STRATEGIC PARTNER

dbWatch
DATABASE CONTROL

Adrian Chodkowski

Azure Databricks – performance tuning

O mnie

- Adrian Chodkowski
- Microsoft Data Platform MVP
- Data engineer & Architekt w Elitmind
- Specjalizacja: Platforma danych Microsoft (Azure & On-premise)
- Data Community
- seequality.net
- Adrian.Chodkowski@outlook.com
- @Twitter: Adrian_SQL
- LinkedIn: http://tinyurl.com/adrian-sql

AGENDA

- Databricks disk cache
- AutoLoader
- Static & Dynamic Partition pruning
- File pruning
- Z-Ordering
- Additional tips & summary

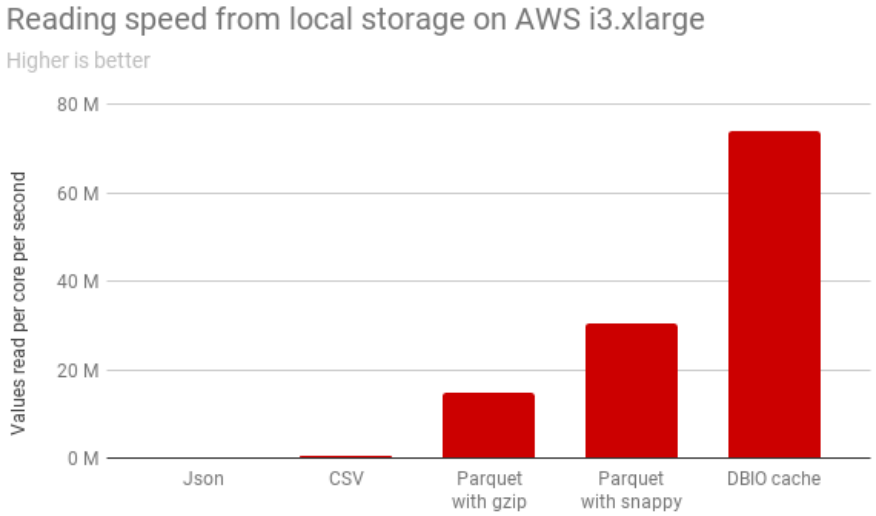
Level: 300

Databricks disk cache

aka Delta cache, DBIO

Disk caching

- Disk caching behavior is a proprietary Databricks feature previously called Delta Cache & DBIO,
- It is different than Spark cache!
- Works with parquet and delta,
- Databricks uses disk caching to accelerate data reads by **creating copies of remote Parquet** data files in nodes' local storage,
- Successive reads of the same data are then performed locally,
- After turning on it can be managed automatically or it can be forced by using CACHE SELECT syntax,
- You benefit most by using cache-accelerated worker instance types



Disk cache vs Spark cache

- **Prefer use of disk cache!**

Feature	disk cache	Apache Spark cache
Stored as	Local files on a worker node.	In-memory blocks, but it depends on storage level.
Applied to	Any Parquet table stored on S3, ABFS, and other file systems.	Any DataFrame or RDD.
Triggered	Automatically, on the first read (if cache is enabled).	Manually, requires code changes.
Evaluated	Lazily.	Lazily.
Force cache	`CACHE SELECT` command	`.cache` + any action to materialize the cache and `.persist`.
Availability	Can be enabled or disabled with configuration flags, enabled by default on certain node types.	Always available.
Evicted	Automatically in LRU fashion or on any file change, manually when restarting a cluster.	Automatically in LRU fashion, manually with `unpersist`.

Disk caching - configuration

- **spark.conf.set("spark.databricks.io.cache.enabled", "[true | false]")**
- Be careful with autoscaling :

> ⓘ Note
>
> When a worker is decommissioned, the Spark cache stored on that worker is lost. So if autoscaling is enabled, there is some instability with the cache. Spark would then need to reread missing partitions from source as needed.

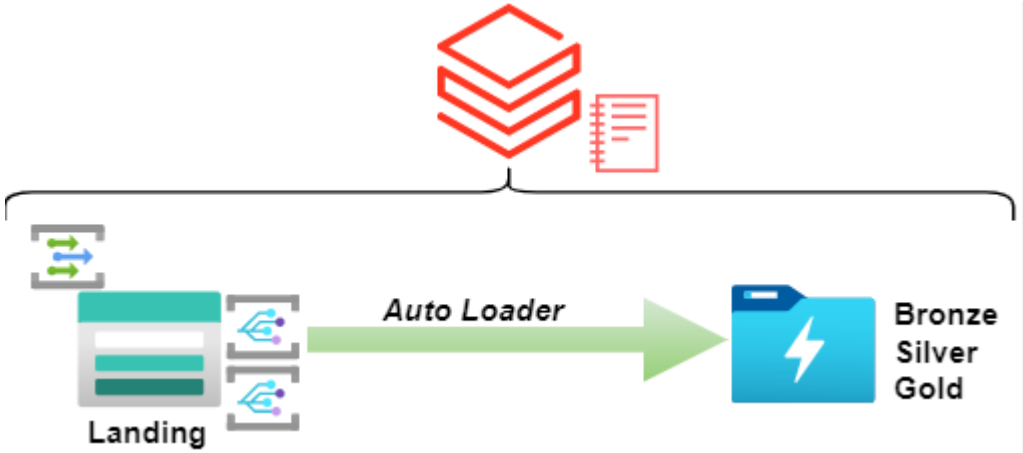
- **Spark.databricks.io.cache.maxDiskUsage** – disk space per node reserved for cached data in bytes,
- **Spark.databricks.io.cache.maxMetaDataCache** – disk space per node reserved for cached metadata in bytes,
- **Spark.databricks.io.cache.compression.enabled** – should the cached data be stored in compressed format.

Ingestion

With Autoloader

Autoloader

- Functionality based on structured streaming that can identify and load new files,
- Works in two modes:
  - Directory Listing – internally lists structure of files and partitions. Can be done incrementally
  - File Notification - leverage Event Grid and storage queues to track new files appearance
- Avoid overwrites the file,
- Have a backfill strategy defined!
- In many cases consider it as a default loading mechanism!
- There is also COPY INTO!



cloudFiles.backfillInterval

Type: `Interval String`

Auto Loader can trigger asynchronous backfills at a given interval, e.g. `1 day` to backfill once a day, or `1 week` to backfill once a week. File event notification systems do not guarantee 100% delivery of all files that have been uploaded therefore you can use backfills to guarantee that all files eventually get processed, available in Databricks Runtime 8.4 (Unsupported) and above. If using the incremental listing, you can also use regular backfills to guarantee the eventual completeness, available in Databricks Runtime 9.1 LTS and above.

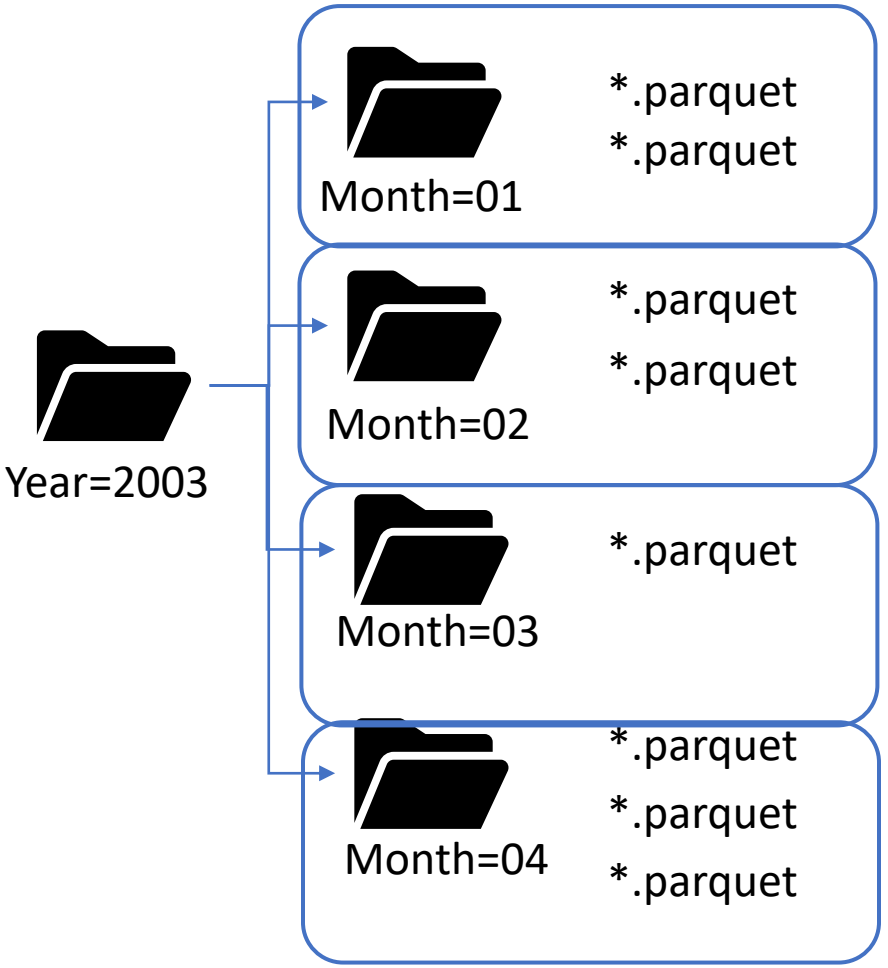
Default value: None

Partitioning

Dynamic Pruning & Design

Partitioning

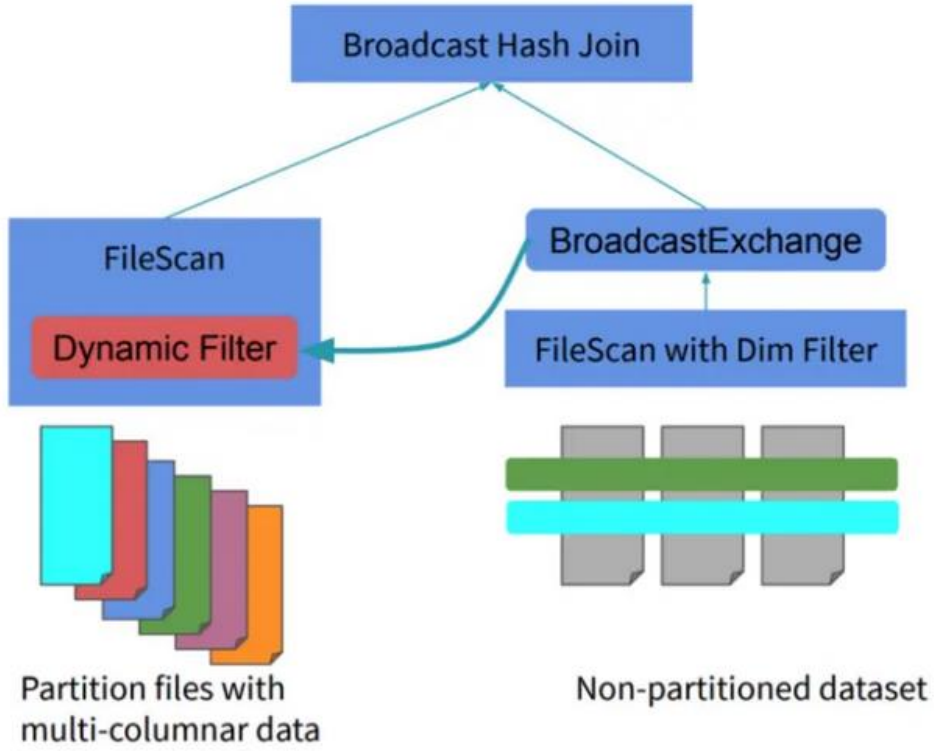
- Division of a table to the hierarchy of folders,
- Can be beneficial when reading data (partition discovery) and optimizing (OPTIMIZE WHERE partition key)
- Databricks recommendation:
  - don’t use it for tables smaller than 1 TB,
  - Each partition >1GB,
- When reading and filter by partition key then only specific partition can be read (rest will be skipped)
- Can be created using PARTITIONED BY or Partition keywords,
- Don’t use CREATE TABLE PARTITION BY AS SELECT – it will add hive overhead and can take ages to finish!



Dynamic Partition Pruning

- Especially useful in star schema
- Introduced in Spark 3.0
- Let's assume query:

```
SELECT d.c1, f.c2
  FROM Fact AS f
JOIN Dimension as d
ON f.join_key = d.join_key
  WHERE d.c2 =10
```



- Since there is a filter on one table (d.c2 = 10), internally DPP can create a subquery:

```
SELECT d.join_key FROM Dimension AS d
  WHERE d.c2=10;
```

- and then broadcast this sub-query result, so that we can use this result to prune partitions for "t1".

Dynamic Partition Pruning

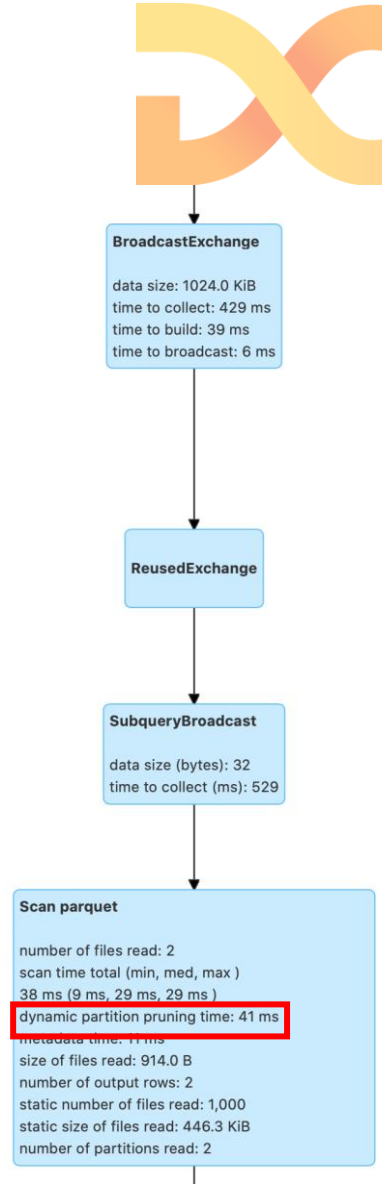
- Especially useful in star schema
- Introduced in Spark 3.0
- Let's assume query:

```
SELECT d.c1, f.c2
  FROM Fact AS f
JOIN Dimension as d
ON f.join_key = d.join_key
  WHERE d.c2 =10
```

- Since there is a filter on one table (d.c2 = 10), internally DPP can create a subquery:

```
SELECT d.join_key FROM Dimension AS d
  WHERE d.c2=10;
```

- and then broadcast this sub-query result, so that we can use this result to prune partitions for "t1".



File pruning

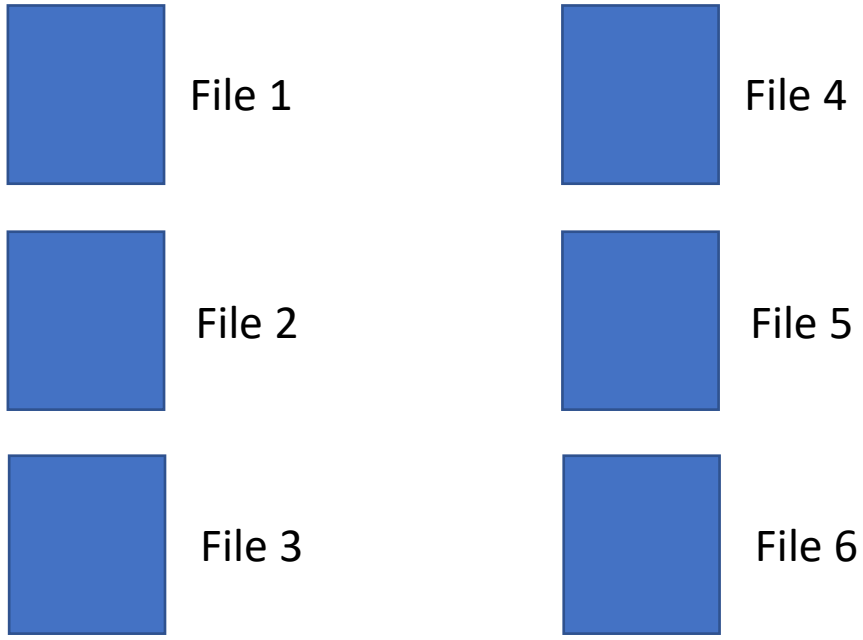
By using stats

File pruning

- In addition to eliminating data at partition granularity, Delta Lake on Databricks dynamically skips unnecessary files when possible.
- Delta Lake automatically collects metadata about data files (files can be skipped without data file access)
- Statistics are taken for the first 32 columns (can be changed).

```
SELECT * FROM
Dimension AS d
WHERE
d.category_id IN
(1,2,5);
```

File	Column	Min	Max
File 1	Category_id	1	2
File 2	Category_id	1	10
File 3	Category_id	6	10
File 4	Category_id	8	20
File 5	Category_id	4	100
File 6	Category_id	1	1

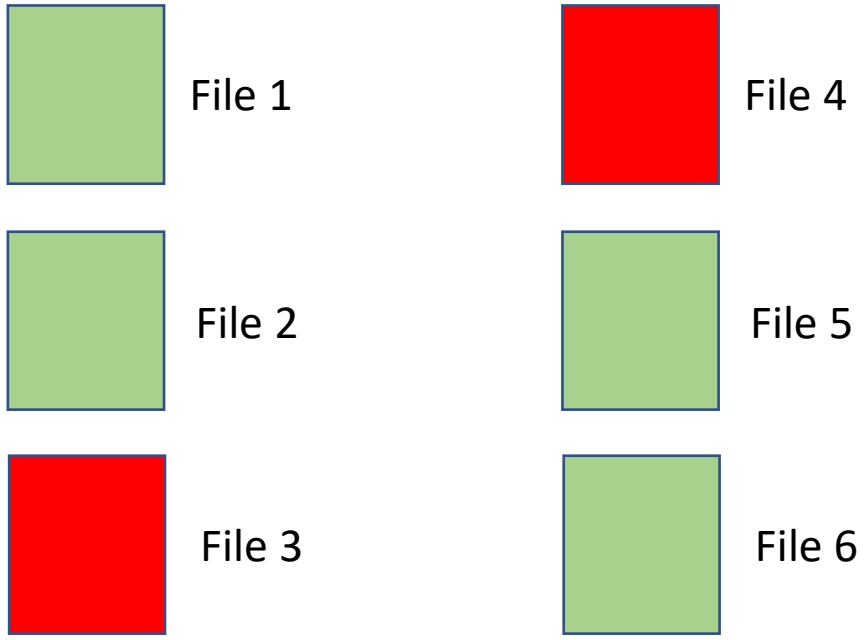


File pruning

- In addition to eliminating data at partition granularity, Delta Lake on Databricks dynamically skips unnecessary files when possible,
- Delta Lake automatically collects metadata about data files (files can be skipped without data file access),
- Statistics are taken for the first 32 columns (can be changed).

```
SELECT * FROM
Dimension AS d
WHERE
d.category_id IN
(1,2,5);
```

File	Column	Min	Max
File 1	**Category_id**	**1**	**2**
File 2	**Category_id**	**1**	**10**
File 3	Category_id	6	10
File 4	Category_id	8	20
File 5	**Category_id**	**4**	**100**
File 6	**Category_id**	**1**	**1**



Zordering

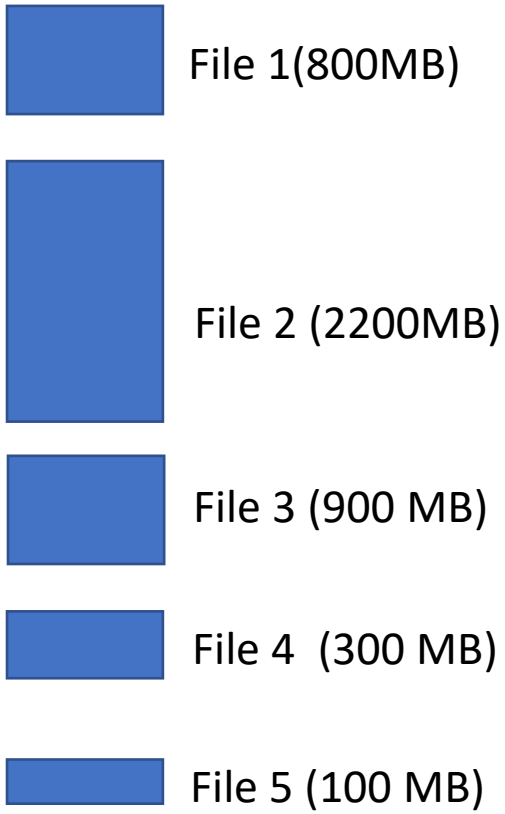
By using OPTIMIZE

OPTIMIZE Z-Order

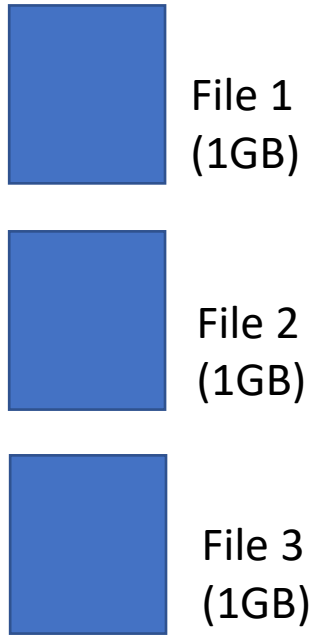
- **OPTIMIZE** optimizes the layout of Delta Lake,
- By Default OPTIMIZE set max file size to **1GB** (1073741824),
- You can control size by using **spark.databricks.delta.optimize.maxFileSize**
- Can be used based on column or bin-packing optimization:
  - **Bin-packing** – idempotent technique that aims to produce evenly-balanced data files with respect to their **size on disk** (not number of rows),
  - **Z-ordering** not-idempotent technique that aims to produce evenly-balanced data files with respect to the **number of rows** (not size on disk)

```
OPTIMIZE events OPTIMIZE events WHERE
 >= '2017-01-01' OPTIMIZE events WHERE
 >= current_timestamp() -
 1 day ZORDER BY (eventType)
```

OPTIMIZE Z-Order




```
OPTIMIZE table
ZORDER BY (category_id)
```



File	Column	Min	Max
File 1	**Category_id**	**1**	**2**
File 2	**Category_id**	**1**	**10**
File 3	Category_id	6	10
File 4	Category_id	8	20
File 5	**Category_id**	**4**	**100**

File	Column	Min	Max
File 1	**Category_id**	**1**	**10**
File 2	**Category_id**	**11**	**20**
File 3	Category_id	20	100

Dynamic File pruning

- Files can be skipped based on join not literal values
- To make it happen following requirements must be met:
  - Inner table (probe) being joined is in Delta format
  - Joint type is INNER or LEFT-SEMI
  - Join Strategy is **BROADCAST HASH JOIN**
  - Number of files in the inner table is greater than value set in **spark.databricks.optimizer.deltaTableFilesThreshold** (default 1000)
  - **Spark.databricks.optimizer.dynamicFilePruning** should be True (default)
  - Size of inner table should be more than **spark.databricks.optimizer.deltaTableSize** (default 10GB)

	Per Partition	Per File (Delta Lake on Databricks only)
Static (based on filters)	Partition Pruning	File Pruning
Dynamic (based on joins)	*Dynamic* Partition Pruning	*Dynamic* File Pruning (NEW!)

Other

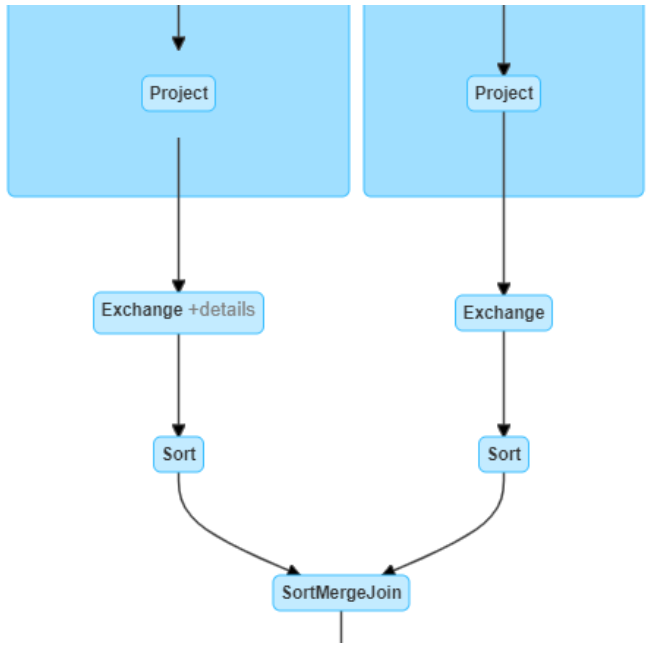
- **Use newest Databricks Runtime,**
- For fast Update/Merge re-write the least amount of files:

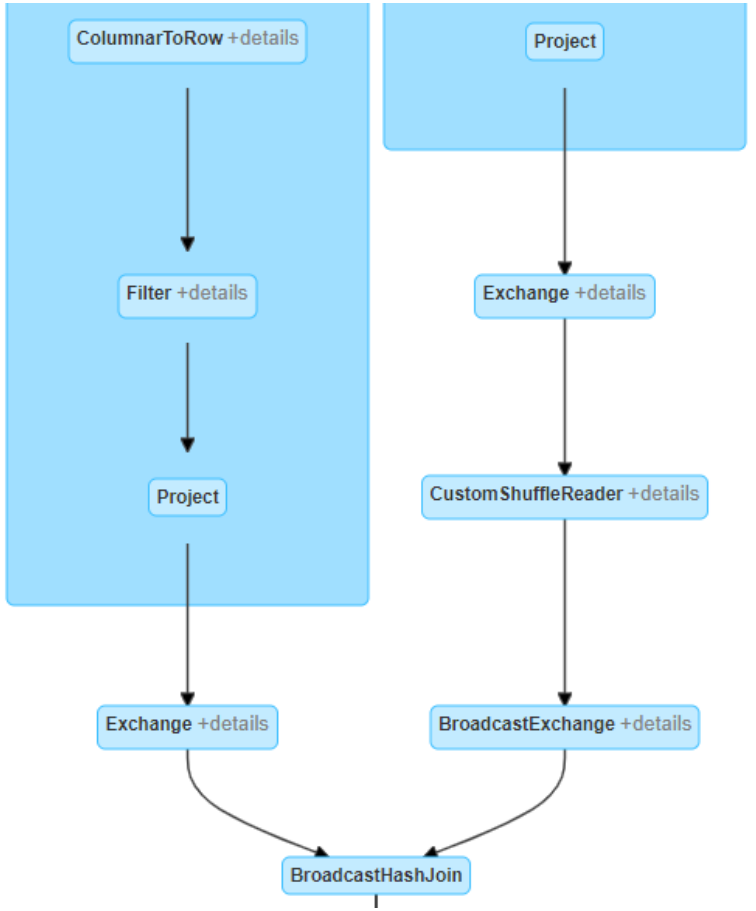
spark.databricks.delta.optimize.maxfilesize 16-128MB or turn on optimized writes.

- **Don’t use Python or Scala UDF if native function exists** – transfer data between Python and Spark = serialization is needed = drastically slows down queries
- **Move numericals, keys**, high cardinality query predicates **to the left**, long string that are not distinct enough for stats collection move to the left (only 32 columns has statistics)
- OPTIMIZE benefits from **Compute Optimized clusters** (because of a lot of encoding and decoding parquet files)
- Think about spot instances,
- For some operations consider **Databricks Standard**.

Adaptive Query Processing

- Game changer in Spark 3.x
- For example: Initially **SortMergeJoin** chosen but once the ingest stages completes plan will be updated to use **BroadcastHashJoin**





Other

- **Turn Adaptive Query Execution** (default)
- **Turn Coalesce Partitions** on (spark.sql.adaptive.coalescePartitions.enabled)
- **Turn Skew Join On** (spark.sql.adaptive.skewJoin.enabled)
- **Turn Local Shuffle Reader** on (spark.sql.adaptive.localShuffleReader.enabled)
- **Broadcast Join threshold** (spark.sql.autoBroadcastJoinThreshold)
- **Don't prefer Sort MergeJoin** spark.sql.join.prefersortmergejoin -> false

- Turn off stats collection
- dataSkippingNumIndexedCols 0

- Optimize Zorder by merge keys between Bronze & Silver
- Turn Optimized Writes
- Restructure columns for skipping

- Optimize ZOrder by join keys or High Cardinality columns used in WHERE
- Turn Optimized Writes
- Enable Databricks IO cache
- Consider using Photon
- Consider using Premium Storage
- Build preaggregate tables

Dziękuję!

STRATEGIC PARTNER

dbWatch
DATABASE CONTROL