# Delta and Databricks vs SQL Server

# Who am I

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


databricks
LAKEHOUSE
Data Engineer Professional

@annawykes

@annawykes

MVP Microsoft® Most Valuable Professional

Cloud Data Driven
ONLINE USER GROUP MEETING
DELIVER FREE CONTENT RELATED TO DATA PLATFORM

sqlbits
2024

# Who am I

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

Microsoft® Most Valuable Professional

welldata

sqlbits ***2024***

# Who am I

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

PyData

[DATA].[BRISTOL]

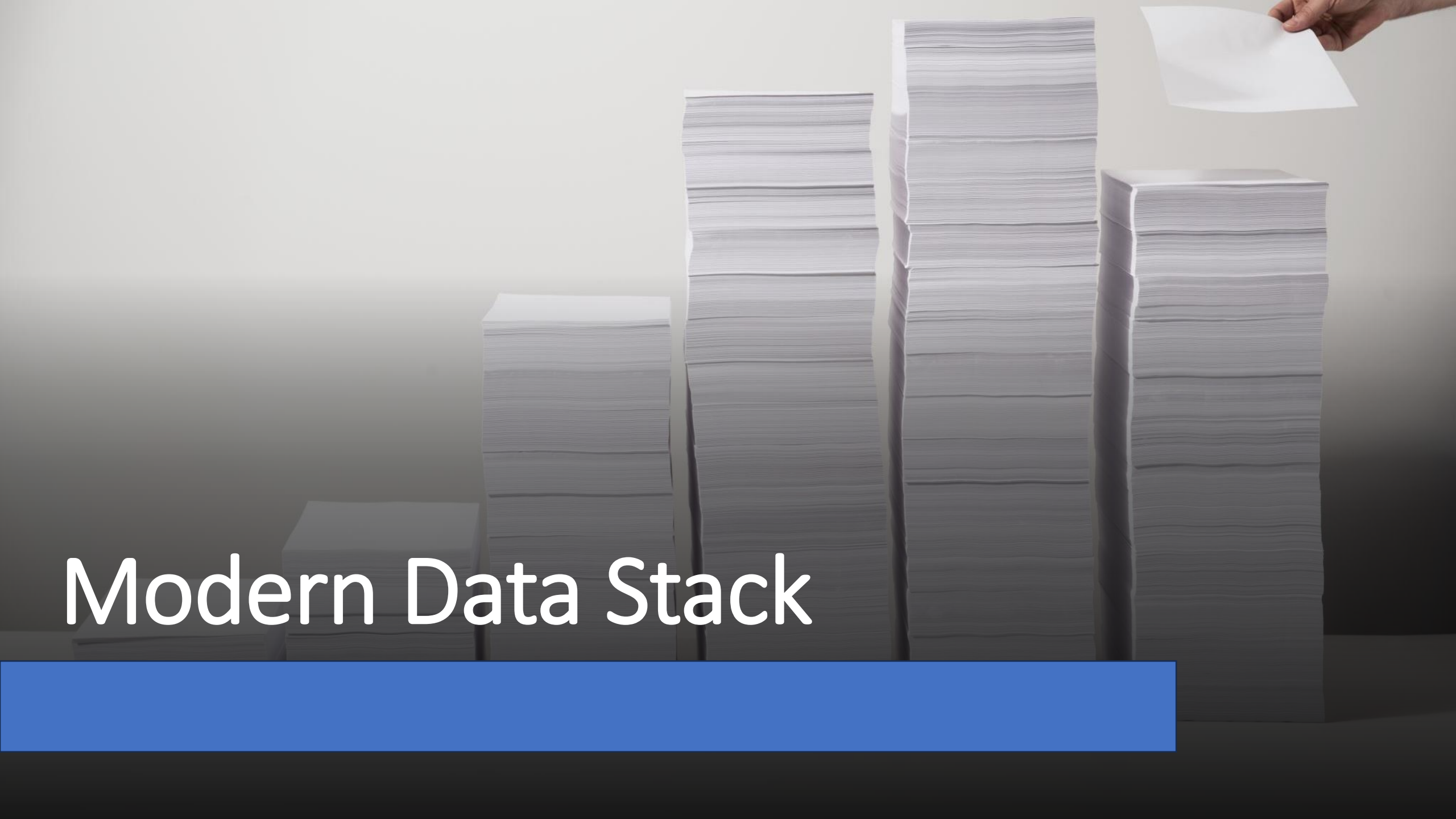
james-c-yarrow

sqlbits
***2024***

# Why?

# Agenda

- Modern Data Stack and where we fit
- Databricks Clusters vs SQL Engine
- ACID Transactions
- Parquet/Delta Storage vs Storage Structures
- Partitioning Delta vs SQL Server
- Z-ordering and Liquid Clustering vs Index Maintenance
- Change Data Capture (CDC)
- Delta History and Time Travel vs SQL Server Temporal Tables
- Delta Vacuum vs Data Retention
- Identity Columns, Primary & Foreign Keys
- Security (Unity Catalog) vs SQL Server Security

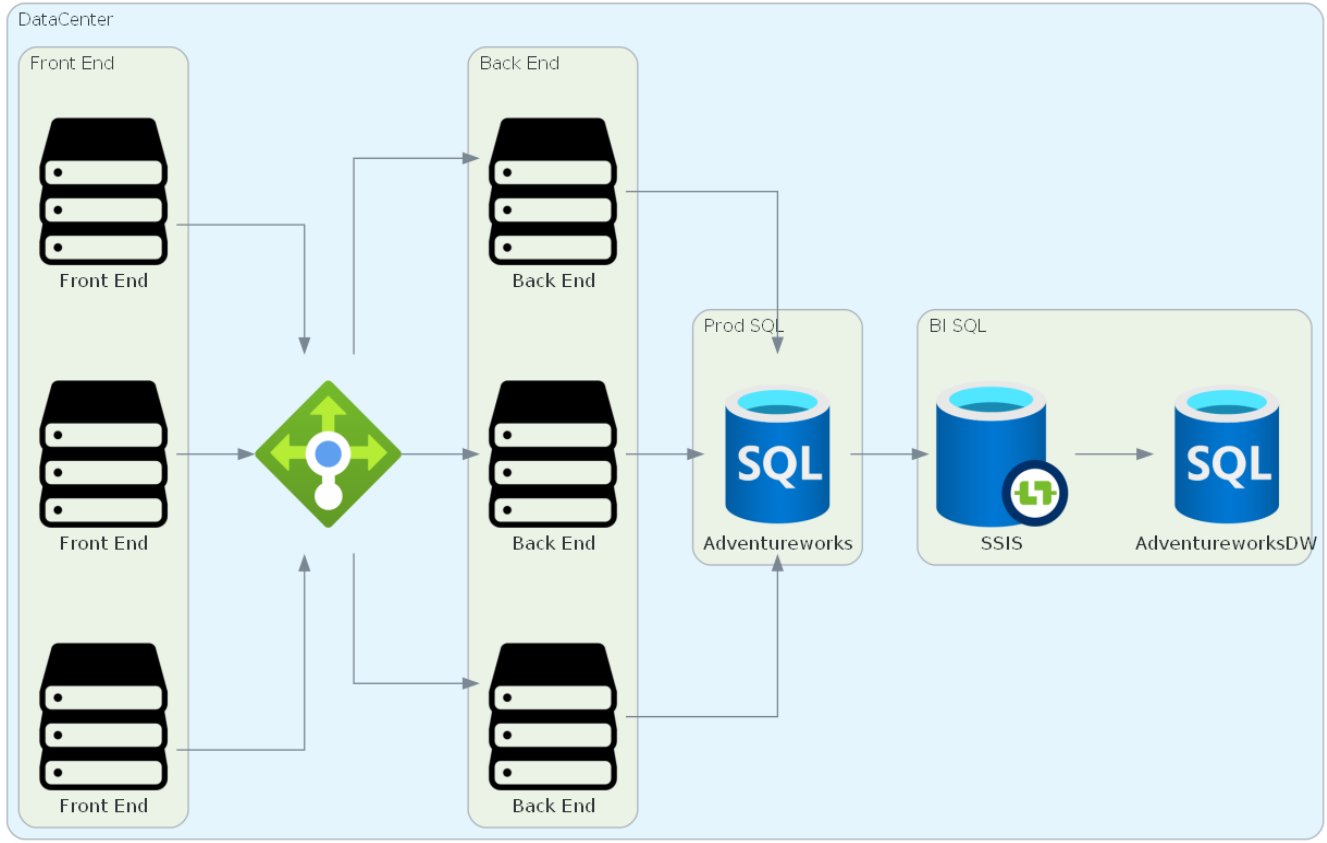


# Modern Data Stack

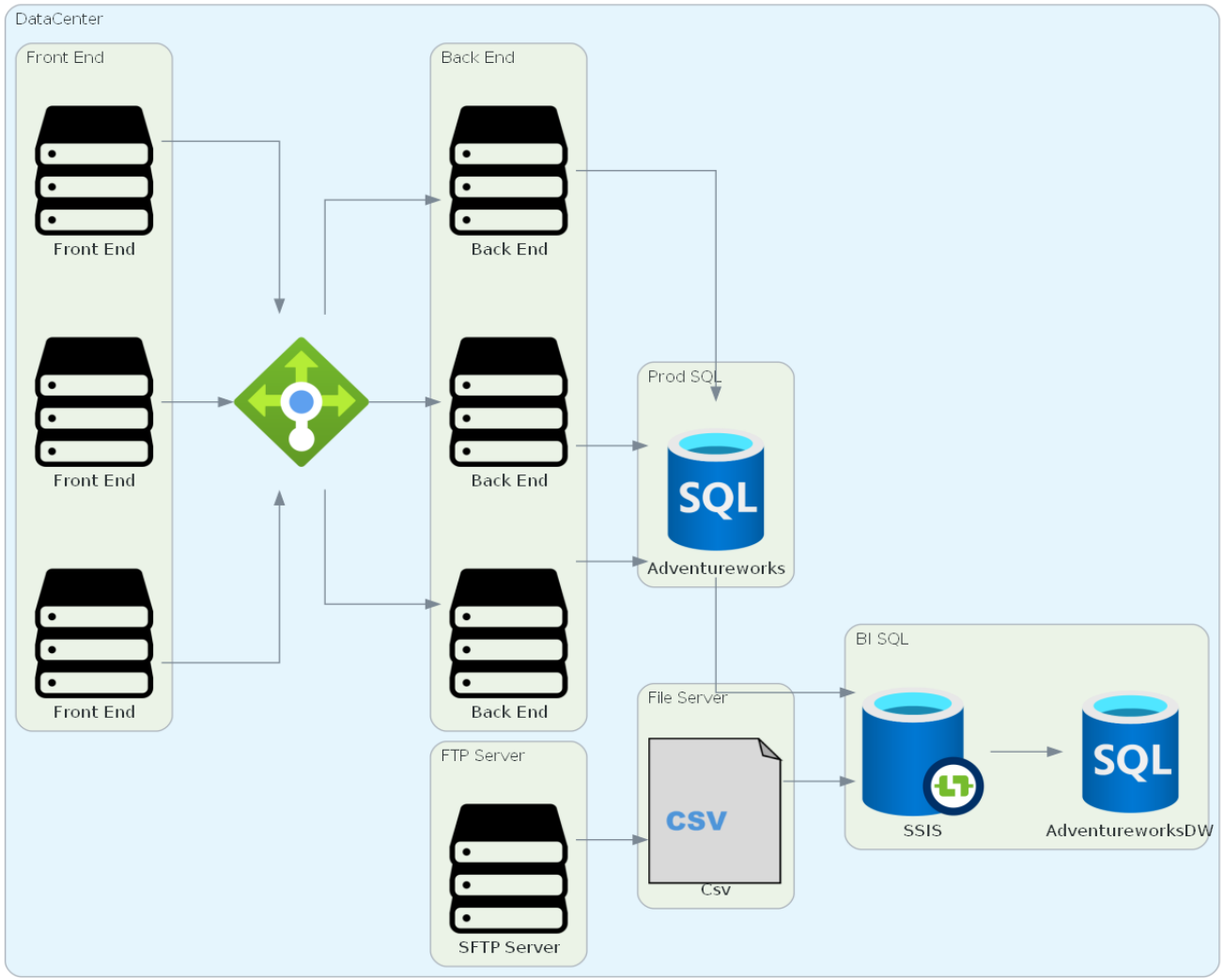
# Modern Data Stack

- The same data stack with modern components and the capabilities they bring
- Same data model challenges
- More Distribution
- More Centralisation
- Let take a quick journey from our traditional SQL Server to SQL Server ETL

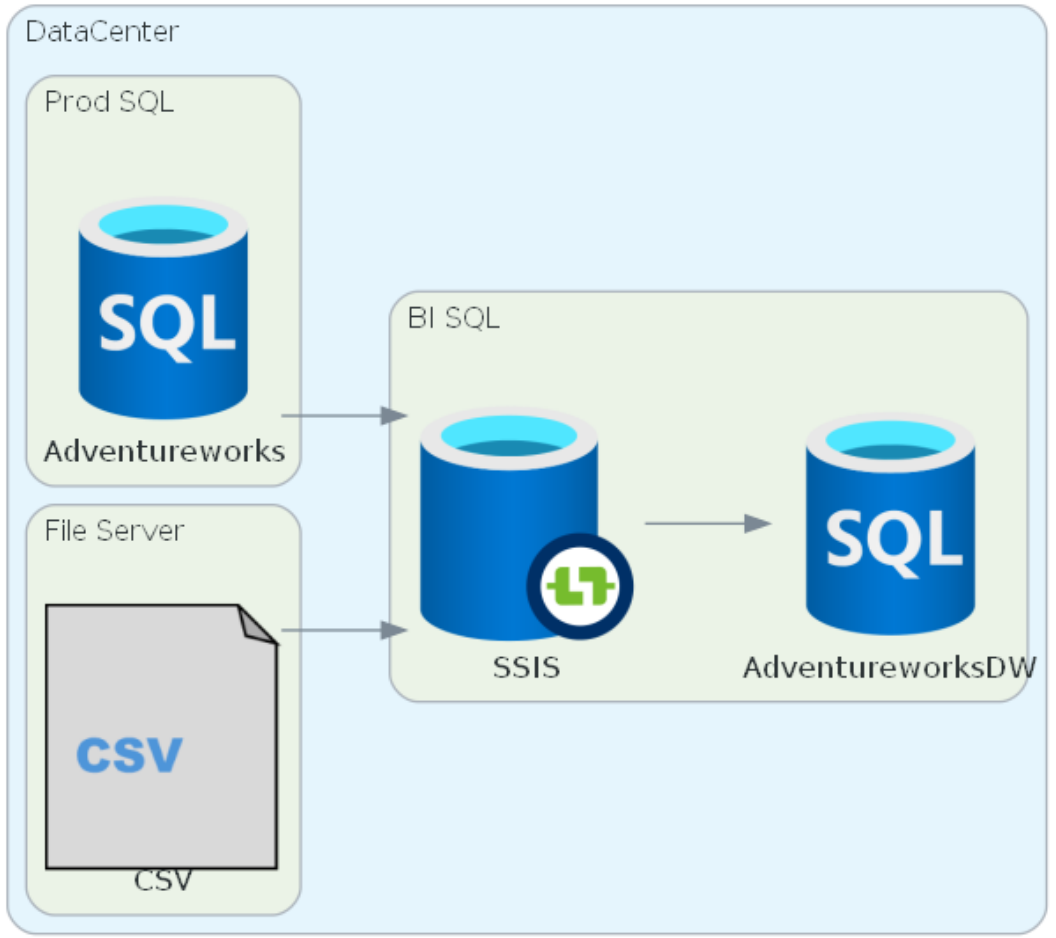
# Modern Data Stack



# Modern Data Stack

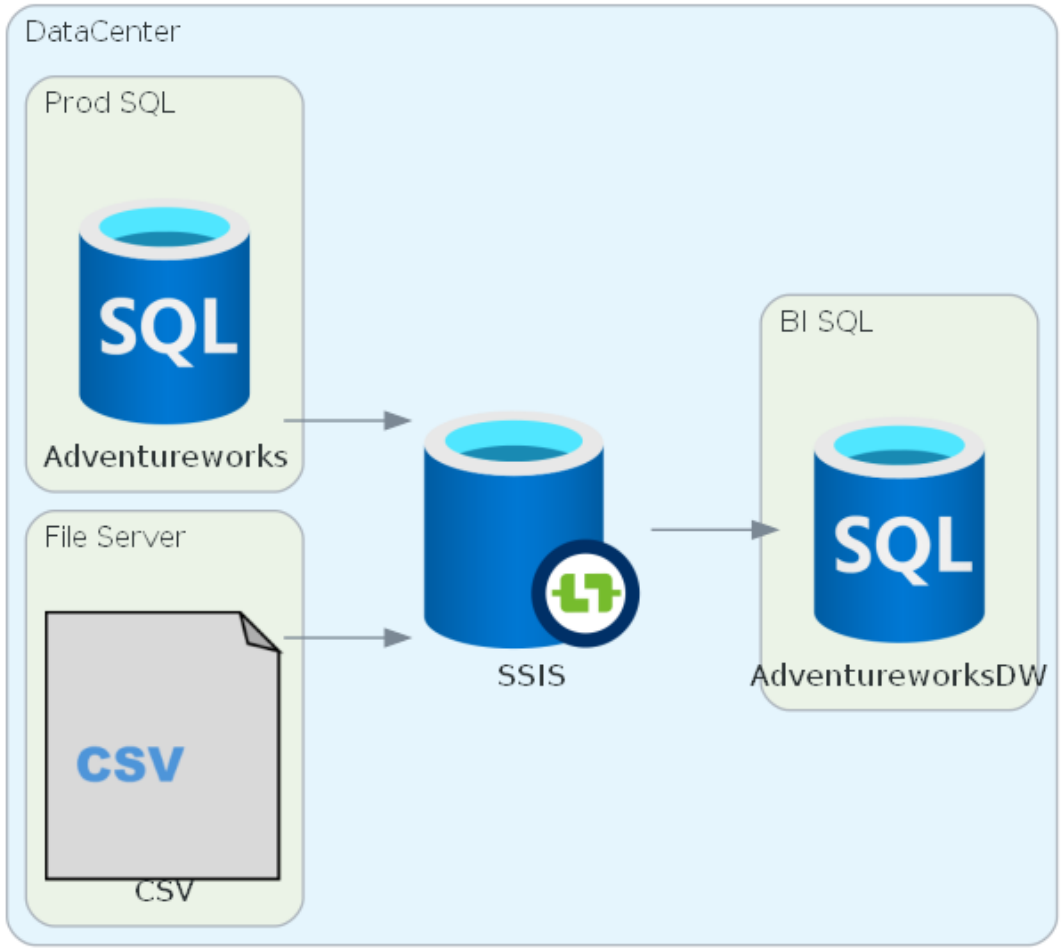


# Modern Data Stack

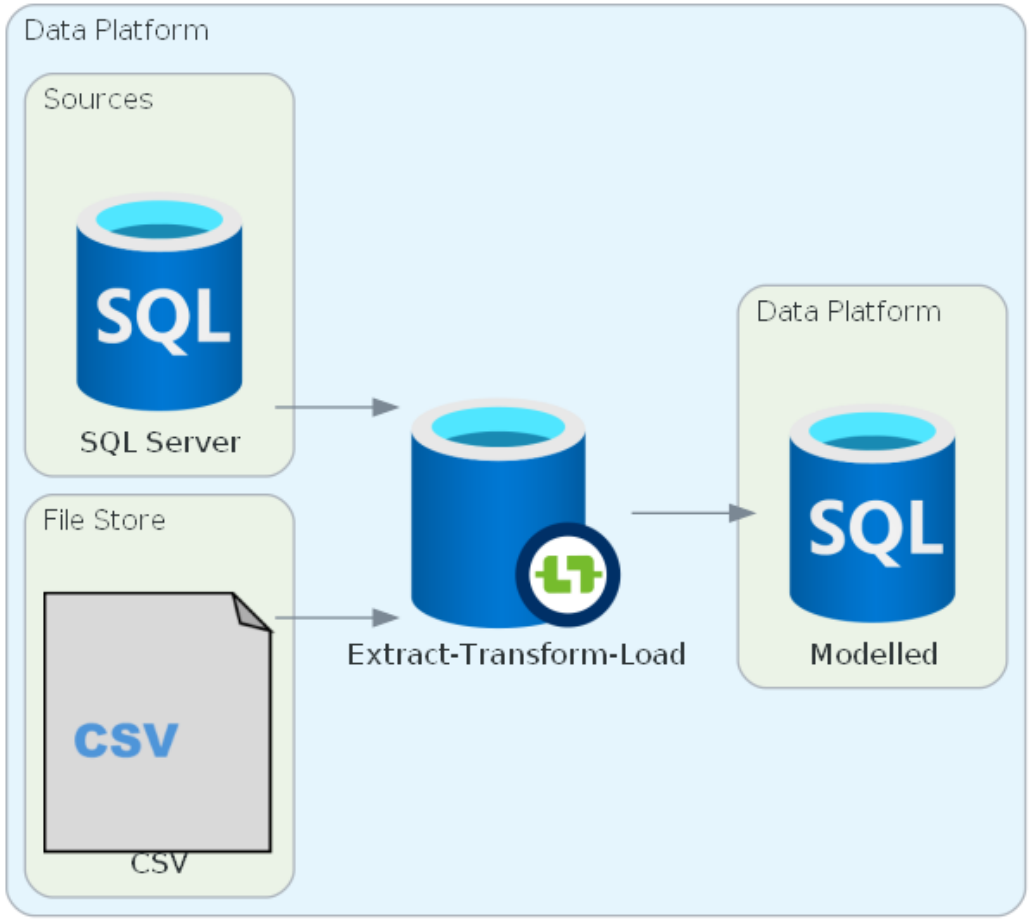


# Modern Data Stack

SSIS does a lot!

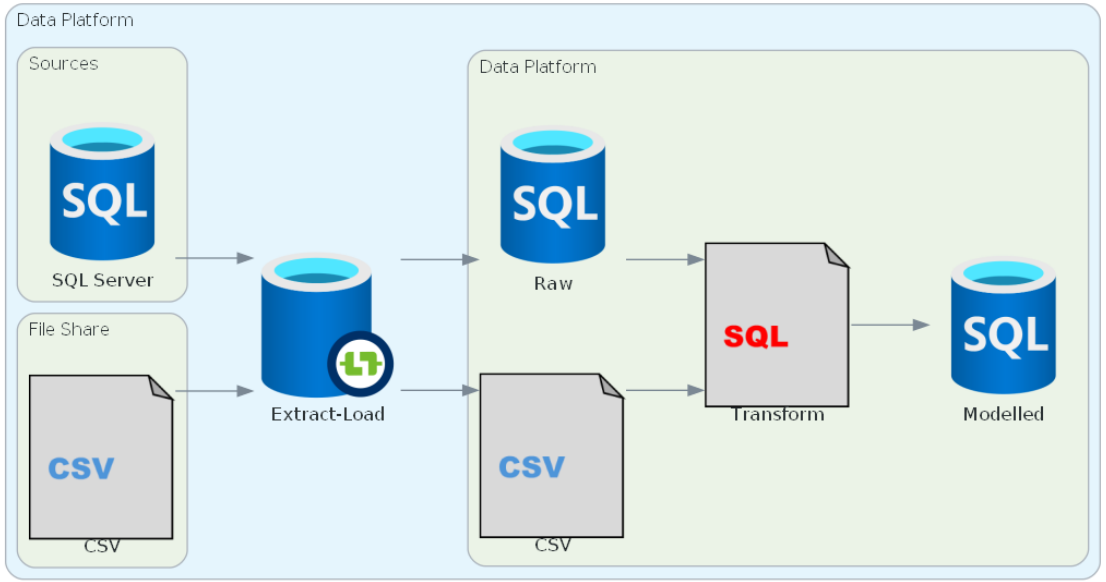


# Modern Data Stack



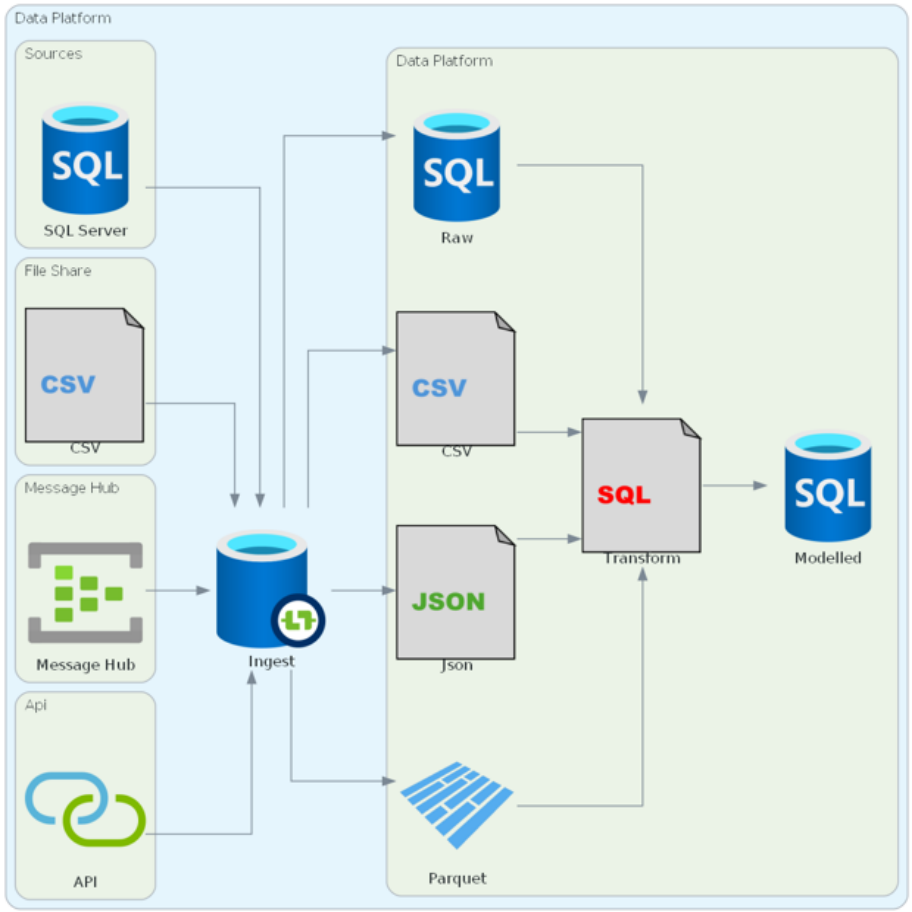
# Modern Data Stack

Storage got cheaper



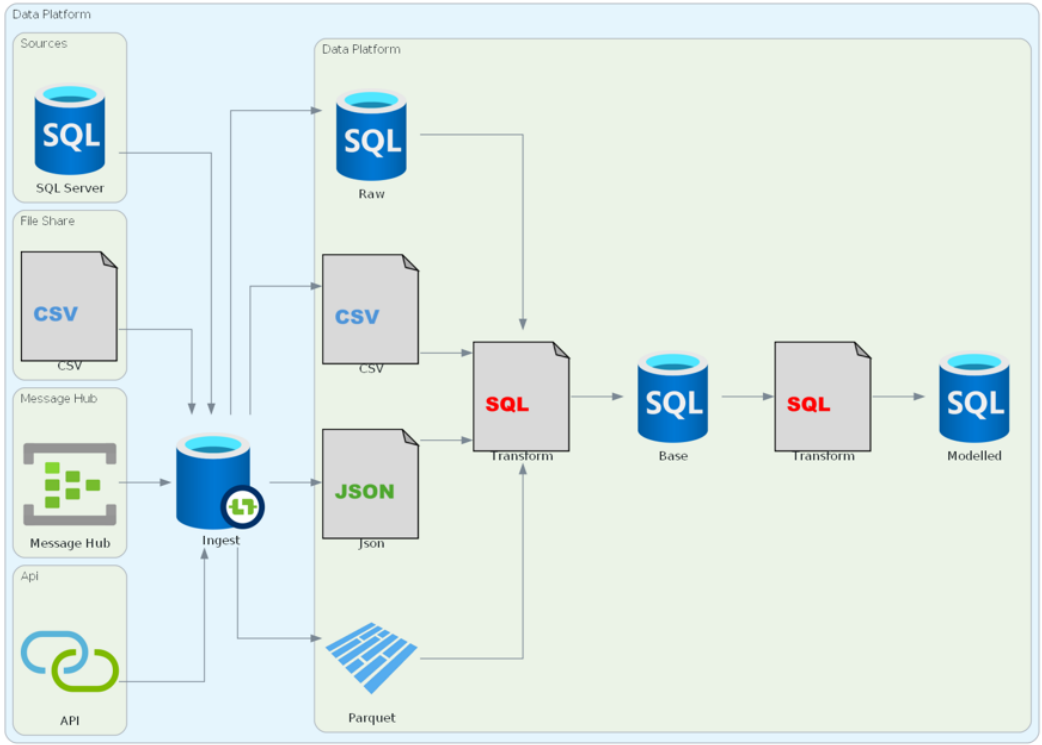
# Modern Data Stack

More sources!

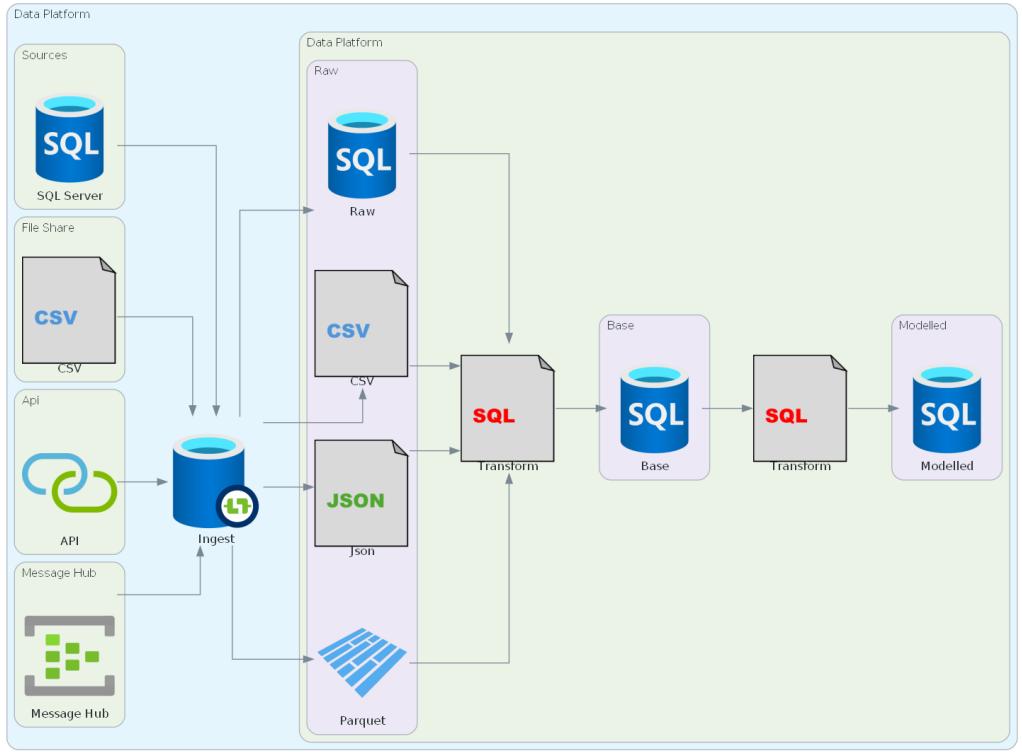


# Modern Data Stack

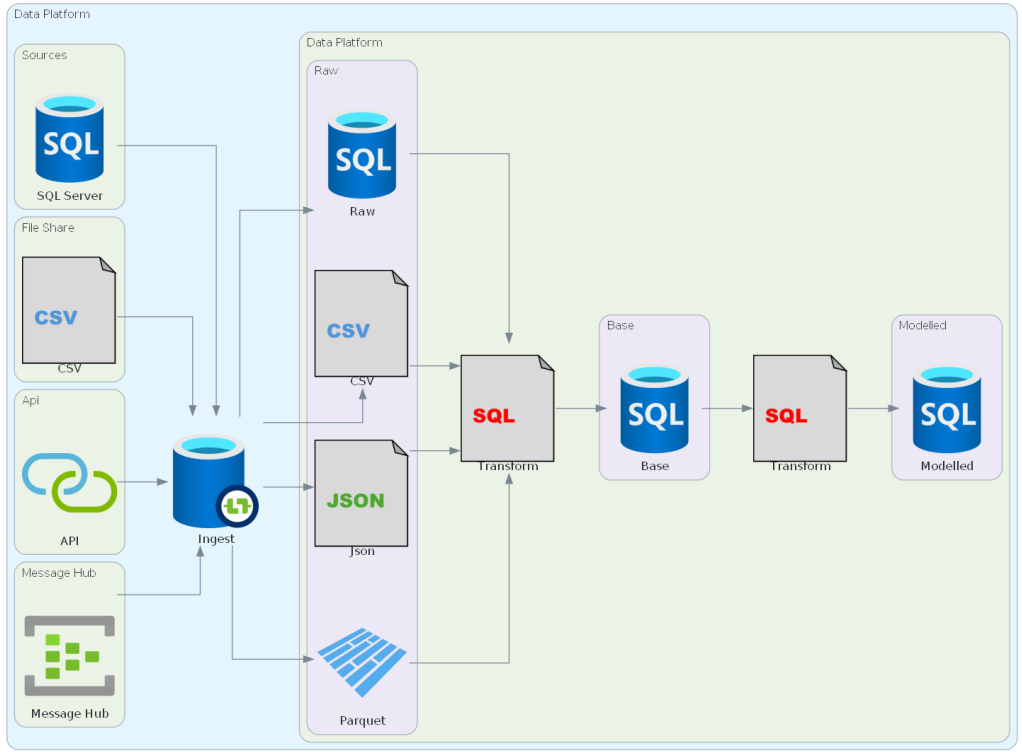
Transformation is hard!



# Modern Data Stack

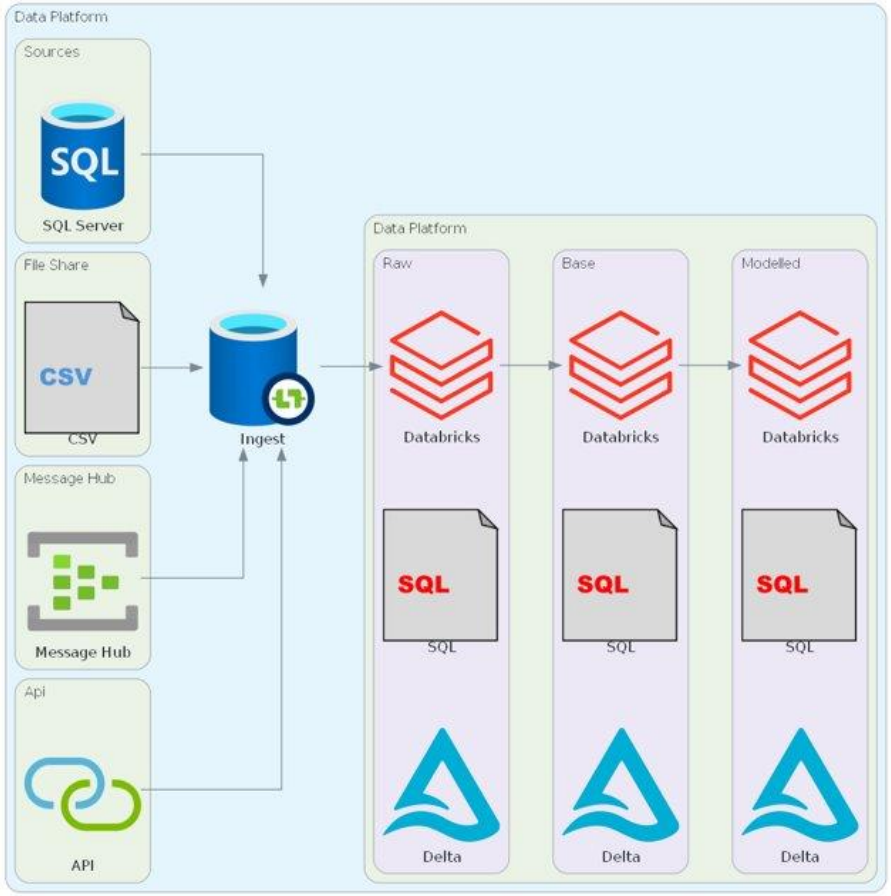


# Modern Data Stack



# Modern Data Stack

More Saving is on demand compute



# Storage & Compute

# What problem are we trying to solve?

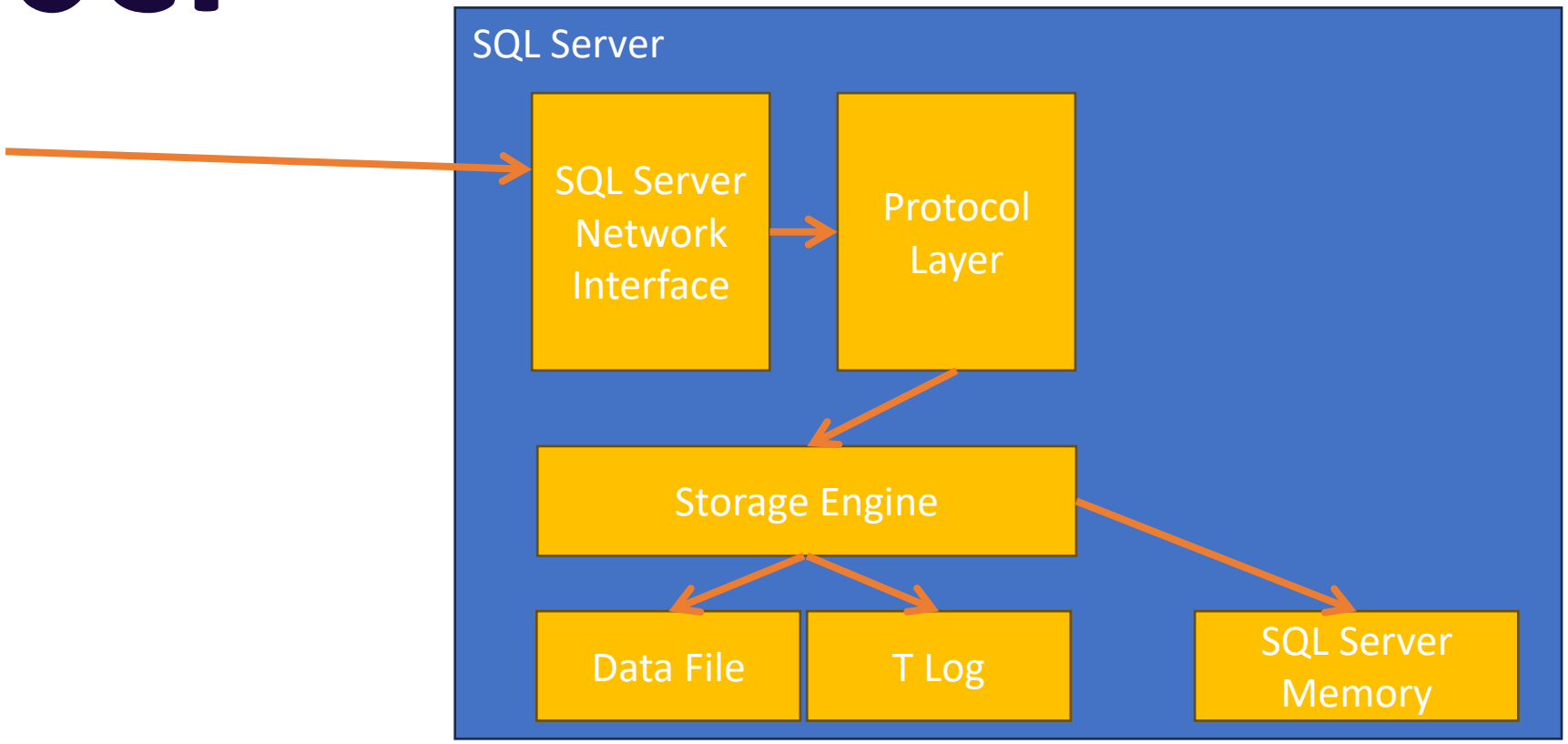
- What compute engine are we using?

# SQL Server

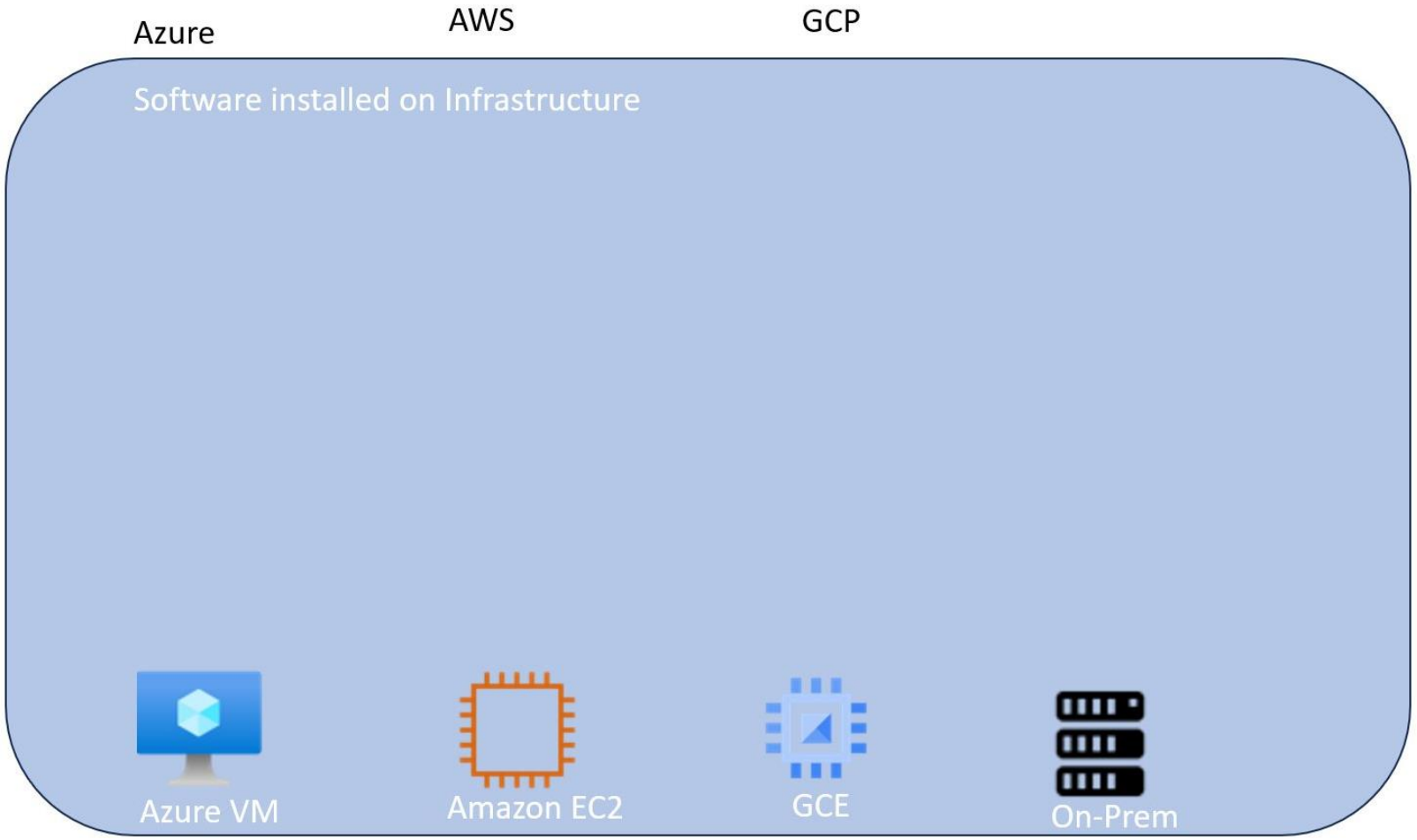
# SQL Server

What is Computer?

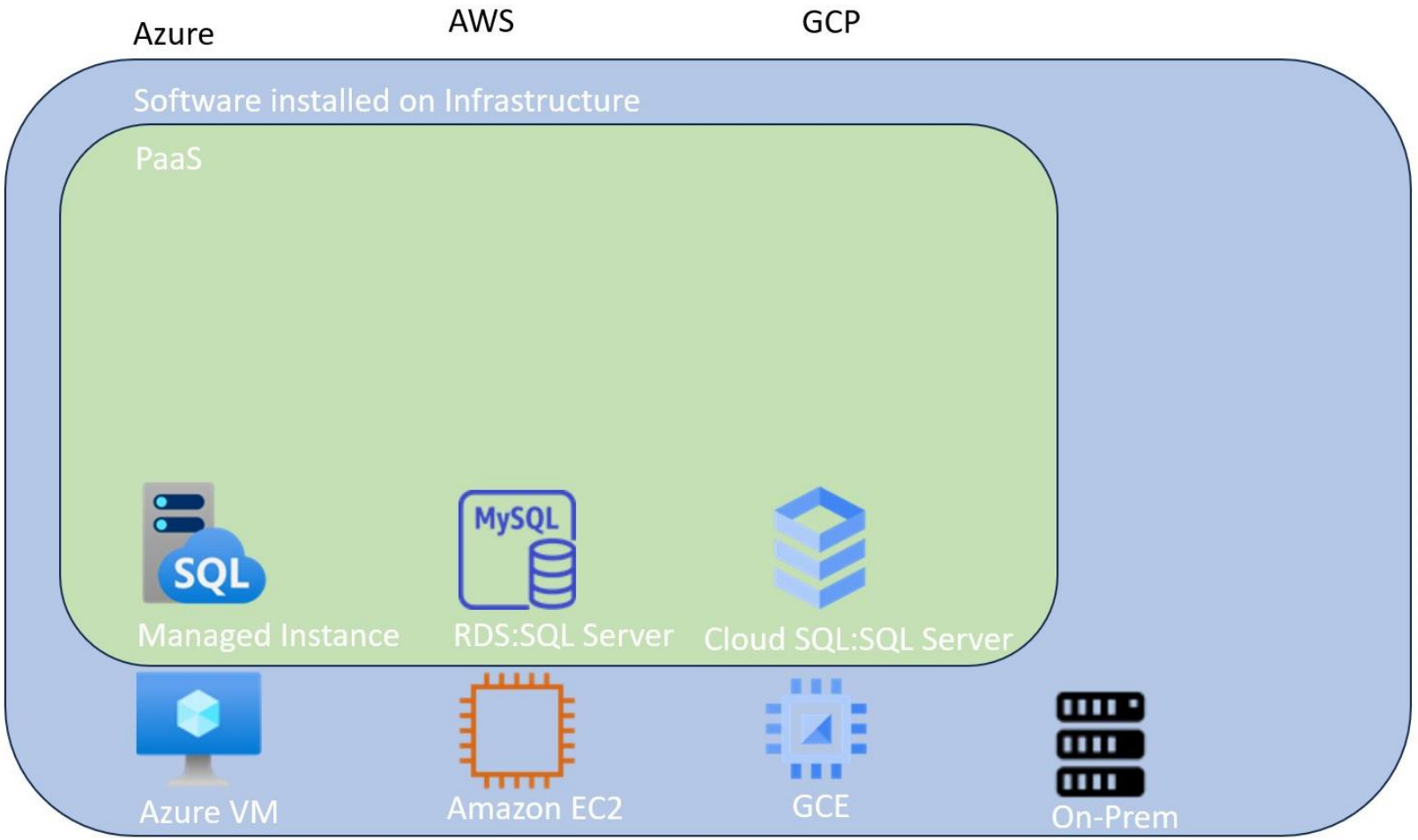
What is Storage?



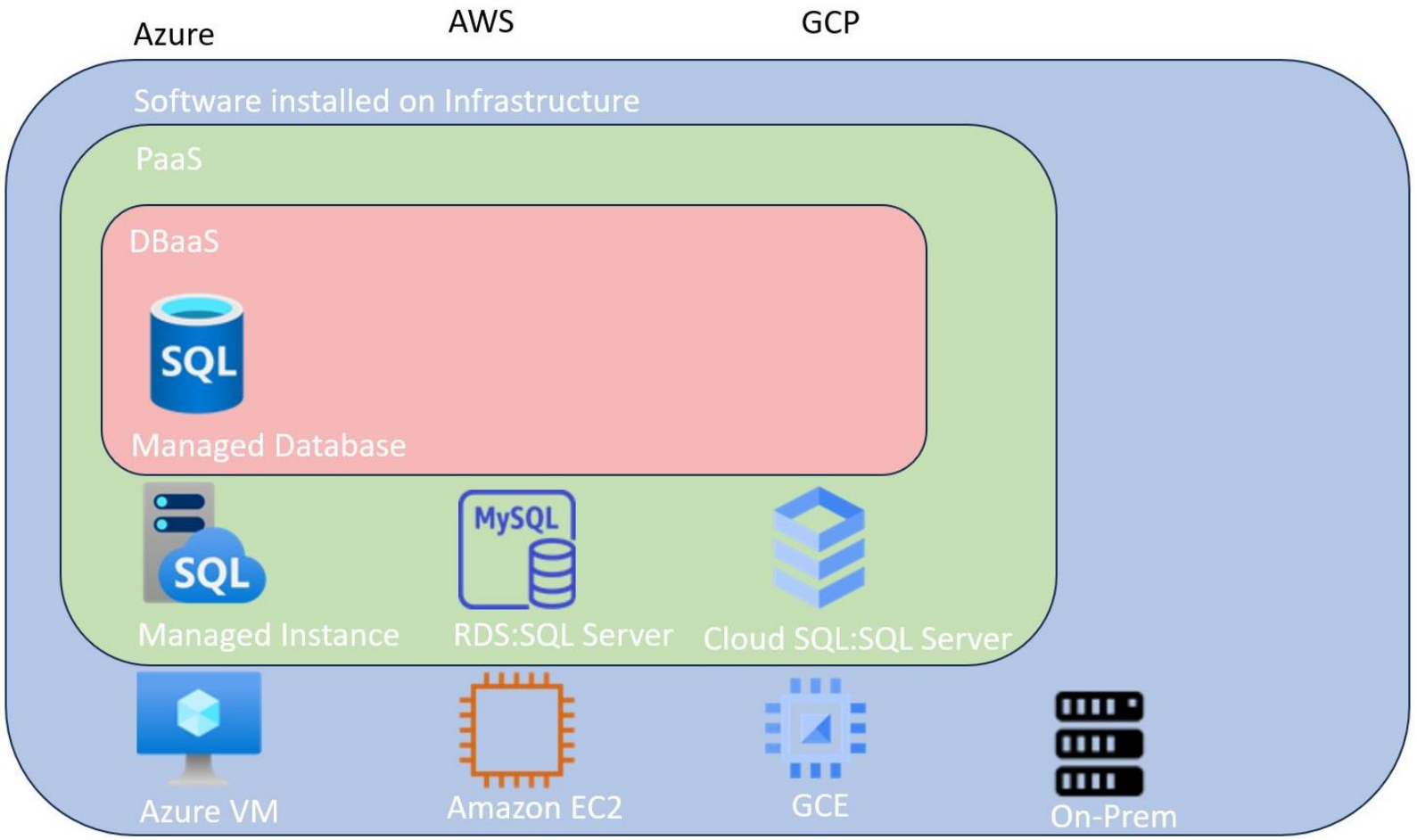
# SQL Server


Azure
AWS
GCP
Software installed on Infrastructure
Azure VM
Amazon EC2
GCE
On-Prem

# SQL Server


Azure
AWS
GCP
Software installed on Infrastructure
PaaS
SQL
Managed Instance
MySQL
RDS:SQL Server
Cloud SQL:SQL Server
Azure VM
Amazon EC2
GCE
On-Prem

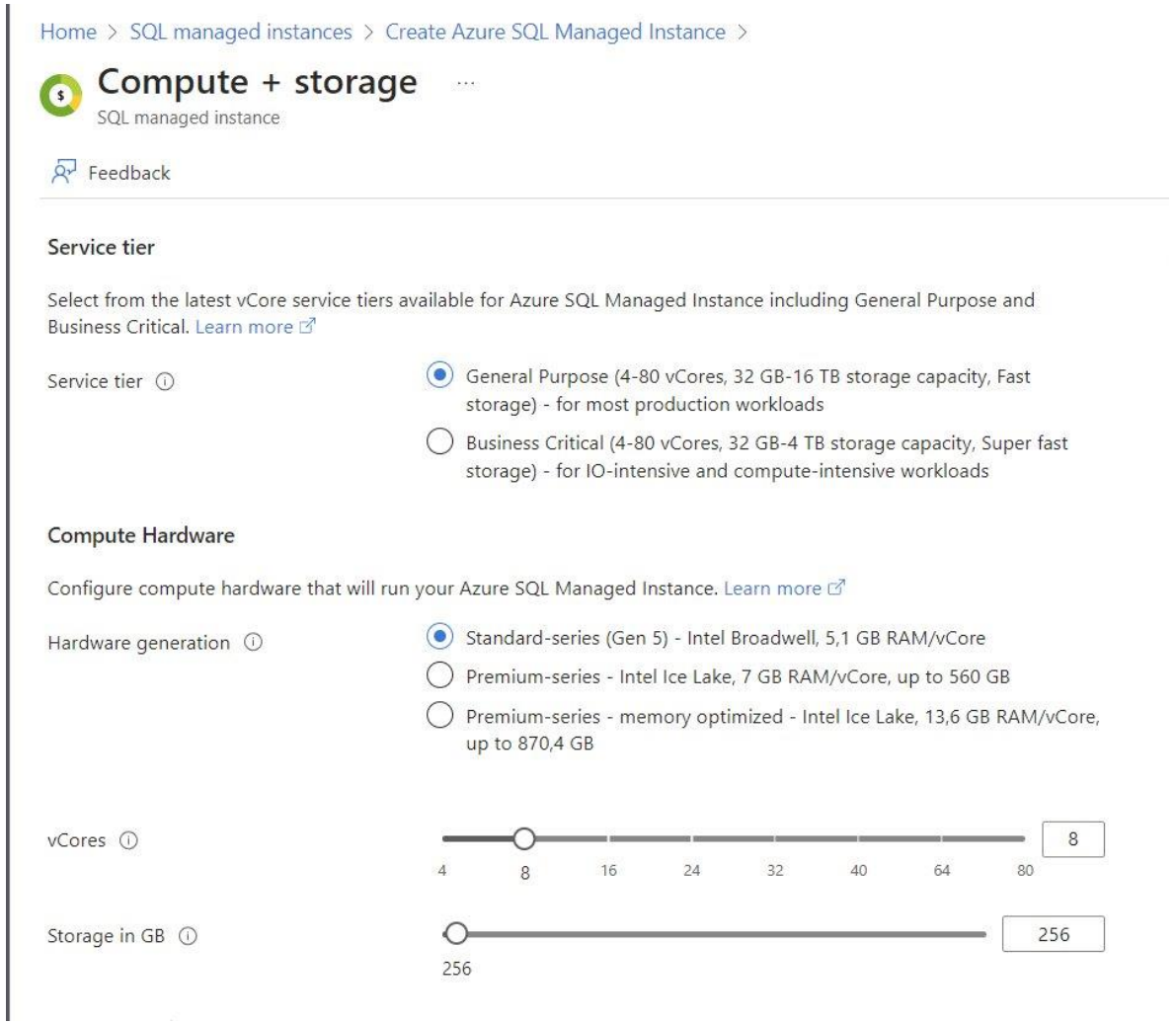
# SQL Server


Azure
AWS
GCP
Software installed on Infrastructure
PaaS
DBaaS
SQL
Managed Database
SQL
Managed Instance
MySQL
RDS:SQL Server
Cloud SQL:SQL Server
Azure VM
Amazon EC2
GCE
On-Prem

# SQL Server

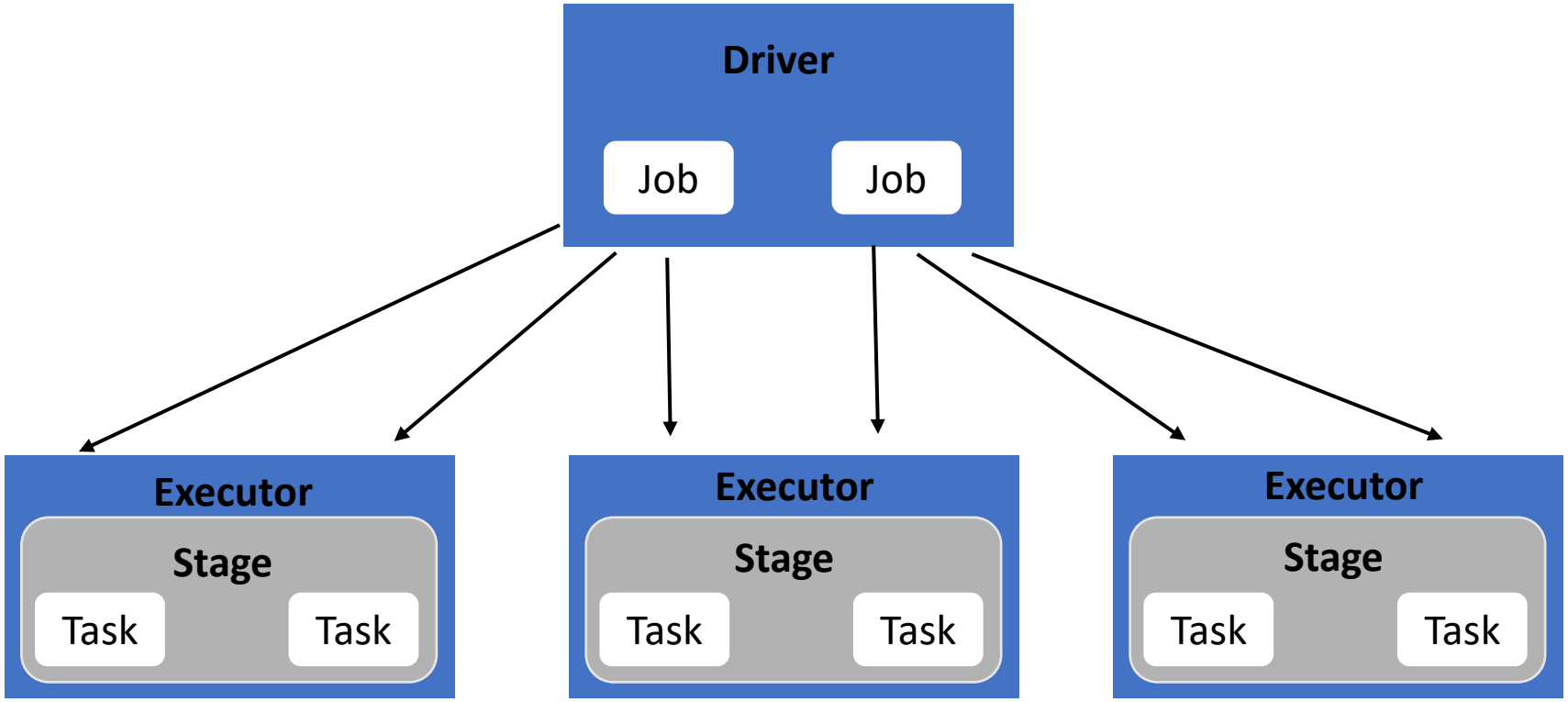
Managed Instance:

- Standard
- Premium
- Memory Optimized Premium



# Databricks Clusters

# How Databricks Compute Works



# Databricks Runtime

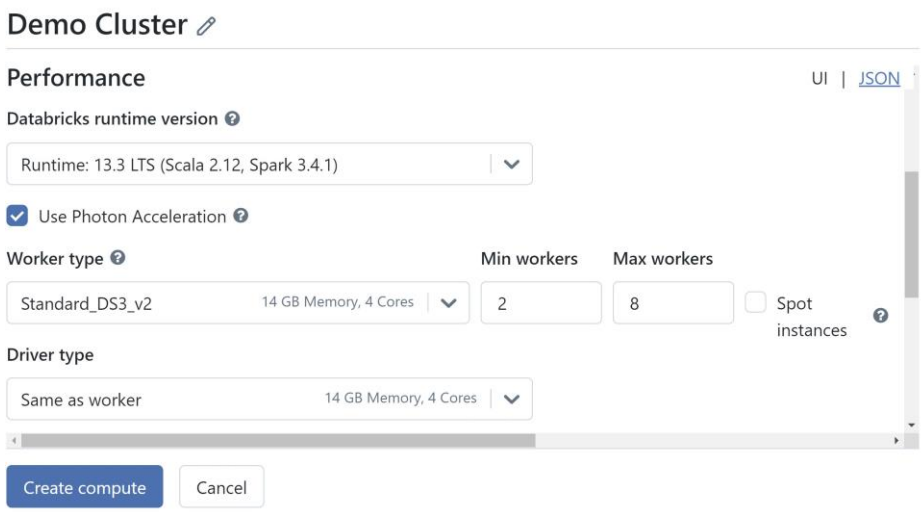
Databricks Runtime's are version of **Apache Spark** with **additional libraries** and **performance enhancements** provided by Databricks. Examples include:

- **Databricks Runtime for Apache Spark**
- **Databricks Runtime ML**
- **Databricks Runtime for SQL Analytics**

# Different types of Databricks Compute

- All-Purpose compute
- Job compute
- Instance pools
- Serverless SQL warehouses

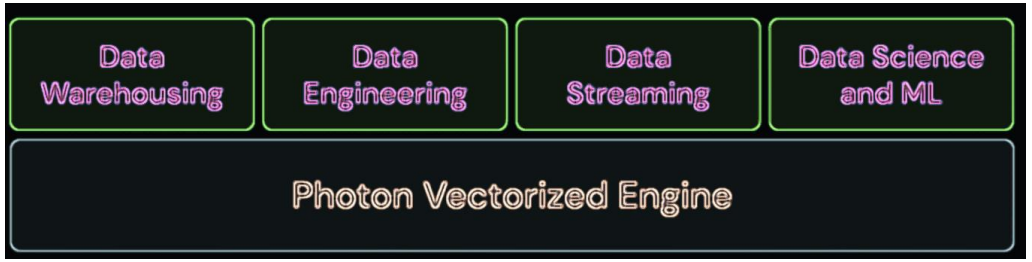
# Create a Databricks Cluster




Databricks supports creating and managing clusters with different configurations based on the **cluster mode**, **instance type**, and **autoscaling**. Standard clusters can be used for **ad-hoc data exploration**, **batch jobs**, and **data engineering tasks**.

# Photon

Databricks-native vectorized query engine that runs your **SQL workloads** and **DataFrame API calls** faster




**Limitations**

- Does **NOT support** Structured Streaming: Photon currently supports **stateless streaming ONLY** with Delta, Parquet, CSV, and JSON. Stateless Kafka and Kinesis streaming is supported when writing to a Delta or Parquet sink.
- **Does NOT support UDFs** or **RDD** APIs.
- **WONT** impact queries that normally run in under two seconds

# ACID Transactions

# Atomicity, Consistency, Isolation, and Durability

| | |
|---|---|
| Atomicity | The Transaction can be treated as a single unit that can succeed or fail |
| C | |
| I | |
| D | |

# 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 |
| I | |
| D | |

# 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 |
| D | |

# 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

| | Atomicity | Consistency | Isolation | Duratbility |
|---|---|---|---|---|
| Parquet | ✗ | ✗ | ✗ | ✗ |
| Delta Format | ✓ | ✓ | ✓ | ✓ |
| Simple Recovery | ✓ | ✓ | ✓ | ✓ |
| Full Recovery | ✓ | ✓ | ✓ | ✓ |

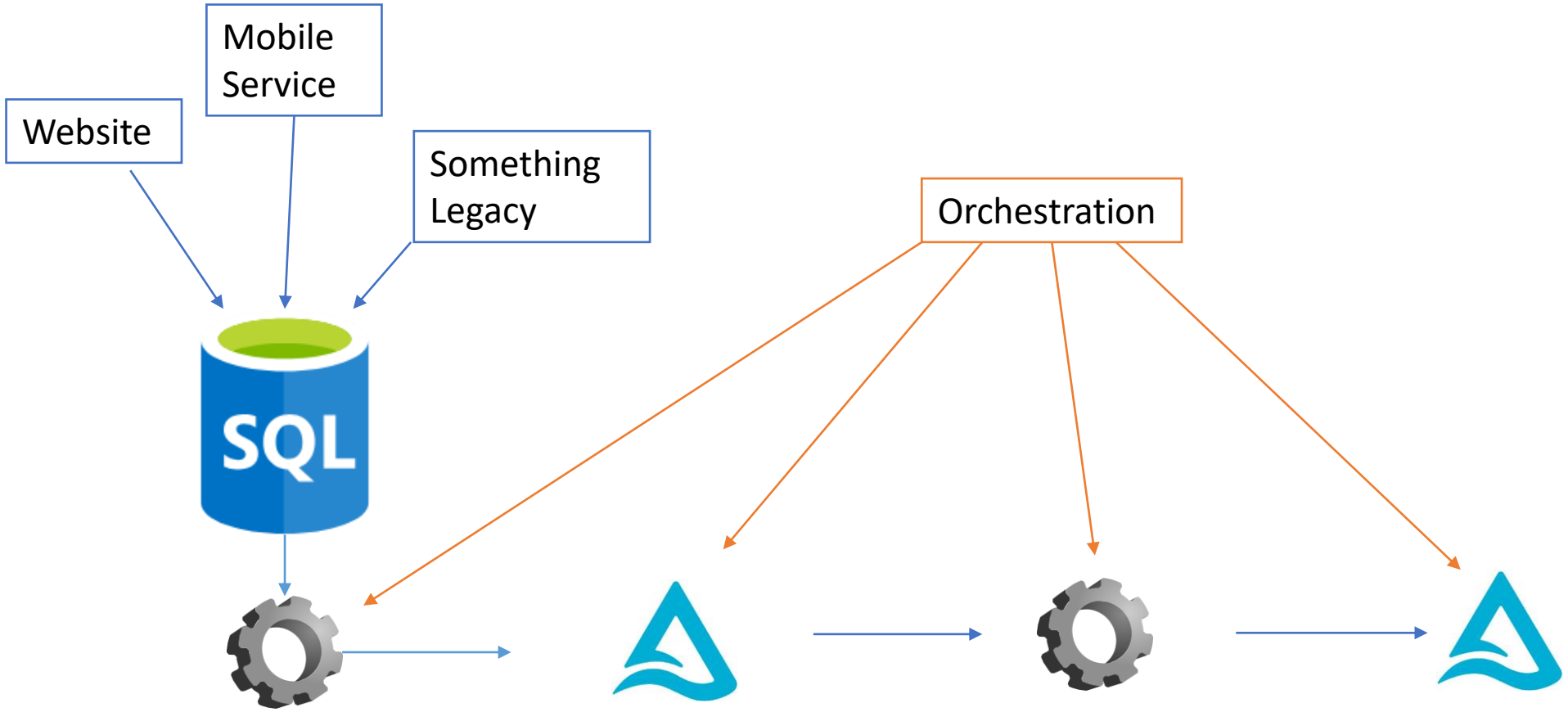
Isolation Levels

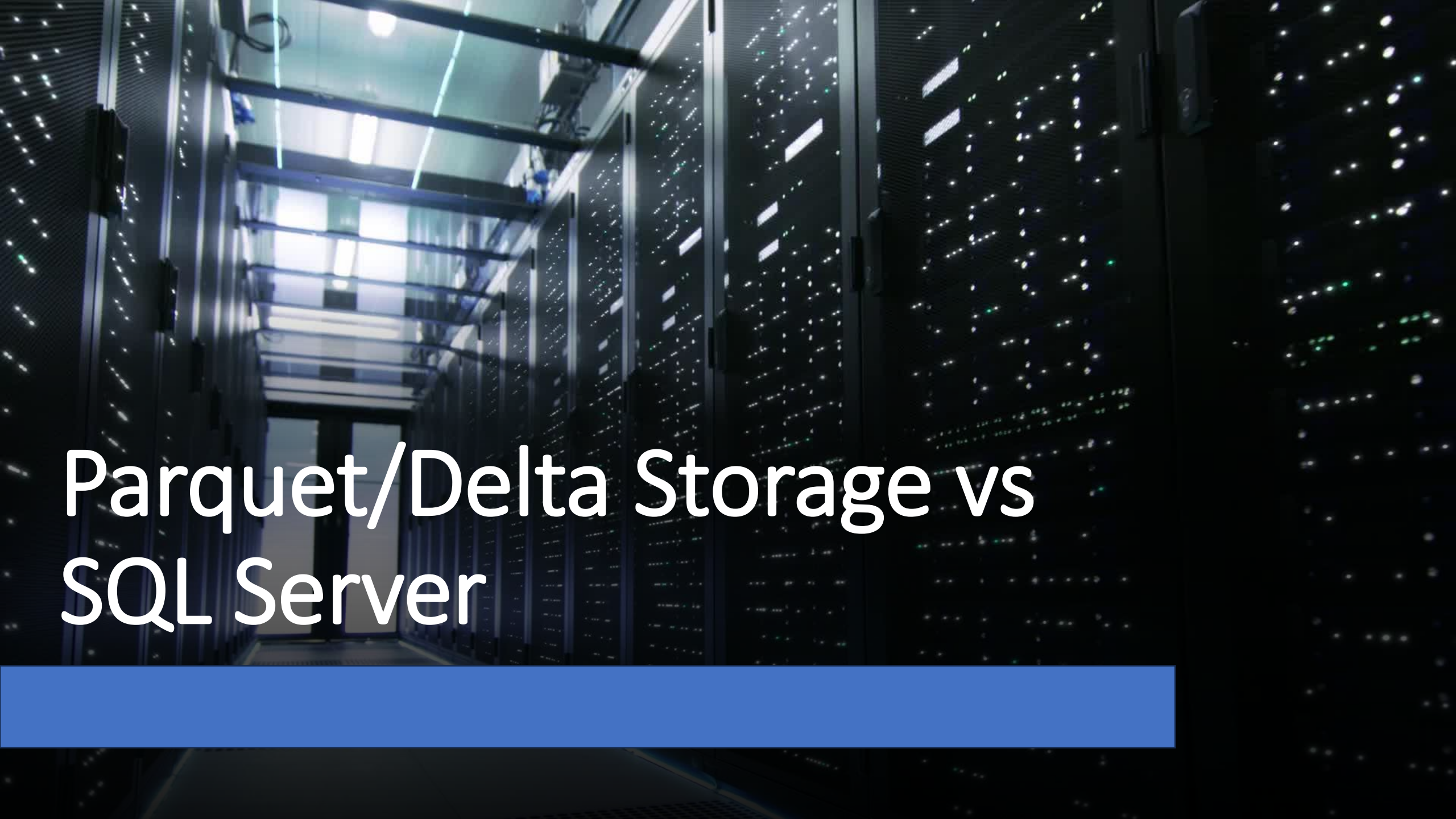
| SQL Server | Delta Format |
|---|---|
| Read Uncommitted | Write Serializable |
| Repeatable Read | ✗ |
| Snapshot | ✗ |
| Serializable | Serializable |

# Atomicity, Consistency, Isolation, and Durability

- Databricks uses Delta format to get ACID properties
- Delta format is open source!


# Modern Data Stack





# Parquet/Delta Storage vs SQL Server

# What problem are we trying to solve?

Where and how to store our data, what format do we use?

# Storage in SQL Server

# How do we store data to disk?

We have to persist data to disk

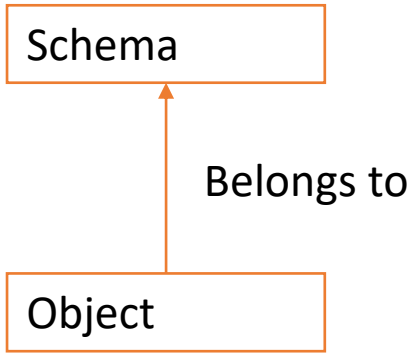
- Data stored in Data files
- SQL Server exclusively access the files

| | SQL Server | Delta Format | Parquet |
|---|---|---|---|
| File Structure | *.MDF,<br>*.NDF | *.parquet | *.parquet |
| Log File Structure | *.LDF | /_delta_log<br>/000000.json | |

# How do we store data to disk?

```
CREATE TABLE [Person] [Person](
[BusinessEntityID] [int] NOT NULL,
[PersonType] [nchar](2) NOT NULL,
[NameStyle] [nvarchar](15) NOT NULL,
[Title] [nvarchar](8) NULL,
[FirstName] [nvarchar](50) NOT NULL,
[MiddleName] [nvarchar](50) NULL,
[LastName] [nvarchar](50) NOT NULL,
[Suffix] [nvarchar](10) NULL,
[EmailPromotion] [int] NOT NULL,
[AdditionalContactInfo] [xml] NULL,
[Demographics] [xml] NULL,
[rowguid] [uniqueidentifier] ROWGUIDCOL  NOT NULL,
[ModifiedDate] [datetime] NOT NULL,

) ON [PRIMARY] GO
```

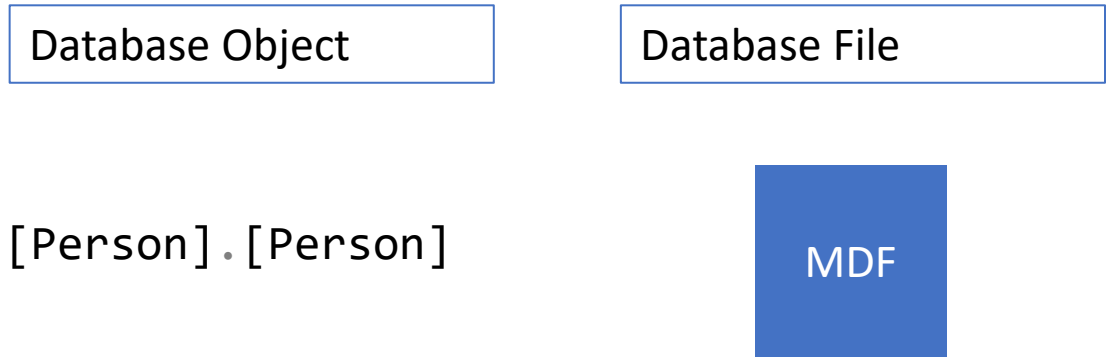



# How do we store data to disk?

Object created on Primary will be in the primary file group

```
sp_spaceused 'TableName’
```

```
SELECT *
 FROM sys.filegroups;
```



# Parquet/Delta Storage

# What is Parquet?

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

| | | |
|---|---|---|
| A | 1 | Fred |
| A | 1 | Bob |
| A | 1 | Fred |
| B | 1 | Bob |
| C | 1 | Fred |
| C | 1 | Bob |
| C | 1 | Fred |
| B | 1 | Bob |

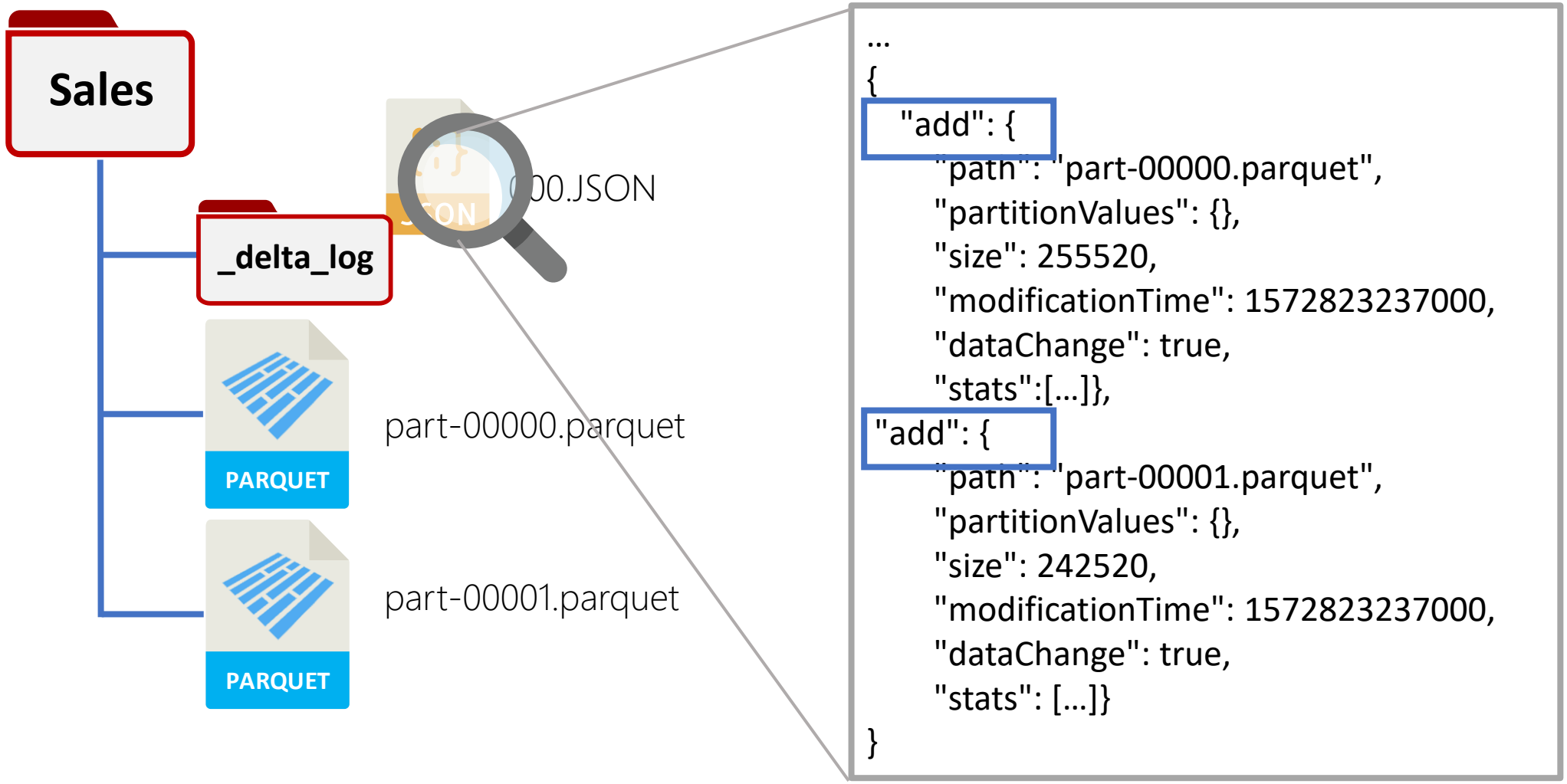
CSV File

| Col1 | Col2 | Col3 |
|---|---|---|
| A*3 | 1*8 | 1 |
| B | | 2 |
| C*3 | | 1 |
| B | | 2 |
| | | 1 |
| | | 2 |
| | | 1 |
| | | 2 |

Parquet File

Parquet works in a **very similar way** to **SQL ColumnStore**

It's a **ColumnStore compression** format


Sales
_delta_log
00.JSON
PARQUET
part-00000.parquet
PARQUET
part-00001.parquet
...
{
"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": [...]}
}

# Delta Features

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

# 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

We can mix these to meet difference performance benefits

Let's stay with person.person

We have different access patterns from different queries

Let’s try and search for some by `[BusinessEntityID]`

# Storage Structures

```
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
      ,[AdditionalContactInfo]
      ,[Demographics]
      ,[rowguid]
      ,[ModifiedDate]
  FROM [AdventureWorks2022].[Person].[Person]
  WHERE [BusinessEntityID] = 4000
```

SQL Server Engine

[PK_Person_BusinessEntityID]

[IX_Person_LastName_FirstName_MiddleName]

[AK_Person_rowguid]

[PXML_Person_AddContact]

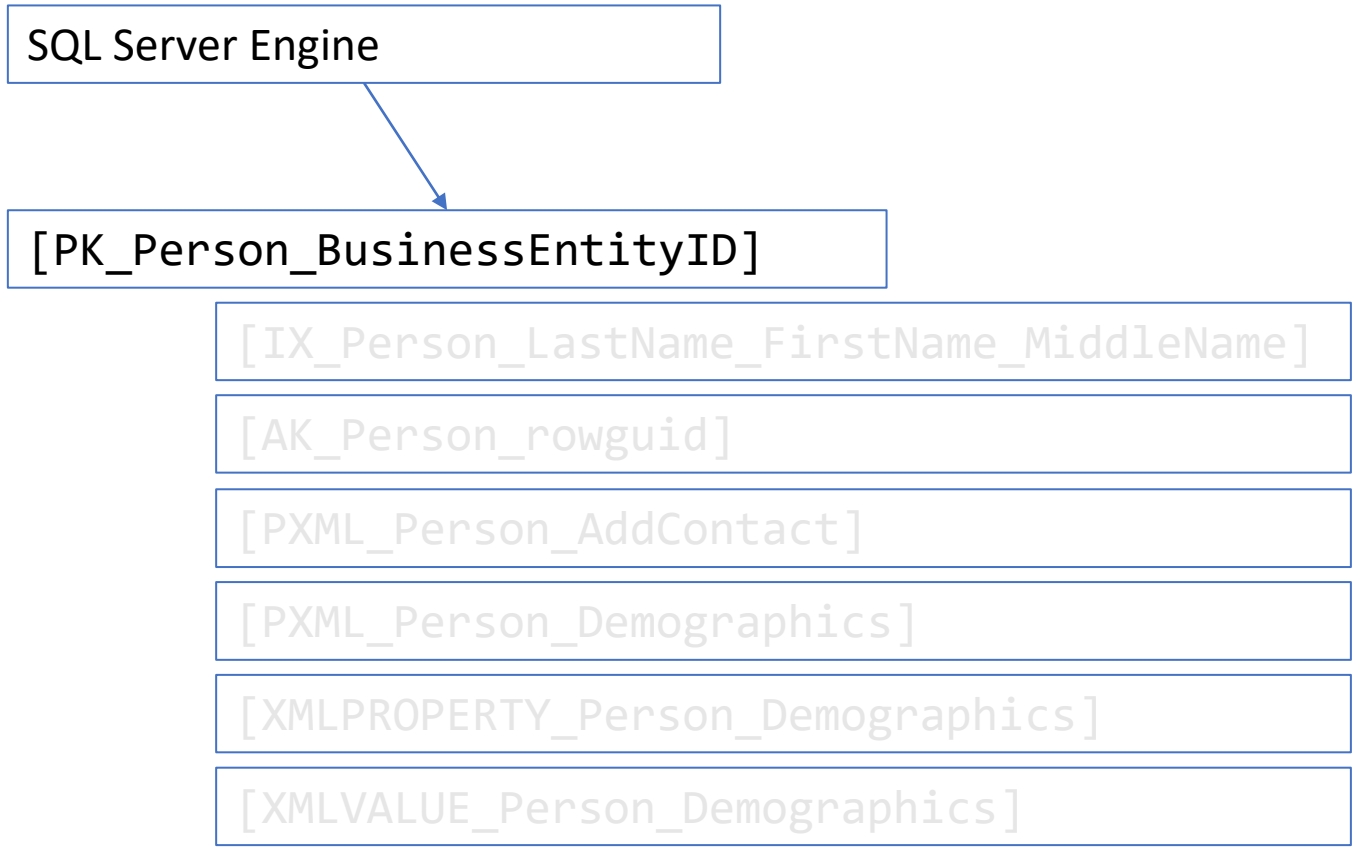
[PXML_Person_Demographics]

[XMLPROPERTY_Person_Demographics]

[XMLVALUE_Person_Demographics]

# Storage Structures

```
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
      ,[AdditionalContactInfo]
      ,[Demographics]
      ,[rowguid]
      ,[ModifiedDate]
  FROM [AdventureWorks2022].[Person].[Person]
  WHERE [BusinessEntityID] = 4000
```



# Storage Structures

```
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
      ,[AdditionalContactInfo]
      ,[Demographics]
      ,[rowguid]
      ,[ModifiedDate]
 FROM [AdventureWorks2022].[Person].[Person]
 WHERE [FirstName] = 'Jeremiah'
 AND [LastName] = 'Martin'
```

SQL Server Engine

[PK_Person_BusinessEntityID]

[IX_Person_LastName_FirstName_MiddleName]

[AK_Person_rowguid]

[PXML_Person_AddContact]

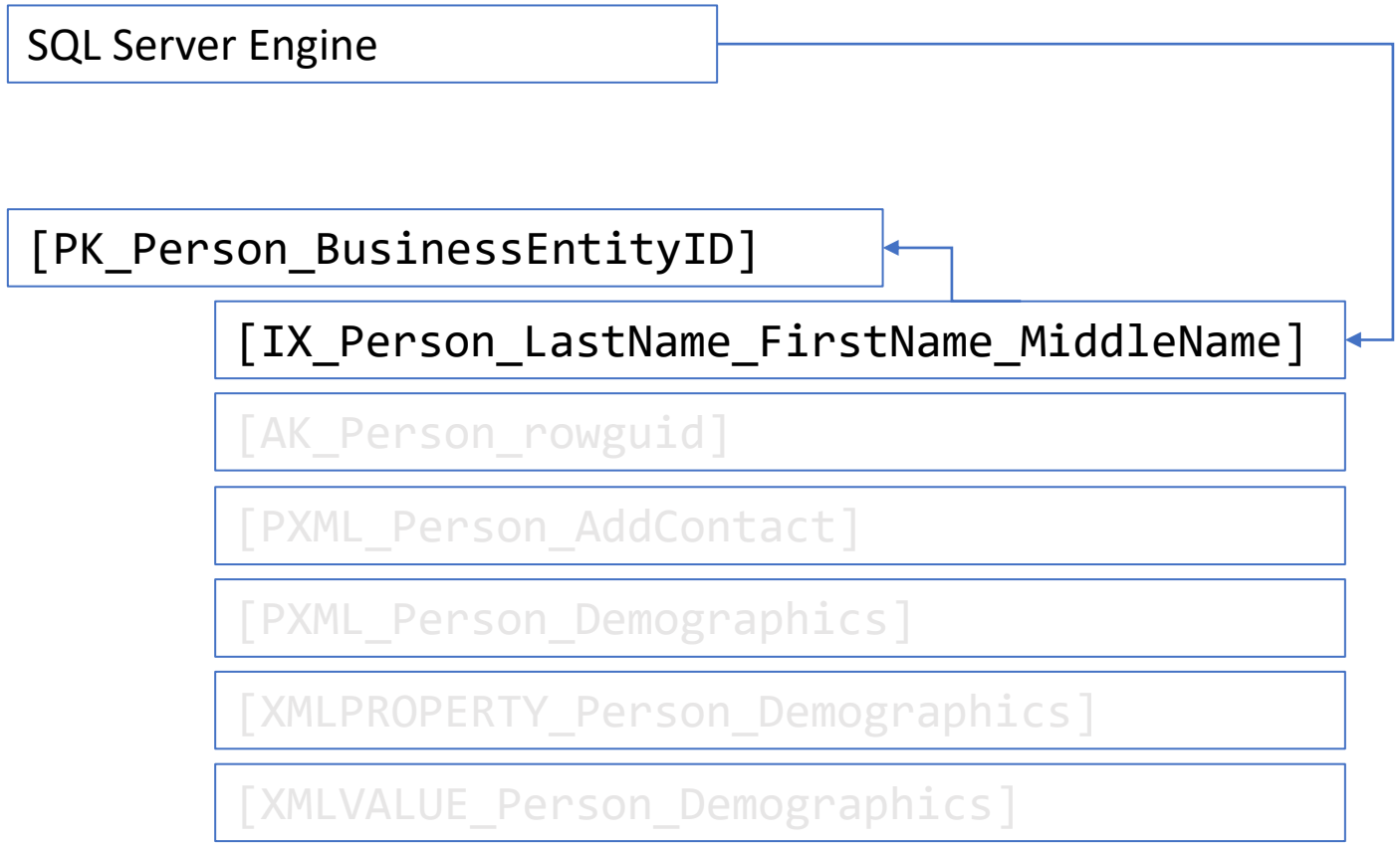
[PXML_Person_Demographics]

[XMLPROPERTY_Person_Demographics]

[XMLVALUE_Person_Demographics]

# Storage Structures

```
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
      ,[AdditionalContactInfo]
      ,[Demographics]
      ,[rowguid]
      ,[ModifiedDate]
 FROM [AdventureWorks2022].[Person].[Person]
 WHERE [FirstName] = 'Jeremiah'
 AND [LastName] = 'Martin'
```



# 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,<br>B Tree,<br>Clustered Column Store,<br>Order Clustered Column Store | Columnal | Columnal |
| Secondary Data Copies | Non Clustered Index,<br>Non Clustered Column Store | | |

# Storage Structures

We can have multiple, transactionally identical structure to meet our use cases

We can index for row level and aggerate queries allow SQL Server to meet this use cases individually or concurrently!

But this is does have its own overheads!


Partitioning in Delta and SQL Server

# What problem are we solving?

- 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 Tables

SQL Server has two type of partitioning.

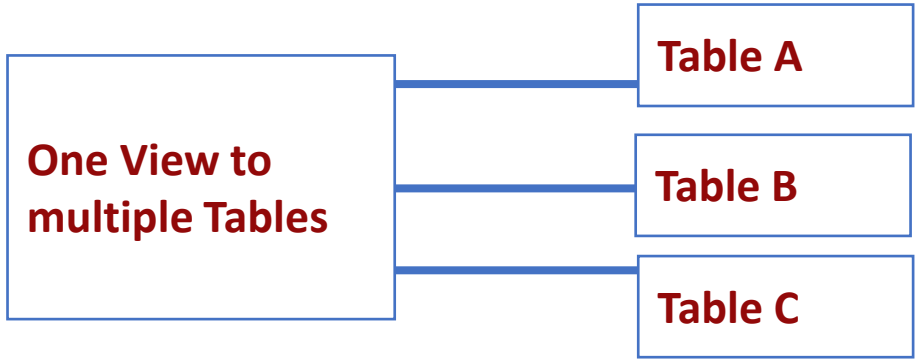
- Partition Table
- Partition Views

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

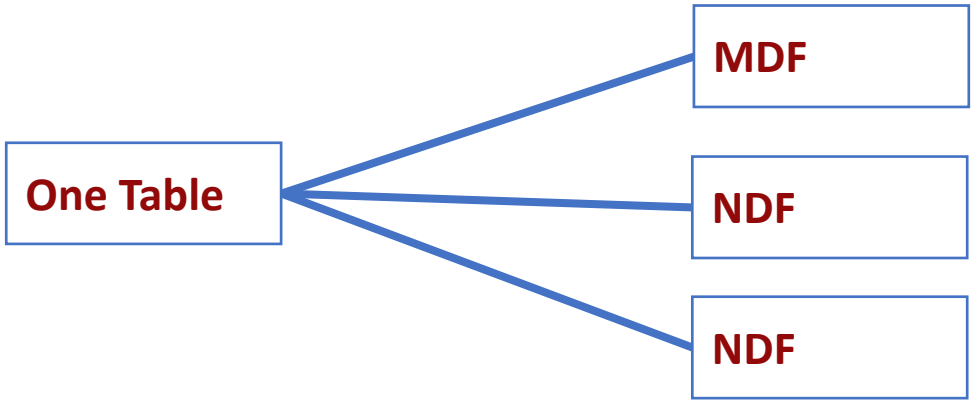
Partition views can be place over tables to partition different keys.

# Partition View and Tables

View

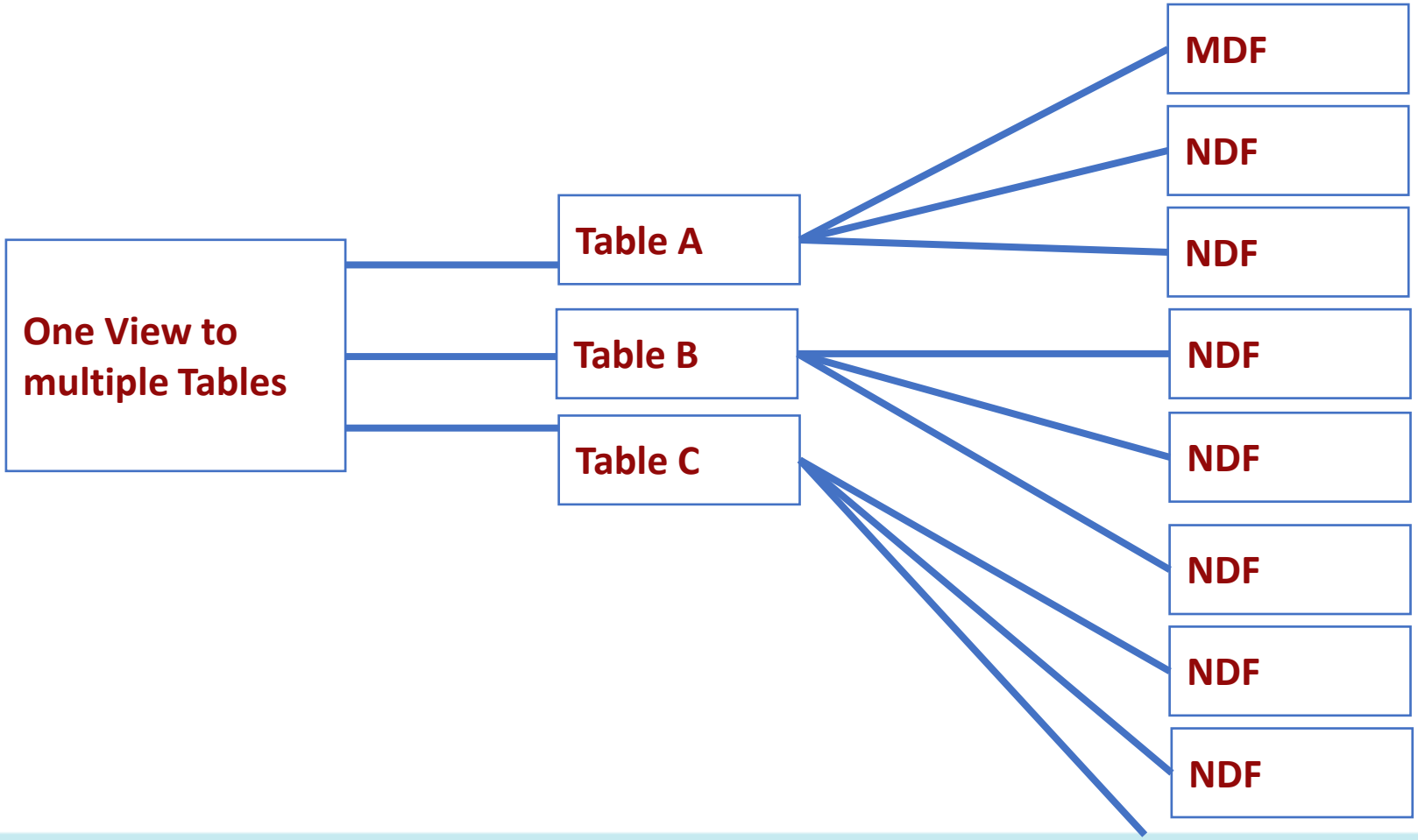



Table




Each table will have its own file group

# Partition View and Tables



# Partition View and Tables

So, lets look at adventure works, let looks at the `[Purchasing].[PurchaseOrderHeader]` table

```
CREATE TABLE [Purchasing].[PurchaseOrderHeader](
[PurchaseOrderID] [int] IDENTITY(1,1) NOT NULL,
[RevisionNumber] [tinyint] NOT NULL,
[Status] [tinyint] NOT NULL,
```

# Partition View and Tables

So, lets look at adventure works, let looks at the [Purchasing].[PurchaseOrderHeader] table

```
CREATE TABLE [Purchasing].[PurchaseOrderHeader_1](
[PurchaseOrderID] [int] INTEGER DEFAULT NEXT VALUE FOR PurchaseOrderHeader_Seq,
[RevisionNumber] [tinyint] NOT NULL,
[Status] [tinyint] NOT NULL CONSTRAINT [PK_PurchaseOrderHeader_Status_1] CHECK ( [Status] = 1 ),
```

# Partition View and Tables

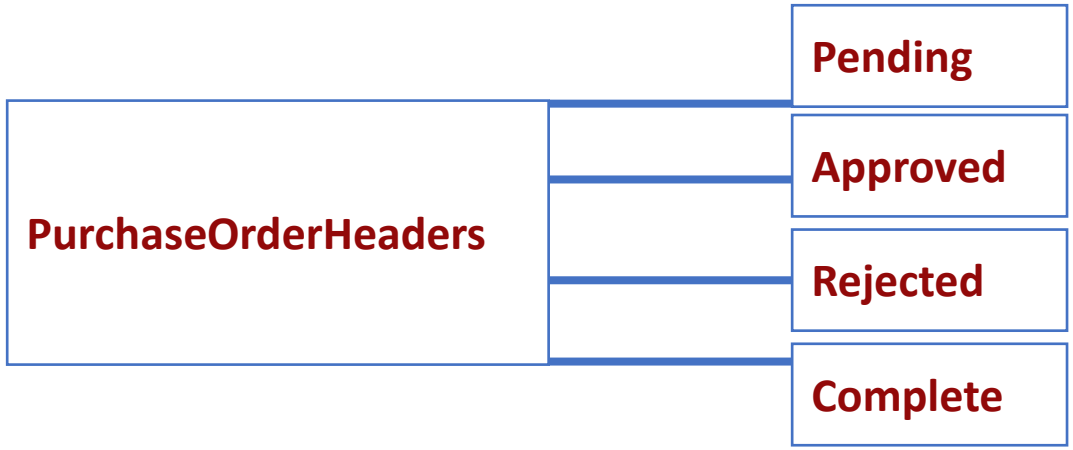
We can start with a partition view, with will be a view that spans the tables

We already have `[PurchaseOrderID]`

We can now also use the `[Status]` so that we can split out, Pending, Approved, Rejected and Complete

# Partition View and Tables

So we will now create these tables with additional constraints so that he data will be in the correct table



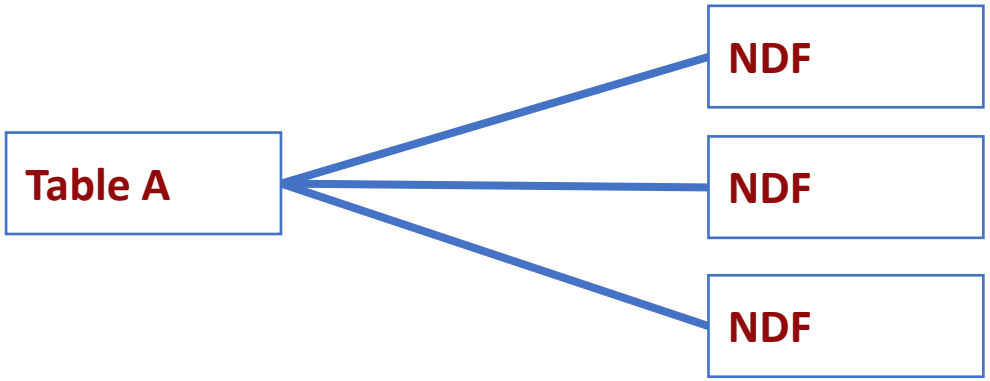
# Partition View and Tables

Demo Time!

# Partition View and Tables

We now have a partition view made of tables, this tables all have the same schema and so we will apply partitioning to them in a group.

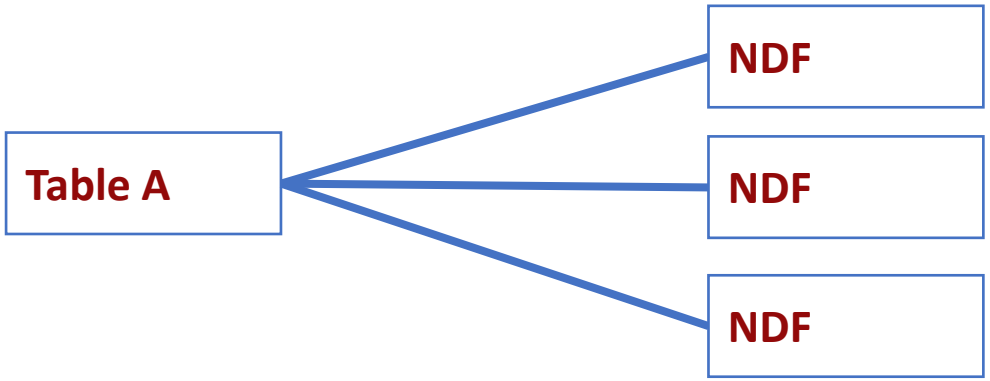
We need to create a Partition Schema and Partition Function



# Partition View and Tables

Our partition Schema is where things can be put!

Our Partition Function how what is going to be based on!



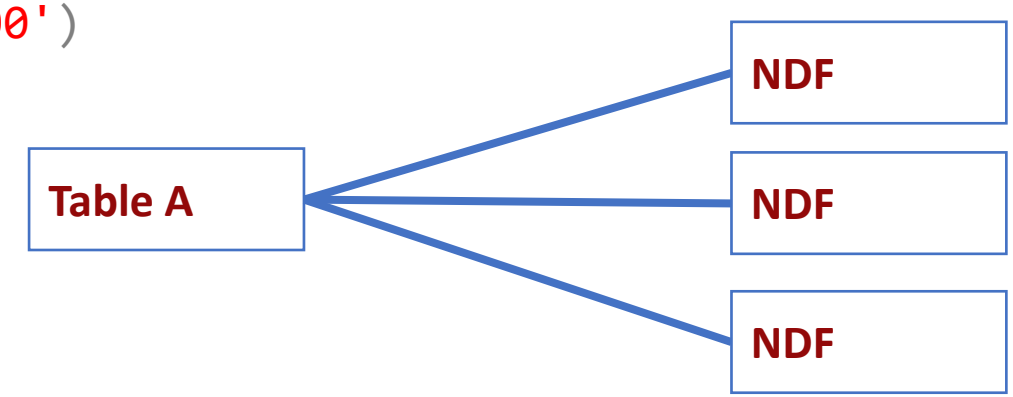
# Partition View and Tables

```
CREATE PARTITION SCHEME [PartSch_Purchasing_PurchaseOrderHeader] AS PARTITION
[PartFun_Purchasing_PurchaseOrderHeader] TO ([Part_2011], [Part_2012], [Part_2013],
[Part_2014])

GO

CREATE PARTITION FUNCTION [PartFun_Purchasing_PurchaseOrderHeader](datetime) AS RANGE
RIGHT FOR VALUES (N'2012-01-01T00:00:00.000', N'2013-01-01T00:00:00.000', N'2014-01-
01T00:00:00.000')

GO
```



# Partition View and Tables

Demo Time!

File Edit View Help

Search

CONNECTIONS

SERVERS

<default> (Inte...

Databases

Security

Server Objects

AZURE

SqlBits-2024-Demo1.ipynb

C: > Users > james > SqlBits-2024-Demo1.ipynb

+ Cell | Run all | Kernel SQL | Attach to AdventurePlay (Integrated)

Create Schema and Sequence

[11]

```sql
IF NOT EXISTS (SELECT * FROM sys.schemas WHERE name = 'Purchasing')
BEGIN
EXEC('CREATE SCHEMA [Purchasing]')
END
/*Create Seq*/
IF EXISTS (SELECT * FROM sys.sequences WHERE name = 'Purchasing')
BEGIN
CREATE SEQUENCE PurchaseOrderHeader_Seq
 AS INTEGER
 START WITH 1
 INCREMENT BY 1
END
/*Drop view fro schema binding*/
IF EXISTS (SELECT * FROM sys.views WHERE name = 'PurchaseOrderHeader')
BEGIN
EXEC('DROP VIEW [Purchasing].[PurchaseOrderHeader]')
END
```

SQL

Commands completed successfully.

Total execution time: 00:00:00.035

Create New tables for Partitioning across status

All tables on Primary as this is part of a Demo

[12]

```sql
/*Demo Starts here*/
/* Create Table*/
IF EXISTS (SELECT * FROM sys.tables WHERE name = 'PurchaseOrderHeader_1')
BEGIN
EXEC('DROP TABLE [Purchasing].[PurchaseOrderHeader_1]')
END

CREATE TABLE [Purchasing].[PurchaseOrderHeader_1](
    [PurchaseOrderID] INTEGER DEFAULT NEXT VALUE FOR PurchaseOrderHeader_Seq,
    [RevisionNumber] [tinyint] NOT NULL,
    [Status] [tinyint] NOT NULL CONSTRAINT [PK_PurchaseOrderHeader_Status_1] CHECK ( [Status] = 1 ),
    [EmployeeID] [int] NOT NULL,
    [VendorID] [int] NOT NULL,
    [ShipMethodID] [int] NOT NULL,
    [OrderDate] [datetime] NOT NULL,
    [ShipDate] [datetime] NULL,
    [SubTotal] [money] NOT NULL,
    [TaxAmt] [money] NOT NULL,
    [Freight] [money] NOT NULL,
    --[TotalDue]  AS (isnull(([SubTotal]+[TaxAmt])+[Freight],(0))) PERSISTED NOT NULL,
    [ModifiedDate] [datetime] NOT NULL,
     CONSTRAINT [PK_PurchaseOrderHeader_PurchaseOrderID_1] PRIMARY KEY CLUSTERED
    (
        [PurchaseOrderID],[Status] ASC
    ) ON [PRIMARY]
) ON [PRIMARY]
GO
```

SQL

Commands completed successfully.

Total execution time: 00:00:00.049

Repeate for `PurchaseOrderHeader_2`, `PurchaseOrderHeader_3` and `PurchaseOrderHeader_4`

# 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.

# Delta Partitioning

```
# 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

No Partitioning

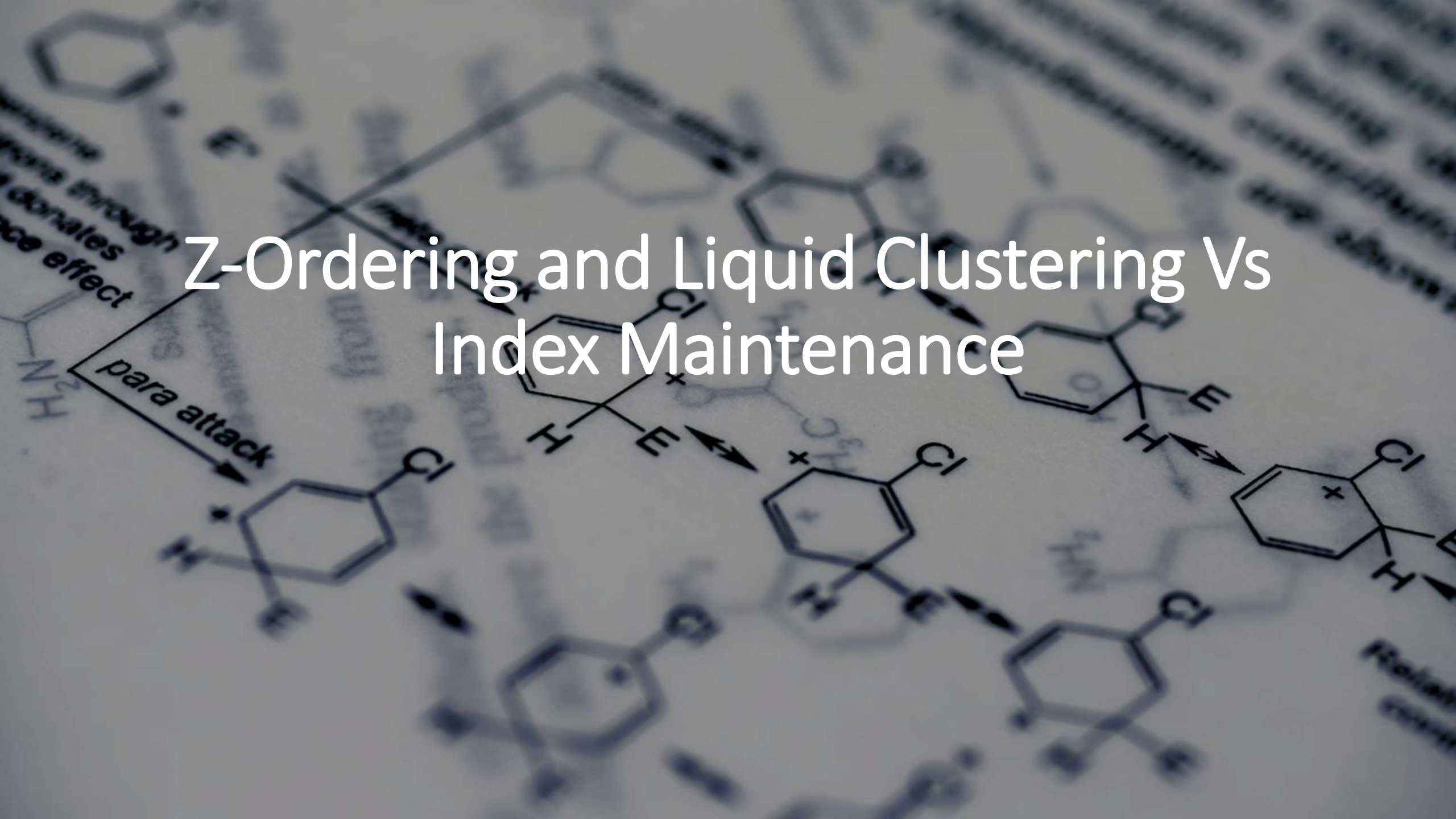
| | path | name | size |
|---|---|---|---|
| 1 | dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/_delta_log/ | _delta_log/ | 0 |
| 2 | dbfs:/mnt/deltalake/MasteringDelta/NYCTaxi/Yellow_TripData/Exported/part-00000-829f8ebc-6be6-4491-a1ba-f9bb93311d5f-c000.snappy.parquet | part-00000-829f8ebc-6be6-4491-a1ba-f9bb93311d5f-c000.snappy.parquet | 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 |

Partitioned

| | 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 |

# Delta Demo

- Partitioning




# Z-Ordering and Liquid Clustering Vs Index Maintenance

# What problem are we solving?

- 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

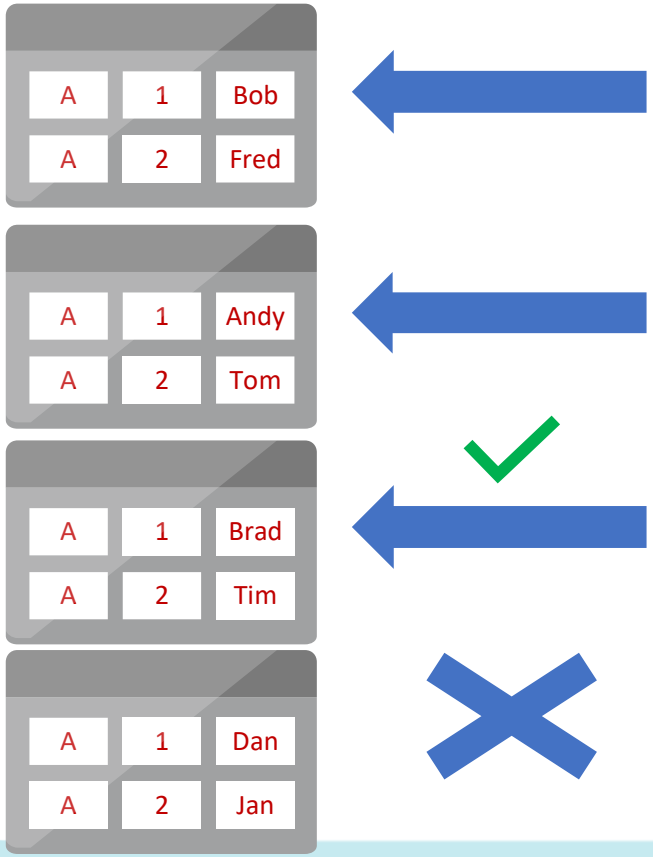
# Z-Ordering

*"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

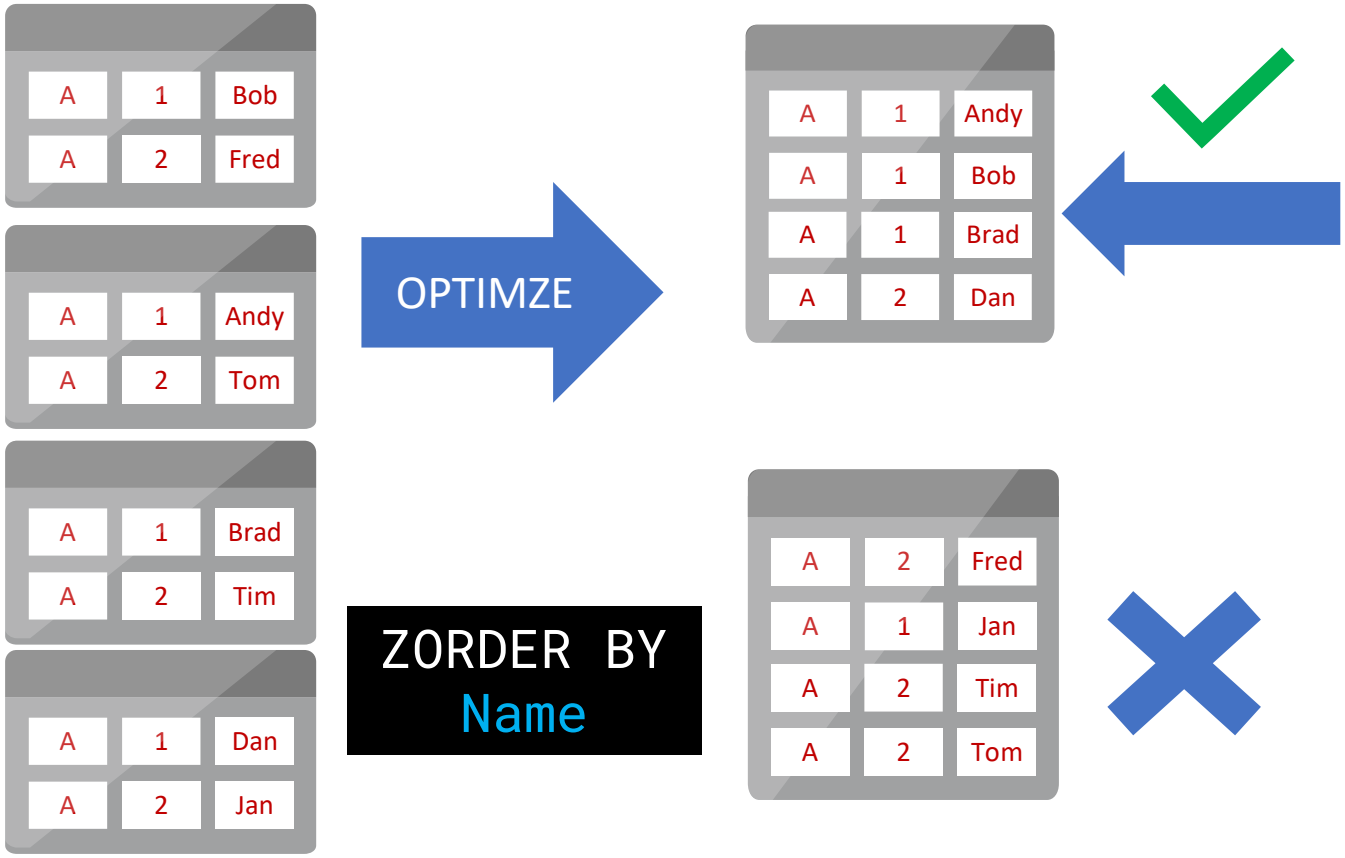
# Z-Ordering Explained




```
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 is similar to a clustered index - organising your data on disk to maximise queries for specific columns

# Demo

- ZOrdering

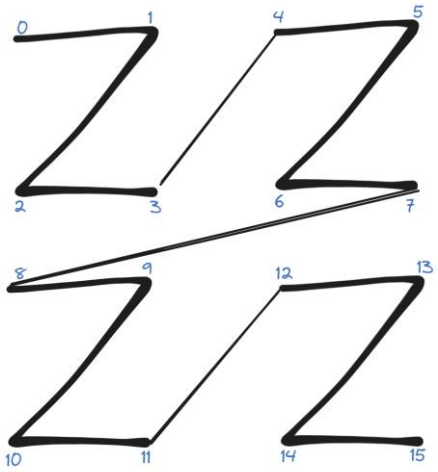
# Databricks Liquid Clustering

Provides flexibility to redefine clustering columns without rewriting existing data, allowing data layout to evolve alongside analytic needs over time.

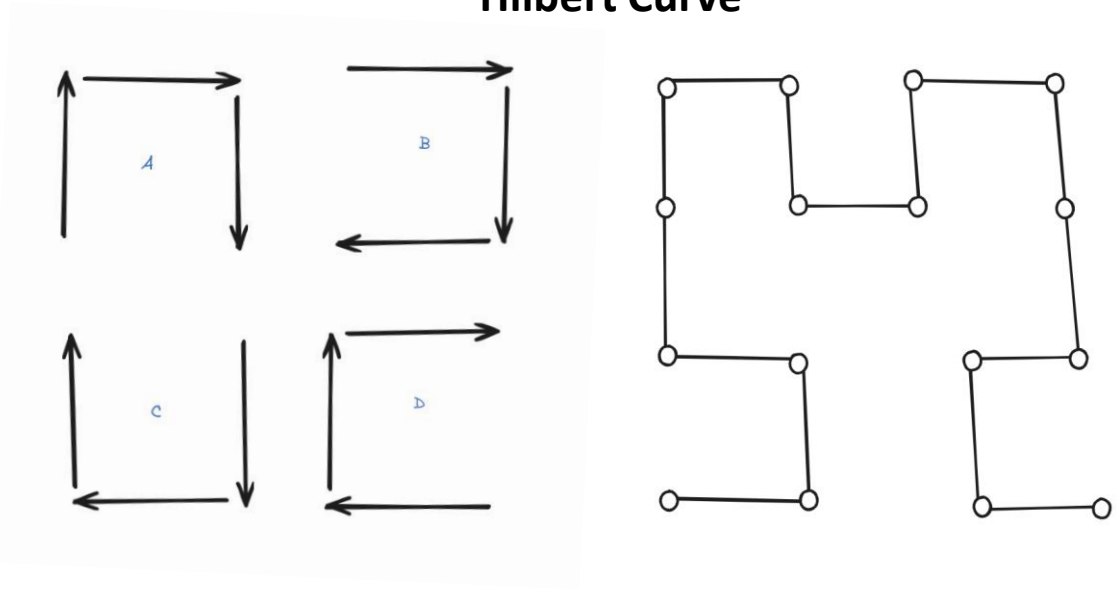
# Hilbert Curve

The Hilbert curve is a way of mapping the multidimensional space into the one-dimensional space

**ZOrdering**




**Hilbert Curve**



# How to Implement

```
CREATE TABLE table1(col0 int, col1 string) USING DELTA CLUSTER BY (col0);

CREATE EXTERNAL TABLE table2 CLUSTER BY (col0) LOCATION 'table_location'
AS SELECT * FROM table1;
```


Indexing in SQL Server

# Z-Ordering is Index Maintenance

The order of the data is set by the Clustered Index

Index Maintenance

- Rebuilds
- Reorganization

# Z-Ordering is Index Maintenance

Lets look at bit deeper into what is happening and why this is needed.

We have two different problem for row-store and column-store indexes as they used the two different structures

Lets start with Row based!

# Z-Ordering is Index Maintenance

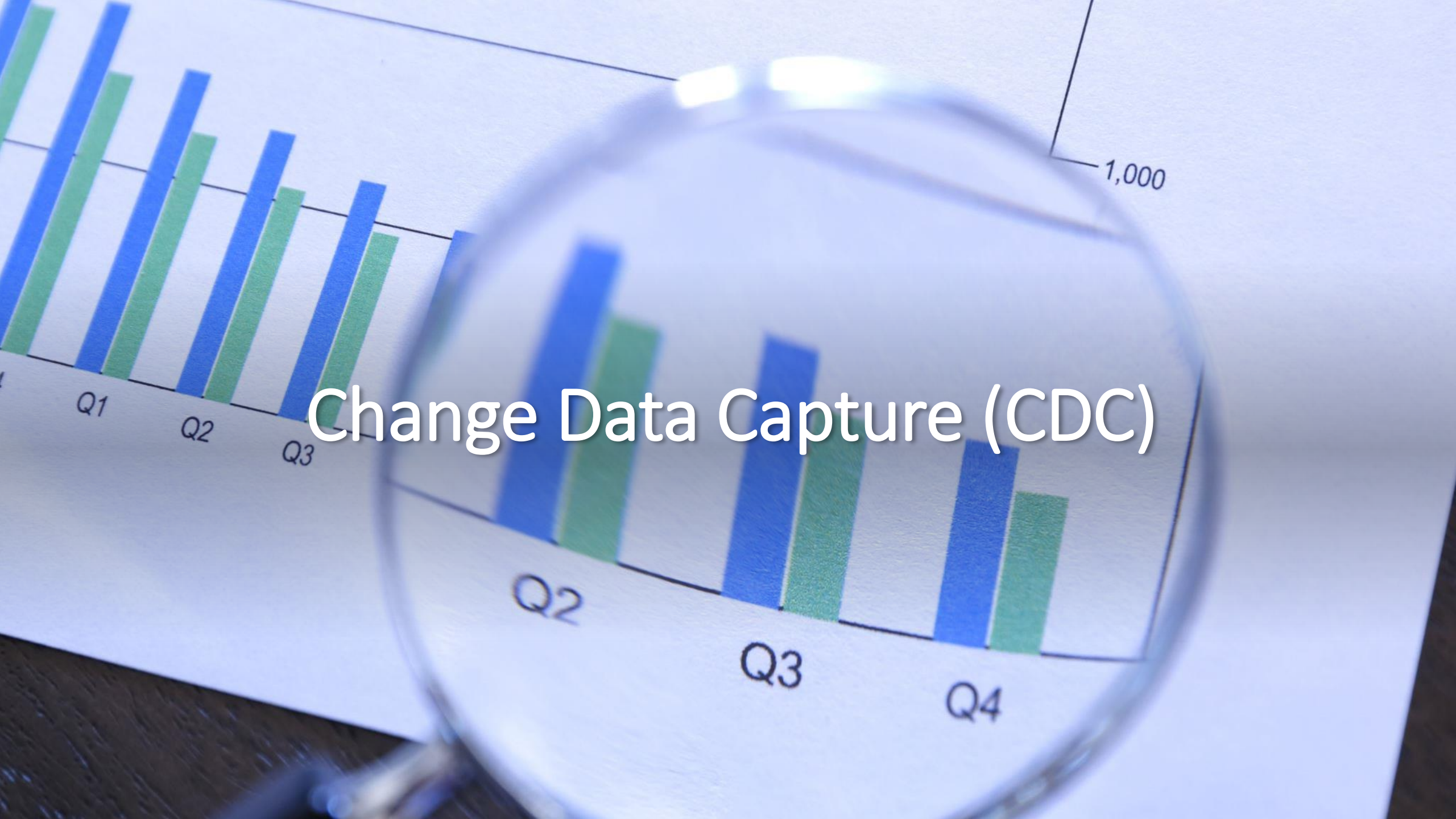
When new records are added to the tables, the indexes are updated!

Lets keep with Person.Person

We have a B-Tree with and see what can happen when we make some changes!

# Z-Ordering is Index Maintenance

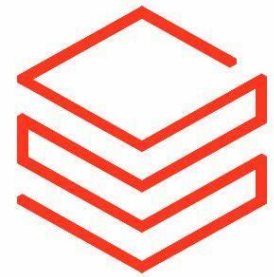
Column-store works in works in a different way.




# Change Data Capture (CDC)

# The only constant is Change!

# What problem are we solving?

| | Can you tell me what has changed? | Can you tell me what this look like at this time? |
|---|---|---|
|  | Change Data Capture<br>Change Tracking* | Temporal Tables |
|  | Change Data Capture | Delta Time Travel |

# Change Data Capture in SQL Server

sqlbits
***2024***

# Change Data Capture

We can track changes of tables by enabling CDC at a database level.

We then need to add each table.

Once this is done, we can see what has change and what it was previously.

Let take a look at Person.Person

Retention, These can get big, Log and table space!

# Change Data Capture

```
/*Enable on the database */

EXEC sys.sp_cdc_enable_db
    GO

/*Enable on the table, Requires a PK */

EXEC sys.sp_cdc_enable_table
    @source_schema = N'Person',
    @source_name   = N'PersonCDC',
    @role_name     = NULL,
    @supports_net_changes = 1
    GO
```

# Change Data Capture

```
/* Select with cdc function */

SELECT[__$start_lsn]
      ,[__$end_lsn]
      ,[__$seqval]
      ,[__$operation]
      ,[__$update_mask]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[ModifiedDate]
  ,sys.fn_cdc_map_lsn_to_time(__$start_lsn) AS 'CDCModifiedDate'
  FROM [AdventurePlay].[cdc].[Person_PersonCDC_CT]
```

# Change Data Capture

```
UPDATE [Person].[PersonCDC]
  SET [FirstName] = 'James',
  [MiddleName] = 'C',
  [LastName] = 'Yarrow'
  WHERE [BusinessEntityID] = 4000
```

Results | Messages

| | __$start_lsn | __$end_lsn | __$seqval | __$operation | __$update_mask | FirstName | MiddleName | LastName | ModifiedDate | CDCModifiedDate |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0x0000003100014E9A0003 | NULL | 0x0000003100014E9A0002 | 3 | 0x0070 | Jeremiah | E | Martin | 2014-02-02 00:00:00.000 | 2024-02-26 18:32:27.410 |
| 2 | 0x0000003100014E9A0003 | NULL | 0x0000003100014E9A0002 | 4 | 0x0070 | James | C | Yarrow | 2014-02-02 00:00:00.000 | 2024-02-26 18:32:27.410 |

# Change Data Capture in Databricks

# Change Data Capture (CDC)

Databricks has a feature to handle CDC called Change Data Feed  (CDF), which allows Databricks to track row-level changes between versions of a Delta table

# How does it work

**Original Table**

| PK | B |
|---|---|
| A1 | B1 |
| A2 | B2 |
| A3 | B3 |

**Change Data**

| PK | B |
|---|---|
| A1 | Z2 |
| A3 | B3 |
| A4 | B3 |

**Change Data Feed Output**

| PK | B | Change Data | Time | V |
|---|---|---|---|---|
| A1 | B2 | Preimage | 12:00 | 2 |
| A1 | Z2 | Postimage | 12:00 | 2 |
| A3 | B3 | Delete | 12:00 | 2 |
| A4 | B4 | Insert | 12:00 | 2 |

**We have four change types:**

- **Preimage**
- **Postimage**
- **Delete**
- **Insert**

# How to Implement

```
CREATE TABLE student (id INT, name STRING, age INT) TBLPROPERTIES
(delta.enableChangeDataFeed = true)
```

```
ALTER TABLE myDeltaTable SET TBLPROPERTIES (delta.enableChangeDataFeed =
true)
```

```
set spark.databricks.delta.properties.defaults.enableChangeDataFeed = true;
```

# How to Read

```
# version as ints or longs
spark.read.format("delta")
.option("readChangeFeed", "true")
.option("startingVersion", 0)
.option("endingVersion", 10)
.table("myDeltaTable")
```

```
# timestamps as formatted timestamp
spark.read.format("delta")
.option("readChangeFeed", "true")
.option("startingTimestamp", '2021-04-21 05:45:46’)
.option("endingTimestamp", '2021-05-21 12:00:00')
.table("myDeltaTable")
```

# Delta History and Time Travel vs Temporal Tables

# What problem are we trying to solve?

Can you tell me what this look like at this time?

# SQL Server Temporal Tables

sqlbits
***2024***

# Temporal Tables

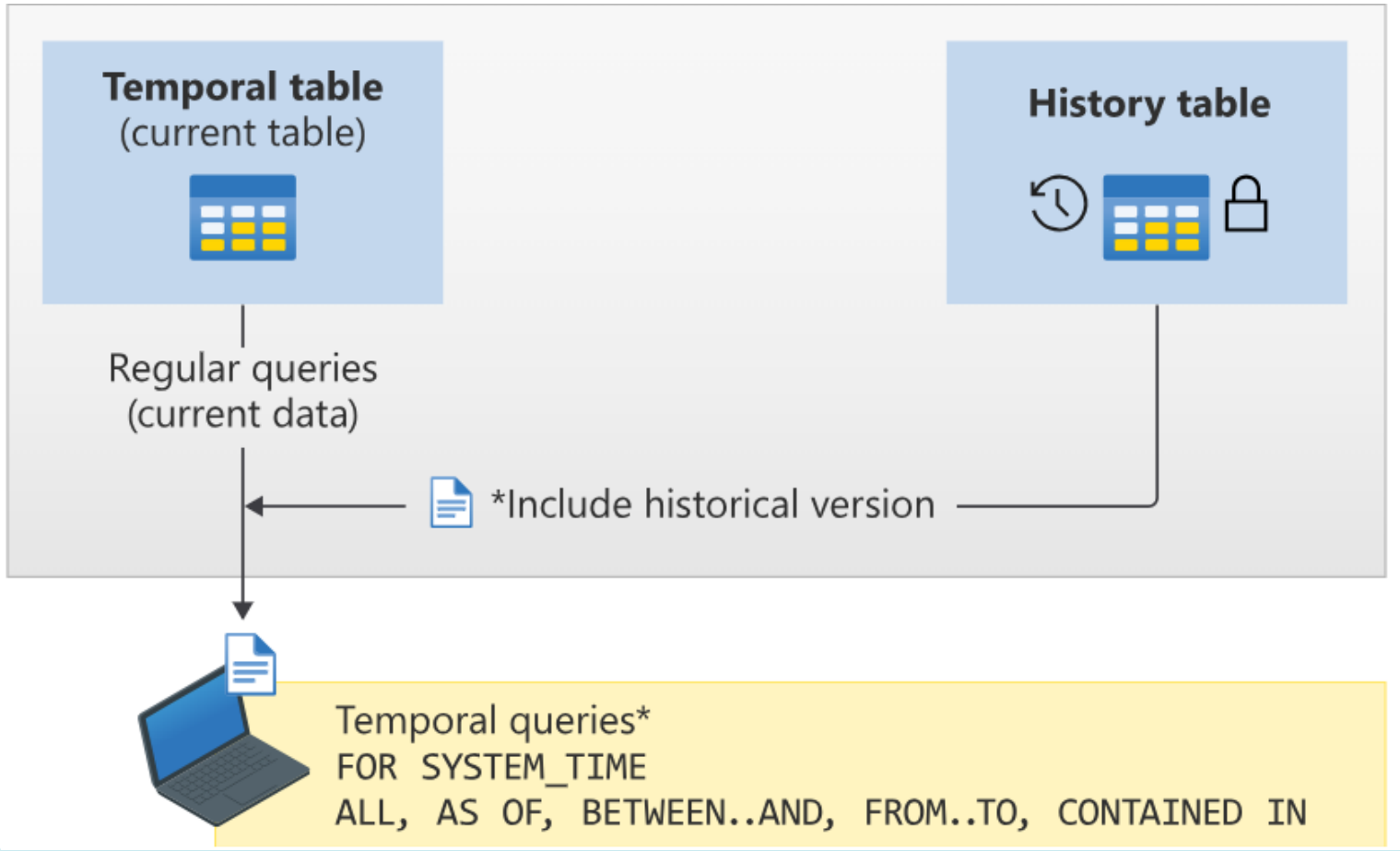
- Allow you to keep history of data changes
- Point in time analysis
- Used for auditing
- Retention

# How do they work?

- Columns to define the period of validity
  - Period Start column
  - Period End Column

- History table

# How can you query the data?



# Demo

File Edit View Help

Demos (Workspace)

Welcome | DemoScripts.ipynb | SQLvsDeltaBricks.ipynb

C: > Users > annet > OneDrive > Presentations > DeltaVsSQLServer > SQLvsDeltaBricks.ipynb

+ Cell | Run all | Kernel SQL | Attach to NOODLE\BRANDS, Adventur

To edit a current table

- Add in the 2 Period start and end columns
- State the history table

```sql
Alter table Person.Person
    Add
        PeriodFrom datetime2 GENERATED ALWAYS AS ROW START NOT NULL
        DEFAULT SYSUTCDATETIME()
      , PeriodTo datetime2 GENERATED ALWAYS AS ROW END NOT NULL
        DEFAULT CAST('9999-12-31 23:59:59.9999999' AS datetime2),
    PERIOD FOR SYSTEM_TIME (PeriodFrom,PeriodTo);
```

SQL

```sql

Alter table Person.Person
set
    (SYSTEM_VERSIONING = ON
        (HISTORY_TABLE = Person.Person_history))

```

SQL

Now lets look at the data in the table to see what it looks like

PROBLEMS 2 | OUTPUT | TERMINAL | TASKS

Tasks

⊗ 2 ⚠ 0

# Limitations of Temporal Tables

- History tables have to be in the same database as the source table
- You can only delete from temporal tables you can't truncate them
- You can't make any changes to the history table
- History tables can't have any constraints e.g. primary or foreign keys

# Delta History and Time Travel in Databricks

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

```
-- 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 |

# Delta Time Travel Demo

- History
- Version

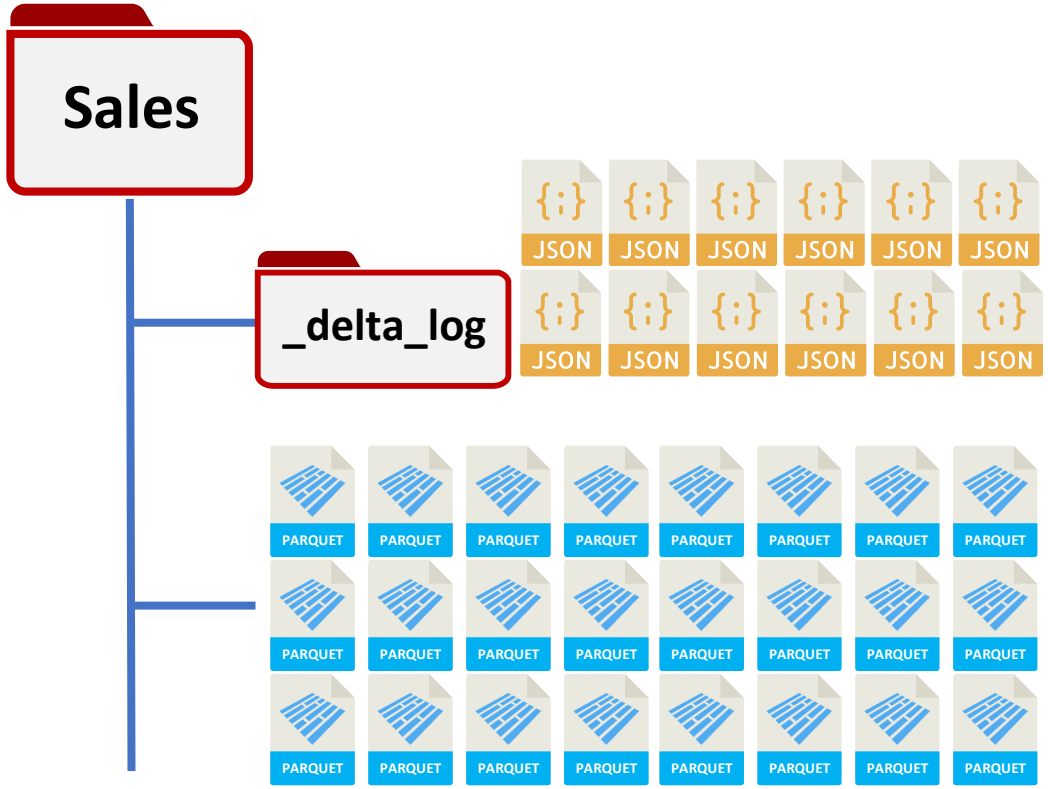
# Delta Vacuum vs SQL Transactions

# What problem are we trying to solve?

Clean up old data

# Delta Vacuum

# Obsolete Files




Removing historical 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

# SQL Server Retention

# Data Retention

- Audit Logs
- Temporal Tables
- Backups
- Change Data Capture

# Primary/Foreign Keys and Constraints

# What problem are we solving?

- 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

**Governance Layer**

databricks

Unity Catalog

# Delta Constraints – Unity Catalog

- Primary Keys
- Foreign Keys
- Identity Columns
- Not Null
- Unique
- Check

# Delta Constraints – Databricks Only

**Primary Keys / 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>);
```

# SQL Primary and Foreign keys

# Primary and Foreign Keys

SQL Server Primary Key

- One per table and Always Unique
- At least one Column
- Not Null
- Just a constraint
- Can be extracted for your data checks down the pipeline

SQL Server will often create your cluster index on your PK


Databricks Security vs SQL Security

# SQL Server Security

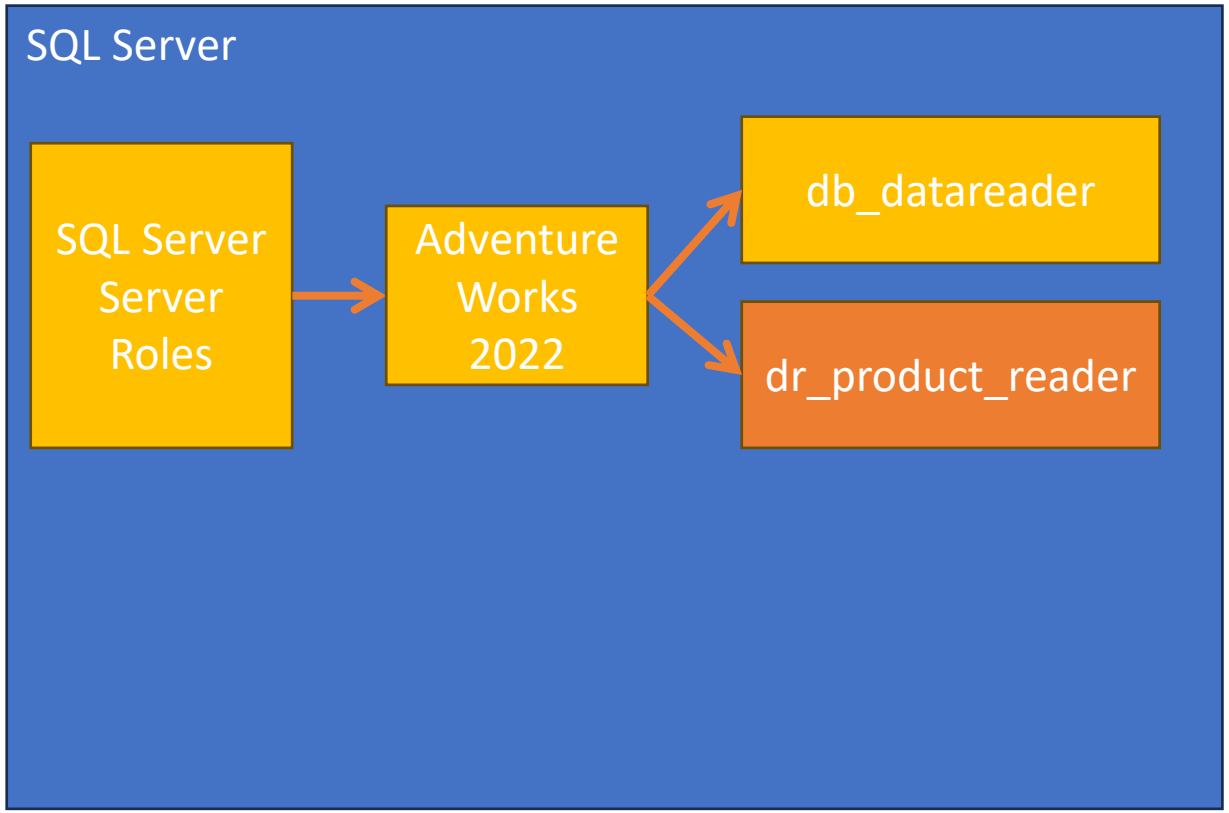
sqlbits
***2024***

# SQL Server Security

- All (Most) Data is stored with in the database
- Access is control via SQL Server
- SQL Server can be it only authority with SQL Logins
- SQL Server hold the file locks at all time when started


# SQL Server Security

- SQL Server have a server within this databases
- Server Roles are applied, then database roles are applied if needed
- Database roles have built-in Roles



# SQL Server Security

We can map ~~Azure Active Directory~~ Entra ID, First we map to a Login at the Server level.

```
USE [master]
GO
--CREATE LOGIN [EntraIDName] FROM EXTERNAL PROVIDER
CREATE LOGIN [MaskingTestUser] WITHOUT LOGIN
```

We then need to create a User at the Database level.

```
USE [AdventureWorks2022]
GO
CREATE USER [MaskingTestUser] FOR LOGIN [MaskingTestUser]
GO
ALTER ROLE [dr_sales_reader] ADD MEMBER [MaskingTestUser]
GO
```

# SQL Server Security

There are built in roles that can grant read across the database [db_reader] but lets limit this to the sales schema

```
USE [AdventureWorks2022]
GO
CREATE ROLE [dr_sales_reader]
GO
GRANT SELECT ON SCHEMA::[Sales] TO [dr_sales_reader]
GO
```

# SQL Server Security

We can then test this with the user we created.

```
EXECUTE AS USER = 'MaskingTestUser';
SELECT TOP (10) *
FROM [Sales].[CreditCard];
SELECT TOP (10) *
FROM [Person].[Person];
REVERT;
```

| | CreditCardID | CardType | CardNumber | ExpMonth | ExpYear | ModifiedDate |
|---|---|---|---|---|---|---|
| 1 | 1 | SuperiorCard | 33332664695310 | 11 | 2006 | 2013-07-29 00:00:00.000 |
| 2 | 2 | Distinguish | 55552127249722 | 8 | 2005 | 2013-12-05 00:00:00.000 |
| 3 | 3 | ColonialVoice | 77778344838353 | 7 | 2005 | 2014-01-14 00:00:00.000 |
| 4 | 4 | ColonialVoice | 77774915718248 | 7 | 2006 | 2013-05-20 00:00:00.000 |
| 5 | 5 | Vista | 11114404600042 | 4 | 2005 | 2013-02-01 00:00:00.000 |
| 6 | 6 | Distinguish | 55557132036181 | 9 | 2006 | 2014-04-10 00:00:00.000 |
| 7 | 7 | Distinguish | 55553635401028 | 6 | 2007 | 2013-02-01 00:00:00.000 |
| 8 | 8 | SuperiorCard | 33336081193101 | 7 | 2007 | 2013-06-30 00:00:00.000 |
| 9 | 9 | Distinguish | 55553465625901 | 2 | 2005 | 2013-09-23 00:00:00.000 |
| 10 | 10 | SuperiorCard | 33332126386493 | 8 | 2008 | 2011-08-31 00:00:00.000 |

```
Msg 229, Level 14, State 5, Line 15
The SELECT permission was denied on the object 'Person',
database 'AdventureWorks2022', schema 'Person'.
```

# SQL Server Security

We can also apply data masking at the column level.

```
ALTER TABLE [Sales].[CreditCard]
ALTER COLUMN [CardNumber] ADD MASKED WITH (FUNCTION = 'default()’);
```

This can be tested with a local user for test which can also but put in your pipeline!

```
EXECUTE AS USER = MaskingTestUser';
SELECT TOP (10) *
FROM [Sales].[CreditCard];
REVERT;
```

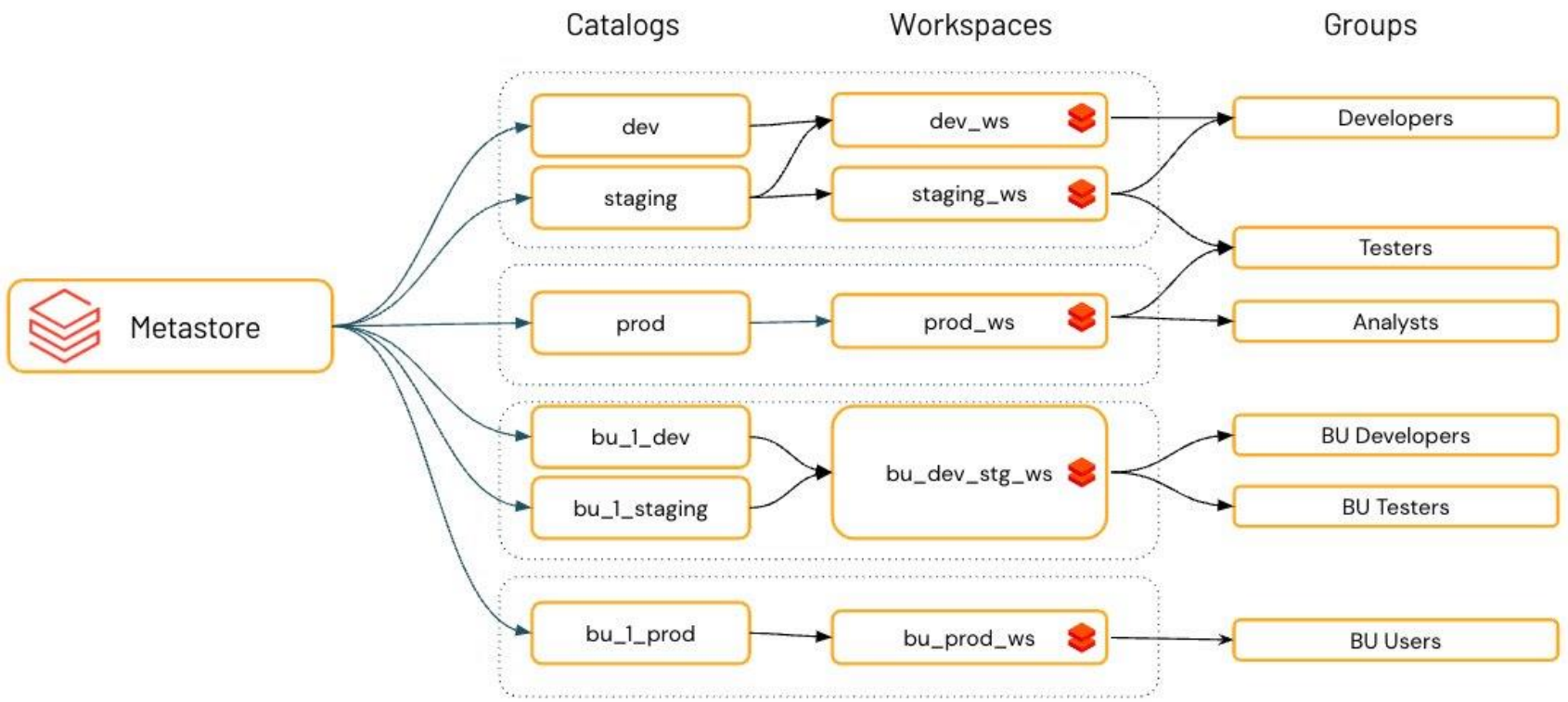
| | CreditCardID | CardType | CardNumber | ExpMonth | ExpYear | ModifiedDate |
|---|---|---|---|---|---|---|
| 1 | 1 | SuperiorCard | xxxx | 11 | 2006 | 2013-07-29 00:00:00.000 |
| 2 | 2 | Distinguish | xxxx | 8 | 2005 | 2013-12-05 00:00:00.000 |
| 3 | 3 | ColonialVoice | xxxx | 7 | 2005 | 2014-01-14 00:00:00.000 |
| 4 | 4 | ColonialVoice | xxxx | 7 | 2006 | 2013-05-20 00:00:00.000 |
| 5 | 5 | Vista | xxxx | 4 | 2005 | 2013-02-01 00:00:00.000 |
| 6 | 6 | Distinguish | xxxx | 9 | 2006 | 2014-04-10 00:00:00.000 |
| 7 | 7 | Distinguish | xxxx | 6 | 2007 | 2013-02-01 00:00:00.000 |
| 8 | 8 | SuperiorCard | xxxx | 7 | 2007 | 2013-06-30 00:00:00.000 |
| 9 | 9 | Distinguish | xxxx | 2 | 2005 | 2013-09-23 00:00:00.000 |
| 10 | 10 | SuperiorCard | xxxx | 8 | 2008 | 2011-08-31 00:00:00.000 |

# Databricks (UC) Security

# Previously

- Secured Data on ADLS instance via  Access Control Lists (ACLs)
- Couldn't sync AD (Active Directory) groups to Databricks

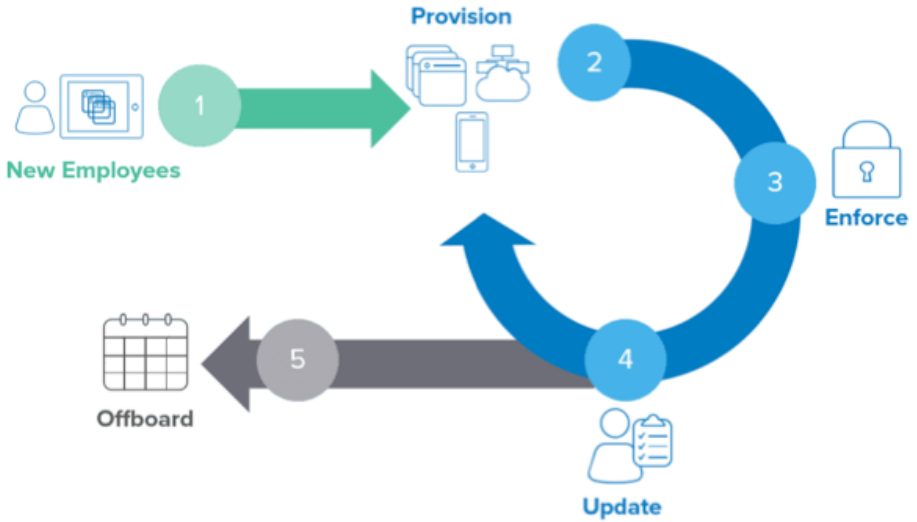
# Security in Unity Catalog



# SCIM Connector

System for Cross domain Security management

We can do this Using Microsoft Entra ID (formerly Azure Active Directory) when working with Unity Catalog



# UC Security Examples

```
-- Let's grant all users a SELECT
GRANT SELECT ON TABLE customers TO `account users`;

-- We'll grant an extra MODIFY to our Data Engineer
-- Note: make sure you created the dataengineers group first as an account admin!
GRANT SELECT, MODIFY ON TABLE customers TO `dataengineers`;
```

# UC Security Examples

```
-- get the current user (for informational purposes)
SELECT current_user(), is_account_group_member('account users');


-- Apply access rule to customers table.
-- country will be the column sent as parameter to our SQL function (region_param)
ALTER TABLE customers SET ROW FILTER region_filter ON (country);
```

# UC Security Examples

```
CREATE OR REPLACE FUNCTION region_filter(region_param STRING)
RETURN
  is_account_group_member('bu_admin') or  -- bu_admin can access all regions
  region_param like "US%";                -- non bu_admin's can only access regions containing US

-- Grant access to all users to the function for the demo by making all account users owners.  Note: Don't do this in production!
GRANT ALL PRIVILEGES ON FUNCTION region_filter TO `account users`;
```

# Summary

- We have seen the modern challenges to data platforms
- We have different approaches to Compute and Storage and the additional complexity of Databricks
- We can have ACID transaction and Partitioning in both
- Both have Ordering data on a granular row by row level for more optimal reading
- Both have point in time querying and changed data capture
- We have a lot of overlap these days, but this also means they can play well
- Similarities in security approaches but still very different components to secure
- We have different advantages but also limitations

Thank you all!

Please fill out the online feedback!

This has dropped off  in our completions rate, since we have switched away for paper feedback.

Don’t make us kill the trees, Please do the online feedback!

https://sqlb.it/?10976

https://github.com/AnnaWykes/delta-dbx-for-dba