DELTA AND DATABRICKS FOR THE DBA

WHO AM I

- 17+ Years Experience
- Data Engineering Consultant
- Intensive Software & Data Engineering Experience
- Microsoft AI MVP
- Public Speaker
- Community Organiser

X @annawykes   in @annawykes

Microsoft® MVP Most Valuable Professional

WHO AM I

- SQL Server DBA/Developer
- 20 years + experience
- Microsoft Data Platform MVP
- User Group Leader
- SQLBits Organiser

welldata

WHO AM I

- 9+ Years Experience
- Data Engineering Consultant
- Database Engineer
- Public Speaker
- Data Bristol Organiser
- Pydata Bristol Organiser

adatis
Part of Telefónica Tech

in james-c-yarrow

WHY?

AGENDA

- ACID Transactions
- Parquet/Delta Storage vs Storage Structures
- Partitioning Delta vs SQL
- Z-ordering vs Clustered Indexing and Column Store Indexing
- Delta History and Time Travel vs SQL Temporal Tables
- Delta Vacuum vs Data Retention
- Identity Columns, Primary & Foreign Keys
- Security (Unity Catalog) vs SQL Security
- Databricks Clusters vs SQL Engine

ACID TRANSACTIONS

ATOMICITY, CONSISTENCY, ISOLATION, AND DURABILITY

Atomicity :	The Transaction can be treated as a single unit that can succeed or fail
Consistency:	The data must be consistent before and after the Transaction
Isolation :	The Transaction is independent and can be attempted without interference
Durability :	A successful transaction persists if the system fails

ATOMICITY, CONSISTENCY, ISOLATION, AND DURABILITY

SQL Server uses a single write ahead transaction log per database.
Databrick gains the functionality of a log file from delta format.

	Atomicity	Consistency	Isolation	Durability
Parquet	✗	✗	✗	✗
Delta Format	✓	✓	✓	✓
Simple Recovery	✓	✓	✓	✓
Full Recovery	✓	✓	✓	✓

Isolation

SQL Server	Delta Format
Read Uncommitted	Write Serializable
Repeatable Read	✗
Snapshot	✗
Serializable	Serializable

ATOMICITY, CONSISTENCY, ISOLATION, AND DURABILITY

Databricks uses Delta format to get ACID properties

Delta format is open source!

STORAGE STRUCTURES

HOW DO WE STORE OUR DATA TO DISK?

We have to persist data to disk

- Data stored in files
- SQL Server exclusively access the files

	SQL Server	Delta Format	Parquet
File Structure	*.MDF, *.NDF	*.parquet	*.parquet
Log File Structure	*.LDF	/_delta_log /000000.json	

HOW DO WE STORE OUR DATA TO DISK?

	SQL Server
File Structure	MDF, NDF
Log File Structure	LDF

- Many tables to a file

[Person].[Address]
[Person].[Person]
[Production].[Product]
[Sales].[Customer]

MDF

LDF

- We can have multiple tables in the same file

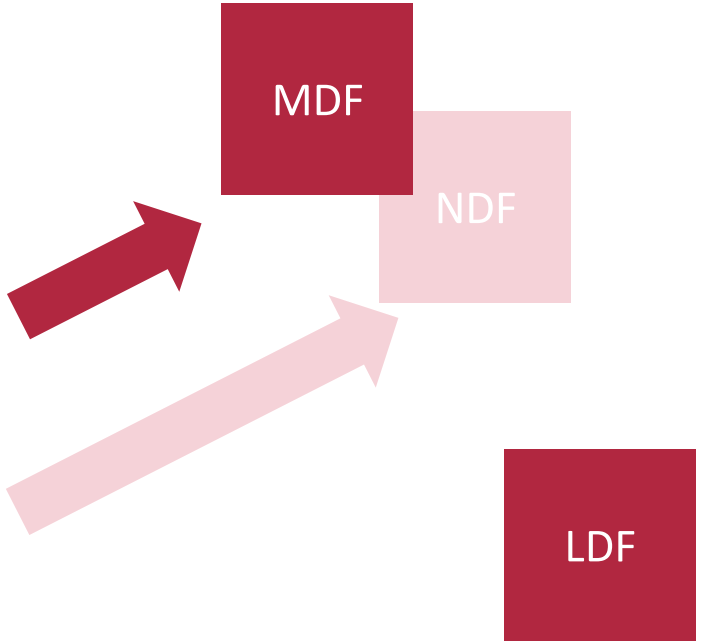
HOW DO WE STORE OUR DATA TO DISK?

	SQL Server
File Structure	MDF, NDF
Log File Structure	LDF

- Many tables to a files
- Multiple files

[Person].[Address]
[Person].[Person]

[Production].[Product]
[Sales].[Customer]

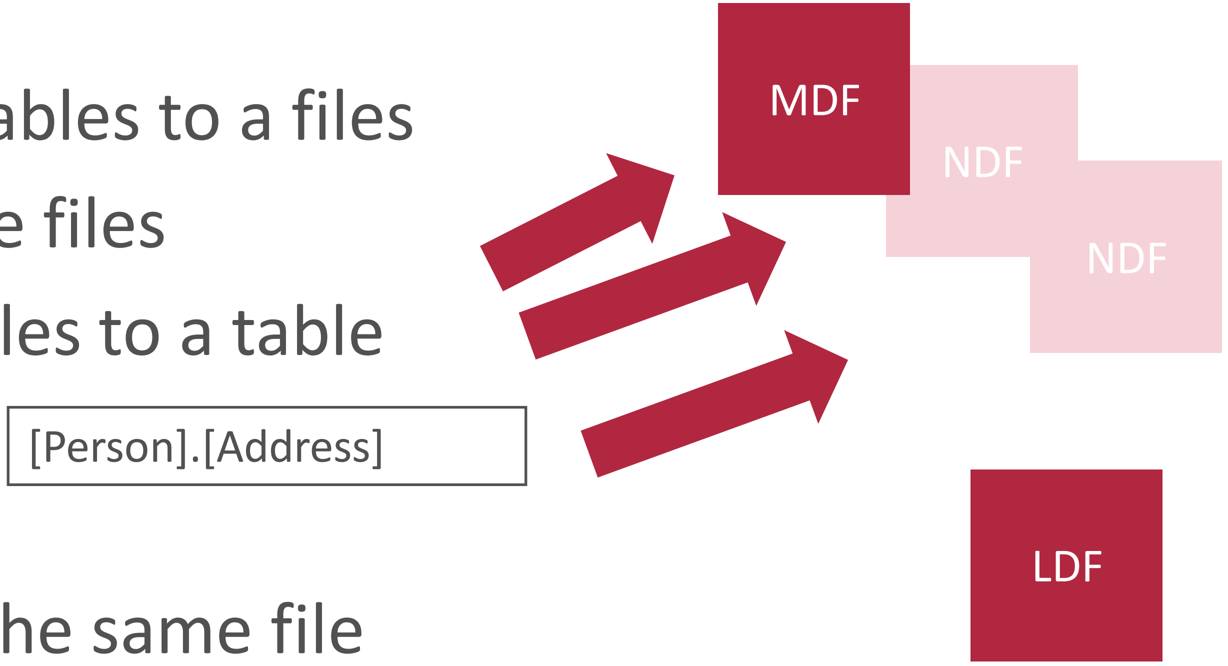


- We can have multiple tables in the same file
- We can have tables in different files groups

HOW DO WE STORE OUR DATA TO DISK?

	SQL Server
File Structure	MDF, NDF
Log File Structure	LDF

- Many tables to a files
- Multiple files
- Many files to a table



- We can have multiple tables in the same file
- We can have tables in different files groups
- We can have one table per file group
  unless the table has been partitioned
- If additional log files are used, they are still shared

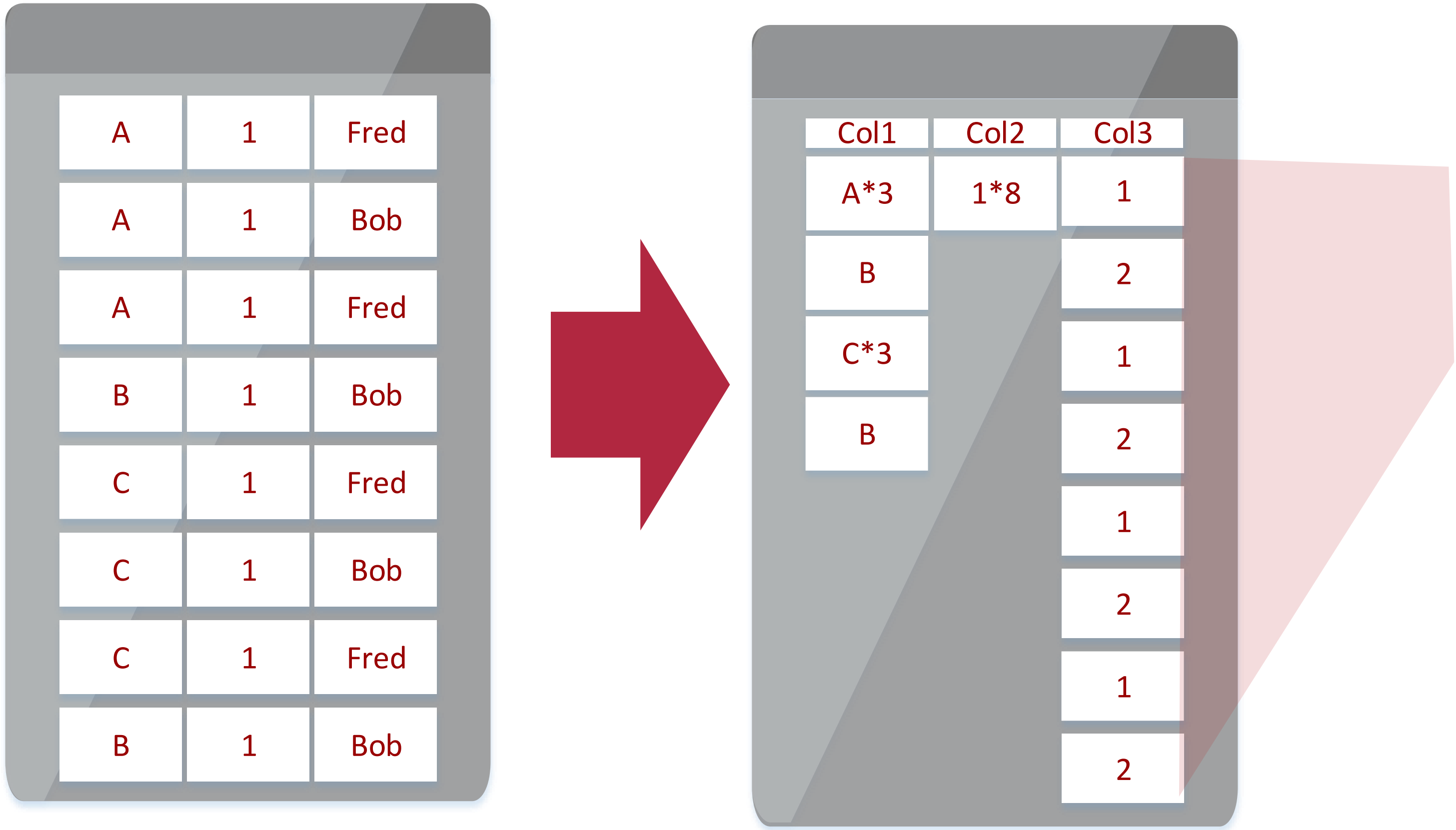
ADVANCING ANALYTICS

PARQUET/DELTA STORAGE

WHAT IS PARQUET?

- Open source, column-oriented data **file format**

Metadata – Schema, Structure, Dictionaries



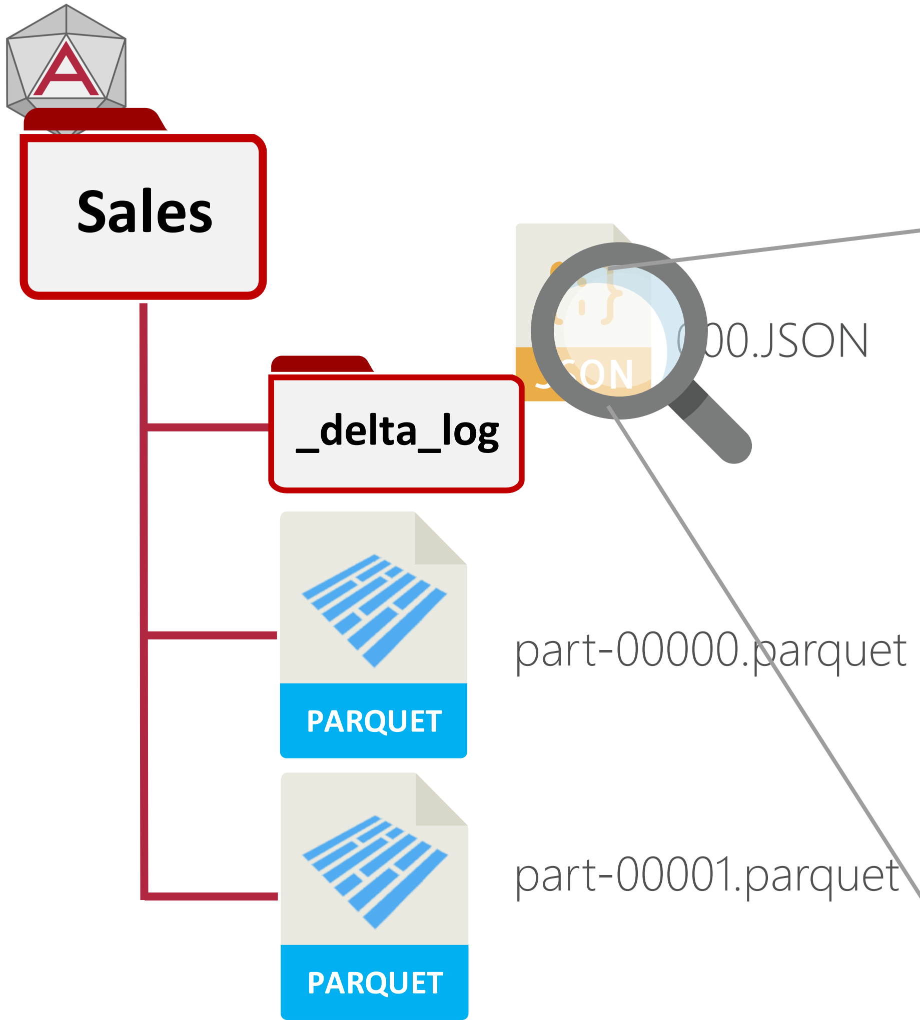
CSV File

Parquet File

Parquet works in a **very similar way** to **SQL ColumnStore** – it's a **ColumnStore compression** format

DELTA FEATURES

- ACID Transactions
- Scalable Metadata
- Time Travelling
- Audit History
- Batch & Streaming Support
- Schema Enforcement
- Schema Evolution
- Familiar SQL Commands
- Open Format
- Compatible with Spark




```
…
{
   "add": {
      "path": "part-00000.parquet",
      "partitionValues": {},
      "size": 255520,
      "modificationTime": 1572823237000,
      "dataChange": true,
      "stats":[…]},
 "add": {
      "path": "part-00001.parquet",
      "partitionValues": {},
      "size": 242520,
      "modificationTime": 1572823237000,
      "dataChange": true,
      "stats": […]}
}
```

SQL STORAGE STRUCTURES

STORAGE STRUCTURES

Within these files, SQL Server has three types of storage structures that are Heap, clustered and column store indexes

- Heap
- Clustered Index
- Column store Index

This are where the data is physically stored on Disk

STORAGE STRUCTURES

SQL Server allow us to have a transactional copy of the data as a non-clustered index that can optimise reads.

In contrast, we would create copy with a subset in Databricks!

	SQL Server	Delta	Parquet
Data Structure	Heap, B Tree, Clustered Column Store, Order Clustered Column Store	Columnal	Columnal
Secondary Data Copies	Non Clustered Index, Non Clustered Column Store		

PARTITIONING IN DELTA VS SQL SERVER

WHAT PROBLEM ARE WE TRYING TO SOLVE?

- Chunking data into logical buckets for reading and writing without having to work with the entire dataset
- Large Data Set can have issues at scale with data manipulation and storage size

PARTITIONING IN SQL SERVER

PARTITION VIEW AND TABLE PARTITION

SQL Server has two type of partitioning.

- Partition Table
- Partition Views

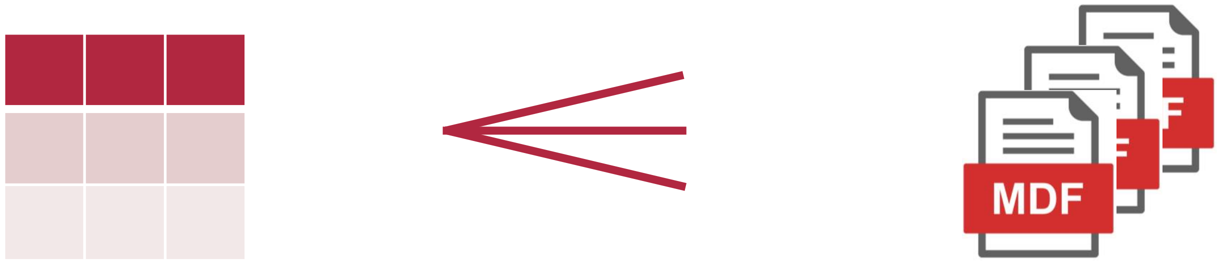
Partition Tables are tables partitioned on a key across multiple files.

Partition views can be place over tables to partition different keys.

PARTITION VIEW AND TABLE PARTITION

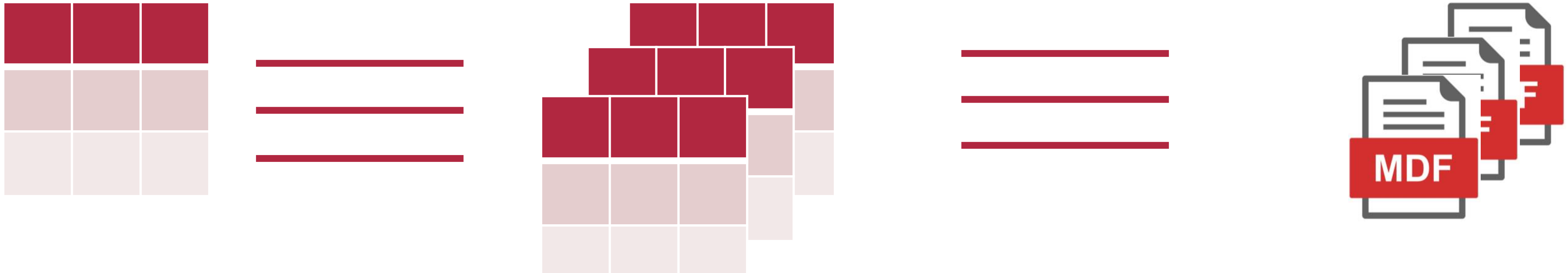
View | Table | File Structure

T a b l e V i e w



One Table

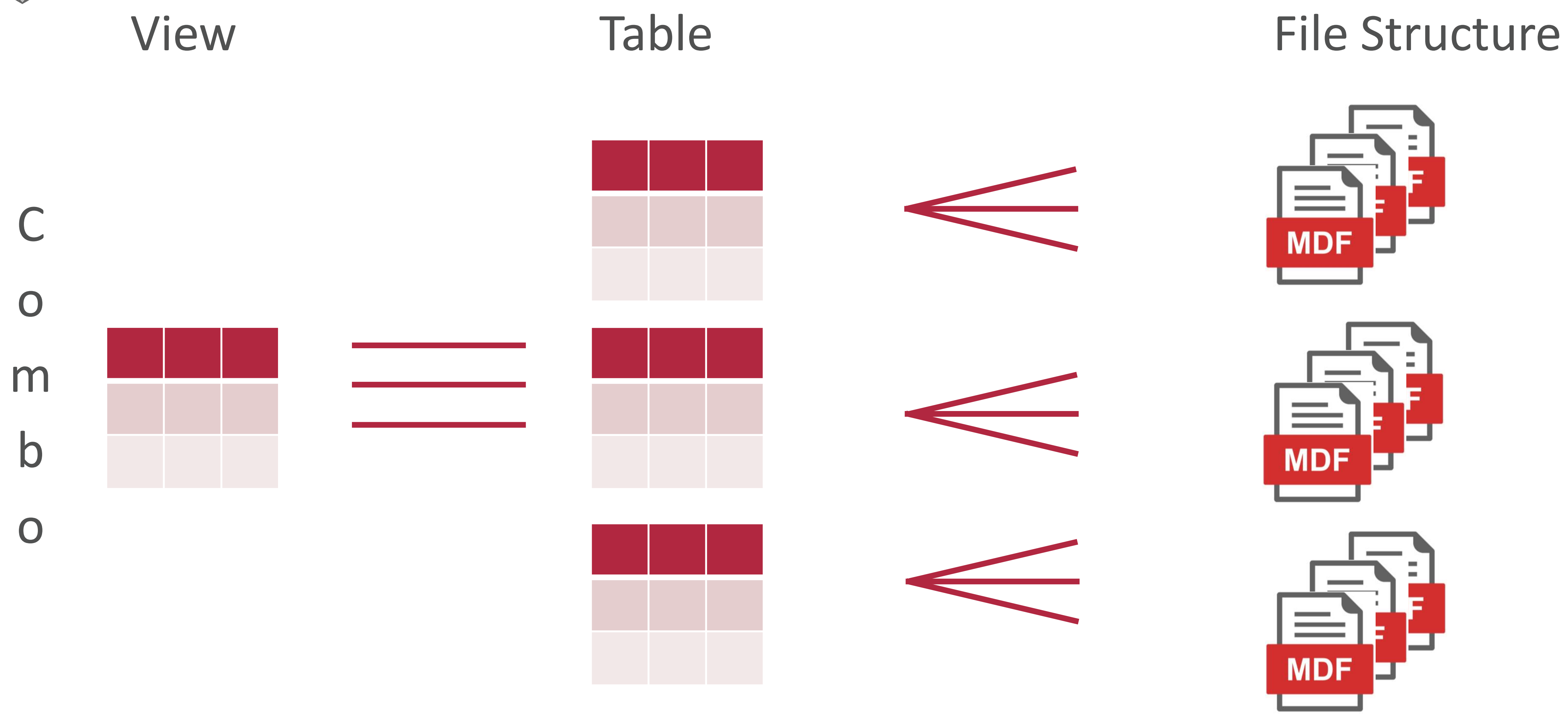
One Table to multiple Files



One View to multiple Tables

Each table will have its own file group

PARTITION VIEW AND TABLE PARTITION



PARTITIONING IN DELTA

DELTA PARTITIONING

Delta data is stored as multiple small files when loaded.

Delta Partitioning divides those data file into useful slices so they can be retrieved quickly and effectively.

	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

ADVANCING ANALYTICS

WRITING DELTA PARTITION

```
# Define DataFrame over root entity folder
( df.write
    .format("delta")
    .mode("overwrite")
    .partitionBy("<ColumnName>")
    .save("deltaFilePath")
)
```

To write delta partition you use the **.partitionBy()** command

	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

- Storage
- Partitions

Z-ORDERING VS CLUSTERED INDEXING AND COLUMN STORE INDEXING

WHAT PROBLEM ARE WE TRYING TO SOLVE?

- Improve performance when reading data
- We don’t want to have to scan the entire dataset to find rows that match our query
- Ordering data on a granular row by row level for more optimal reading

Z-ORDERING IN DELTA

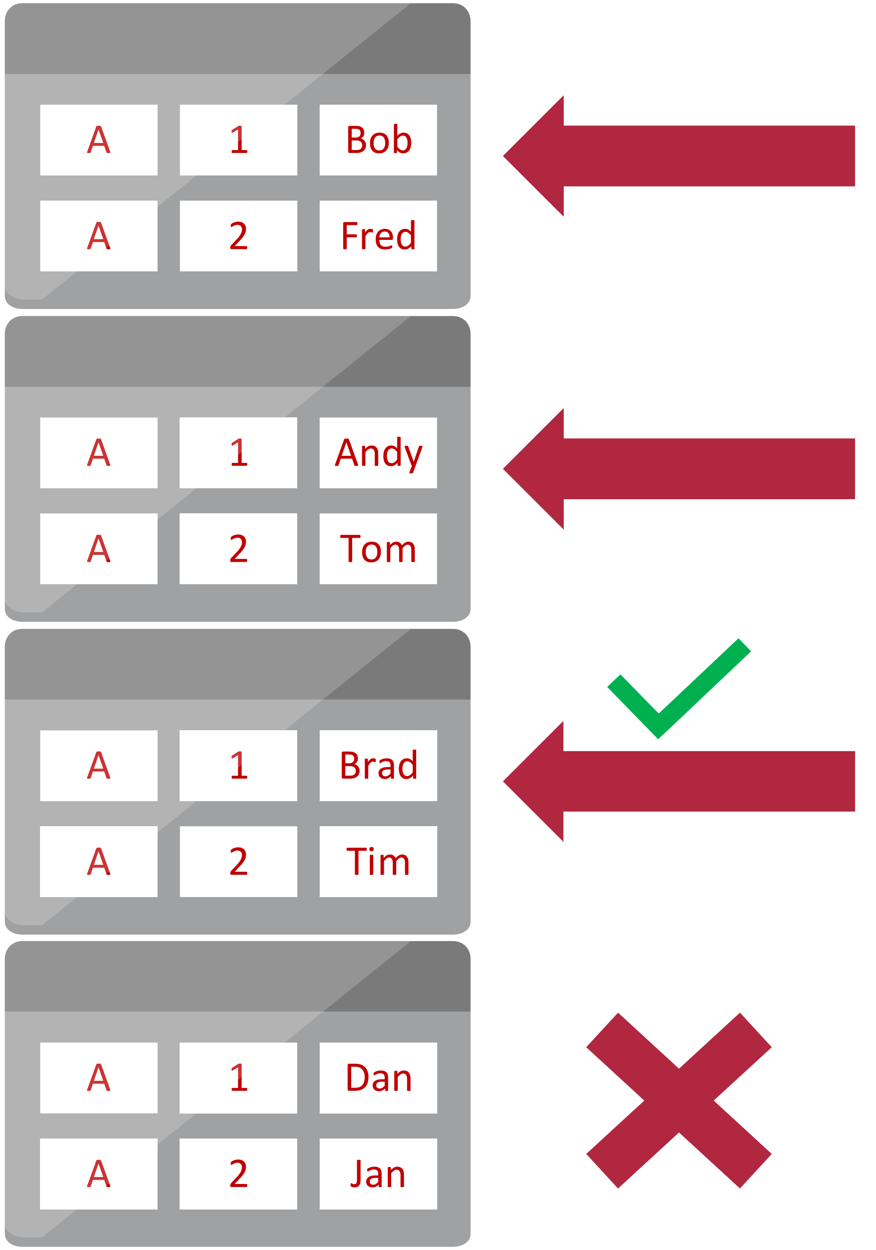
Z-ORDERING

"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

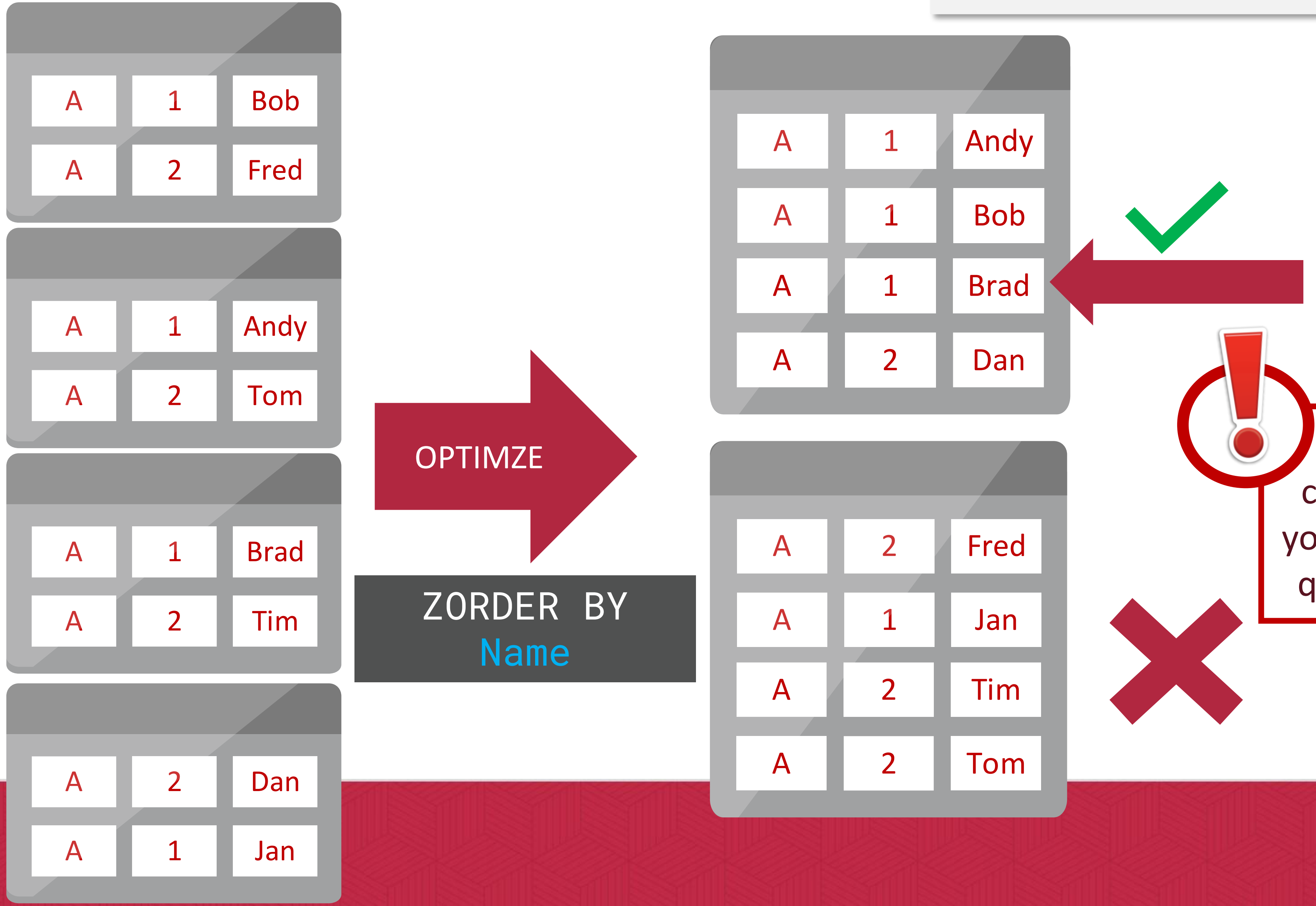



```sql
SELECT count(*) FROM Employees
WHERE Name = 'Brad'
```

- 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’



Z-Order as similar to a clustered index - organising your data on disk to maximise queries for specific columns

DEMO

- Z-ordering

INDEXING IN SQL SERVER

Z_ORDERING IS INDEX MAINTAINENCE

The order of the data is set by the Clustered Index

Index Maintenance

- Rebuilds
- Reorganization

DELTA HISTORY AND TIME TRAVEL VS SQL TEMPORAL TABLES

WHAT PROBLEM ARE WE TRYING TO SOLVE?

- Keeping a history of changes for data
- Being able to query against point in time

DELTA HISTORY AND TIME TRAVEL

DELTA TIME TRAVEL

- Delta Lake allow you to query older version/snapshots of a Delta table using Time Travel.
- Delta Lake stored a 30-days version history by default.
- Useful for debugging, auditing, rolling back for data quality or to reproduce experiments.
- Can query an older versions using Python or SQL syntax.

DELTA TIME TRAVEL

```sql
-- SQL syntax
SELECT count(*)
FROM
<databaseName>.<tableName>
VERSION AS OF 0
```

```python
-- DataFrameReader options
( spark.read
        .format("delta")

.option("versionAsOf", 0)
```

	version	timestamp
4	4	2021-08-09T10:48:28.000+0000
5	3	2021-08-09T10:47:53.000+0000
6	2	2021-08-09T10:45:52.000+0000
7	1	2021-08-09T10:45:49.000+0000
8	0	2021-08-09T10:45:29.000+0000

DEMO

- History
- Versions

SQL TEMPORAL TABLES

TEMPORAL TABLES

- Allow you to keep history of data changes
- Used for auditing
- Retention

CREATING A TEMPORAL TABLE

```sql

CREATE TABLE dbo.TmprlEmployee
(
  [EmployeeID] int IDENTITY NOT NULL PRIMARY KEY CLUSTERED
  , [Name] nvarchar(100) NOT NULL
  , [Position] varchar(100) NOT NULL
  , [Department] varchar(100) NOT NULL
  , [ValidFrom] datetime2 GENERATED ALWAYS AS ROW START
  , [ValidTo] datetime2 GENERATED ALWAYS AS ROW END
  , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
 )
 -- Set the history table, this is automatically created
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.TmprlEmployeeHistory,
     HISTORY_RETENTION_PERIOD = 6 MONTHS ));
```

INSERTING INTO A TEMPORAL TABLE

```sql
INSERT INTO TmprlEmployee
([Name], Position, Department)
VALUES
('Fred Flintstone'   ,'Team Leader', 'Cave Building'),
('Barney Rubble'     ,'Assistant',   'Cave Building')
```

INSERTING INTO A TEMPORAL TABLE

[6]

```sql
-- We can see the new record
SELECT * FROM TmprlEmployee
-- but there is nothing in the history table because currently nothing has changed
SELECT * FROM TmprlEmployeeHistory
```

(2 rows affected)

(0 rows affected)

Total execution time: 00:00:00.016

	EmployeeID	Name	Position	Department	ValidFrom	ValidTo
1	1	Fred Flintstone	Team Leader	Cave Building	2023-09-29 18:08:40.5018095	9999-12-31 23:59:59.9999999
2	2	Barney Rubble	Assistant	Cave Building	2023-09-29 18:08:40.5018095	9999-12-31 23:59:59.9999999

UPDATING A TEMPORAL TABLE

```sql
UPDATE TmprlEmployee
SET Position = 'Manager'
WHERE [EmployeeID] = 1
```

UPDATING A TEMPORAL TABLE

```sql
-- We can see the new record
SELECT * FROM TmprlEmployee
-- but there is nothing in the history table because currently nothing has changed
SELECT * FROM TmprlEmployeeHistory
```

(2 rows affected)

(1 row affected)

Total execution time: 00:00:00.013

	EmployeeID	Name	Position	Department	ValidFrom	ValidTo
1	1	Fred Flintstone	Manager	Cave Building	2023-10-01 12:54:53.6886117	9999-12-31 23:59:59.9999999
2	2	Barney Rubble	Assistant	Cave Building	2023-09-29 18:08:40.5018095	9999-12-31 23:59:59.9999999

	EmployeeID	Name	Position	Department	ValidFrom	ValidTo
1	1	Fred Flintstone	Team Leader	Cave Building	2023-09-29 18:08:40.5018095	2023-10-01 12:54:53.6886117

DELETING FROM A TEMPORAL TABLE

```sql
-- Now what happens if we remove a record
DELETE FROM TmprlEmployee
WHERE employeeid = 2
```

DELETING FROM A TEMPORAL TABLE

```
-- the current table is showing the new from date
SELECT * FROM TmprlEmployee
-- and the history table is showing the old record
SELECT * FROM TmprlEmployeeHistory
```

(1 row affected)

(2 rows affected)

Total execution time: 00:00:00.012

	EmployeeID	Name	Position	Department	ValidFrom	ValidTo
1	1	Fred Flintstone	Manager	Cave Building	2023-10-01 12:54:53.6886117	9999-12-31 23:59:59.9999999

	EmployeeID	Name	Position	Department	ValidFrom	ValidTo
1	1	Fred Flintstone	Team Leader	Cave Building	2023-09-29 18:08:40.5018095	2023-10-01 12:54:53.6886117
2	2	Barney Rubble	Assistant	Cave Building	2023-09-29 18:08:40.5018095	2023-10-01 12:56:35.2509339

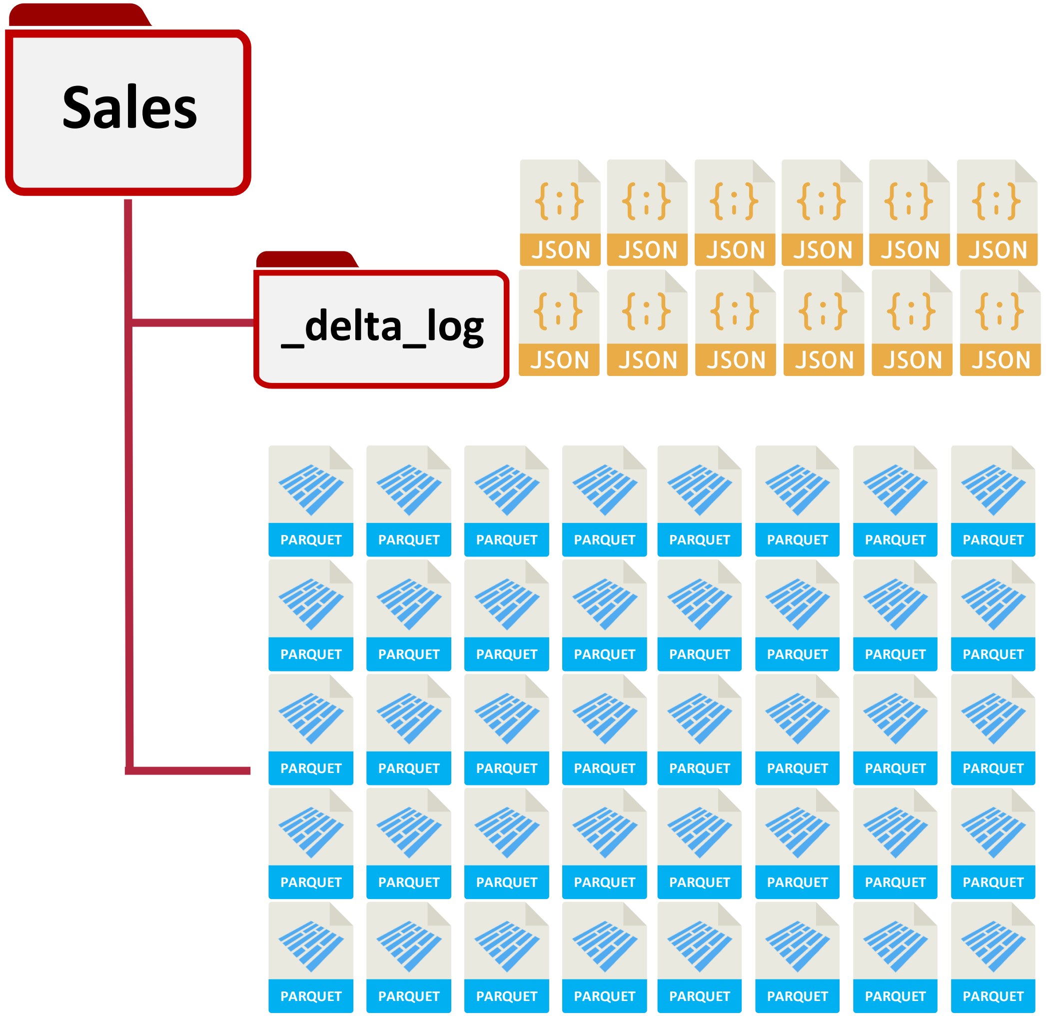
DELTA VACUUM VS DATA RETENTION

WHAT PROBLEM ARE WE TRYING TO SOLVE?

- Removing historical information that is no longer required

DELTA 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

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)
```

DEMO

- Vacuum

DATA RETENTION

DATA RETENTION

- Audit logs
- Change Data Capture (CDC)
- Temporal Tables

PRIMARY & FOREIGN KEYS

WHAT PROBLEM ARE WE TRYING TO SOLVE?

- How can we keep referential integrity?
- How can we join data together across data?

DELTA PRIMARY & FOREIGN KEYS

WHAT IS DATABRICKS UNITY CATALOG?

Unity Catalog provides centralized access control, auditing, lineage, and data discovery capabilities across Databricks workspaces.

- Define once, secure everywhere
- Standards-compliant security model
- Built-in auditing and lineage
- Data discovery
- System tables

databricks

UNITY CATALOG

DELTA CONSTRAINTS – UNITY CATALOG

Delta Table
Primary Keys
Foreign Keys
Identity Columns

databricks

UNITY CATALOG

Governance Layer

DELTA CONSTRAINTS – DATABRICKS ONLY

- **Primary Keys** and **Foreign Keys**
- **Identity Columns** (Surrogate Keys)

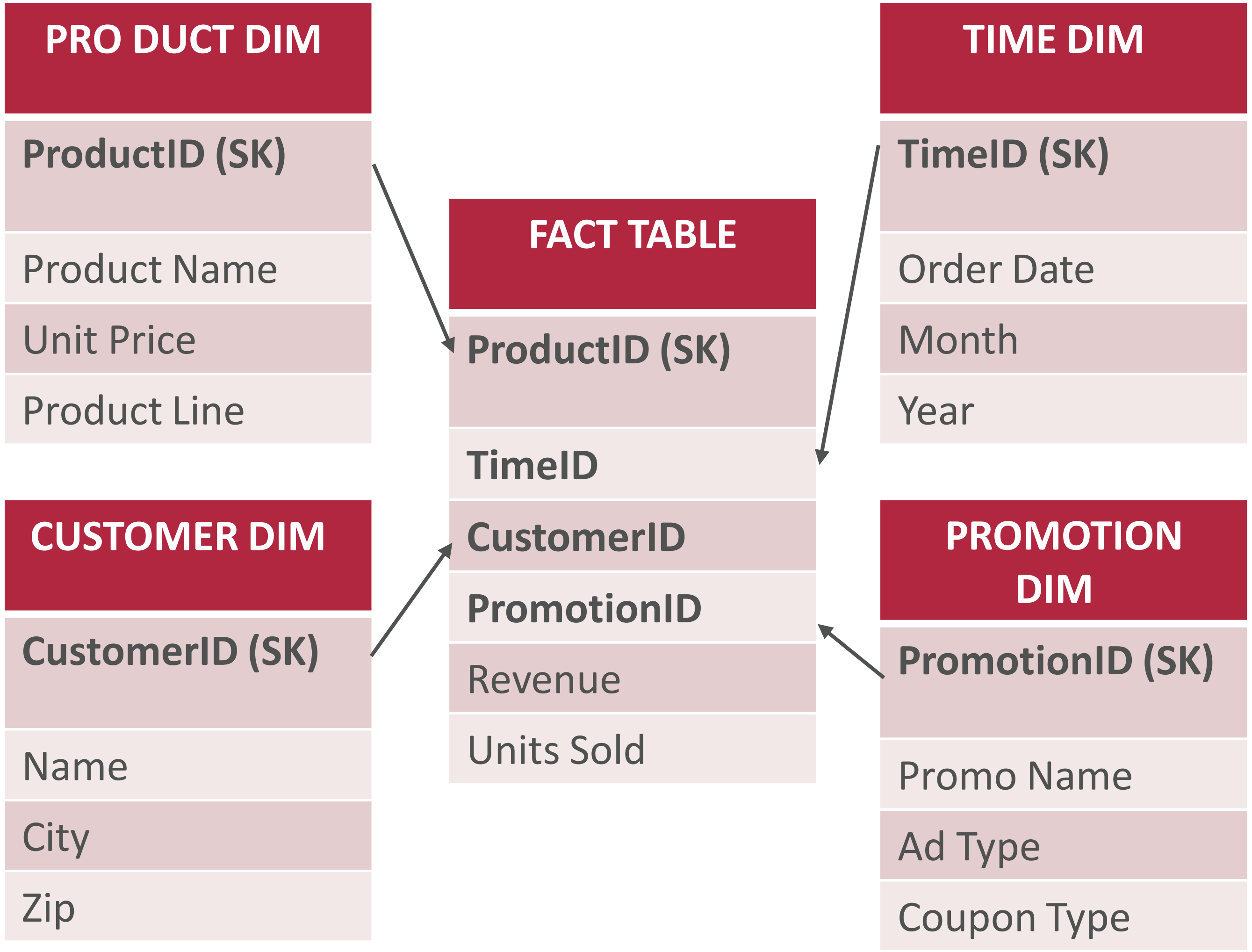
```
# Primary Key
CREATE TABLE <tableName> (
  <col1> INTEGER NOT NULL,
  <col2> INTEGER NOT NULL,
  CONSTRAINT <name> PRIMARY KEY(<col1>, <col2>)
);
```

```
# Foreign Key
CREATE TABLE <tableName> (
  <pkCol> INTEGER NOT NULL PRIMARY KEY,
  <col1> INTEGER,
  <col2> INTEGER,
  CONSTRAINT <name> FOREIGN KEY(<col1>, <col2>)
  REFERENCES <refName>
 );
```

```
# Identity Columns
CREATE OR REPLACE TABLE <tableName>  (
  <pkCol> BIGINT GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1) PRIMARY KEY,
  <col1>,
  <col2>
);
```

IDENTITIES

- Traditionally used when created SK (Surrogate Keys) for Star Schemas
- This was long awaited, but tricky for Databricks to implement due to the nature of distributed compute (unlike SQL Server)
- With Databricks Runtime 10.4 LTS we finally had identity columns



SQL PRIMARY & FOREIGN KEYS

PRIMARY AND FOREIGN KEYS

A primary key a constraint that enforces uniqueness, This can be a single column or multiple columns.

A foreign Key is constraint that is linked to a primary key constraint.

This link enforces that the value in the foreign key must exist in the primary key.

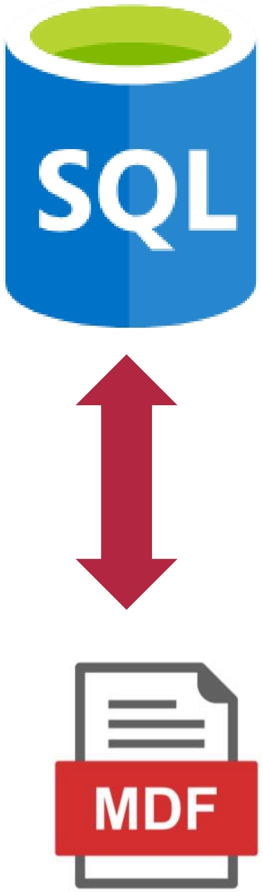
DATABRICKS SECURITY VS SQL SECURITY

SQL SERVER SECURITY

SQL SERVER SECURITY

- SQL Server has an object level security model.
- Data must be accessed via the SQL Server.
- SQL Server holds a file lock at all time.

We have AD/AAD with users, groups, Manage Identities and SQL Auth!



Application hold file locks at all time

Application hold file all time

DATABRICKS (UC) SECURITY

PREVIOUSLY

- Secured Data on ADLS instance via Access Control Lists (ACLs)

- Couldn't sync AD (Active Directory) groups to Databricks

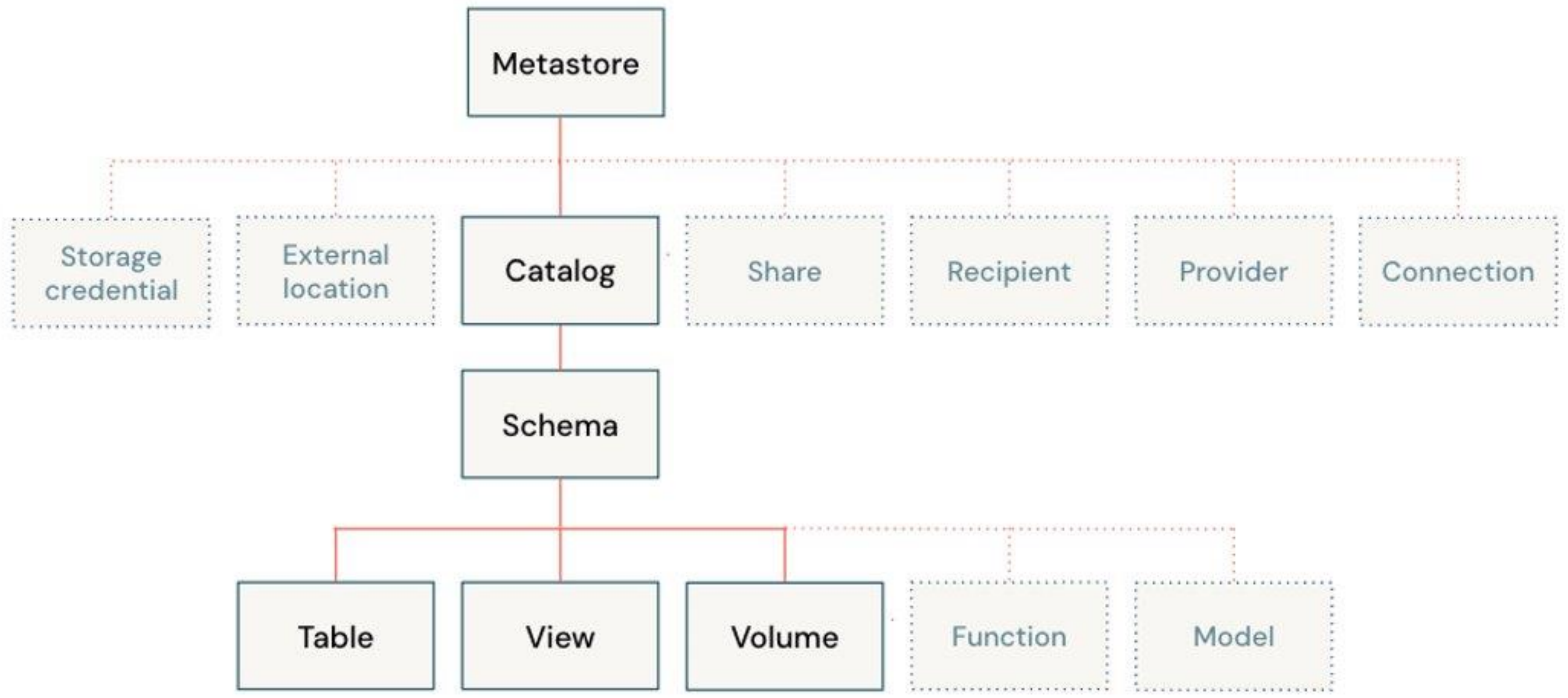
SECURITY IN UNITY CATLOG

Unity Catalog gives you the ability to choose between centralized and distributed governance models:

- In the centralized governance model, your governance administrators are owners of the metastore and can take ownership of any object and grant and revoke permissions.

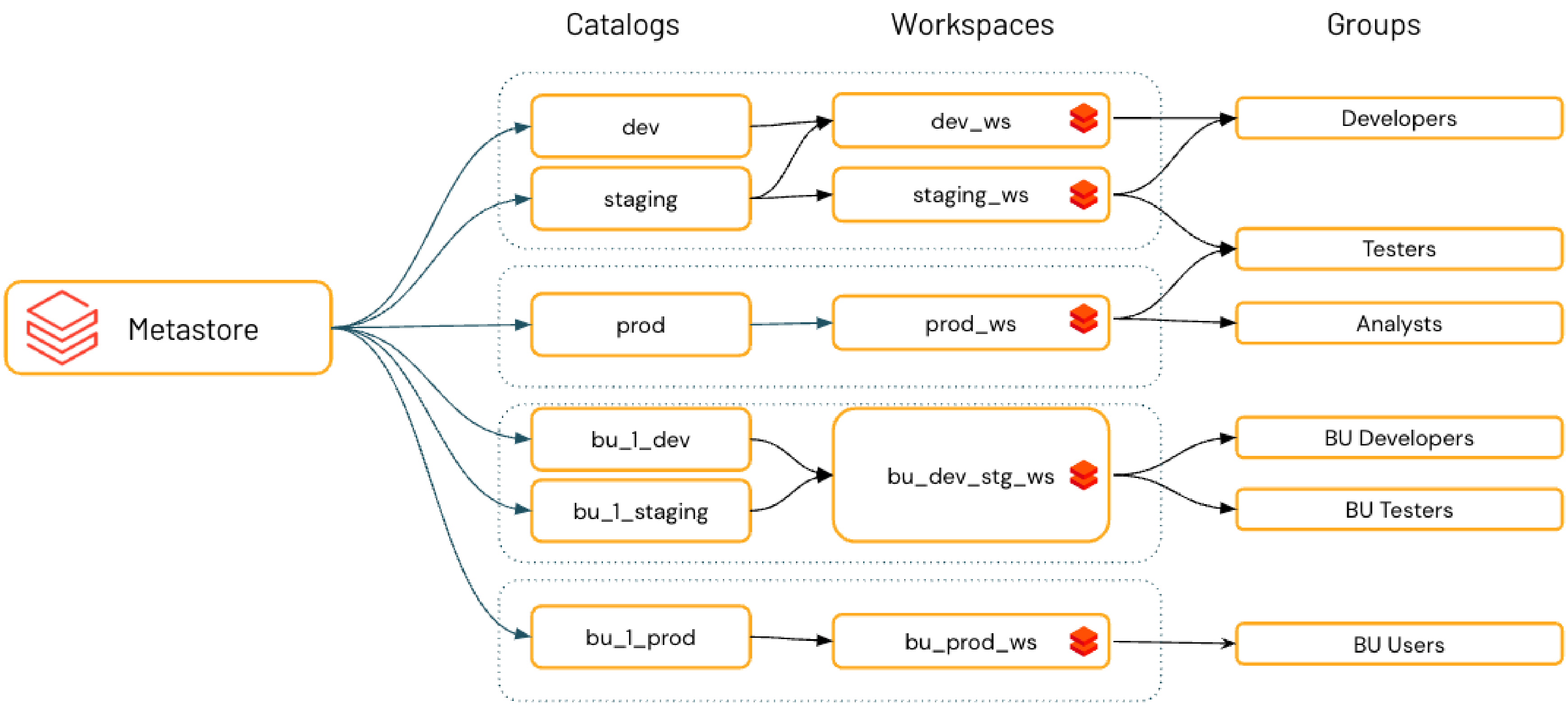
- In a distributed governance model, the catalog or a set of catalog’s is the data domain

SECURITY IN UNITY CATLOG



ADVANCING ANALYTICS

SECURITY IN UNITY CATLOG



STORAGE AND COMPUTE

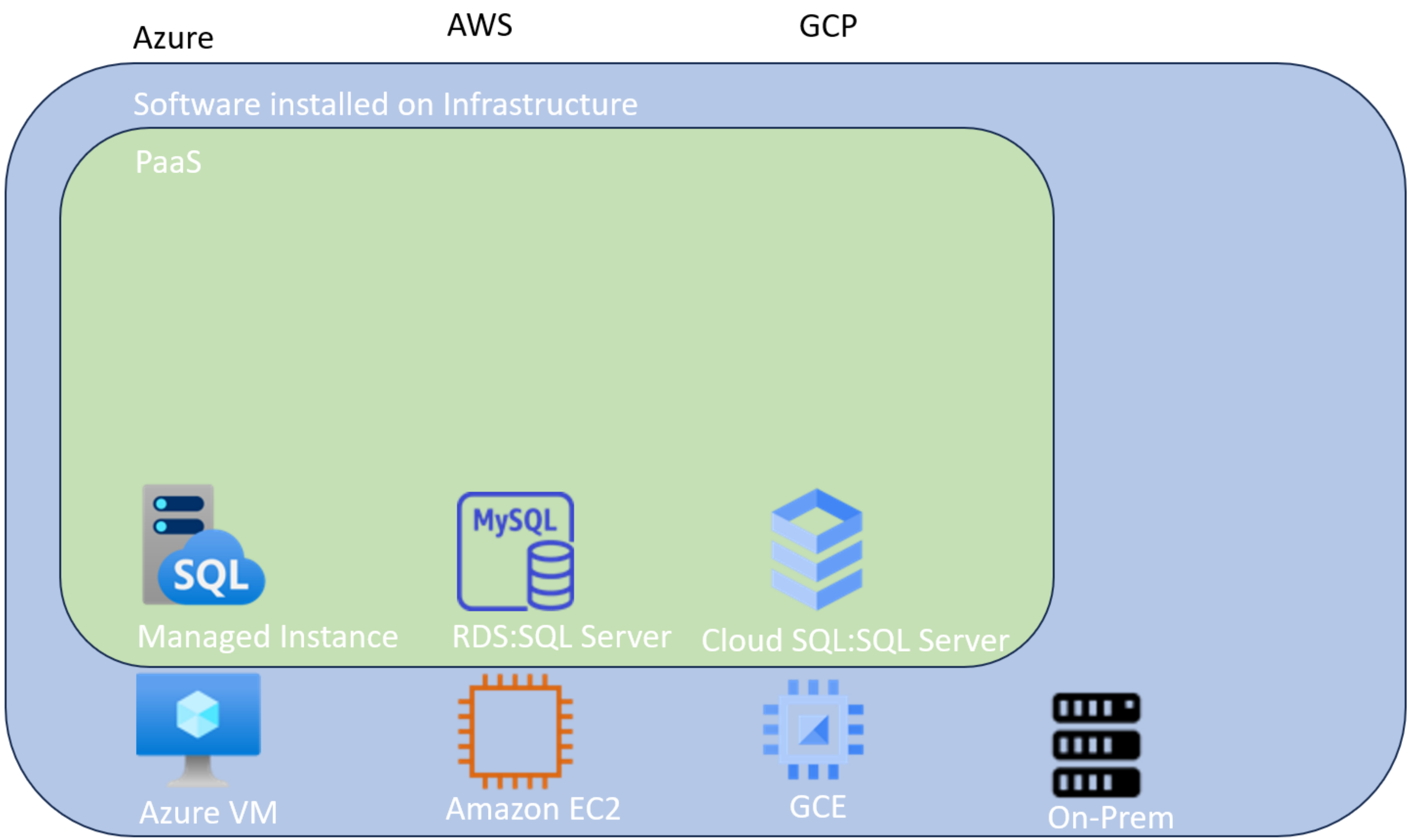
WHAT PROBLEM ARE WE TRYING TO SOLVE?

- What compute engine are we using?

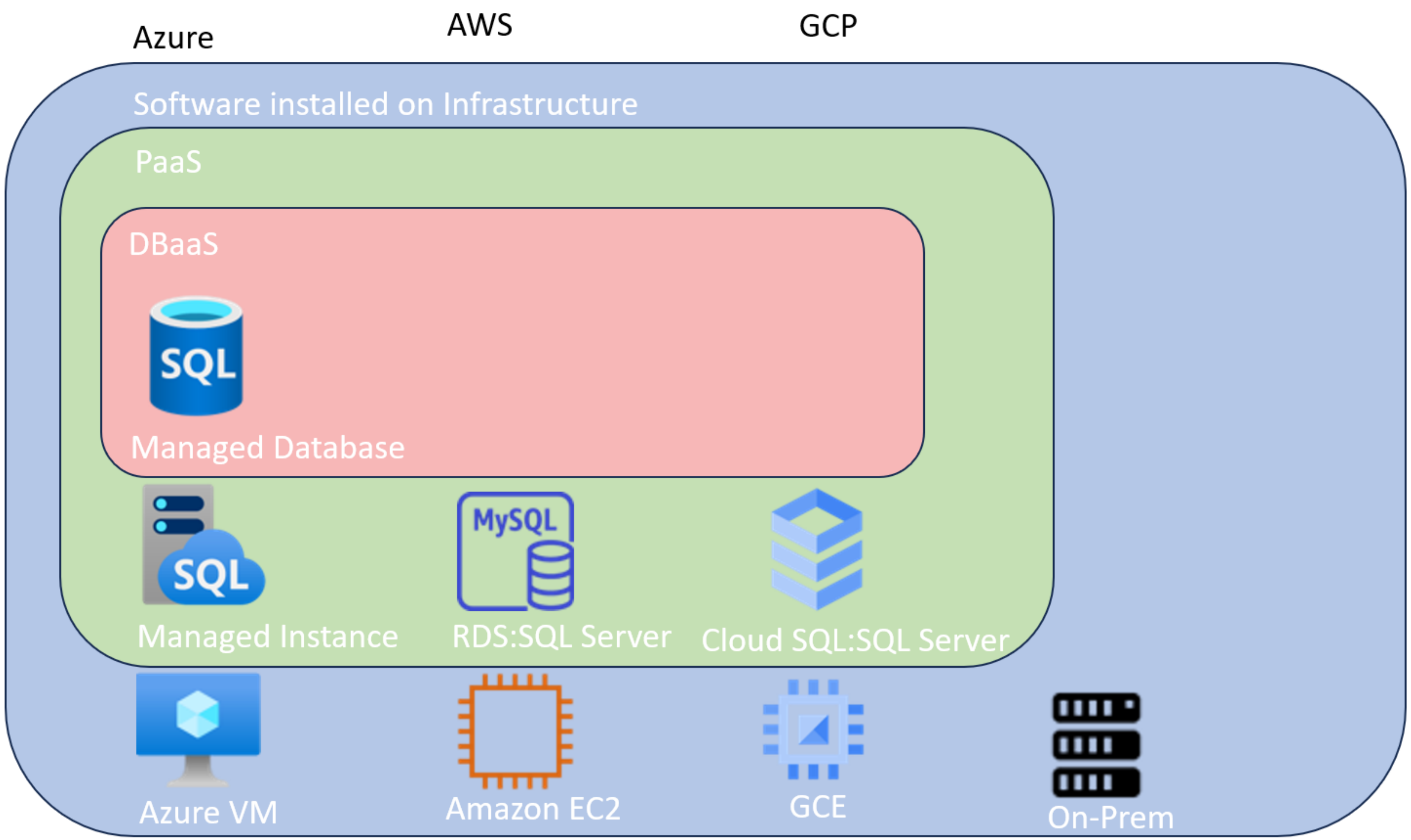
SQL SERVER

SQL SERVER AND IT DEPLOYMENT

SQL SERVER AND IT DEPLOYMENT



SQL SERVER AND IT DEPLOYMENT

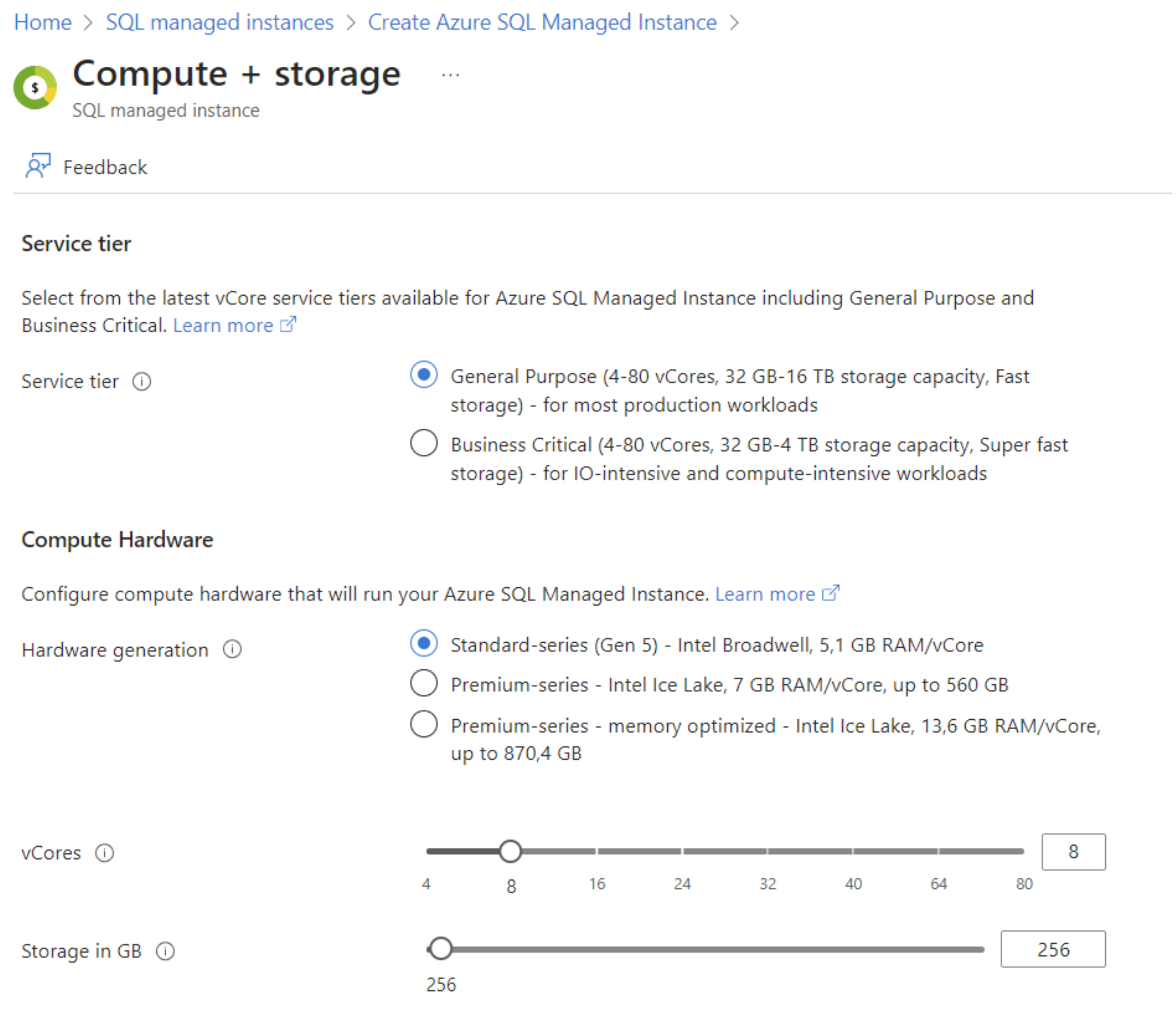


ADVANCING ANALYTICS

SQL SERVER IN AZURE

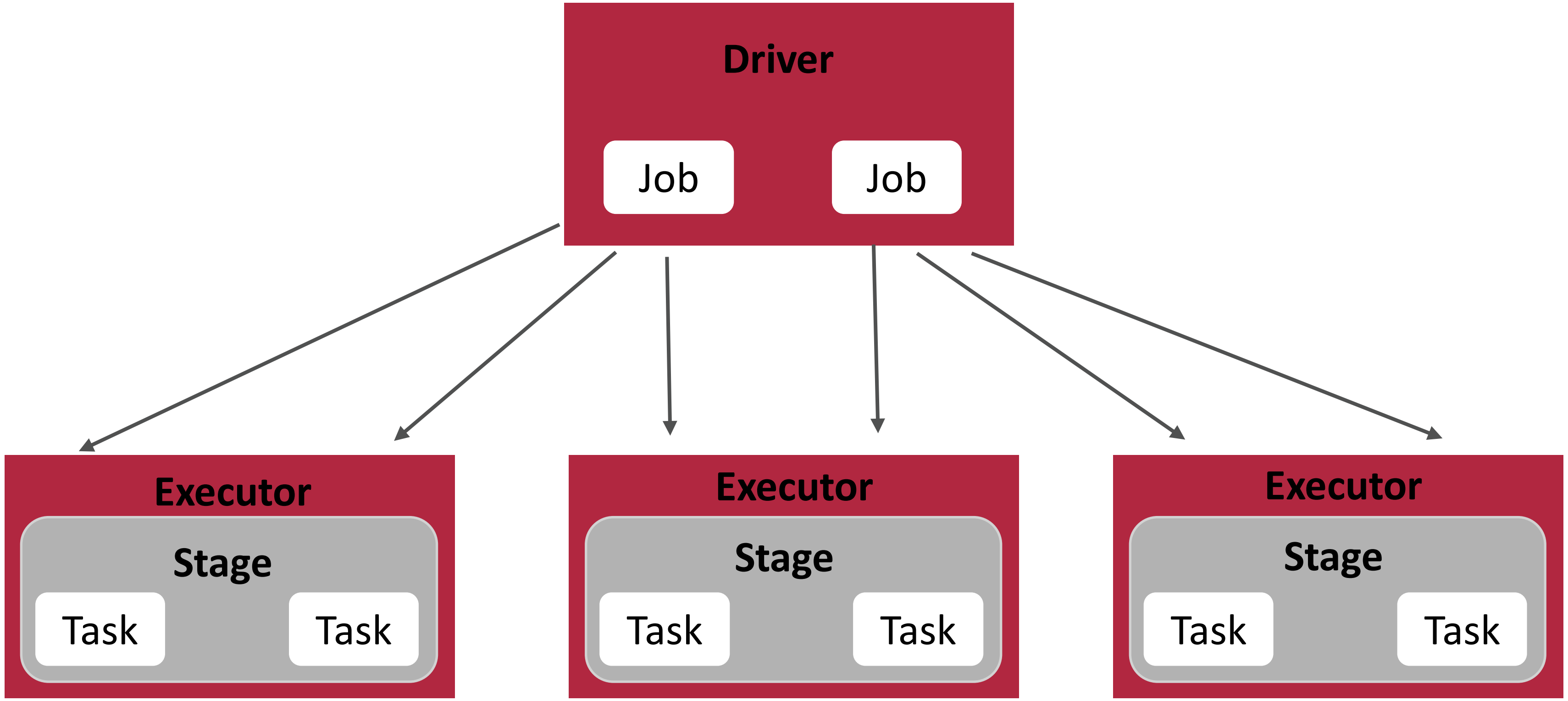
Managed Instance:

- Standard
- Premium
- Memory Optimized Premium

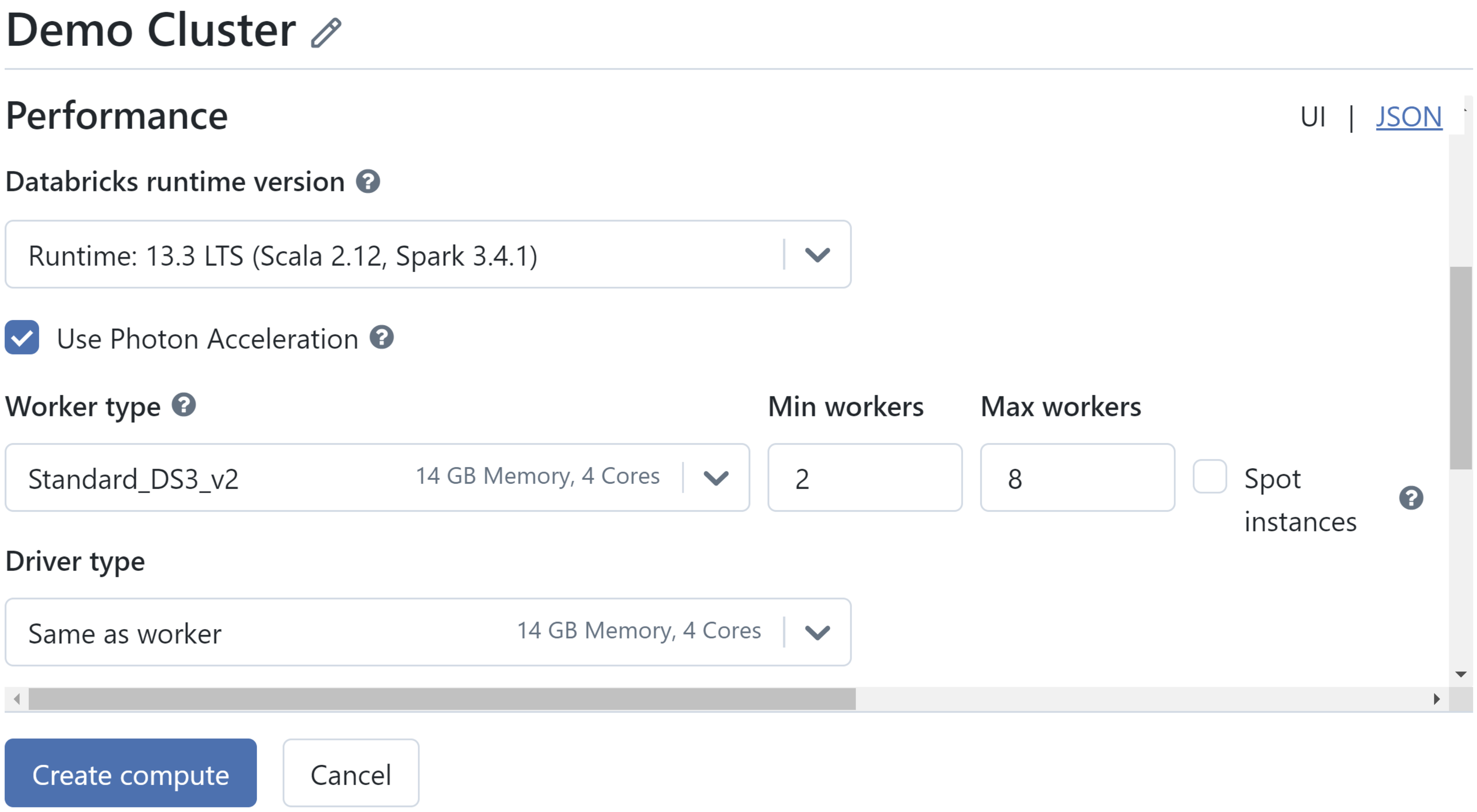


DATABRICKS CLUSTERS

HOW DATABRICKS COMPUTE WORKS



DATABRICKS CLUSTERS



SUMMARY

- We have ACID transactions and Partitioning in both
- They have different approaches to storage
- Ordering data on a granular row by row level for more optimal reading available in both: Clustered Indexes, Z-Ordering
- Replay history is achievable in both but very different
- Compute is very different in both, with Databricks being more complex
- Similarities in security approaches but still very different