters, **Data Filters**, Pushed Filters apply in that order
  - TimeStamp and String types aren't always very useful
    - Precision/Truncation prevent exact matches, have to fall back to files sometimes
- Avoid collecting stats on long strings
  - Put them outside first 32 columns or collect stats on fewer columns
    - alter table change column col after col32
    - set spark.databricks.delta.properties.defaults.dataSkippingNumIndexedCols = 3

For effective data skipping and Z-ordering in Delta Lake, it's necessary to collect statistics on your data. By default, Databricks Delta Lake gathers statistics for the first 32 columns in a table, recording values like min and max for each column in the file metadata. These stats let Spark identify which files are likely to contain relevant query results, allowing it to skip over files that don't match the filter criteria. For instance, metadata-only queries such as finding a column's maximum value can be answered without reading any data files, as long as stats are available.
However, some limitations exist. Filters are applied in order: partition filters, data filters, and then pushed filters. Because of possible precision or truncation issues, statistics for timestamp and string columns may not always lead to exact matches, requiring a fallback to file scanning. It's also best to avoid collecting statistics for columns with long strings—either by placing them outside the first 32 columns or updating configuration settings—to maintain query speed and system efficiency.

What about Partitioning?

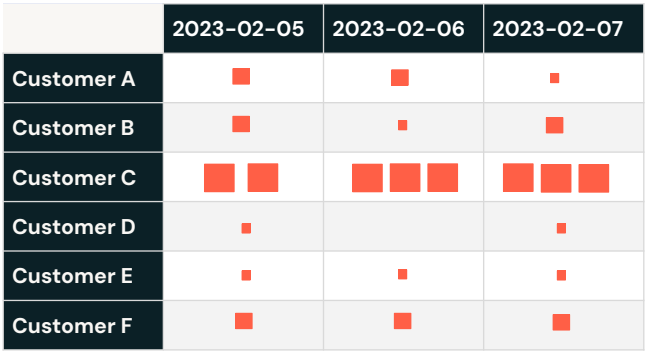
- Generally not recommended!
  - Partitioning is usually misused/overused
  - Tiny file problems or Skew
- Good use-cases for partitioning
  - Isolating data for separate schemas (single->multiplexing)
  - GDPR/CCP use cases where you commonly delete a partitions worth of data
  - Use cases requiring a physical boundary to isolate data SCD Type 2, partition on current or not for better performance.
- If you partition
  - Choose column with low cardinality
  - Try to keep each partition less than 1tb and greater than 1gb
  - Tables expected to grow TBs
  - Partition (usually) on a date, zorder on commonly used predicates in where clauses

Partitioning is a classic technique for organizing data, allowing Spark to avoid scanning unnecessary files by segmenting data according to specific columns. However, Databricks generally advises against using partitioning as it's often overused or misapplied, leading to issues like a proliferation of small files or unevenly distributed data (data skew). Despite these drawbacks, partitioning can be useful for certain cases—such as when you need to isolate data for different schemas, comply with GDPR or CCPA by efficiently deleting entire partitions, or create clear physical data boundaries (as with SCD type 2 tables).

When partitioning is needed, it's best to use columns with low cardinality (few unique values) to avoid generating many tiny files. Aim to keep each partition between 1 GB and 1 TB. Partitioning is especially helpful for tables expected to grow above a terabyte, and is commonly done on a date column. Z-ordering can also be used together with partitioning to optimize queries that filter on frequently used columns in WHERE clauses.

Challenges with Disk Partitioning

Partition by customer ID and date + optimize



- **Many small files.**
  - High metadata operations overhead.
  - Slow read operations.
- **Data skew.**
  - Inconsistent file sizes across partitions.

While partitioning can be useful in some scenarios, Databricks currently recommends liquid clustering as a more flexible and efficient approach. The main challenges with partitioning include the risk of creating many small files, which increases metadata overhead and slows down read operations. Additionally, partitioning can lead to data skew, where some partitions contain very little data while others have a lot, resulting in inconsistent file sizes. This imbalance makes query performance and optimization less effective.

Introducing Liquid Clustering

What it is

Innovative technique to clustering data layout to support efficient query access and reduce data management and tuning overhead. It's flexible and adaptive to data pattern changes, scaling, and data skew

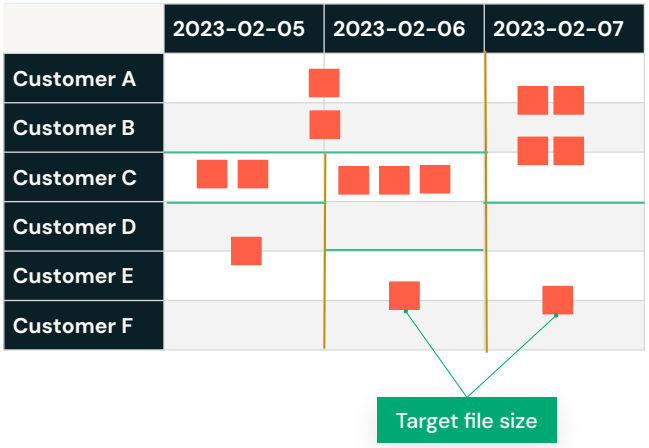
Benefits

- Best performance out of the box
  - Clustering on write
- Most consistent data skipping
  - Immune to data skew
- Minimal write amplification on table maintenance
  - True incremental optimize
- Row Level Concurrency
  - Simplify logic of concurrent writers
- Reduced Cognitive Overhead
  - No worrying about cardinality

Liquid clustering is an innovative technique that Databricks has developed to reduce data management overhead and provide more support for efficient query access. It offers a flexible and adaptive approach that changes with your data and scales very well, helping to avoid data skew. With liquid clustering, clustering is automatically enabled so you get optimum performance right out of the box. It is immune to data skew and enables more consistent data skipping. Additionally, there is minimal write amplification during table maintenance thanks to true incremental optimization and row-level concurrency, which simplifies logic for concurrent writers and reduces cognitive load. With liquid clustering, there is no need to worry about whether you are clustering on high or low cardinality columns. The approach is as the name suggests—liquid and adaptable as your data evolves.

Liquid Clustering

Liquid cluster by customer ID and date



- Liquid is not subject to rigid boundaries
  - Liquid intelligently decides what ranges of data to combine
- Data skew is gone
  - Data sizes are consistent
- Liquid stores metadata
  - New data can be clustered into existing clusters on write

With liquid clustering, the main objective is to target a specific file size, allowing for much more flexibility compared to traditional rigid boundaries. The “liquid” aspect means that data isn’t confined to strict partitions—Databricks can intelligently decide which ranges of data to combine so that file sizes stay roughly the same. This dramatically reduces data skew, resulting in consistent data sizes across files. Liquid clustering also involves storing metadata, which is then used to cluster new data into existing clusters on write, making both writing and reading operations faster and more efficient.

Table Statistics

Keeping table statistics up to date for best results with cost-based optimizer

- Collects statistics on all columns in table
- Helps Adaptive Query Execution
  - Choose proper join type
  - Select correct build side in a hash-join
  - Calibrating the join order in a multi-way join

```
ANALYZE TABLE mytable COMPUTE STATISTICS FOR ALL COLUMNS
```

Table statistics are calculated for any of these optimization techniques and play a crucial role in improving table performance. By collecting statistics on the table columns, the system can optimize how data files are read and processed. These statistics are especially helpful for Adaptive Query Execution (AQE), which uses them to choose the best join type, select the appropriate build side in hash joins, and optimize the join order in multi-way joins. To gather these statistics, it is possible to use the ANALYZE TABLE command and specify 'compute statistics for all columns.' These detailed statistics then support more efficient query planning and execution.

Predictive Optimization

What is Predictive Optimization?

- Predictive optimization refers to using predictive analytics techniques to automatically optimize and enhance the performance of systems, processes, or workflow.
- It involves leveraging data-driven insights to proactively identify and implement optimizations, improving efficiency, cost-effectiveness, and overall system performance.

Predictive optimization in Databricks leverages predictive analytics to automatically improve the performance of systems, workflows, and processes. This technique uses data-driven insights to proactively identify opportunities for optimization before issues impact efficiency or costs. By analyzing usage patterns, Databricks can determine the most suitable optimization strategies for a given workload, ensuring operations run at peak performance and cost effectiveness.

With predictive optimization enabled, Databricks continually monitors workload activity, learns from system behavior, and implements tailored adjustments. This proactive approach results in greater efficiency, reduced resource consumption, and improved overall system performance—essentially delivering the best optimization outcomes without requiring manual intervention. The system aims to provide the greatest possible value by automatically fine-tuning configurations and processes for each unique workload environment.

Predictive Optimization

Key Features

Automatic Maintenance

- It automates the execution of background maintenance tasks on Delta tables.

Support Maintenance Operations

- It supports maintenance operations, including **OPTIMIZE** to improve query performance by optimizing file sizes and **VACUUM** to reduce storage costs by deleting unused data

Set and Forget Approach

- It intelligently and automatically runs maintenance jobs without requiring ongoing user supervision.

Serverless Computing

- It utilizes serverless compute, eliminating the need for users to manually manage compute cluster.

Predictive optimization in Databricks provides automatic maintenance for Delta tables, taking over routine optimization tasks that used to require manual effort. This includes background automation of maintenance activities such as OPTIMIZE and VACUUM, so there's no need to worry about scheduling or supervising these jobs. Once predictive optimization is set up, it intelligently runs all necessary maintenance without user intervention, allowing you to "set it and forget it." These benefits are currently available for Delta tables only. In addition to maintenance automation, predictive optimization supports serverless computing, removing the need to manage compute clusters manually. The system automatically adjusts compute resources and performs required maintenance, further reducing operational overhead and streamlining both performance management and resource allocation.

Thank you for completing this lesson and continuing your journey to develop your skills with us.