ADVANCING ANALYTICS

Performance Tuning

Performance Tuning & Optimisation

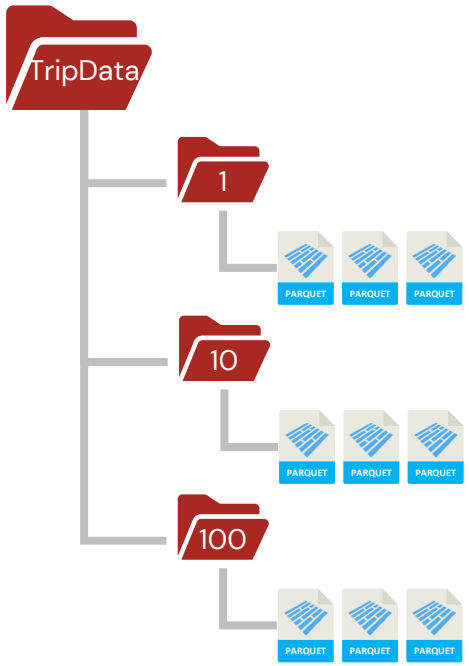
Partitioning

Partition

- Data is stored as multiple small files in folder

- Partition divides those data file into useful slices

	path	name	size
1	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/_delta_log/	_delta_log/	0
2	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/part-00000-829f8ebc-6be6-4491-a1ba-f9bb93311d5f-c000.snappy.parquet	part-00000-829f8ebc-6be6-4491-a1ba-f9bb93311d5f-c000.snappy.parquet	161807024
3	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/part-00001-a95fa1fc-a3c5-4249-b09b-a2e475442757-c000.snappy.parquet	part-00001-a95fa1fc-a3c5-4249-b09b-a2e475442757-c000.snappy.parquet	154872635
4	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/part-00002-459fd8dc-f4ce-4dcf-8fd9-d87bdf2f9c25-c000.snappy.parquet	part-00002-459fd8dc-f4ce-4dcf-8fd9-d87bdf2f9c25-c000.snappy.parquet	153406419
5	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/part-00003-2f681144-3098-4adf-a917-b5800c8557d5-c000.snappy.parquet	part-00003-2f681144-3098-4adf-a917-b5800c8557d5-c000.snappy.parquet	154855624
6	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/part-00004-e30f11fb-008d-4029-87bd-7953f9346e20-c000.snappy.parquet	part-00004-e30f11fb-008d-4029-87bd-7953f9346e20-c000.snappy.parquet	147625269
7	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/part-00005-b470a4e3-a8fb-413e-85c4-320e8d82ca3e-	part-00005-b470a4e3-a8fb-413e-85c4-320e8d82ca3e-c000.snappy.parquet	144943806



	path	name	size
1	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=1/	PULocationID=1/	0
2	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=10/	PULocationID=10/	0
3	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=100/	PULocationID=100/	0
4	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=101/	PULocationID=101/	0
5	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=102/	PULocationID=102/	0
6	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=104/	PULocationID=104/	0
7	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=105/	PULocationID=105/	0

Writing Partitions

Use the PARTITIONED BY or .partitionBy() command

```
-- %sql

CREATE TABLE IF NOT EXISTS [table]
  PARTITIONED BY ("[columnName]")
  ...
```

```
-- %python

( df.write
    .format("delta")
    .mode("overwrite")
    .partitionBy("[columnName]")
    .save("[deltaFilePath]")
)
```

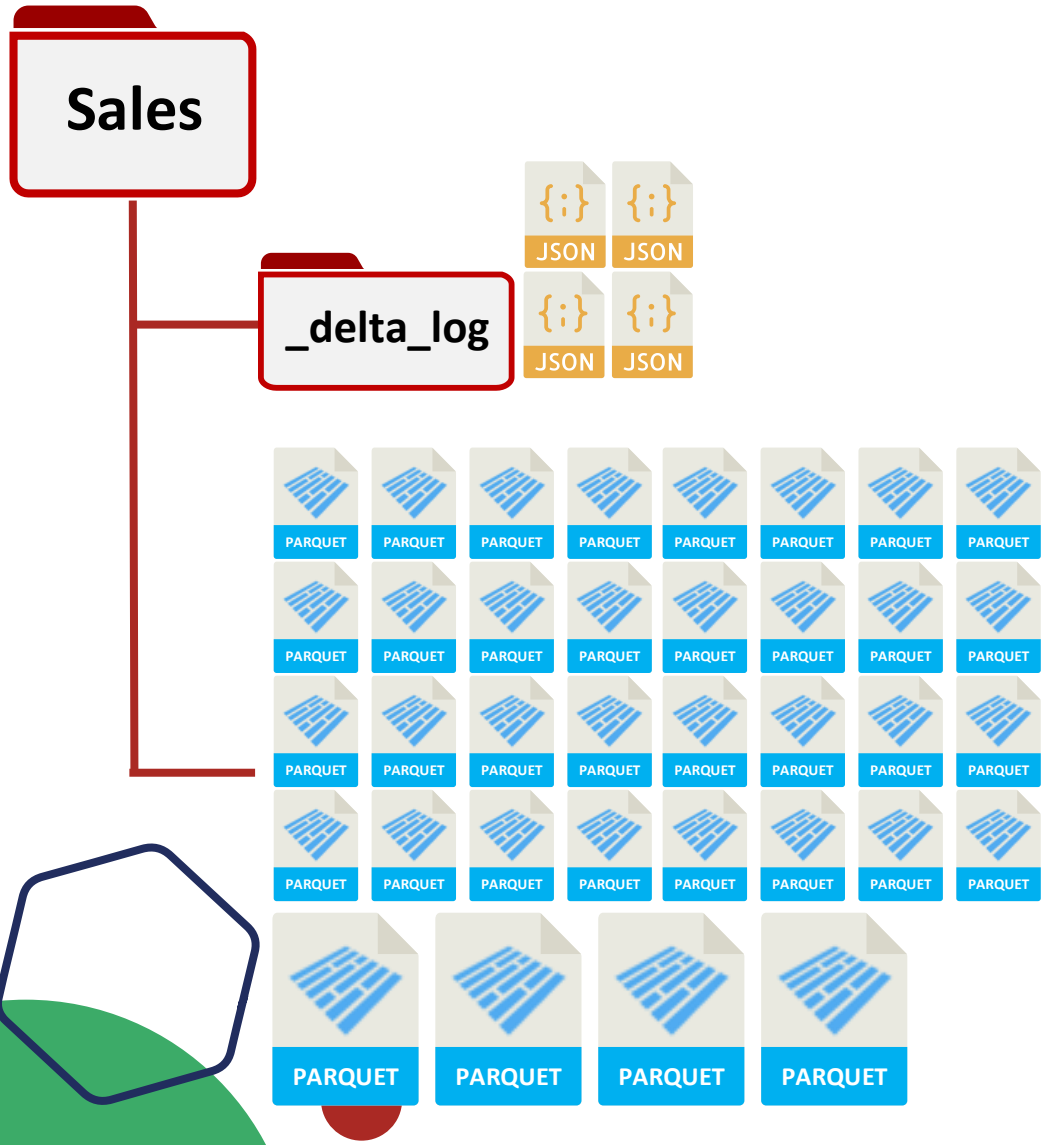
	path	name	size
1	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=1/	PULocationID=1/	0
2	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=10/	PULocationID=10/	0
3	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=100/	PULocationID=100/	0
4	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=101/	PULocationID=101/	0
5	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=102/	PULocationID=102/	0
6	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=104/	PULocationID=104/	0
7	dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Partitioned/PULocationID=105/	PULocationID=105/	0

Demo: Partitioning

Partition a Tsable

Optimization

Small/Incremental Updates Problem



- Parquet files have an **optimal size**
- Updates **small and inefficient files**
- `OPTIMIZE` command compacts small files into larger
- `OPTIMIZE` command performed on entire table
- Files are NOT deleted
- Add to the JSON transaction log

ADVANCING ANALYTICS

OPTIMIZE using Bin-packing

```
--Optimize an entire table
OPTIMIZE <database>.<table>

--Optimize a specific partition (this MUST be a partition column)
OPTIMIZE <database>.<table> WHERE purchaseMonth = 202007
```

```
1 %sql
2 optimize optimal.NYCMonth ZORDER BY (PickupDate_day)
```

▸ (4) Spark Jobs

	path	metrics
1	null	▾ object numFilesAdded: 10 numFilesRemoved: 200 ▸ filesAdded: {"min": 45382662, "max": 285639606, "avg": 240297886, "totalFiles": 10, "totalSize": 2402978867} ▸ filesRemoved: {"min": 11851782, "max": 11921682, "avg": 11889963, "totalFiles": 200, "totalSize": 2377992788} partitionsOptimized: 0 ▸ zOrderStats: {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputOtherFiles": {"num": 200, "size": 2377992788}, "inputNumCubes": 0, "mergedFiles": {"num": 200, "size": 2377992788}, "mergedNumCubes": 0} numBatches: 1

OPTIMIZE using Bin-packing

- Python/ Scala operations
- Work on a “DeltaTable” object rather than a dataframe
- Using .optimize() command
- Using .where() command

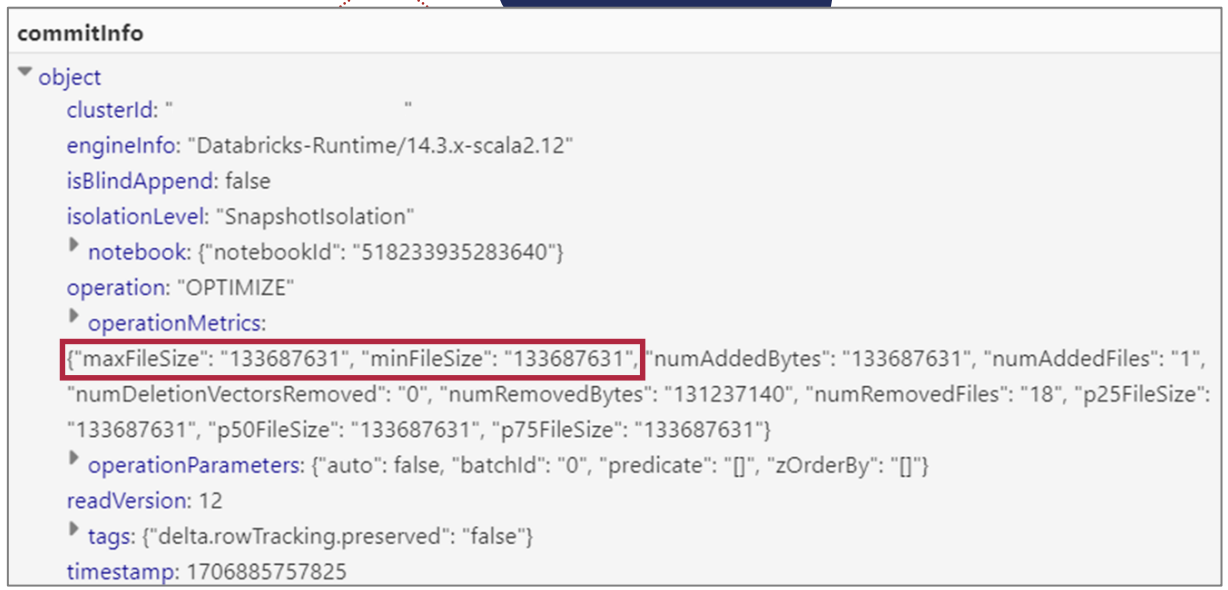
```
# This requires a “deltaTable” object
     delta.tables          *

deltaTable = DeltaTable.forPath(spark, <Path to Delta Table>)
# OR
deltaTable = DeltaTable.forName(spark, <HIVE Table Name>)`
```

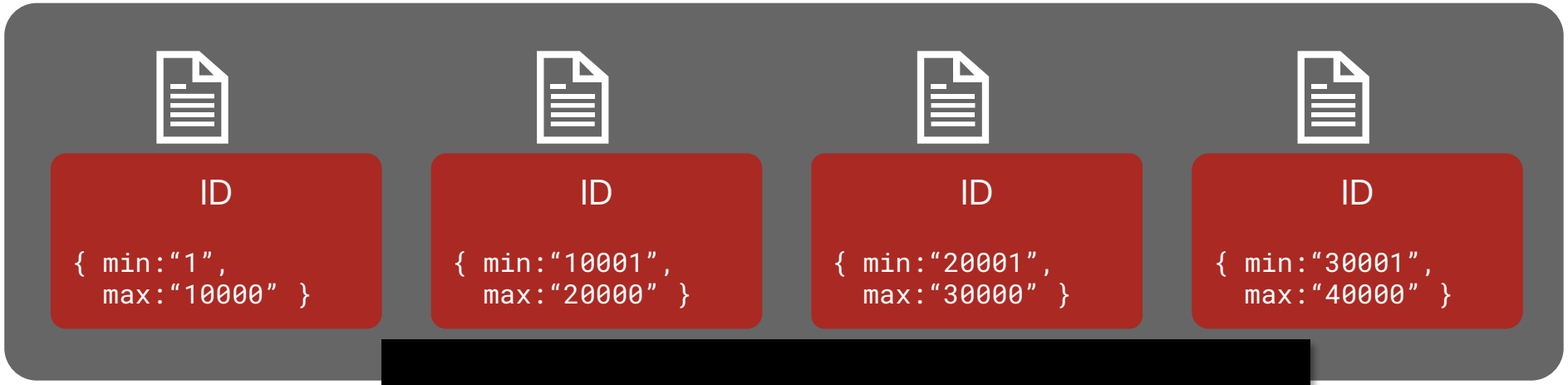
```
deltaTable.optimize().executeCompaction()
```

```
deltaTable.optimize().where("date='<date>'").executeCompaction()
```

Data Skipping

- Collect Statistics in Transaction Log
- Collects “min/max” Value For First 32 Columns
- Selectively Ignore Files






```
SELECT * FROM [table] WHERE ID < “20000”
```

- Do not need to read all the file and can skip it

Data Skipping

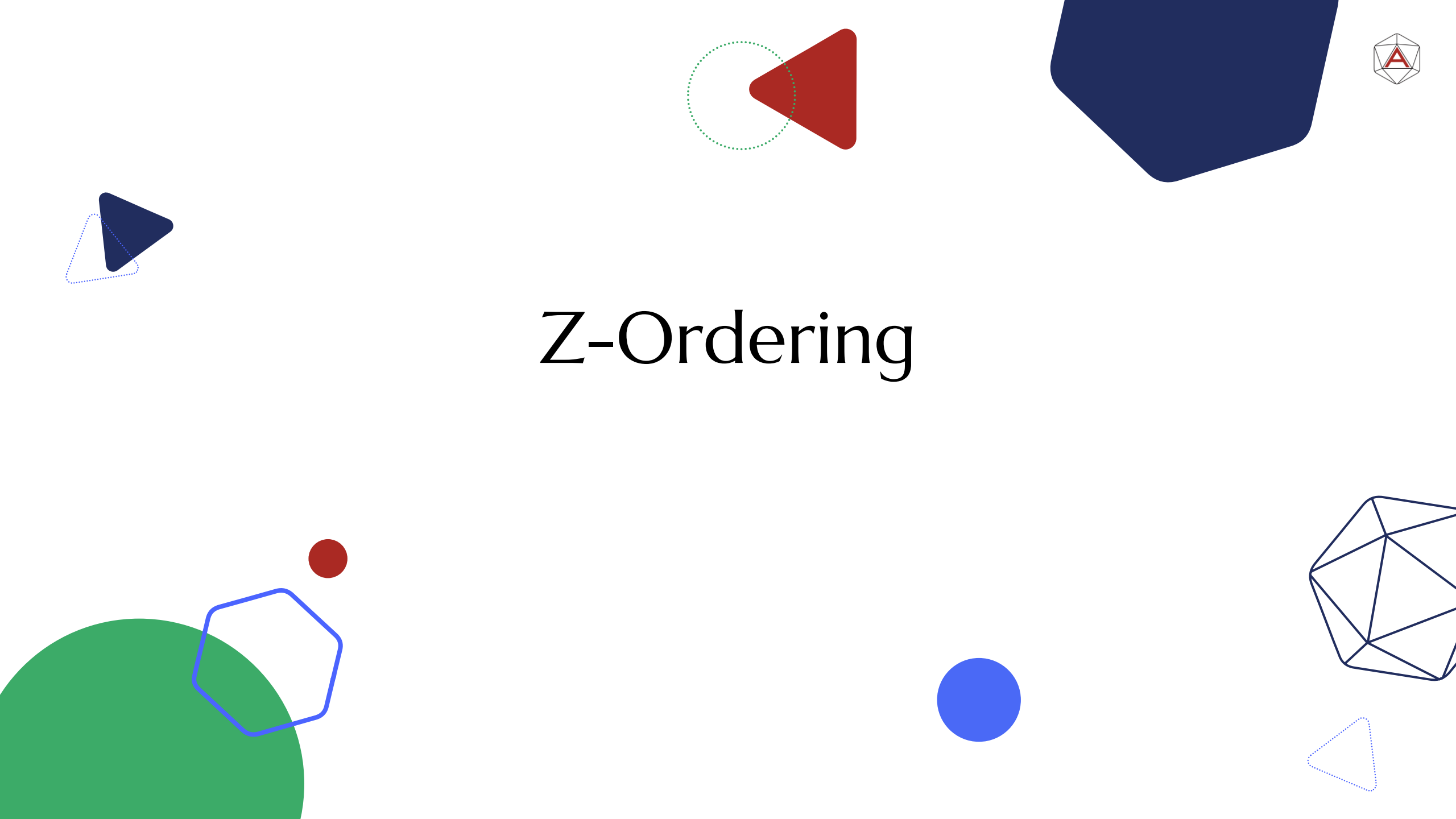
- Specify Delta Statistics Columns

```
delta.dataSkippingNumIndexedCols
```

- More Effective on Keys, Number, High Cardinality

- More Less Effective on Long Strings and Timestamps

Z-Ordering



Z-Ordering

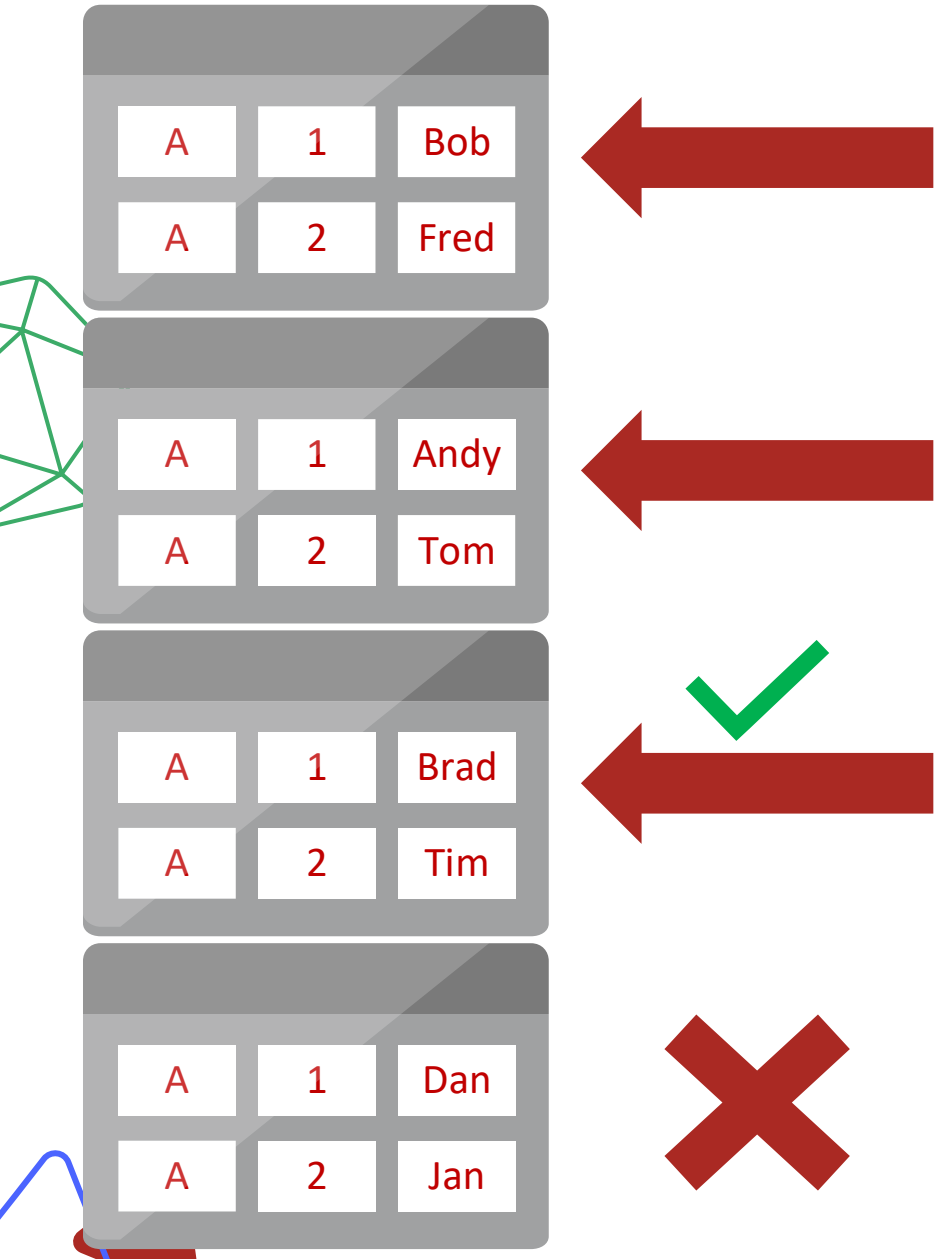
"Sort the data on specific columns before writing to files, to optimize data skipping"

```
--Optimize an entire table
OPTIMIZE [database].[table] ZORDER BY [ColumnName]
```

Z-Order can be expense, as with any sort-based Operation. It is sometimes better to perform this as an out-of-hours maintenance operation

Z-Ordering Explained

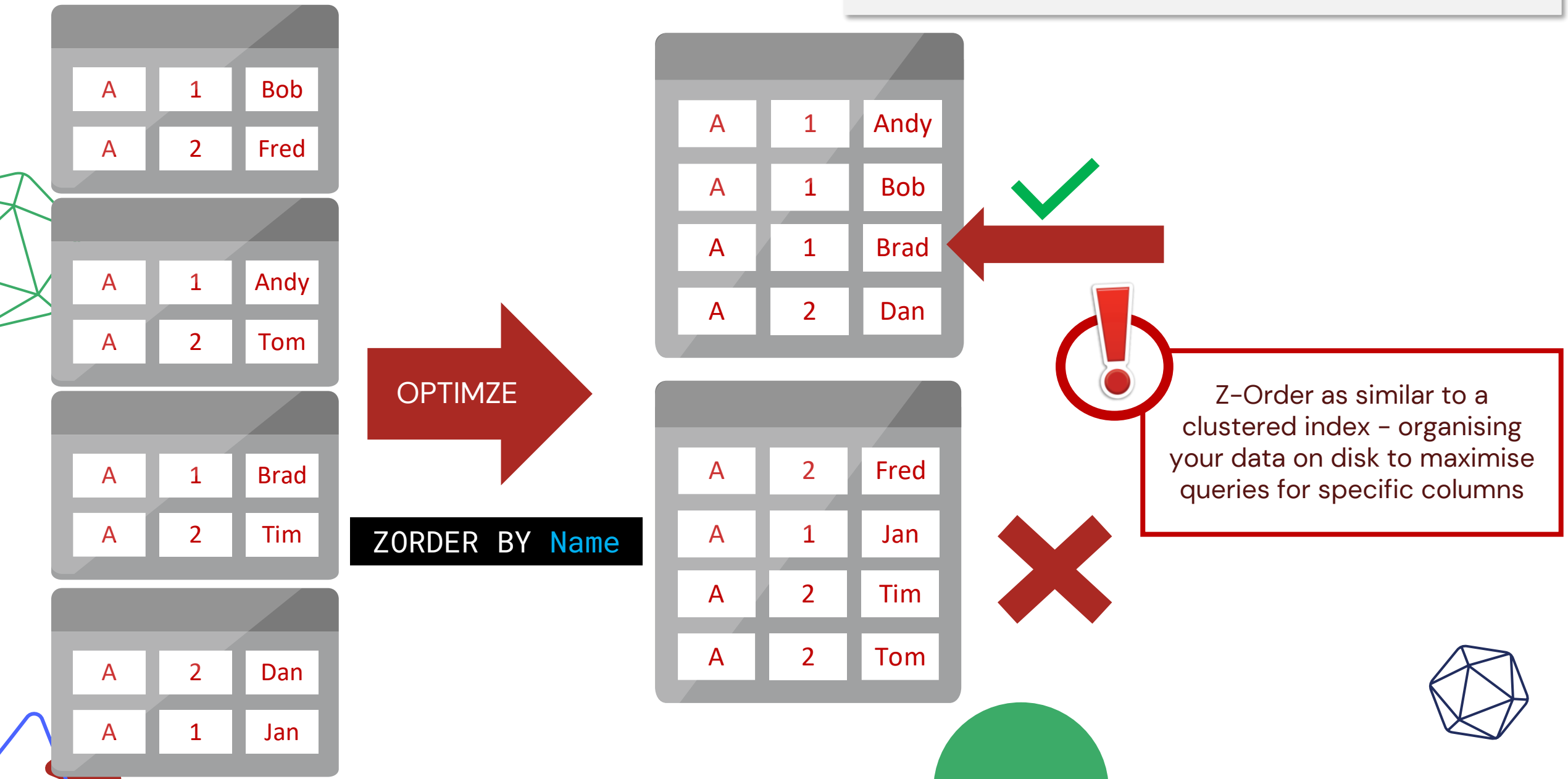
```
SELECT count(*) FROM Employees
WHERE Name = 'Brad'
```



- The small files are not ordered
- SQL statement to query data
- Data skipping will look at all files until it finds what it's looking for

Z-Ordering Explained

```
SELECT count(*) FROM Employees
WHERE Name = 'Brad'
```



Auto-Optimize

Auto Optimize

- Optimize at specifically at certain times when **required**
- Optimize automatically every time you **perform a write** using ***Auto-Optimize***

```
--Enable AutoOptimize
set spark.databricks.delta.properties.defaults.autoOptimize.optimizeWrite = true;
set spark.databricks.delta.properties.defaults.autoOptimize.autoCompact = true;
```

Options:

- **Optimized Writes:**
  - Dynamically optimizes **partition sizes**
  - Attempts to **write out 128 MB files** for each table partition
- **Auto Compaction:**
  - Run a **lightweight optimize job** after the **write** has finished
  - Further file **compaction opportunities**

Auto Optimise vs Optimize

OPTIMIZE is specified **manually**

- Compacts small files or co-locate common data using Z-ORDER
- SQL command

```
--Optimize an entire table
OPTIMIZE <database>.<table> ZORDER BY <columnName>
```

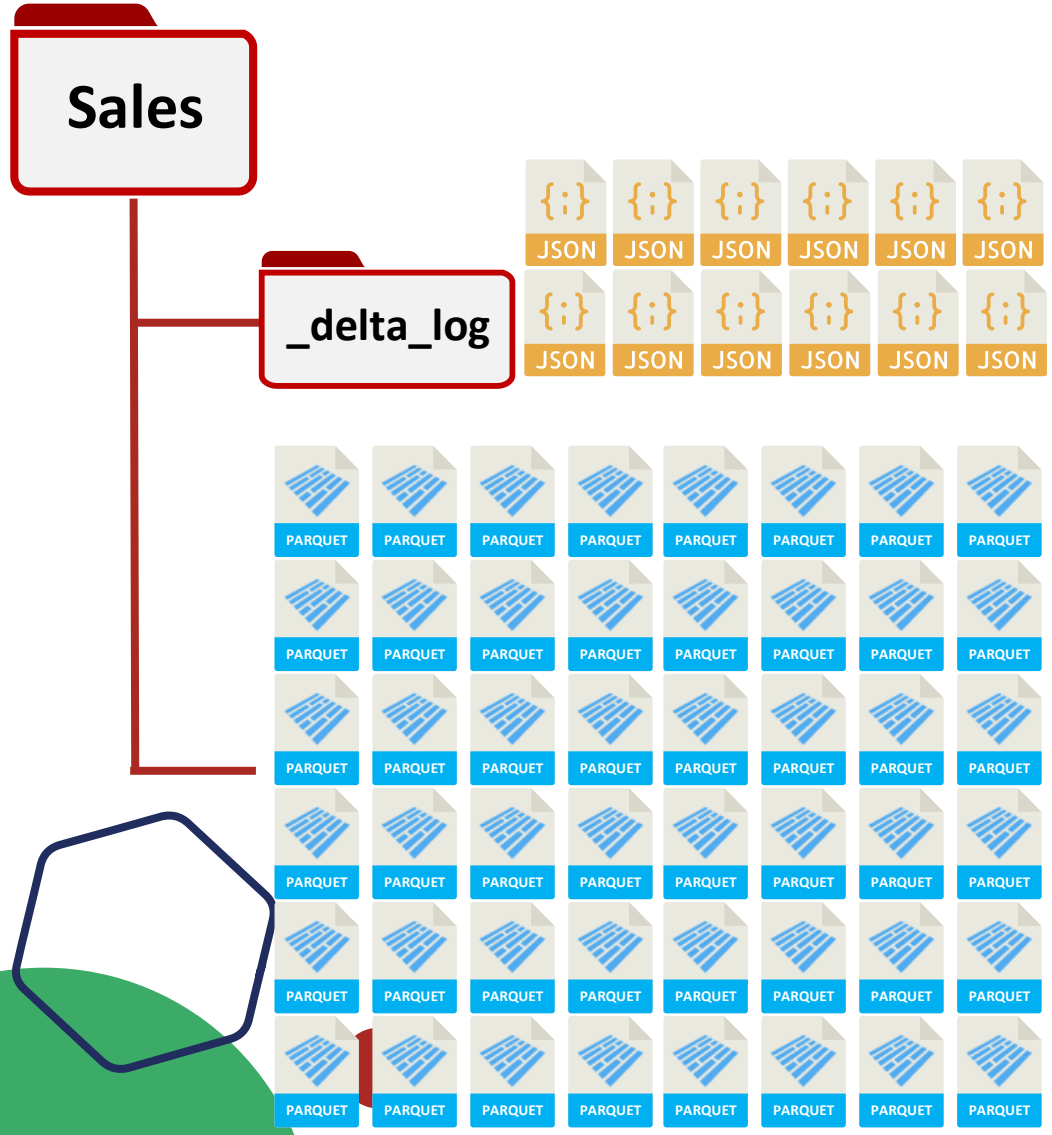
Auto-Optimize is **automatically**

- Performs during every write action, compacting into small files
- Spark Configuration settings change

```
--Enable AutoOptimize
set spark.databricks.delta.properties.defaults.autoOptimize.optimizeWrite = true;
set spark.databricks.delta.properties.defaults.autoOptimize.autoCompact = true;
```

Vacuum

Obsolete Files



To remove obsolete history files, Delta has the **VACUUM** command

This command physically deletes data files older than a specified date

You CANNOT time travel past dates where history has been vacuumed

ADVANCING ANALYTICS

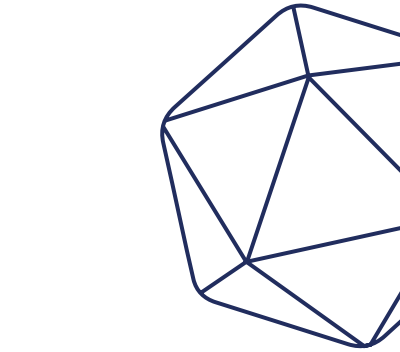
Using the Vacuum Command

Vacuuming in **SQL**:

```
--Vacuum Table using defaults
VACUUM [database].[table]

--Vacuum using path not Hive table
VACUUM '/mnt/lake/BASE/myTable/'

--VACUUM for a non-default time period
VACUUM [database].[table] RETAIN 168 HOURS

--TEST THE VACUUM BEFORE YOU RUN IT
VACUUM [database].[table] RETAIN 168 HOURS DRY RUN
```

Using the **Python deltaTable** object:

```
# Vacuum Table using defaults
deltaTable.vacuum()

# Vacuum Table for files older than 7 days (168 hours)
deltaTable.vacuum(168)
```



ADVANCING ANALYTICS

Demo: Optimization

Optimize a Delta Table

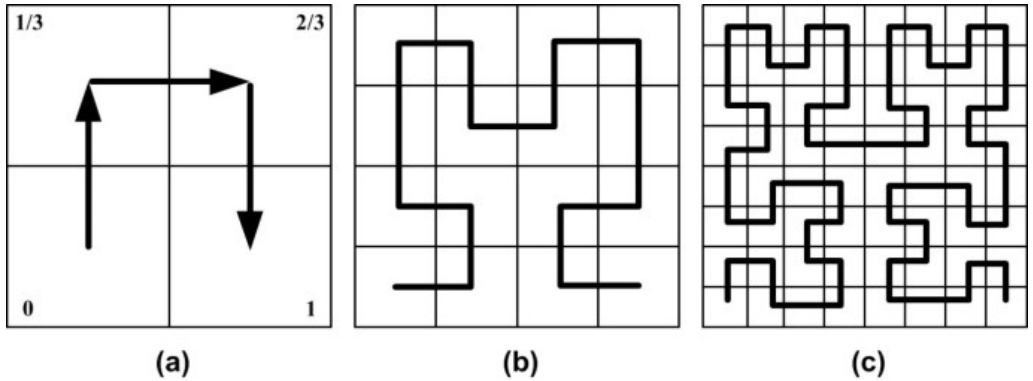
LAB01: Optimization

Optimize & Vacuum

Liquid Clustering

Liquid Clustering

- New Optimizing Approach
- Flexible
- Simplifies Data Layout
- Enhances Query Performance
- Streamlines Optimization Process
- Hilbert Curves Algorithm

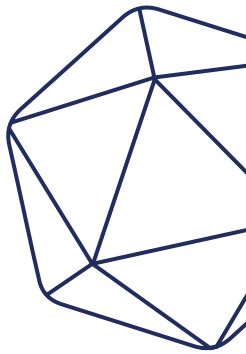


Source: *https://asp-eurasipjournals.springeropen.com/articles/10.1186/1687-6180-2012-181/figures/1*

ADVANCING ANALYTICS

Liquid Clustering

- Dynamic Data Layout
- Adaptable Clustering Management
- Simplified Command Execution
- Eliminates Partitioning and Z-Ordering
- Efficient Table Creation Process

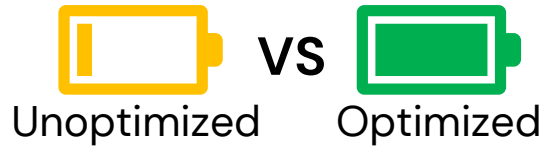


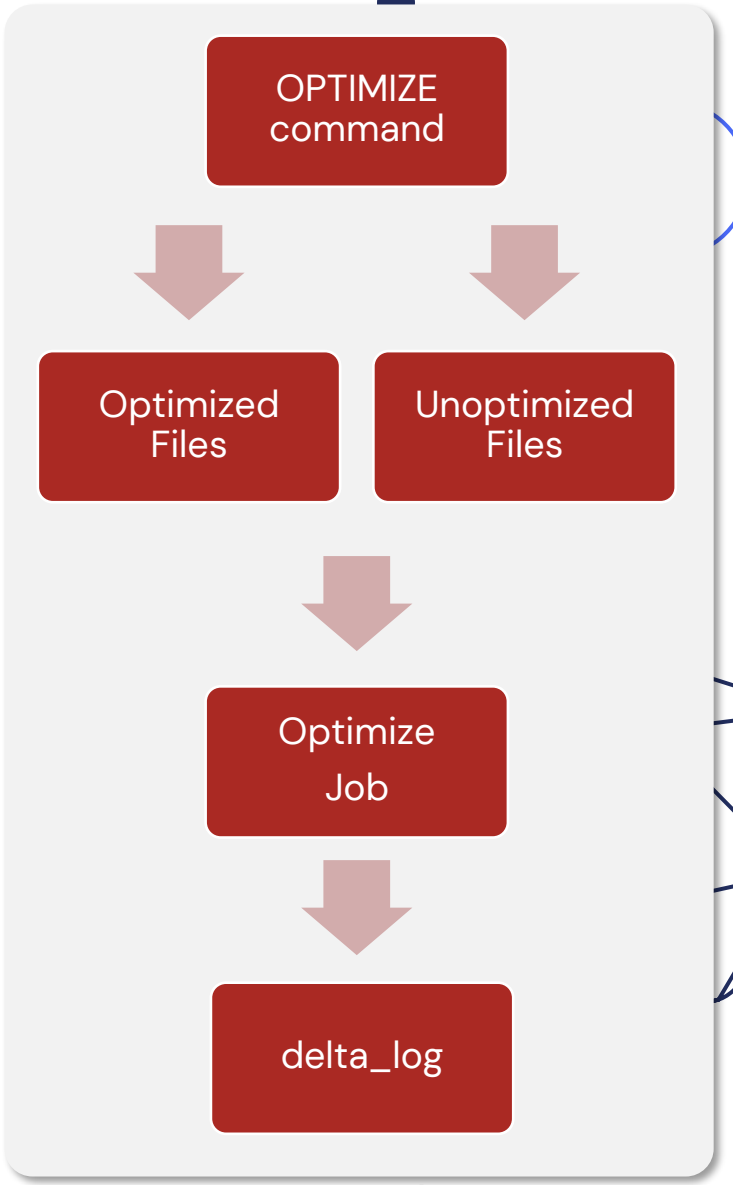
Liquid Clustering - Scenarios

- High Cardinality Columns
- Skew Data Distribution
- Rapidly Growing Data with High Maintenance
- Concurrent Write Operations
- Inadequate or Excessive Partitions

Liquid Clustering – Explained

- Incremental Clustering
- Optimize Flow
- Metadata Integration





ADVANCING ANALYTICS

Liquid Clustering – Commands

- Clustering Columns in Table Definitions

```
-- Create table
CREATE TABLE [database].[table] CLUSTER BY [columnName]
```

- Managing Clustering Columns

```
-- Modified Table
ALTER TABLE [database].[table] CLUSTER BY [columnName]

-- Rename Column - Column mapping needs to be enabled
ALTER TABLE [database].[table] RENAME COLUMN [oldName] TO [newName]

-- Deleted Cluster Column
ALTER TABLE [database].[table] CLUSTER BY NONE
```

- Triggering Clustering Columns

```
--Optimize Table
OPTIMIZE [database].[table]
```

- Viewing Metadata

```
--View Metadata
DESCRIBE TABLE [table] or DESCRIBE DETAIL [table]
```

Liquid Clustering – Consideration

- Default Choice
- Compatibility

```
> Databricks Runtime 13.3 LTS
```

- Uses Specific Delta Versions

- Not Compatible With Previous Strategies
- Eliminates OPTIMIZE ZORDER BY

ADVANCING ANALYTICS

Liquid Clustering – Limitations

- Columns With Statistics
- 4 Columns
- Structured Streaming

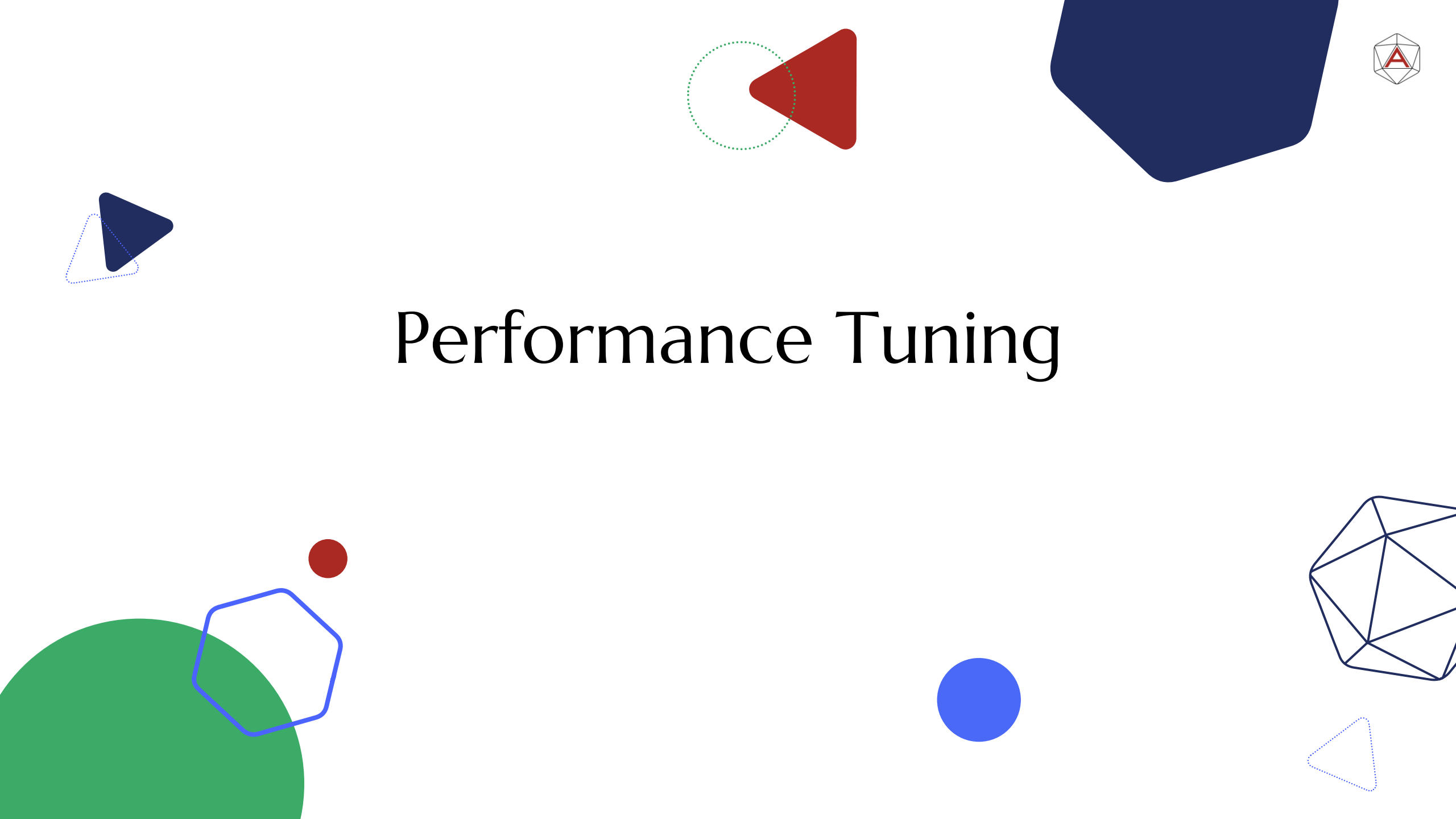
ADVANCING ANALYTICS

Demo: Liquid Clustering

Liquid Clustering

LAB02: Liquid Clustering

Liquid Clustering & Vacuum

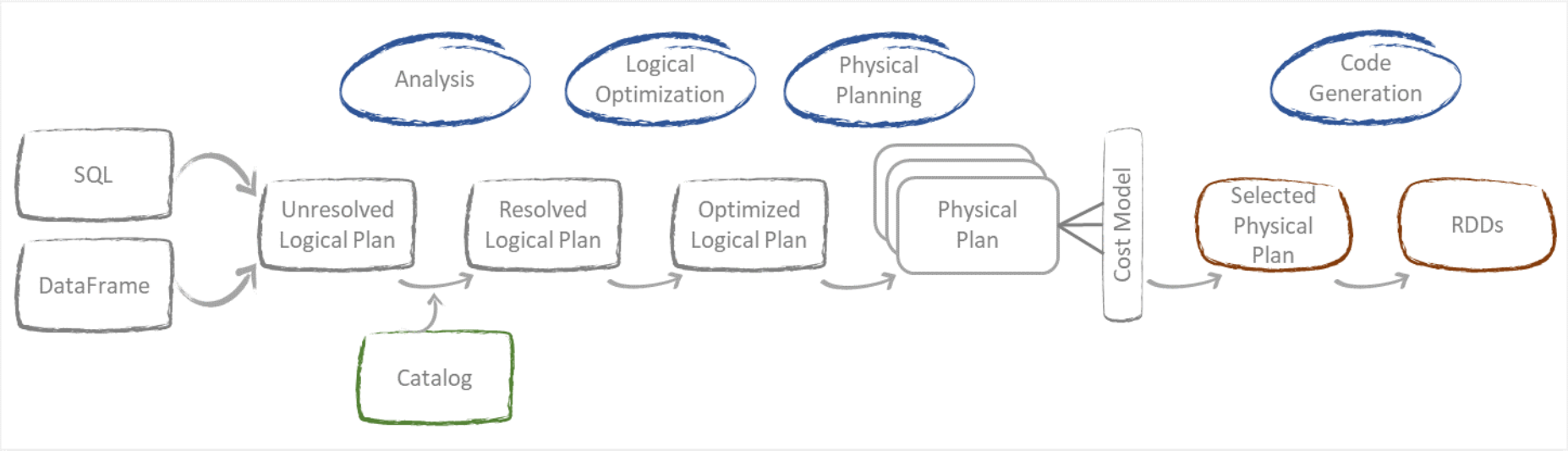


Performance Tuning

Execution Plan

Execution Plan

- Optimising based on the Execution Plans
- Identifies the most effective method to execute the Spark operation
- Use Catalyst Optimizer



Logical Plan

Logical Plan is broken down into three sections:

- **Unresolved Logical Plan** (Parsed Logical Plan)
  - Identifies the `Unresolved` objects
- **Resolved Logical Plan** (Analyzed Logical Plan)
  - Validates the `Unresolved` objects
- **Optimized Logical Plan**
  - Applies predicates or rules to further optimize the plan

```
== Parsed Logical Plan ==
'Sort ['customer_id ASC NULLS FIRST], true
+- Aggregate [customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759], [customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759, sum((units_purchased#1759 * cast(10 as double))) AS total_units_purchased#1785]
   +- Filter cast(customer_id#1751 as string) IN (cast(95432188 as string),cast(98765432 as string),cast(12345678 as string),cast(99999999 as string),cast(11123757 as string))
      +- SubqueryAlias spark_catalog.MasteringDelta.Customers
         +- Relation[customer_id#1751,customer_name#1752,street#1753,city#1754,district#1755,state#1756,postcode#1757,region#1758,units_purchased#1759,valid_from#1760,country#1761] parquet

== Analyzed Logical Plan ==
customer_id: int, customer_name: string, street: string, city: string, units_purchased: double, total_units_purchased: double
Sort [customer_id#1751 ASC NULLS FIRST], true
+- Aggregate [customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759], [customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759, sum((units_purchased#1759 * cast(10 as double))) AS total_units_purchased#1785]
   +- Filter cast(customer_id#1751 as string) IN (cast(95432188 as string),cast(98765432 as string),cast(12345678 as string),cast(99999999 as string),cast(11123757 as string))
      +- SubqueryAlias spark_catalog.MasteringDelta.Customers
         +- Relation[customer_id#1751,customer_name#1752,street#1753,city#1754,district#1755,state#1756,postcode#1757,region#1758,units_purchased#1759,valid_from#1760,country#1761] parquet

== Optimized Logical Plan ==
Sort [customer_id#1751 ASC NULLS FIRST], true
+- Aggregate [customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759], [customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759, sum((units_purchased#1759 * 10.0)) AS total_units_purchased#1785]
   +- Project [customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759]
      +- Filter cast(customer_id#1751 as string) IN (95432188,98765432,12345678,99999999,11123757)
         +- Relation[customer_id#1751,customer_name#1752,street#1753,city#1754,district#1755,state#1756,postcode#1757,region#1758,units_purchased#1759,valid_from#1760,country#1761] parquet
```

Physical Plan

- Is how the Logical Plan will be **executed** on the cluster
- Generates different **execution strategies**
- Compares them through a **Cost Model**
- Selects the **best optimal plan/strategy** as the “Best Physical Plan”

```
== Physical Plan ==
Sort [customer_id#1751 ASC NULLS FIRST], true, 0
+- Exchange rangepartitioning(customer_id#1751 ASC NULLS FIRST, 200), ENSURE_REQUIREMENTS, [id=#2608]
   +- *(2) HashAggregate(keys=[customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759], functions=[finalmerge_sum(merge sum#1879) AS sum((units_purchased#1759 * 10.0))#1784])
      +- Exchange hashpartitioning(customer_id#1751, customer_name#1752, street#1753, city#1754, units_purchased#1759, 200), ENSURE_REQUIREMENTS, [id=#2604]
         +- *(1) HashAggregate(keys=[customer_id#1751, customer_name#1752, street#1753, city#1754, knownfloatingpointnormalized(normalizenanandzero(units_purchased#1759)) AS units_purchased#1759], functions=[partial_sum((units_purchased#1759 * 10.0)) AS sum#1879])
            +- *(1) Filter cast(customer_id#1751 as string) IN (95432188,98765432,12345678,99999999,11123757)
               +- *(1) ColumnarToRow
                  +- FileScan parquet masteringdelta.customers[customer_id#1751,customer_name#1752,street#1753,city#1754,units_purchased#1759] Batched: true, DataFilters: [cast(customer_id#1751 as string) IN (95432188,98765432,12345678,99999999,11123757)], Format: Parquet,
Location: PreparedDeltaFileIndex[dbfs:/user/hive/warehouse/masteringdelta.db/customers], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<customer_id:int,customer_name:string,street:string,city:string,units_purchased:double>
```

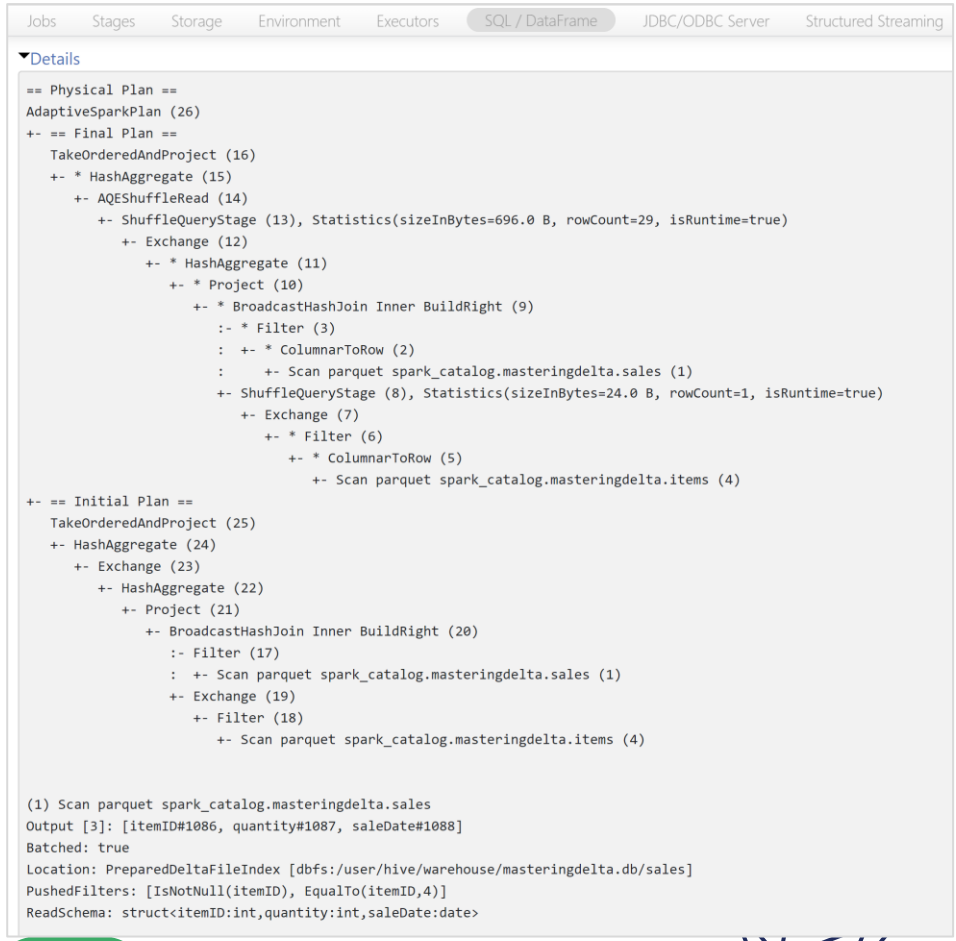
Generate Execution Plan

- Execution Plan can be generated using SQL or Python syntax
- Applying additional parameters show different output format of plan
- Can view in Spark UI

```
EXPLAIN EXTENDED
SELECT statement
```

```
%python
df.explain("extended")
```

```
EXPLAIN [ EXTENDED | CODEGEN | COST | FORMATTED ]
statement
```

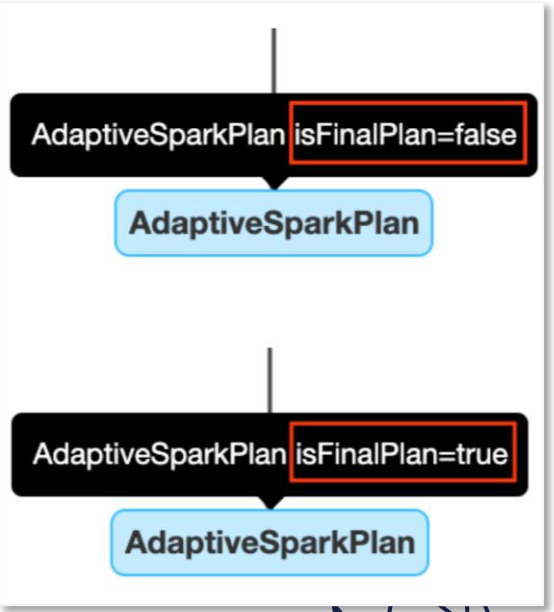



Adaptive Query Execution

Adaptive Query Execution (AQE)

- Optimizes further by changing the Query Plan:
  - Uses Runtime Statistics
  - Increases Query Performance
- Visible in Spark UI
- Enable using Spark Configuration settings

```
AdaptiveSparkPlan isFinalPlan=true
+- == Final Plan ==
   *(3) BroadcastHashJoin [key#13], [a#23], Inner, BuildLeft, false
   :- BroadcastQueryStage 2, Statistics(sizeInBytes=1024.0 KiB, rowCount=1,
   :  +- BroadcastExchange
   :     ...
+- == Initial Plan ==
   SortMergeJoin [key#13], [a#23], Inner
   :- Sort [key#13 ASC NULLS FIRST], false, 0
   :  +- Exchange hashpartitioning(key#13, 5), true, [id=#117]
   :     ...
```




```
spark.conf.set("spark.sql.adaptive.enabled", "true")
```

Demo: Execution Plan

Execution Plan

Query Hints

Query Hints

- Specify the approach

Partitioning Hints

- COALESCE, REPARTITION, REPARTITION_BY_RANGE, REBALANCE

```
SELECT /*+ [COALESCE | REPARTITION | REPARTITION_BY_RANGE |
REBALANCE](n) */
       <columnName>
FROM <t1>
```

Join Hints

- BROADCAST, MERGE, SHUFFLE_HASH, SHUFFLE_REPLICATE_NL

```
SELECT /*+ [BROADCAST | MERGE | SHUFFLE_HASH | SHUFFLE_HASH](t1) */
       <columnName>
FROM <t1>
INNER JOIN <t2> ON t1.key = t2.key;
```

LAB03: Execution Plan

Execution Plan

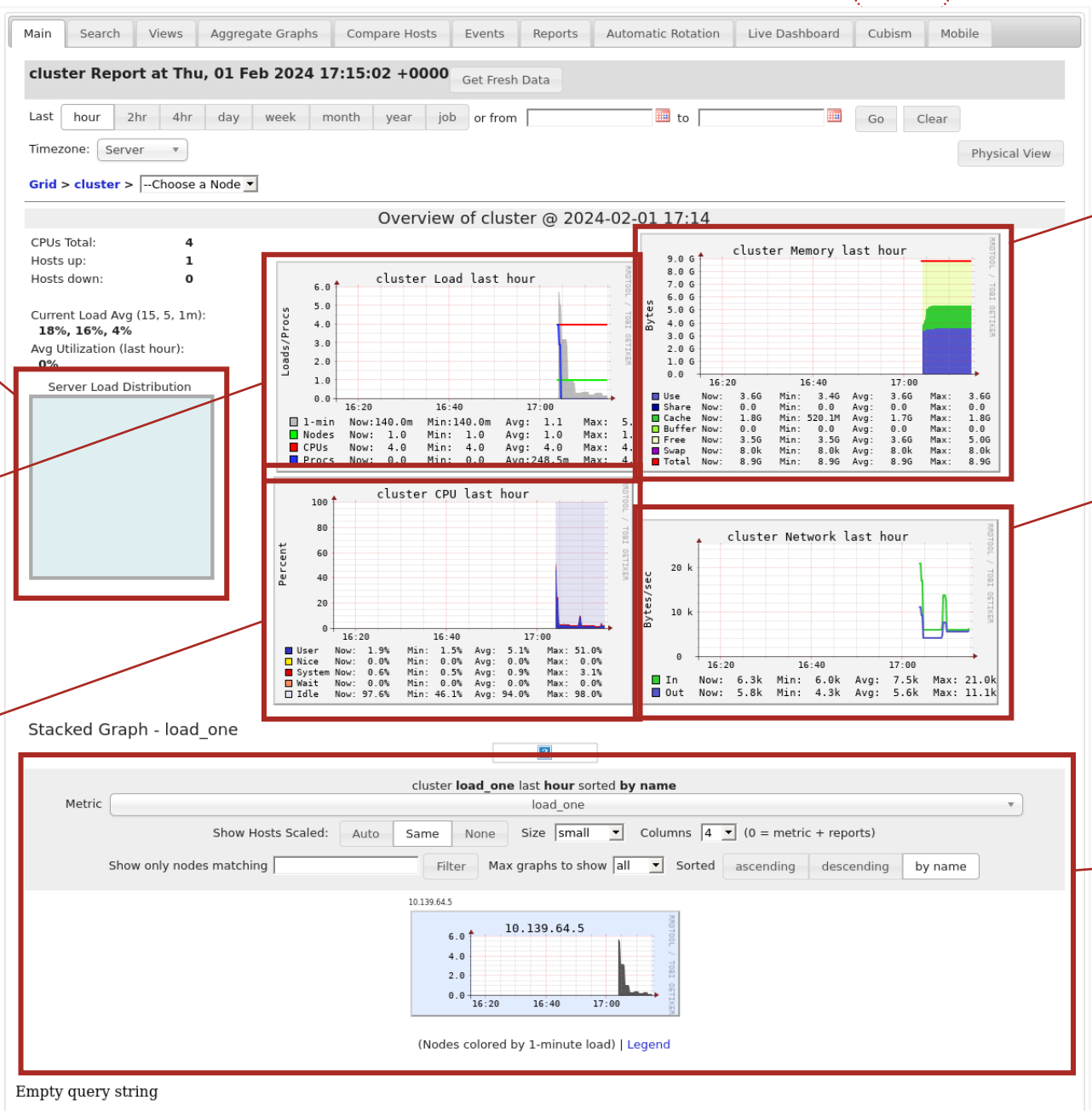
Cluster Monitoring

Ganglia

- Native Monitoring
- Databricks Runtime < 12.0
- Gathers Live Cluster Metrics (CPU, Memory, etc.)
- Generates Snapshot of Usage
- Historical Metrics Stored

https://www.youtube.com/watch?v=y3VCWVbzAKA

Ganglia Snapshot



Server Load Distribution

Cluster Load

Cluster CPU

Cluster Memory

Networking

Cluster Nodes

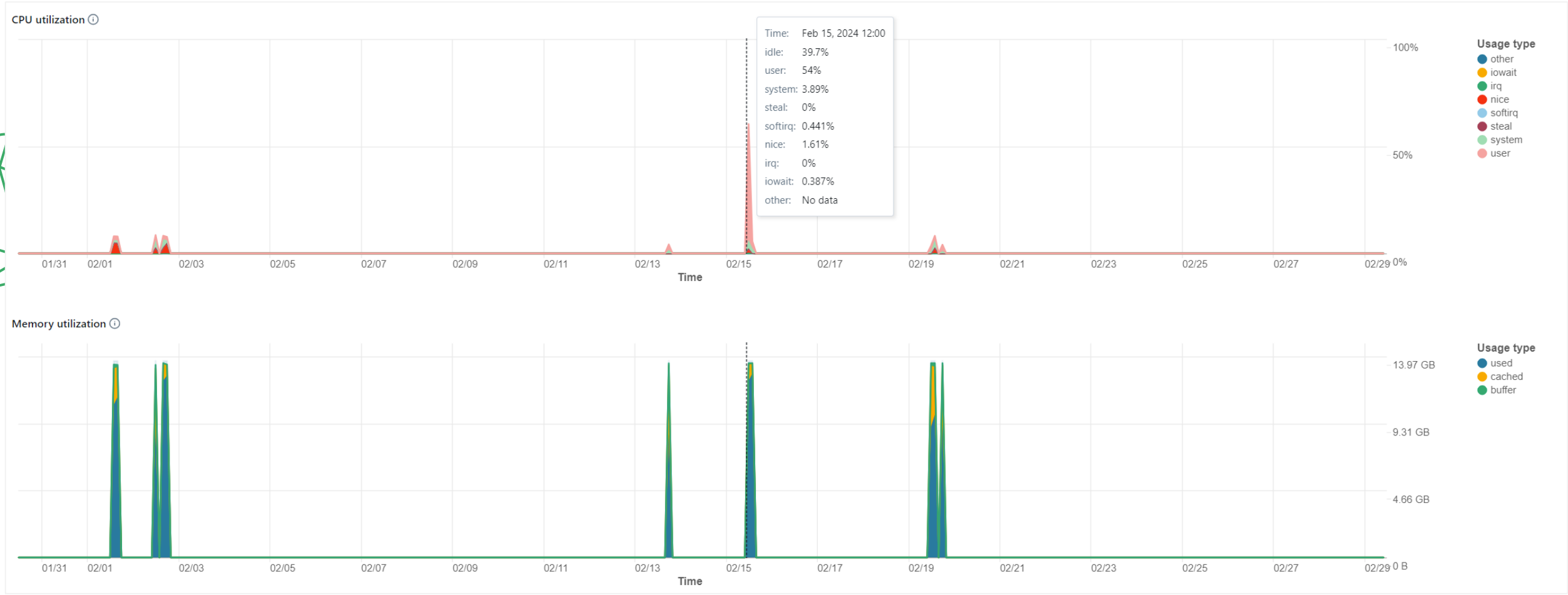
Metrics

- Native Monitoring
- Databricks Runtime 13.0+
- Metrics replaces Ganglia
- Gathers Key Hardware & Spark Metrics
- Stored in Azure Databricks-Managed Storage

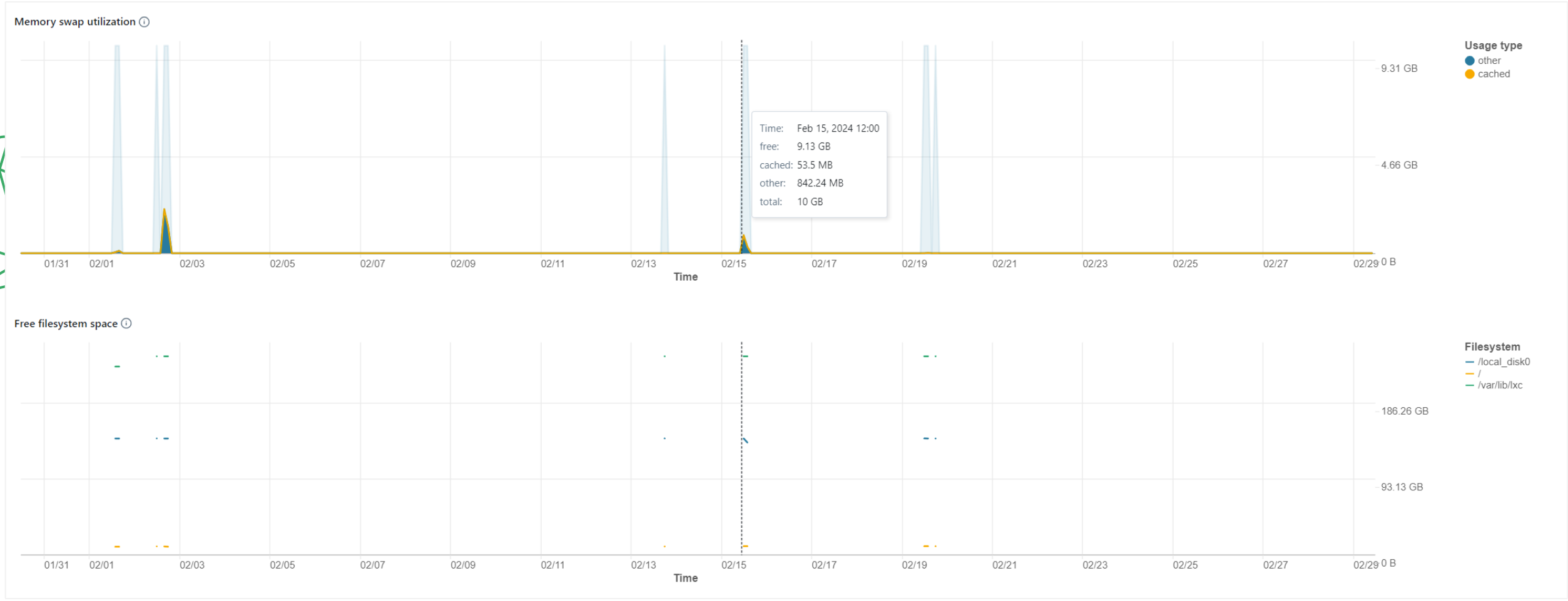
- Metrics
  - Hardware
  - Spark
  - GPU

https://www.youtube.com/watch?v=9fa8dnKbfsU

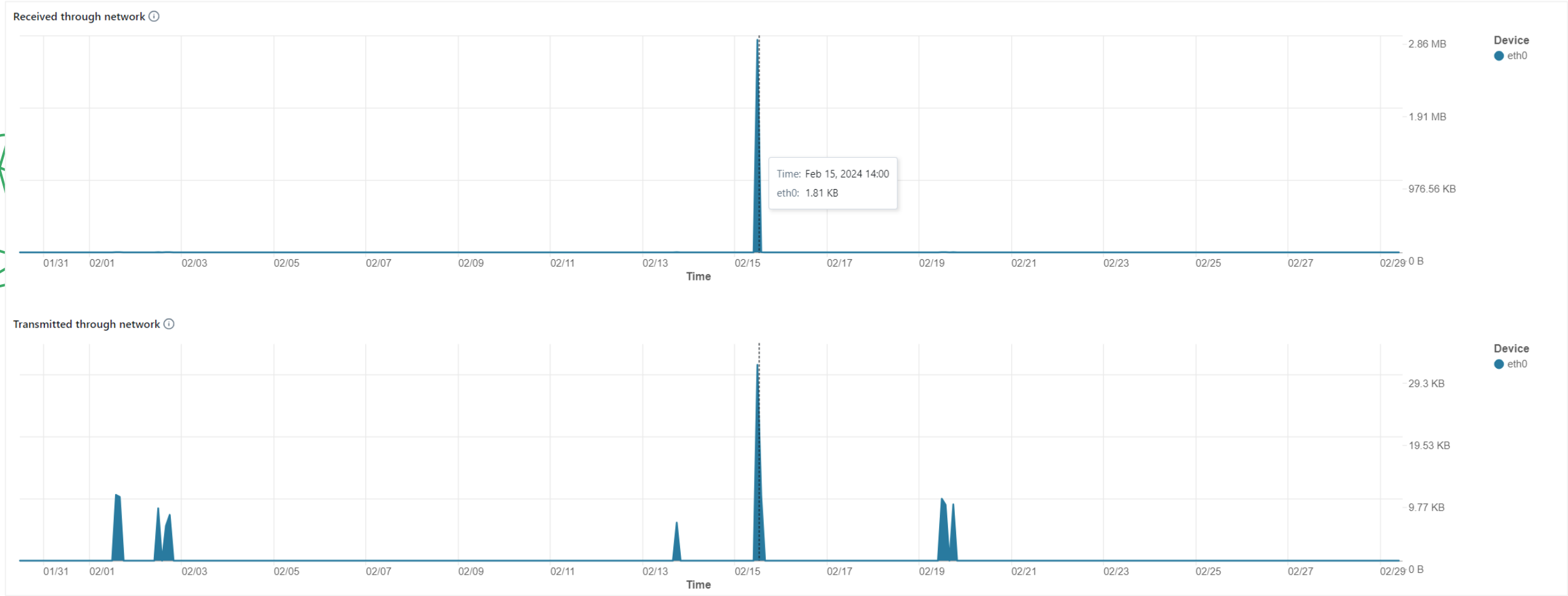
Hardware Metrics – CPU & Memory



Hardware Metrics - Memory



Hardware Metrics - Networking

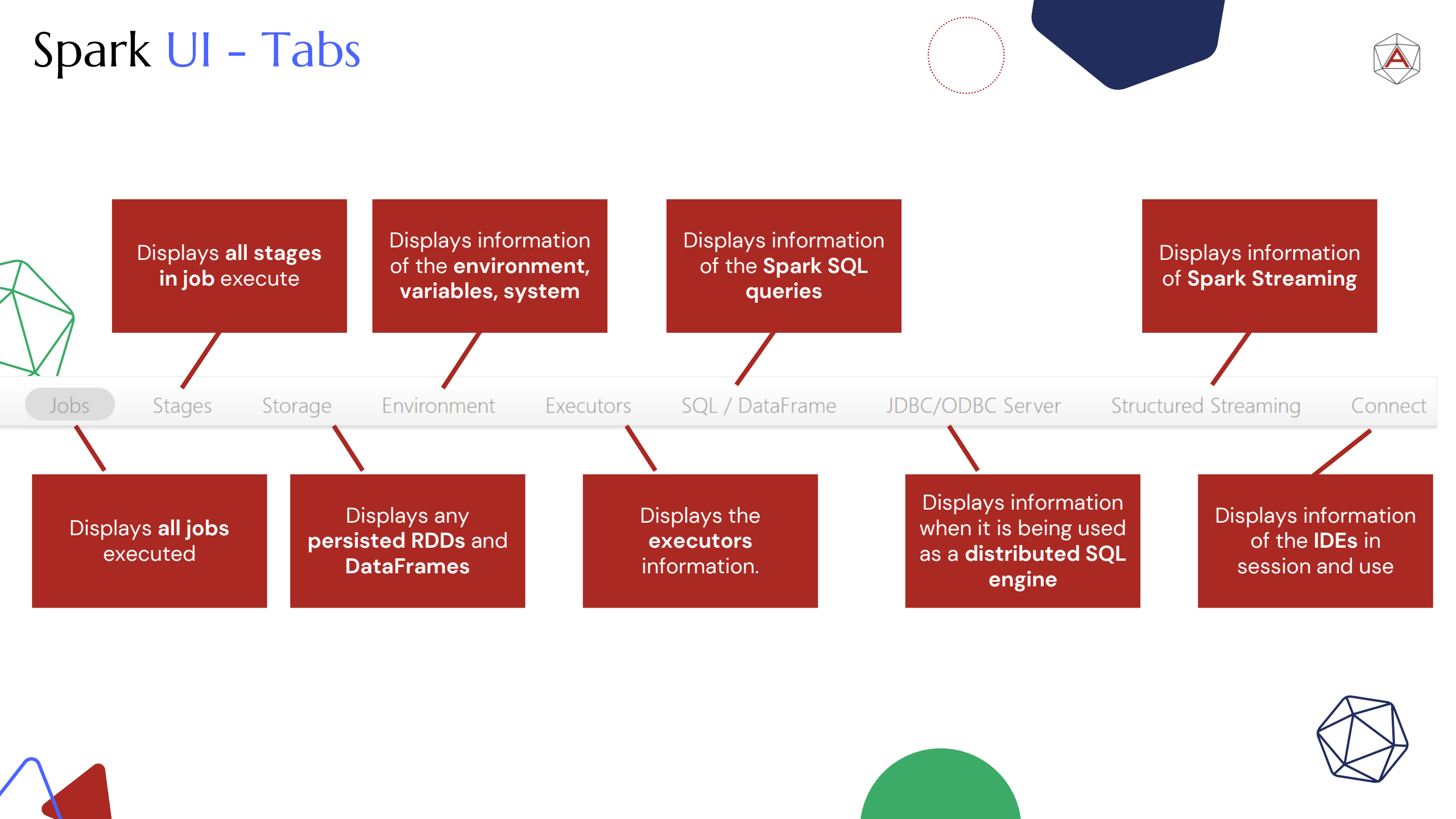


Spark UI

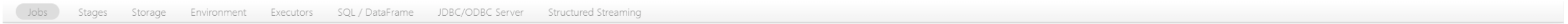
Spark UI

- Monitor Spark Application
- Insight Into Executions and Workload
- Debugging
- Displays queries, jobs, DAG, and query plans

Spark UI - Tabs



Spark UI - Jobs



Spark Jobs (?)

User: root
Total Uptime: 4.6 min
Scheduling Mode: FAIR
Completed Jobs: 11

▸ Event Timeline

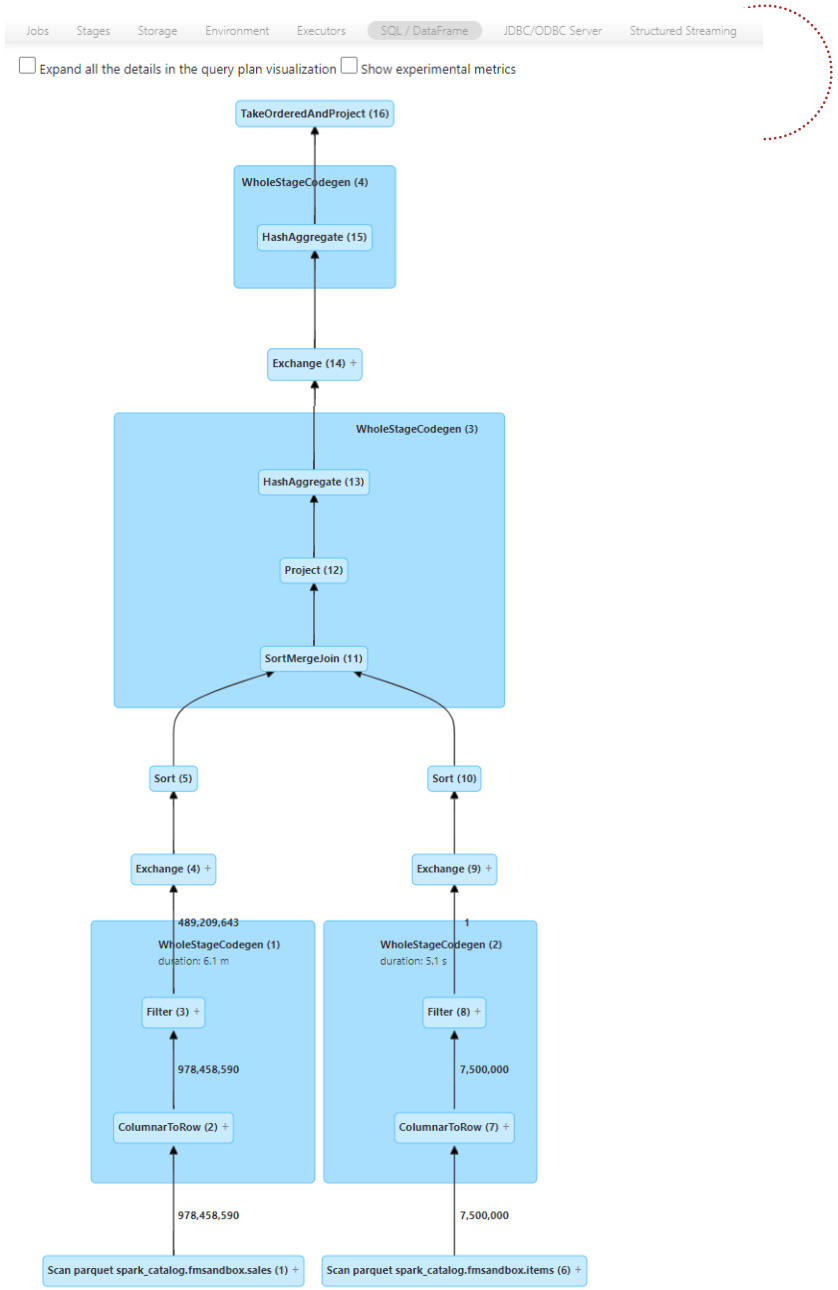
▾ Completed Jobs (11)

Page: 1

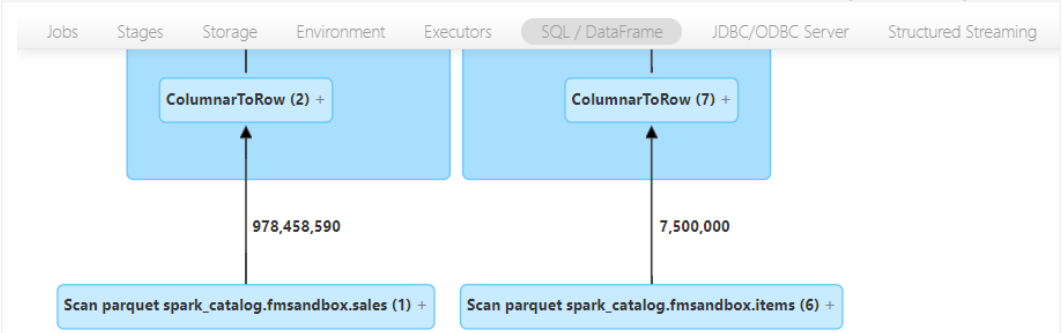
1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
10 (3147736733626348505_6685574054404766343_326aa3d54f44435ea5e7107c817fbe4b)	try: def ___databricks_percent_sql(): i... executeCollect at DatasetRefCache.scala:71	2023/02/02 15:54:58	0.3 s	1/1	4/4
9 (3147736733626348505_6685574054404766343_326aa3d54f44435ea5e7107c817fbe4b)	try: def ___databricks_percent_sql(): i... executeCollect at DatasetRefCache.scala:71	2023/02/02 15:54:58	0.3 s	1/1	4/4
8 (3147736733626348505_8306147359562026961_53e17d4711ef4d1c9cea51c4107e89a9)	# Drop "items" and "sales" delta table spark.sq... first at Snapshot.scala:238	2023/02/02 15:54:56	0.2 s	2/2 (1 skipped)	2/2 (4 skipped)
7 (3147736733626348505_8306147359562026961_53e17d4711ef4d1c9cea51c4107e89a9)	# Drop "items" and "sales" delta table spark.sq... collect at Checksum.scala:387	2023/02/02 15:54:56	0.3 s	2/2	5/5
6 (3147736733626348505_8306147359562026961_53e17d4711ef4d1c9cea51c4107e89a9)	# Drop "items" and "sales" delta table spark.sq...	2023/02/02 15:54:55	0 ms	0/0	0/0
5 (3147736733626348505_8306147359562026961_53e17d4711ef4d1c9cea51c4107e89a9)	# Drop "items" and "sales" delta table spark.sq... write at WriteIntoDeltaCommand.scala:70	2023/02/02 15:52:40	2.2 min	1/1	4/4

Spark UI - DAG


Jobs
Stages
Storage
Environment
Executors
SQL / DataFrame
JDBC/ODBC Server
Structured Streaming
Expand all the details in the query plan visualization
Show experimental metrics
TakeOrderedAndProject (16)
WholeStageCodegen (4)
HashAggregate (15)
Exchange (14)
WholeStageCodegen (3)
HashAggregate (13)
Project (12)
SortMergeJoin (11)
Sort (5)
Sort (10)
Exchange (4)
Exchange (9)
489,209,643
1
WholeStageCodegen (1)
duration: 6.1 m
WholeStageCodegen (2)
duration: 5.1 s
Filter (3)
Filter (8)
978,458,590
7,500,000
ColumnarToRow (2)
ColumnarToRow (7)
978,458,590
7,500,000
Scan parquet spark_catalog.fmsandbox.sales (1)
Scan parquet spark_catalog.fmsandbox.items (6)

Spark UI - Query Plan



▸ Text Execution Summary

▾ Details

```
== Physical Plan ==
TakeOrderedAndProject (16)
+- * HashAggregate (15)
   +- Exchange (14)
      +- * HashAggregate (13)
         +- * Project (12)
            +- * SortMergeJoin Inner (11)
               :- Sort (5)
               :  +- Exchange (4)
               :     +- * Filter (3)
               :        +- * ColumnarToRow (2)
               :           +- Scan parquet spark_catalog.fmsandbox.sales (1)
               +- Sort (10)
                  +- Exchange (9)
                     +- * Filter (8)
                        +- * ColumnarToRow (7)
                           +- Scan parquet spark_catalog.fmsandbox.items (6)


(1) Scan parquet spark_catalog.fmsandbox.sales
Output [3]: [itemID#1241, quantity#1242, saleDate#1243]
Batched: true
Location: PreparedDeltaFileIndex [dbfs:/user/hive/warehouse/fmsandbox.db/sales]
PushedFilters: [IsNotNull(itemID), EqualTo(itemID,4)]
ReadSchema: struct<itemID:int,quantity:int,saleDate:date>

(2) ColumnarToRow [codegen id : 1]
Input [3]: [itemID#1241, quantity#1242, saleDate#1243]

(3) Filter [codegen id : 1]
Input [3]: [itemID#1241, quantity#1242, saleDate#1243]
```

Demo: Cluster Monitoring

Spark UI

Metrics

Recap

Recap

- Small File Problem
- Optimization
  - Bin-packing
  - Data Skipping
  - Z-Ordering
- Liquid Clustering
- Performance Tuning
  - Execution Plans
  - Adaptive Query Execution
  - Query Hints
- Cluster Monitoring

ADVANCING ANALYTICS