All about Data Engineering

Azure Databricks Power Guide

170+ Pages

Azure Databricks Mastery:

Hands-on project

with

Unity Catalog,

Delta Lake,

Medallion Architecture vs 27

Azure Databricks Free notes

End to end project with Unity Catalog

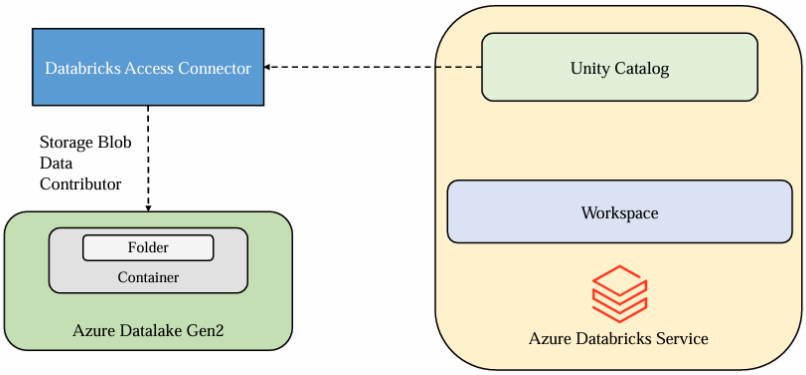
Azure Databricks Mastery: Hands-on project with Unity Catalog, Delta lake, Medallion Architecture

Day 1: Create DataBricks resource using Azure Portal

Environment Setup: Login to your Azure Portal

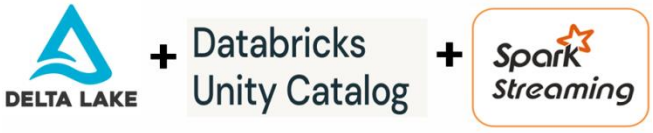
Step 1: Creating a budget for project: search and type budget, "ADD" on Cost Management, "Add Filter" in "Create budget", select Service Name: Azure Databricks in drop down menu.

Step 2: Set alerts as well in next step. Finally click on "Create".



Step 3: Create a Databricks resource, for "pricing tier", click here for more details: https://azure.microsoft.com/en-us/pricing/details/databricks/

Hence select for Premium (+ Role based access controls), skip "Managed Resource Group Name", not any changes required in "Networking", "Encryption", "Security", "Tags" also.



Continuous Integration + Continuous Deployment

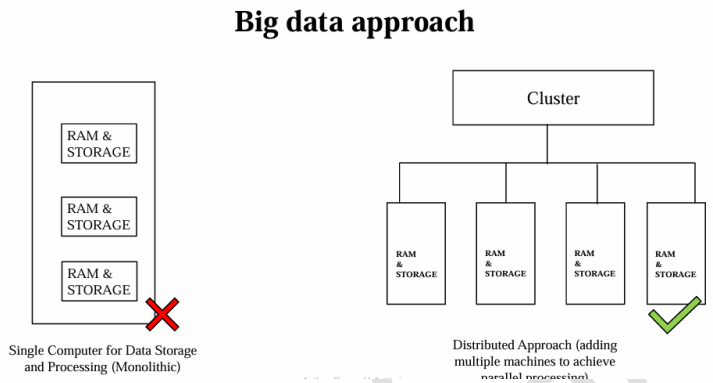
Step 4: Create a "Storage Account" from "Microsoft Vendor", select "Resource Group" as previous one, "Primary Service" as "ADLS Gen 2", select "Performance" as "standard", "Redundancy" as "LRS", not any changes required in "Networking", "Encryption", "Security", "Tags" also.

Step 5: Walkthough on databricks Workspace UI: click on "Launch Workspace" or go through URL: looks like https://_____azuredatbricks.net, Databricks keep updating UI, click on "New" for "Repo"

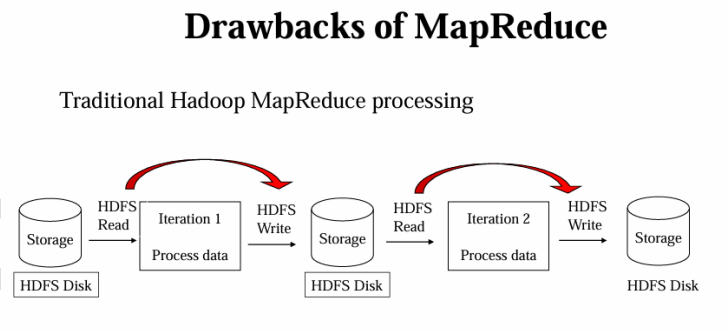
as CI/CD, “Add data” in “New”, “Workflow” are just like Pipeline at high level, “Search” bar for searching also.

- **Databricks admin types: https://learn.microsoft.com/en-us/azure/databricks/admin/**

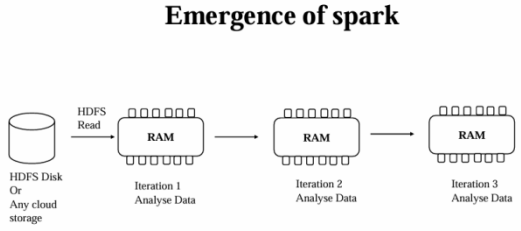
Theory 1: What is Big Data approach?: Monolithic is used for Single Computer and distributed Approach using Cluster which is group of computers.



Theory 2: Drawbacks of MapReduce: In HDFS, in the each iteration, Read and Write operation from disk which will take place high I/O disk costs, developer also have to write complex program, Hadoop is only single super Computer.



Theory 3: Emergence of Spark: First it uses HDFS or Any cloud Storage then further process takes place in RAM, it uses in-memory process which is 10-100 times faster than Disk based application, here database is detached from memory and process aloof.



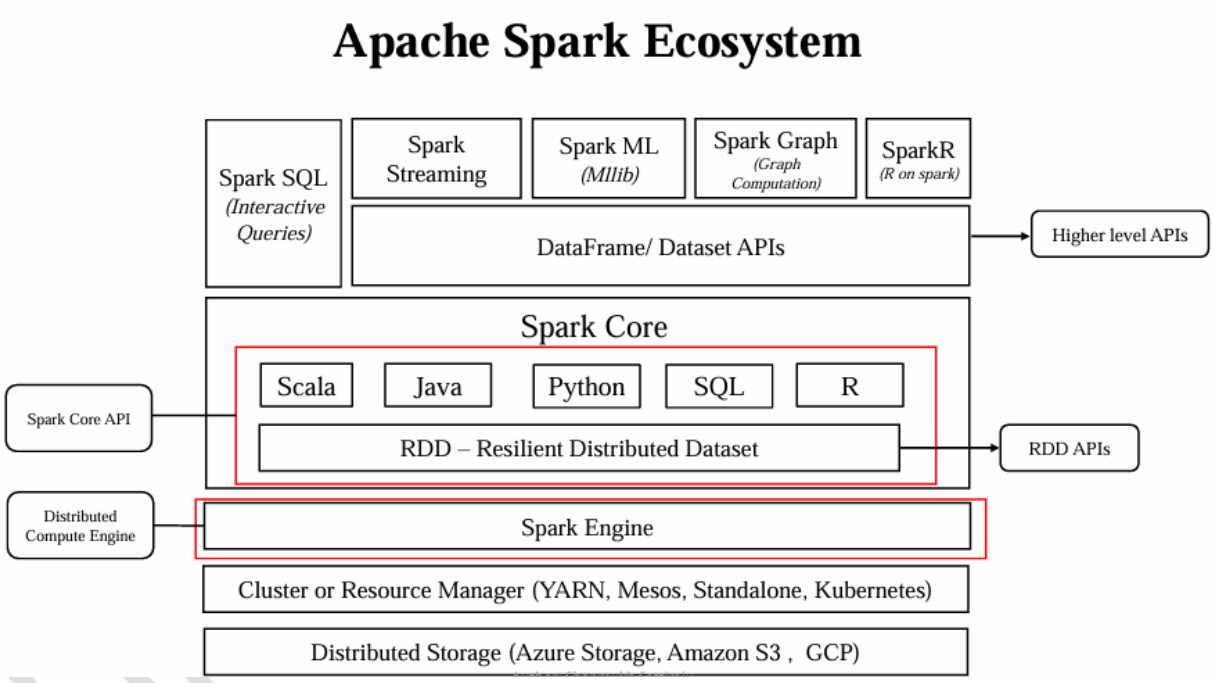
Theory 4: Apache Spark: it is an in-memory application framework.

Apache Spark

Apache Spark is an **open** source **in-memory** application framework for **distributed data processing** and iterative analysis on massive data volumes

In simple terms, Spark is a
- Compute Engine
- Unified data processing System

Theory 5: Apche Spark Ecosystem: Spark Core, special data structure RDD, this is collection of items distributed across the compute nodes in the cluster, these will be processed in parallel, but RDDs are difficult to use for complex operations and they are difficult to optimize , now we are making use of Higher level APIs and libraries like Data Frames and Data Set APIs. Also, uses other high level APIs like Spark SQL, Spark Streaming, Spark ML etc.



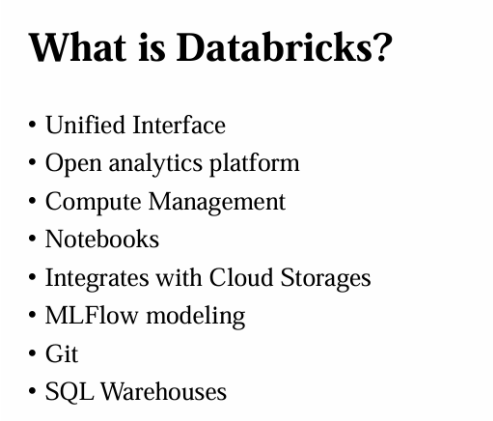
In the real time, we do not use RDD but higher level APIs to do our programming or coding, data frame APIs to interact with spark and these data frames can be invoked using any languages like Java, Python, SQL or R and internally spark has two parts: set of core APIs, and the Spark Engine: this distributed Computing engine is responsible for all functionalities, there is an OS which will manage this group of computers (cluster) is called Cluster Manager, In Spark, there are many Cluster Managers in which you can use like YARN Resource Manager or Resource standalone, Mesos or Kubernetes.

So, Spark is a distributed data processing solution not a storage system, Spark does not come with storage system, can be used like Amazon S3, Azure Storage or GCP.

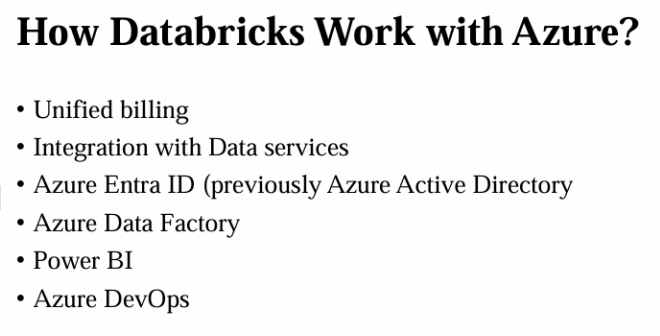
We have Spark Context, which is Spark Engine, to break down the task and scheduling the task for parallel execution.

So, what is Databricks? The founders of the Spark developed a commercial product and this is called Databricks to work with Apache Spark in more efficient way, Databricks is available on Azure, GCP and AWS also.

Theory 6: What is Databricks?: DB is a way to interact with Spark, to set up our own clusters, manage the security, and use the network to write the code. It provides single interface where you can manage data engineering, data science and data analyst workloads.

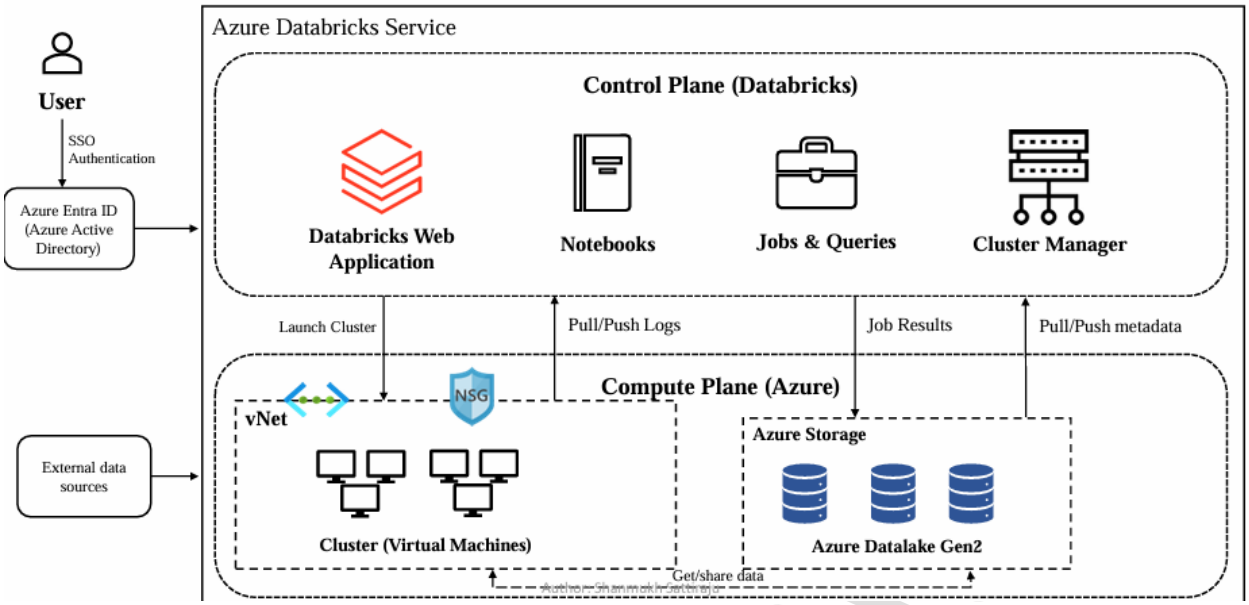


Theory 7: How Databricks Works with Azure? DB can integrate with data services like Blod storage, Data Lake Storage and SQL Database and security Entra ID, Data Factory, Power BI and Azure DevOps.



Theory 8: Azure Databricks Architecture: Control plane is taken care by DB and Compute Plane is taken care by Azure.

Azure Databricks Architecture



Theory 9: Cluster Types: All purpose Cluster and Job cluster. Multi-node cluster is not available in Azure Free subscription because it's allowed to use only maximum of four CPU cores.

Azure Databricks Compute

- Cluster is a set of computation resources and configurations to run your workloads
- Workloads can be:
  1. Set of commands in a notebook
  2. A job that you run as a automated workflow
- Cluster types:
  1. **All purpose Cluster**
     - To execute set of commands in a notebook
  2. **Job Cluster**
     - To execute a job that you run as a automated workflow

In DB workspace: (inside Azure Portal), "create cluster", select "Multi-node": Driver node and worker node are at different machines. In "Access mode", if you will select "No isolation shared" then "Unity Catalogue" is not available. Always uncheck "Use Photon Acceleration" which will reduce your DBU/h, can be seen from "Summary" pane at right top.

Cluster Types

1. **All purpose Cluster**
   - To interactively run the commands in your notebook
   - Multiple users can share such clusters to do collaborative interactive analysis.
   - You can terminate, restart, attach , detach these clusters to multiple notebooks
   - You can choose
     - Multi-node cluster = Driver node and executor nodes will be on separate machine
     - Single node cluster = Only there will be a single driver node with single machine
2. **Job Cluster**
   - To run a job that you run as a automated workflow
   - It runs a new job cluster and terminates the cluster automatically when the job is complete.
   - You cannot restart a job cluster.

Theory 10: Behind the scenes when creating cluster: click on "Databricks" instance in Azure portal before clicking on Databricks "Launch Workspace", there is "Managed Resource Group": open this link; there is a Virtual network and Network security group and Storage account.

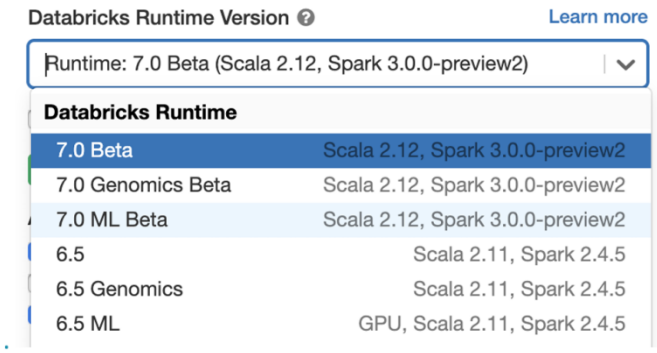
Cluster Access modes

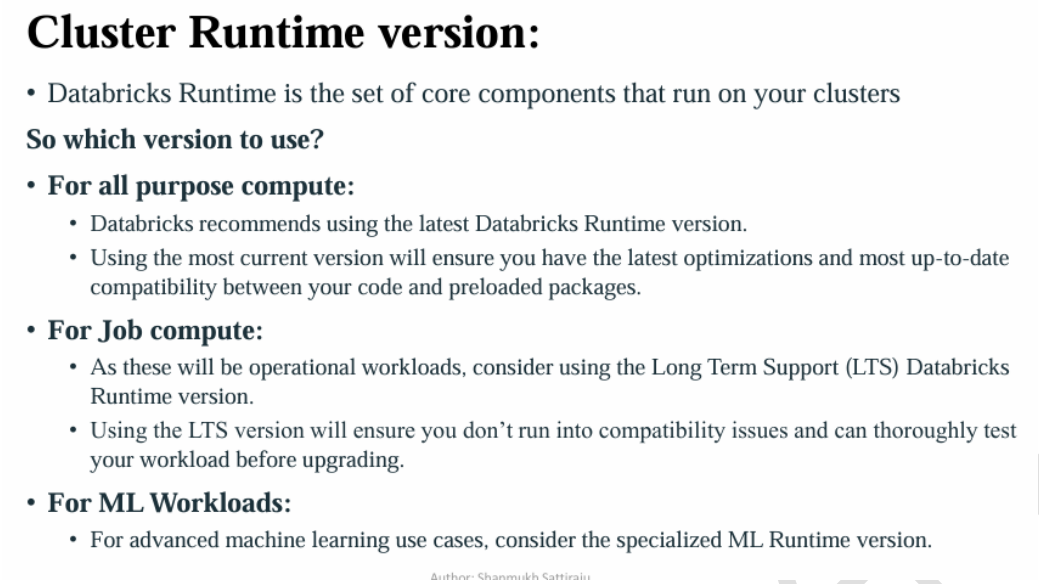
Access Mode	Visible to user	UC Support	Supported Languages	Notes
Single user	Always	Yes	Python, SQL, Scala, R	Can be assigned to and used by a single user.
Shared	Always (**Premium plan or above required**)	Yes	1. Python (on Databricks Runtime 11.1 and above), 2. SQL, 3. Scala (on Unity Catalog-enabled clusters using Databricks Runtime 13.3 and above)	Can be used by multiple users with data isolation among users.
No Isolation Shared	Yes, Admins can hide this cluster type by enforcing user isolation in the admin settings page.	No	Python, SQL, Scala, R	There is a related account-level setting for No Isolation Shared clusters.

This Storage account is going to store Meta Data of it, we will see Virtual Machine, when we will create any compute Resource, now go to Databricks workspace, create any compute resource and then come back here, will find some disks, Public IP address and VM. For all these, we will be charged as DBU/h.

Question: What is Databricks Runtime?

The set of core components that run on the clusters managed by Databricks. Consists of the underlying Ubuntu OS, pre-installed languages and libraries (Java, Scala, Python, and R), Apache Spark, and various proprietary Databricks modules (e.g. DBIO, Databricks Serverless, etc.). Azure Databricks offers several types of runtimes and several versions of those runtime types in the Databricks Runtime Version drop-down when you create or edit a cluster.





Question: What are the types of Databricks Runtimes?

There are major 4 types of Databricks Runtimes.

a) Databricks Runtime for Standard
b) Databricks Runtime for Machine Learning
c) Databricks Runtime for Genomics
d) Databricks Light

Databricks Runtime for Standard

Databricks Runtime includes Apache Spark but also adds a number of components and updates that substantially improve the usability, performance, and security of big data analytics.

Databricks Runtime for Machine Learning

Databricks Runtime ML is a variant of Databricks Runtime that adds multiple popular machine learning libraries, including TensorFlow, Keras, PyTorch, and XGBoost. ML also supports additional GPU supporting libraries clusters. Graphics processing Units Speeding up Machine Learning models. GPUs can drastically lower the cost because they support efficient parallel computation.

Databricks Runtime for Genomics

Databricks Runtime for Genomics is a variant of Databricks Runtime optimized for working with genomic and biomedical data.

Databricks Light

Databricks Light provides a runtime option for jobs that don't need the advanced performance, reliability, or autoscaling benefits provided by Databricks Runtime.

Databricks Light does not support:

- Delta Lake

- Autopilot features such as autoscaling
- Highly concurrent, all-purpose clusters
- Notebooks, dashboards, and collaboration features
- Connectors to various data sources and BI tools

Cluster policies (in Unity Catalog)

- Policies are a set of rules configured by admins
- These are used to limit the configuration options available to users when they create a cluster
- Policies have access control lists that regulate which users and groups have access to the policies.
- Any user with unrestricted policy can create any type of cluster

Stop our compute resource, nothing is deleted in Azure portal, but when we will click on Virtual Machine, then that will show not "start". But if you delete compute resource from Databricks workspace, check your Azure portal again, will find all resources i.e. disks, Public IP address and VM etc are deleted.

Day 2: Understanding notebook and Markdown basics : Hands-on

Note: this part can be executed in Databricks Community edition, not necessarily to be run in Azure Databricks resource

```
%md
### Heading 3
#### Heading 4
##### Heading 5
###### Heading 6

####### Heading 7
-----------------------------------------------------------
%md
# This is a comment
-----------------------------------------------------------

%md

1. HTML Style <b> Blod </b>
2. Astricks style **Blod**
-----------------------------------------------------------

%md

*Italics* style
-----------------------------------------------------------

%md

`print(df)` is the statement to print something

```
This
is multiline
```
```

code
```

---

%md

- one
- two
- three
---

%md
To highlight something

<span style="background-color: #FFFF00"> Highlight this </span>
---

%md
![Profile Pic](https://media.licdn.com/dms/image/C4E03AQGx8W5WMxE5pw/profile-displayphoto-shrink_400_400/0/1594735450010?e=1705536000&v=beta&t=_he0R75U4AKYCbcLgDRDakzKvYZybksWRoqYvDL-alA)
---

%md
Click on [Profile Pic](https://media.licdn.com/dms/image/C4E03AQGx8W5WMxE5pw/profile-displayphoto-shrink_400_400/0/1594735450010?e=1705536000&v=beta&t=_he0R75U4AKYCbcLgDRDakzKvYZybksWRoqYvDL-alA)

# Day 3: DataBricks in Notebook - Magic Commands : Hands-on

**Magic commands in Databricks: if any SQL command is to be executed then select 'SQL'.**

**Note: this part can be executed in Databricks Community edition, not necessarily to be run in Azure Databricks resource**

**1. Select 'Python' from top and type**
**print('hello')**
**#Comments**
**Default language is Python**
---

## Magic commands

- You can use multiple languages in one notebook
- You need to specify language magic command at the beginning of a cell.
- By default, the entire notebook will work on the language that you choose at the top

| Magic command | Language | Description |
|---|---|---|
| %python | Python | Execute a Python query against Spark Context. |
| %scala | Scala | Execute a Scala query against Spark Context. |
| %sql | Spark SQL | Execute a SparkSQL query against Spark Context. |
| %r | R | Execute a R query against Spark Context. |

**2. %scala**
**print("hello") will work and also #comments will also not work.**
**For comments in Scala use //Comments**
---

**3. Comments in SQL -- Comments**
**now in %sql**
**select 2+5 as sum**
```

4. in %r
x <-"Hello"
print(x)
---

5. There are much more magic commands in DB.
%fs ls
List all things in all the directories inside DBFS ie Databricks File System.

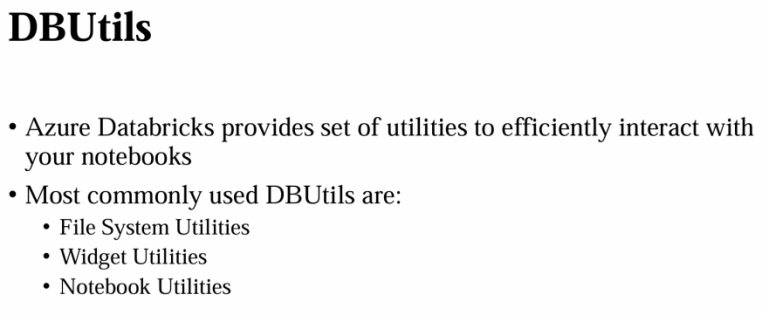
---

6. Know all the Magic commands available:
type:
%lsmagic
---

7. Summary of Magic commands: You can use multiple languages in one notebook and you need to specify language magic commands at the beginning of a cell. By default, the entire notebook will work on the language that you choose at the top.

DBUtils:

DBUtils: Azure Databricks provides set of utilities to efficiently interact with your notebook.
Most commonly used DBUtils are:
1. File System Uttilities
2. Widget Utilities
3. Notebook Utilities



---

1. What are the availble utilities?
# just type:
dbutils.help()
---

2. Lets see File System Utilities
%md
# File System Utilities
# click new cell:
# type:
dbutils.fs.help()

Ls utility

what are available list in particular directory: Enable DBFS, click on "Admin setting" from right top, click on "Workspace Settings",
scroll down, enable 'DBFS File Browser', now you can see 'DBFS' tab, after clicking on 'DBFS' tab, some set on folders are there,
You will find "FileStore" in left pane in “Catalog” button, somewhere, copy path from "spark API format",
path = 'dbfs:/FileStore'

```
dbutils.fs.ls(path)

# why ls, see just above from dbutils.fs.help() details.
```


```
---------------------------------------------------------------

# remove any directory:
# just copy following addres from above:such as FileInfo(path='dbfs:/FileStore/temp/', name='temp/', size=0, modificationTime=0)
dbutils.fs.rm('dbfs:/FileStore/CopiedFolder/',True)

# True is added bcs if this file is not exisiting than it will just reply 'True'
# just check directory list again, that file has been removed.
dbutils.fs.ls(path)

---------------------------------------------------------------

#### mkdir

# why heading are important bcs, left side "Table of Contents" are there, which showing all the headings

dbutils.fs.mkdirs(path+'/SachinFileTest/')

---------------------------------------------------------------

# list all files so that we can see newly created directory is there or not?
dbutils.fs.ls(path)

### put: Inside a folder lets put something,

dbutils.fs.put(path+ '/SachinFileTest/test.csv','1, Test')
---------------------------------------------------------------


# also check using manual "DBFS" tab
### head : read the file conten, which we just written,
filepath = path+ '/SachinFileTest/test.csv'
dbutils.fs.head(filepath)

dbutils.fs.head("/Volumes/main/default/my-volume/data.csv", 25)
This example displays the first 25 bytes of the file data.csv located in /Volumes/main/default/my-volume/.
---------------------------------------------------------------

### Copy: Move this newly created file from one location to another
source_path = path+ '/SachinFileTest/test.csv'
destination_path = path+ '/CopiedFolder/test.csv'
dbutils.fs.cp(source_path,destination_path,True)

---------------------------------------------------------------

# display content from recently pasted values
dbutils.fs.head(destination_path)
---------------------------------------------------------------


# same activity can be done by right click of that file *.csv
# with "Copy path", "Move", "Rename", "Delete"

---------------------------------------------------------------

# Move is cut and paste/move
# copy is just copy and paste

source_path = path+ '/FileTest/test.csv'
destination_path = path+ '/MovedFolder/test.csv'
dbutils.fs.mv(source_path,destination_path,True)
---------------------------------------------------------------
```


```
# remove folder
dbutils.fs.rm(path+ '/MovedFolder/',True)
dbutils.fs.help()
```

Day 4: DBUitls -Widget Utilities : Hands-on

Note: this part can be executed in Databricks Community edition, not necessarily to be run in Azure Databricks resource

Why Widgets: Widgets are helpful to parameterize the Notebook, imagine, in real world you are working in heterogeneous environment, either in DEV env, Test env or Production env, to change everywhere, just parameterize the notebook, instead of hard coding the values everywhere.

Details: Coding:

```
# what are vailavle tools, just type:
dbutils.widgets.help()
```

```
%md
## Widget Utilities
```

```
%md
## Let's start with combo Box
### Combo Box
dbutils.widgets.combobox(name='combobox_name',defaultValue='Employee',choices=['Employee','Developer','Tester','Manager'],label="Combobox Label ")
```

```
# Extract the value from "Combobox Label"
emp=dbutils.widgets.get('combobox_name')

# dbutils.widgets.get retrieves the current value of a widget, allowing you to use the value in your Spark jobs or SQL Queries.

print(emp)
type(emp)
```

```
# DropDown Menu
dbutils.widgets.dropdown(name='dropdown_name',defaultValue='Employee',choices=['Employee','Developer','Tester','Manager'],label="Dropdown Label")
```

```
# Multiselect
dbutils.widgets.multiselect(name='Multiselect_name',defaultValue='Employee',choices=['Employee','Developer','Tester','Manager'],label="MultiSelect Label")
```

```
# Text
dbutils.widgets.text(name='text_name',defaultValue='',label="Text Label")
```

```
dbutils.widgets.get('text_name')
# dbutils.widgets.get retrieves the current value of a widget, allowing you to use the value in your Spark jobs or SQL Queries.
```

```
result = dbutils.widgets.get('text_name')
print(f"SELECT * FROM Schema.Table WHERE Year = {result}")
```

```
# go to Widget setting from right, change setting to "On Widget change"--> "Run notebook", now entire notebook is getting executed

print('execute theseeeSachin ')
```

Day 5: DBUtils - Notebook Utils : Hands-on

Note: this part can be executed in Azure Databricks resource, not in Databricks Community edition, otherwise it will give like: To enable notebook workflows, please upgrade your Databricks subscription.

Create a compute resource with Policy: "Unrestricted", "Single node", uncheck "Use Photon Acceleration", select least node type,

Now go to Workspace-> Users-> your email id will be displayed, add notebook from right, click on "notebook" rename as

Notebook 1: "Day 5: Part 1: DBUtils Notebook Utils: Child"

```
dbutils.notebook.help()
-------------------------
a = 10
b = 20
-------------------------

c = a + b
-------------------------
print(c)
-------------------------
# And I'm going to use the exit here. So basically what exit will do is it is going to execute all the commands before that. And it is going to come here. And if ever there is an exit command, it is going to stop executing the notebook at that particular point and it is going to return the value, whatever you are going to enter here.
dbutils.notebook.exit(f'Notebook Executed Successfully and returned {c}')

# We are going to access this notebook in another Notebook
-------------------------
print('Test')
```

Notebook 2: "Day 5: Part 2: DBUtils Notebook Utils: Parent"

```
print('hello')
-------------------------
```

```
dbutils.notebook.run('Day 5 Part 1 DBUtils Notebook Utils Child',60)
60 is timeout parameter


Click on "Notebook Job", will lend you to "Workflow", where it is executed as job, there are two kinds of clusters, one is interactive and another is "Job", it's executed as a "Job", under "Workflow", check all "Runs".
```

Now "clone" Notebook 1: "Day 5: Part 1: DBUtils Notebook Utils: Child" and Notebook 2: "Day 5: Part 2: DBUtils Notebook Utils: Parent" and rename as "Day 5: Part 3: DBUtils Notebook Utils: Child Parameter" and "Day 5: Part 4: DBUtils Notebook Utils: Parent Parameter"

Notebook 3: "Day 5: Part 1: DBUtils Notebook Utils: Child Parameter"

```
dbutils.notebook.help()

---------------------------
dbutils.widgets.text(name='a',defaultValue='',label = 'Enter value of a ')
```

```
dbutils.widgets.text(name='b',defaultValue='',label = 'Enter value of b ')
---------------------------
a = int(dbutils.widgets.get('a'))
b = int(dbutils.widgets.get('b'))
```
 # The dbutils.widgets.get function in Azure Databricks is used to retrieve the current value of a widget. This allows you to dynamically incorporate the widget value into your Spark jobs or SQL queries within the notebook.

```
---------------------------
c = a + b
---------------------------
print(c)
---------------------------
dbutils.notebook.exit(f'Notebook Executed Successfully and returned {c}')
```

Notebook 4: "Day 5: Part 4: DBUtils Notebook Utils: Parent Parameter"

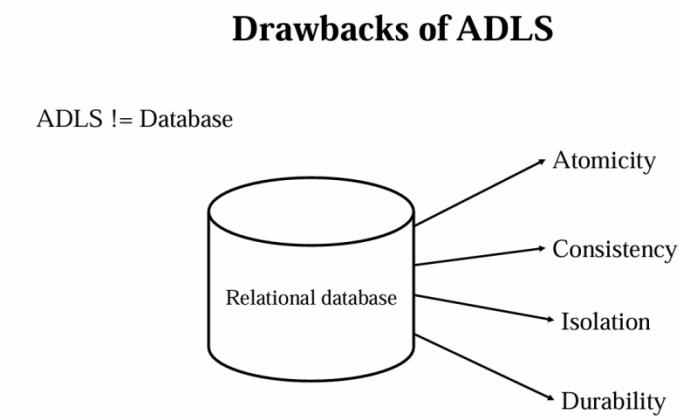
```
print('hello')
-------------------
dbutils.notebook.run(Day 5: Part 1: DBUtils Notebook Utils: Child Parameter',60,{'a' : '50', 'b': '40'})
# 60 is timeout parameter
```

 # go to Widget setting from right, change setting to "On Widget change"--> "Run notebook", now entire notebook is getting executed

On right hand side in "Workflow" → "Runs", there are Parameters called a and b.

Day 6: What is delta lake, Accessing Datalake storage using service principal:

- ✓ **Introduction to section Delta Lake: Delta is a key feature in Azure Databricks designed for managing data lakes effectively. It brings ACID transactions to Apache Spark and big data workloads, ensuring data consistency, reliability, and enabling version control. Delta helps users maintain and track different versions of their data, providing capabilities for rollback and audit.**
- ✓ **Reference: https://learn.microsoft.com/en-us/azure/databricks/delta/#--use-generated-columns**



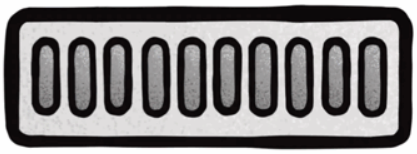
- ✓ **In this section, we will dive into Delta Lake, where the reliability of structured data meets the flexibility of data lakes.**
- ➢ **We'll explore how Delta Lake revolutionizes data storage and management, ensuring ACID transactions and seamless schema evolution within a unified framework.**
- ✓ **Discover how Delta Lake enhances your data lake experience with exceptional robustness and simplicity.**

ACID Explained

by levelupcoding.com

Atomicity

Each transaction is a single unit that either succeeds completely or fails completely.

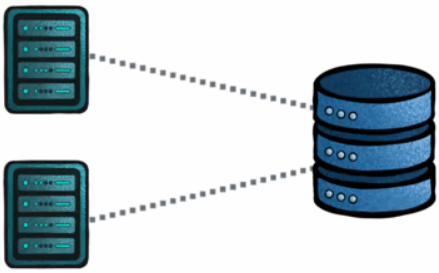


Consistency

Each transaction should follow the database's rules and constraints, keeping the database at a valid state.

Isolation

Concurrent transactions occur separately from each other and are not visible until they're successfully completed.



Durability

Transactions are recorded in non-volatile memory, a transaction will remain committed even if a crash or error occurs afterwards.

Brought to you by

LEVEL UP CODING

@NikkiSiapno @LevelUpCoding

Drawbacks of ADLS

- No ACID properties
- Job failures lead to inconsistent data
- Simultaneous writes on same folder brings incorrect results
- No schema enforcement
- No support for updates
- No support for versioning
- Data quality issues

✓ **We'll cover the key features of Delta Lake, accompanied by practical implementations in notebooks.**

✓ **By the end of this section, you'll have a solid understanding of Delta Lake, its features, and how to implement them effectively.**

❖ **Delta Lake: One of the foundational technologies provided by the Databricks Lakehouse Platform is an open-source, file-based storage format that brings reliability to data lakes. Delta Lake is an open source technology that extends Parquet data files with a file-based transaction log for ACID transactions that brings reliability to data lakes.**

Reference: https://docs.databricks.com/delta/index.html

✓ **ADLS != Database, in RDBMS there is called ACID Properties which is not available in ADLS.**

Data Lake came forward to solve following drawback of ADLS:

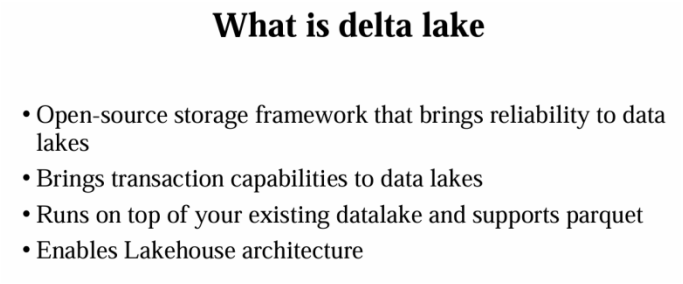
✓ **Drawbacks of ADLS:**

1. **No ACID properties**
2. **Job failures lead to inconsistent data**
3. **Simultaneous writes on same folder brings incorrect results**
4. **No schema enforcement**
5. **No support for updates**
6. **No support for versioning**
7. **Data quality issues**

✓ **What is Delta Lake?**

- **It is an Open-source framework that brings reliability to data lakes.**

- **Brings transaction capabilities to data lakes.**
- **Runs on top of your existing data lake and supports parquet.**
- **Delta Lake is not a data warehouse or a database.**
- **Enables Lakehouse architecture.**



A. Datawarehouse can work only on structure data, which is first generation evolution. However it is supporting ACID properties. One can delete, update and perform data governance on it.

Datawarehouse cannot handle the data other than structure cannot serve a ML use cases.

B. Modern data warehouse architecture: There is Modern data warehouse architecture, which includes usage of Data Lakes for object storage, which is cheaper option for storage, this also called two tier architecture.

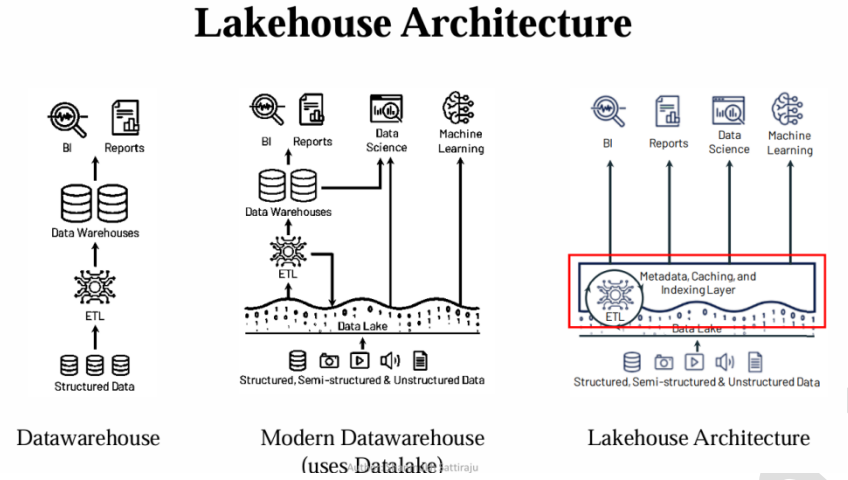
So the best features would be first one.

It supports the any kind of data can be structured or unstructured, and the ingestion of data is much faster. And the data lake is able to scale to any extent. And let us see what the drawbacks here are.

Like we have seen, Data Lake cannot offer the acid guarantees, it cannot offer the schema enforcement, and a data lake can be used for ML kind of use cases, but it cannot serve for BI use case, a BI use case is better served by the data warehouse.

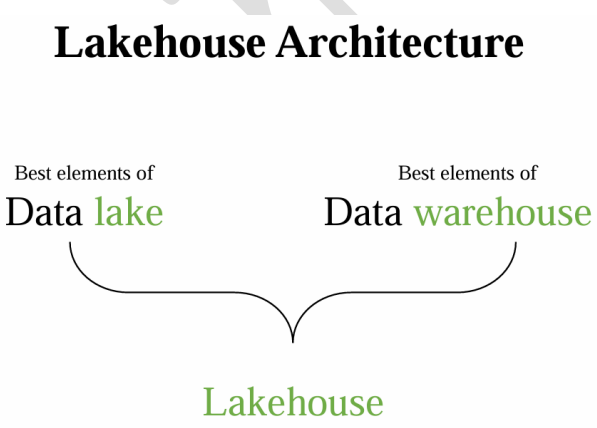
That is the reason we are still using the data warehouse in this architecture.

C. Lakehouse Architecture: Databricks gave a paper on Lakehouse, which proposed the solution by just having a single system that manages both the things.



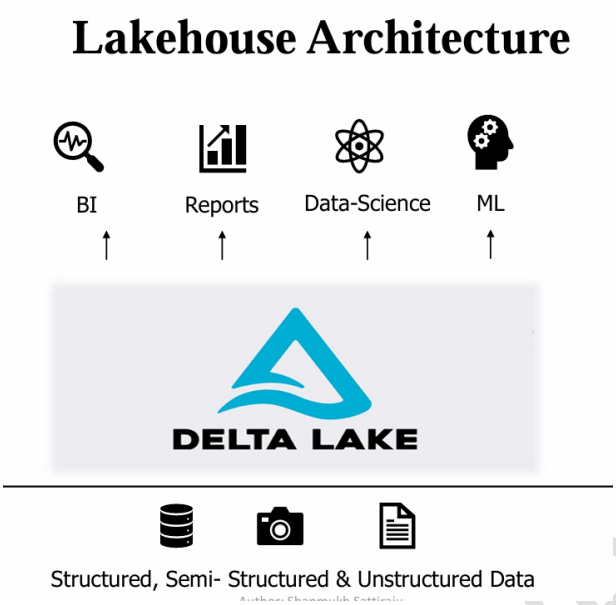
So Databricks has solved this by using Delta Lake. They introduced metadata, which is transaction logs on top of the data lake, which gives us data warehouse like features.

Delta Lake not only stores the raw data in data files but also maintains additional information to ensure data integrity and consistency. This includes **table metadata** (information about the structure and schema of the table), **transaction logs** (which keep track of changes and updates to the data), and **schema history** (allowing for schema evolution and time travel). These features enable Delta Lake to provide ACID transactions and other advanced data management capabilities.



So Delta Lake is one of the implementation that uses the Lakehouse architecture. If you can see in the diagram there is something called metadata caching and indexing layer. So under the hood there will be data lake on the top of the data lake. We are implementing some transaction log feature where that is called the Delta lake, which we will use the Delta Lake to implement Lakehouse architecture.

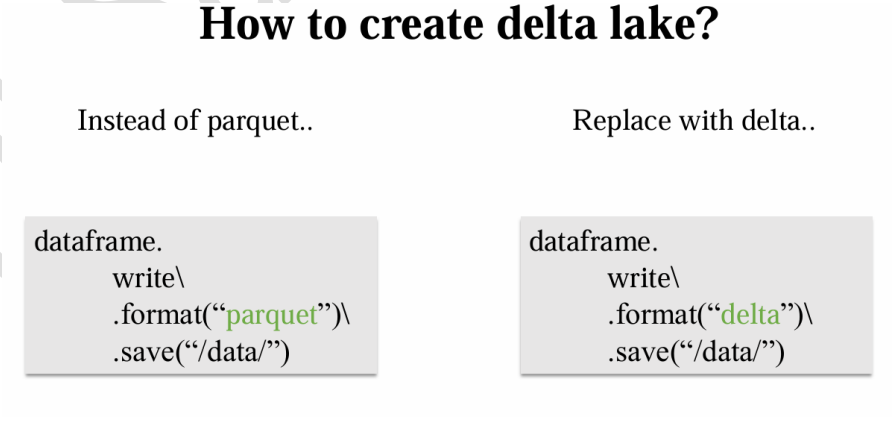
So let's understand about the Lakehouse architecture now. So the combination of best of data warehouses and the data lakes gives the Lakehouse where the Lakehouse architecture is giving the best capabilities of both.



If you can see the diagram, Data Lake itself will be having an additional metadata layer for data management, which having a transaction logs that gives the capability of data warehouse.

So using Delta Lake we can build this architecture. So let's see more about the Lakehouse architecture now. So coming to this we have the data lake and data warehouse which are architecture we have seen. And each is having their own capabilities.

Now Data Lake House is built by best features of both. Now we can see there are some best elements of Data Lake and there are best elements of Data Warehouse. Lake House also provides traditional analytical DBMs management and performance features such as Acid transaction versioning, auditing, indexing, caching, and query optimization.



Create Databricks instances (with standard Workspace otherwise Delta Live tables and SQL warehousing will be disabled) and ADLS Gen 2 instances in Azure Portal.

- **Difference between Data Lake, Data Warehouse and Data Mesh:**

With the rise of modern architectures, terms like Data Lake, Data Warehouse, and Data Mesh often pop up, but how do they differ? Let's break it down:

1 #Data_Lake

Think of it as a vast reservoir—storing raw and unprocessed data in its native format. Perfect for data scientists diving deep into analytics or machine learning use cases. It's flexible and scalable but needs strong #governance to avoid becoming a "data swamp."

💡 Example Tools: #Hadoop, Amazon S3.

📌 Use Case: An e-commerce platform storing clickstream data, user logs, and videos for advanced analytics.

2 #Data_Warehouse

The organized sibling of Data Lakes—storing structured, processed data for immediate analysis. It's built for business intelligence (BI) and reporting, ensuring optimized queries and actionable insights.

💡 Example Tools: Snowflake , Google BigQuery, Amazon Redshift.

📌 Use Case: A bank analyzing customer transaction data for financial forecasting.

3 #Data_Mesh

The modern challenger—focused on decentralized data ownership. Instead of one central repository, teams manage their own domain-specific data products while ensuring self-service and scalability. This is ideal for large organizations tackling agility and scalability issues.

💡 Example Implementation: Leveraging Zhamak Dehghani's principles.

📌 Use Case: A global enterprise empowering cross-functional teams to manage and deliver data products independently.

Delta Lake Importance: Here's a practical walkthrough of some amazing features and techniques for managing Delta Lake tables.

1. Track Changes in Your Delta Table

==========

Delta Lake allows you to view the historical changes made to a table. With simple commands like DESCRIBE HISTORY, you can track updates, additions, and deletions to ensure data integrity and transparency.

DESCRIBE HISTORY student;

2. Access Data at a Specific Timestamp

==========

Ever need to retrieve data as it was at a specific point in time? Delta Lake's TIMESTAMP AS OF lets you query data from the past by specifying a timestamp.

SELECT * FROM student TIMESTAMP AS OF '2025-01-18T13:47:11.000+00:00';

3. Version Control with Delta Tables

==========

Delta Lake allows querying data at specific versions, making it easy to access past states of a table. This helps ensure your data is always recoverable and up to date with precise version control.

SELECT * FROM student VERSION AS OF 1;

4. Restore a Table to a Previous Version

==========

Accidents happen, and sometimes data is deleted or corrupted. Delta Lake's RESTORE feature lets you quickly recover data from a previous version without skipping a beat.

RESTORE TABLE student TO VERSION AS OF 2;

5. Optimizing Tables for Faster Queries

==========

Want faster query performance? Use OPTIMIZE and ZORDER BY to compact smaller files into larger ones, making it easier and quicker to read data by specific columns.

Delta Lake Optimization

Technique	What It Does	Notes
Z-Ordering	Clusters related data on disk to reduce data scanning	Use on high-cardinality columns frequently used in query predicates
Partitioning Strategy	Splits data by key columns using Hive-style partitioning	Only for tables >1TB; ensure partitions have at least 1GB of data
Compaction	Merges small files into larger ones (default 128MB target)	Run OPTIMIZE regularly
Auto Compaction	Automatically merges small files during write operations	Enable with delta.autoOptimize.autoCompact; runs synchronously after writes
Liquid Clustering	Dynamically adapts data layout based on query patterns	Recommended for all new Delta tables; replaces partitioning and Z-ordering
Vacuum Operations	Cleans up old data files and metadata to reduce storage costs	Set retention period based on recovery needs; frees up storage space

OPTIMIZE student ZORDER BY id;

- Delta Lake builds upon standard data formats. Delta lake table gets stored on the storage in one or more data files in Parquet format, along with transaction logs in JSON format.
- In addition, Databricks automatically creates Parquet checkpoint files every 100 commits to accelerate the resolution of the current table state. If you want to have a checkpoint file for delta table for every 10 commits or after any desired commits. You can customize it using the configuration "delta.checkpointInterval"
- Syntax:

```
ALTER TABLE table_name SET TBLPROPERTIES ("delta.checkpointInterval" = "10")
```

---

6. Keep Your Data Clean with Vacuuming

==========

Over time, your Delta Lake tables may accumulate old or uncommitted files. Use the VACUUM command to clean up unnecessary files and remove outdated versions to improve performance and storage.

VACUUM student;

7. Why is Delta Lake important?

==========

Delta Lake combines the scalability and flexibility of a data lake with the reliability and performance of a data warehouse. By incorporating features like version control, time travel, optimization, and vacuuming, it empowers organizations to manage big data efficiently, safely, and with minimal performance impact.

Optimize your databricks workspace:

Here's a quick overview of the 6 key Delta Lake optimization techniques that can cut your query times from hours to minutes.

📋 Z-Ordering

📋 Compaction

Combines small files into larger ones to reduce metadata overhead and improve I/O.

When to use:
→ Many small files (< 100MB each)
→ Streaming workloads creating lots of tiny files
→ Before running large analytical queries

When not to use:
→ Files already optimal size (100MB-1GB)
→ Real-time streaming requirements
→ Limited compute resources

📋 Partitioning

Physically organizes data into separate folders based on column values.

When to use:
→ Clear filtering patterns (date, region, category)

→ Low cardinality columns (< 1000 partitions)
→ Queries consistently filter on partition column

When not to use:
→ High cardinality columns (creates too many small files)
→ No consistent query patterns
→ Columns that change frequently

📋 Auto Compaction

Auto Compaction occurs after a write to a table has succeeded to check if files can further be compacted; if yes, it runs an OPTIMIZE job without Z-Ordering toward a file size of 128 MB. Auto Compaction is part of the Auto Optimize feature in Databricks. it checks after an individual write, if files can further be compacted, if yes, it runs an OPTIMIZE job with 128 MB file sizes instead of the 1 GB file size used in the standard OPTIMIZE.

Auto compaction does not support Z-Ordering as Z-Ordering is significantly more expensive than just compaction.

Automatically triggers compaction during writes without manual intervention.

When to use:
→ Streaming pipelines
→ Frequent incremental loads
→ Teams without dedicated optimization workflows

When not to use:
→ Batch processing with controlled file sizes
→ Cost-sensitive environments (adds compute overhead)
→ Custom compaction logic needed

📋 Liquid Clustering

Dynamically reorganizes data based on query patterns, adapting to your workload without manual tuning.

When to use:
→ Evolving query patterns
→ Multiple clustering columns needed
→ Want to replace both partitioning and z-ordering

When not to use:
→ Simple, stable query patterns
→ Tables smaller than 1GB

📋 Vacuum

Removes old file versions to reclaim storage space.

When to use:
→ After major data changes
→ Storage costs are high
→ Old versions no longer needed for time travel

When not to use:

→ Need historical versions for auditing
→ Within retention period of critical data
→ During active read operations

🏳 Common Mistakes
1. Over-partitioning small datasets
2. Z-ordering on too many columns
3. Skipping vacuum in cost-sensitive environments
4. Using liquid clustering on tiny tables
5. Manual compaction on auto-compaction enabled tables
6. Z-ordering on high cardinality columns like IDs or timestamps
7. Not monitoring file sizes after optimization (missing the target 100MB-1GB range)
8. Applying same optimization strategy across all tables regardless of usage patterns

---

Facts about Delta Lake File Statistics:

Delta Lake automatically captures statistics in the transaction log for each added data file of the table. By default, Delta Lake collects the statistics on the first 32 columns of each table. These statistics indicate per file:

- Total number of records
- Minimum value in each column of the first 32 columns of the table
- Maximum value in each column of the first 32 columns of the table
- Null value counts for in each column of the first 32 columns of the table
- Nested fields count when determining the first 32 columns. Example: 4 struct fields with 8 nested fields will total to the 32 columns.
- The statistics are leveraged for data skipping when executing selective queries.
- The statistics are generally uninformative for string fields with very high cardinality.

These statistics are leveraged for data skipping based on query filters.

- **Hands-on: Accessing Datalake storage using service principal:**

“Day 6 Part 1 Test+access.ipynb”

Source Link: Tutorial: Connect to Azure Data Lake Storage Gen2 - Azure Databricks | Microsoft Learn

Step 1: Create ADLS Gen 2 instance

Inside ADLS Gen 2, create a ADLS Gen 2 with name “deltadbstg”, create a container with name “test”, inside this container add a directory with name “sample”, upload a csv file name “countires1.csv”.

Step 2: Databricks instances and Compute resource

Inside Databricks instances: Create a compute resource with Policy: "Unrestricted", "Single node", uncheck "Use Photon Acceleration", select least node type.

Go give permission, we have unity catalogue.

Step 3: Create a Microsoft Entra ID service principal

- **Go to Azure Entra ID (previously Azure Active directory), inside it, going to create some service principle, click on "App Registration" on left hand side where you can create an app. Click on "New Registration",**
- **Give name: "db-access", leave other settings as it is. Copy "Application (client) ID" and "Directory (tenant) ID" from "db-access" overview.**
- **Eg: Application ID: dbf11b5d-9d81-4cc8-82f3-f14da29c8c98**
- **Directory (tenant) ID: 076af1d9-53a7-48e4-9c43-8ae1c16db3e2**

Step 4: Create new New client secret and Copy Secret Key

- **Inside app registration: Also copy secret key from left, "certificates & secrets" from left, click on "+ New client secret", give "Description" as "dbsecret" and click on "Add".**
- **Copy the "Value" from "dbsecret" now.**

- **Eg: Client Secret Value or service credential:"HnF8Q~braIA_bVu6IVtDXrvVYzm_YCncVouGtcb."**
- **Note three keys "Application (client) ID" and "Directory (tenant) ID" and "Value" from "dbsecret" i.e. secret ID in a text notebook.**
- **Inside notebook, secret ID is "service credential".**

Step 5: Add Role Assignment from Access Control (IAM)

- **To give access to data storage, goto ADLS Gen 2 instances in Azure Portal, go to "Access Control (IAM)", click on "+Add", click on "+Add Role Assignment", "User, Group and service Principal"-> search for "Storage Blob Data Contributor", click on storage blob contributor" and "+select members", type service principle which is "db-access". Select, finally Review and Assign.**

Step 6: Put all these credentials at respective places

Replace everything here:

----

service_credential = dbutils.secrets.get(scope="<scope>",key="<service-credential-key>")

spark.conf.set("fs.azure.account.auth.type.<storage-account>.dfs.core.windows.net", "OAuth")

spark.conf.set("fs.azure.account.oauth.provider.type.<storage-account>.dfs.core.windows.net", "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")

spark.conf.set("fs.azure.account.oauth2.client.id.<storage-account>.dfs.core.windows.net", "<application-id>")

spark.conf.set("fs.azure.account.oauth2.client.secret.<storage-account>.dfs.core.windows.net", service_credential)

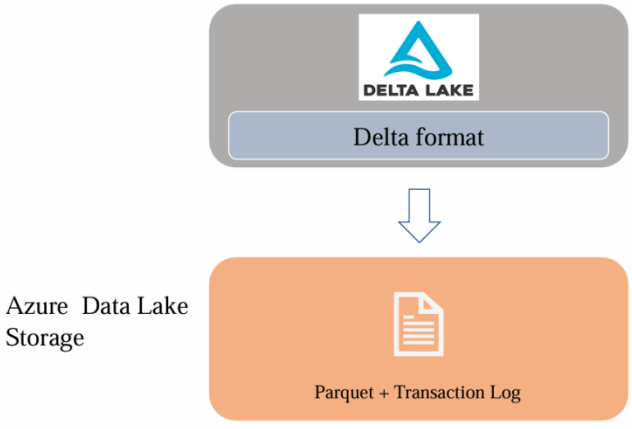
spark.conf.set("fs.azure.account.oauth2.client.endpoint.<storage-account>.dfs.core.windows.net", "https://login.microsoftonline.com/<directory-id>/oauth2/token")

Hands on 2: Drawbacks of ADLS – practical:

Day 6 Part 2 .+Drawbacks+of+ADLS.ipynb

- **Create new directory in "test" container with name "files" and upload csv file "SchemaManagementDelta.csv"**

This hands on showing that using data lake we are unable to perform Update operation. Only in delta lake this operation is supportive.



Even using spark.sql, are unable to perform Update operation. This is one of Drawbacks of ADLS.

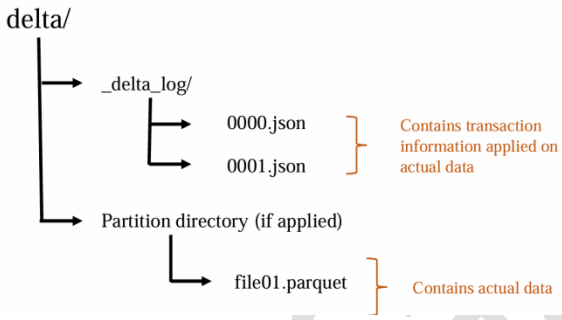
Versioning is also not available in ADLS, which is Drawbacks of ADLS.

Hands on 3: Creating Delta lake:

Day 6 Part 3 +Drawbacks+of+ADLS+-+delta.ipynb

Hands on 4: Understanding Transaction Log:

Day 6 Part 4 Understanding+the+transaction+log.ipynb



Transaction logs tracks changes to delta table and it is responsible for bringing the ACID compliance for Delta lake.

Understanding Transaction log file (Delta Log)

- Contains records of every transaction performed on the delta table
- Files under _delta_log will be stored in JSON format
- Single source of truth

Transaction log contents

JSON File = result of set of actions

- **metadata** – Table's name, schema, partitioning ,etc
- **Add** – info of added file (with optional statistics)
- **Remove** – info of removed file
- **Set Transaction** – contains record of transaction id
- **Change protocol** – Contains the version that is used
- **Commit info** – Contains what operation was performed on this

- **What is Delta Table**?

Delta Tables are a data storage format and technology that allow for efficient management and manipulation of large datasets in data lakes. They are a feature of Delta Lake, an open-source storage layer that runs on top of Apache Spark and Apache Parquet. Delta Tables provide features like ACID transactions, schema enforcement, time travel, and upserts. These features make it easier to work with big data in a consistent, reliable, and efficient manner.

Key Features of Delta Tables:

ACID Transactions: Delta Tables support ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring that data operations are reliable and consistent. This is important in big data systems where multiple jobs or users may be reading and writing to the same table concurrently. With ACID guarantees, Delta Tables ensure that operations like insertions, deletions, and updates are atomic and isolated.

Schema Enforcement and Evolution: Delta Tables ensure that the schema of the data is enforced when writing to the table, which means that the data conforms to a predefined schema. If a column is missing or if the type of data in a column doesn't match the expected schema, it will be rejected. Delta Tables also support schema evolution, which allows you to automatically add new columns or modify existing ones without breaking existing processes or applications.

Time Travel: Delta Tables provide a feature called time travel, which allows users to query previous versions of the table. This can be useful for auditing purposes, debugging, or retrieving historical data. Time travel is achieved through Delta's versioning system, where each write operation (like an insert or update) is tracked with a new version.

Data Versioning: Delta Tables automatically track versions of the data, enabling you to perform queries on a specific version. Each time data is written or modified, a new version is created. You can access these versions by specifying a timestamp or version number.

Upserts (**MERGE**): Delta Tables allow upserts (a combination of "update" and "insert") through the MERGE command. This allows you to efficiently update existing records and insert new ones in a single operation. It is commonly used when data changes over time, and you want to ensure that the latest records are inserted or updated without duplicating data.

Scalable and Efficient Storage: Delta Tables are built on top of Apache Parquet, which is an optimized columnar storage format. This provides significant improvements in query performance, compression, and storage efficiency.

Day 7: Creating delta tables using SQL Command

Day 07 Part 1 pynb

- **Change default language to "SQL", then Create schema with name "delta", before further code, where exactly we can see this table, go to "Catalog", there are two defaults catalogues, "Hive metastore" and "samples". This is not "Unity catalog".**
- **The Hive metastore is a workspace-level object. Permissions defined within the hive_metastore catalog always refer to the local users and groups in the workspace. Hence Unity catalog cannot manage the local hive_metastore objects like other objects. For more refer https://docs.databricks.com/en/data-governance/unity-catalog/hive-metastore.html#access-control-in-unity-catalog-and-the-hive-metastore.**
- **Schema with name "delta" is created in "Hive metastore" catalogue, this is schema not database.**
- **Create table with name "'delta'.deltaFile" , any table which you are creating be default is a delta table in Databricks. Check again Schema with name "delta" which is created in "Hive metastore" catalogue, here one symbol with delta is also showing.**

- **To find exact location of this delta table: Go to "Cataloge"-> "Hive metastore" -> "delta" -> "deltaFile"-> "Details"-> "Location".**
- **There is no parquet file, means we haven't inserted any data.**

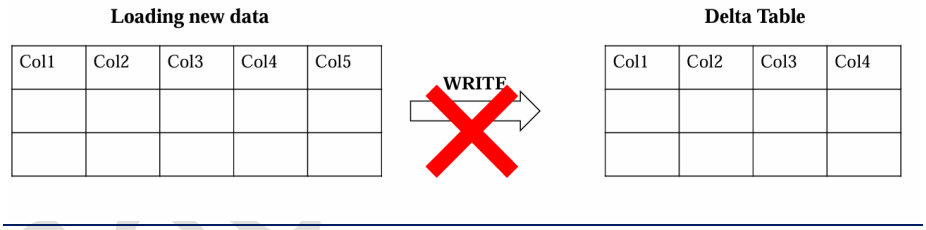
Day 07 Part 2 pynb file

Day 07 Part 3 pynb file

Day 07 Part 4 pynb file

- ✓ **Create "SchemaEvol" in "Test" containers (this is part of Day 6), before running this file upload two csv files, "SchemaLessCols.csv" and "SchemaMoreCols.csv" in "SchemaEvol" directory in "Test".**
- ✓ **Schema Enforcement or Schema Validation: Let's take a delta table, which is maintained strictly, we are ingesting data into this table on daily basis. In one ingestion, if a new data is coming with new "Column" which is not available in this schema.**

Schema Enforcement



- ✓ **Delta Lake uses Schema validation on "Writes".**
- ✓ **Now on a fine day during the data ingestion some data comes with a new column which is not in the schema of our current table, which is being overwritten to the location where our delta lake is present.**

How does schema enforcement works?

Delta lake uses Schema validation on **"writes" .**

Schema Enforcement Rules:

1. Cannot contain any additional columns that are not present in the target table's schema
2. Cannot have column data types that differ from the column data types in the target table.

- ✓ **Now, generally, if you are using the data lake and I'm mentioning it again to prevent any confusion, I mean the data lake, not the delta lake, the general data lake will allow the overwriting of this data and we will lose any of our original schema.**
- ✓ **Like we have seen in the drawback, we try to overwrite to the location where we lost our data and it allowed the write.**

- ✓ But coming to the Delta Lake, we have a feature called schema enforcement or Schema validation, which will check for the schema for whatever the data that is getting written on the Delta Lake table.
- ✓ If the schema does not match with the data which we are trying to write to the destination, it is going to reject that particular data.
- ✓ It will cancel the entire write operation and generates an error stating that the schema is not matching the module of the schema.
- ✓ Validation is to safeguard the delta lake that ensures the data quality by rejecting the writes to a table that do not match the table schema.
- ✓ A classic example is you will be asked to scan your IDs before entering your company premises, so that is going to check if you are the authorized person to enter this.
- ✓ Similarly, schema enforcement acts as a gatekeeper who checks for the right data to enter to the Delta Lake.
- ✓ Now, how does this schema enforcement works exactly?
- ✓ So to understand this, Delta Lake uses the schema validation on writes, which means all the new writes to the new table are checked for the compatibility with the target table.
- ✓ So during the right time it is going to check for the schema compatible or not.
- ✓ If the schema is not compatible, data is going to cancel. The delta lake cancels the transaction altogether.

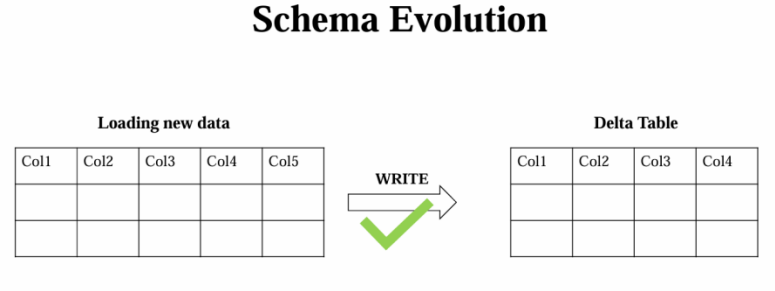
Dr Sachin Saxena
Follow on LinkedIn

- ✓ No data is being written, and it raises an exception to let the user know about the mismatch. And there are certain rules on how the schema enforcement works.
- ✓ And let us see on what conditions the incoming data will be restricted in writing to the delta table. So let's see about the rules now.
- ✓ So it cannot contain any additional columns like we have seen before.
- ✓ If the incoming data is having a column more than the one defined in the schema, it is treated as a violation to the schema enforcement.
- ✓ But if it is having less number of columns than the target table, it is going to allow the write by giving the null value to the existing columns where there is no data for this particular table.
- ✓ But if the incoming data is having more number of columns, it is going to cancel that insert. Now there is one more rule where it cannot have the different data types. If a delta table's column contains the string data, but the corresponding column in the data frame incoming is having the integer data, the schema enforcement will raise an exception and it will prevent the write operation entirely.
    - Now how is this schema enforcement useful?
    - Because it is such a stringent check, schema enforcement is an excellent tool to use as a gatekeeper to get a clean, fully transformed data set that is ready to use for production or consumption.
    - It is typically enforced on tables that directly fed into the machine learning algorithm by dashboard or data analytics or visualization tools, and schema enforcement is used for any production system that is requiring highly structured, strongly typed semantics checks.
    - And it's enough with this theory.
    -

- Trying to append more columns using code. Extra column is “Max_Salary_USD”.
- Source with fewer columns will accept.
-

Day 07 Part 5 pynb file

- **Schema Evolution: Schema evolution in Databricks Delta Lake enables the flexible evolution of table schemas, allowing changes such as adding, removing, or modifying columns without the need for rewriting the entire table. This flexibility is beneficial for managing changes in data structures over time.**



- **So schema evolution is a feature that allows the user to easily change the tables current schema to accommodate the data changing over time.**

- **Question**: How would you handle schema evolution in PySpark?

Schema evolution refers to the ability to handle changes in your dataset's structure—

- ✅ Adding new columns/fields
- ✅ Dropping existing columns/fields
- ✅ Changing the datatypes of existing fields.

PySpark Techniques for Schema Evolution:

1) **Merge Schemas Automatically** : PySpark can merge schemas across files dynamically using

the **mergeSchema** option. This works well with formats like Parquet, ORC, and JSON.

Example :
df= spark.read.option("mergeSchema", "true").parquet("path") .

2)Define and Enforce a Custom Schema : When you need strict control over your data structure, define a schema explicitly to ensure consistency.

Example :
schema = StructType([StructField("Name", StringType(), True),
Struct Field("Age", IntegerType(), True), StructField("Salary", Integer Type(), True)])

df = spark.read.schema(schema).parquet("path/to/files")
df.printSchema().

3) Handle Missing Columns Dynamically:When columns are added or missing, handle them programmatically to avoid breaking the pipeline.

Example :
from pyspark.sql.functions import lit

if 'new_column' not in df.columns:
df = df.withColumn("new_column", lit(None)).

4)Schema Evolution for Streaming Data: Streaming pipelines require special attention to schema changes. Tools like Delta Lake offer seamless schema evolution.

Example :
from delta.tables import DeltaTable

df.write.format("delta").option("mergeSchema",
"true").mode("append").save("path/to/delta/table").
schema registry to validate schema changes.

- **Most commonly, it is used when performing an append or an override operation to automatically adapt the schema to include one or more columns.**
- **"mergeSchema","True" enables this Schema Evolution in Delta tables.**

DataFrameWriter.mode defines the writing behaviour when data or table already exists.
Options include:

- append: Append contents of the DataFrame to existing data.
- `overwrite`: Overwrite existing data.
- `error` or `errorifexists`: Throw an exception if data already exists.
- `ignore`: Silently ignore this operation if data already exists.

Day 07 Part 6 pynb file

Audit Data changes & Time Travel: "Time travel" in Delta Lake enables users to query a historical snapshot of the data at a specific version, facilitating data correction or analysis at different points in time.

Audit Data Changes & Time Travel

- Delta automatically versions every operation that you perform
- You can time travel to historical versions
- This versioning makes it easy to audit data changes, roll back data in case of accidental bad writes or deletes, and reproduce experiments and reports.

Day 07 Part 7 pynb file

- **Vacuum Command:**

- **If you are getting a very high storage cost now, if your organization wish to delete the old data like a 30 days old data, you can make use of the vacuum command.**
- **Now let's see how we can implement this. Now in order to know how many files will be deleted, you can make use of the dry run feature in the vacuum.**

Vacuum in Delta lake

- Vacuum helps to remove parquet files which are not in latest state in transaction log
- It will skip the files that are starting with _ (underscore) that includes _delta_log
- It deletes the files that are older then retention threshold
- Default retention threshold in 7 days
- If you run VACUUM on a Delta table, you lose the ability to time travel back to a version older than the specified data retention period.

- **So let's see how you can implement this. Now I am going to use some feature called dry Run.**
- **So it is not actually going to delete any kind of data. It is just going to show us how many files will be deleted. So it will ideally give a list of first thousand files that will be deleted and it will not actually delete.**
- **It will just show what files will be deleted. Now, by default, the retention period of this vacuum command is seven days. So any data that is having the age of more than seven days that will be deleted by default using this vacuum command.**
- **So we just created our table and we just inserted few records, but we haven't have any data which is older than seven days.**

- **Now, if I just try to run this particular command, it is going to show me nothing. And now you can see he's not returning any results because we are not having any data, which is post seven days old, which is the retention period of this particular vacuum command.**

- **Now for testing purpose, if you want to delete the data, which is less than seven period of time than the default duration, you can make use of the retain command.**
- **There is restriction, just make "retentionDuration" to True.**

Q.2. A data engineer is trying to use Delta time travel to rollback a table to a previous version, but the data engineer received an error that the data files are no longer present.

Which of the following commands was run on the table that caused deleting the data files?

Ans: VACUUM

Overall explanation

Running the VACUUM command on a Delta table deletes the unused data files older than a specified data retention period. As a result, you lose the ability to time travel back to any version older than that retention threshold.

Reference: https://docs.databricks.com/sql/language-manual/delta-vacuum.html

Q.3. In Delta Lake tables, which of the following is the primary format for the data files?

Ans: Parquet

Overall explanation

Delta Lake builds upon standard data formats. Delta lake table gets stored on the storage in one or more data files in Parquet format, along with transaction logs in JSON format.

Reference: https://docs.databricks.com/delta/index.html

Day 07 Part 8 pynb file

Convert to Delta:

Day 8: Understanding Optimize Command – Demo

- ✓ **Optimize command in Databricks primarily reduces the number of small files and compacts data, improving query performance and storage efficiency.**
- ✓ **So on a high level, the optimize command will help us to compact multiple small files into a single file.**

Optimize in Delta lake

Operation	parquet files	_delta_log	Line number Column	State
CREATE TABLE		000.json		
WRITE	aabb.parquet	001.json	100	**Active**
WRITE	ccdd.parquet	002.json	101	Inactive
WRITE	eeff.parquet	003.json	102	Inactive
DELETE 101		004.json		
UPDATE 102	iijj.parquet	005.json	99	**Active**

- ✓ **So this is one of the optimization feature in the delta lake which the name itself indicates the optimize. Now the use of this particular command is to compact multiple files into a single file.**
- ✓ **If you are aware of the small file problem in the spark, where if you have 100 small files, where each transaction is creating a file, if you want to read the content of each and every file, the time to open each and every file is more than reading the actual file.**
- ✓ **It need to open the file, it need to read the content, and it need to close the file. So the time to open and close each and every file is more than actual content reading. So this is why the optimize command helps in a way where it can just combine all the active files into a single file.**

- ✓ **I am mentioning something like active files, because sometimes there are inactive files where. Let us try to understand what exactly is this active and what is an inactive.**

- ✓ **So let's see by taking an example. So we'll be doing some transformations on our data in our Delta lake transformations are nothing but the operations like inserts, deletes and updates and etc..**
- ✓ **So each action or a transformation will be treated as a commit and it will create the parquet file.**

UPSERT (Merge) in delta lake

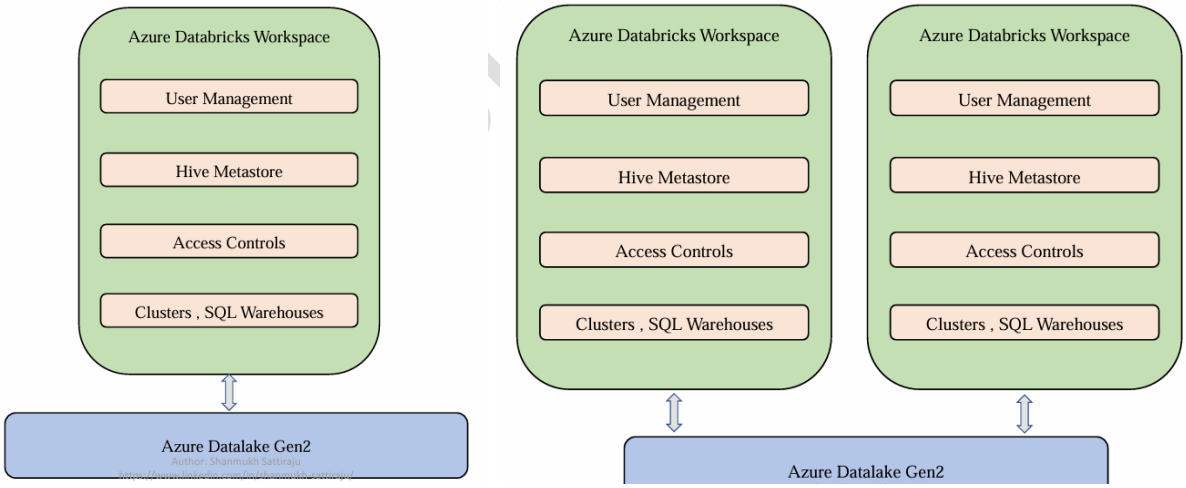
- We can UPSERT (UPDATE + INSERT) data using MERGE command.
- If any matching rows found, it will update them
- If no matching rows found, this will insert that as new row

```
MERGE INTO <Destination_Table>
USING <Source_Table>
        ON <Dest>.Col2 = <Source>.Col2
WHEN MATCHED
THEN UPDATE SET
          <Dest>.Col1 = <Source>.Col1,
          <Dest>.Col2 = <Source>.Col2
WHEN NOT MATCHED
THEN INSERT
        VALUES(Source.Col1, Source.Col2)
```

- ✓ **Along with that it will create the delta log files. So imagine we are creating a empty table because we are doing an empty table creation. It is also an operation where the operation is recorded as a create table.**
- ✓ **And it is not going to create any parquet file, but it is going to create a delta log.**

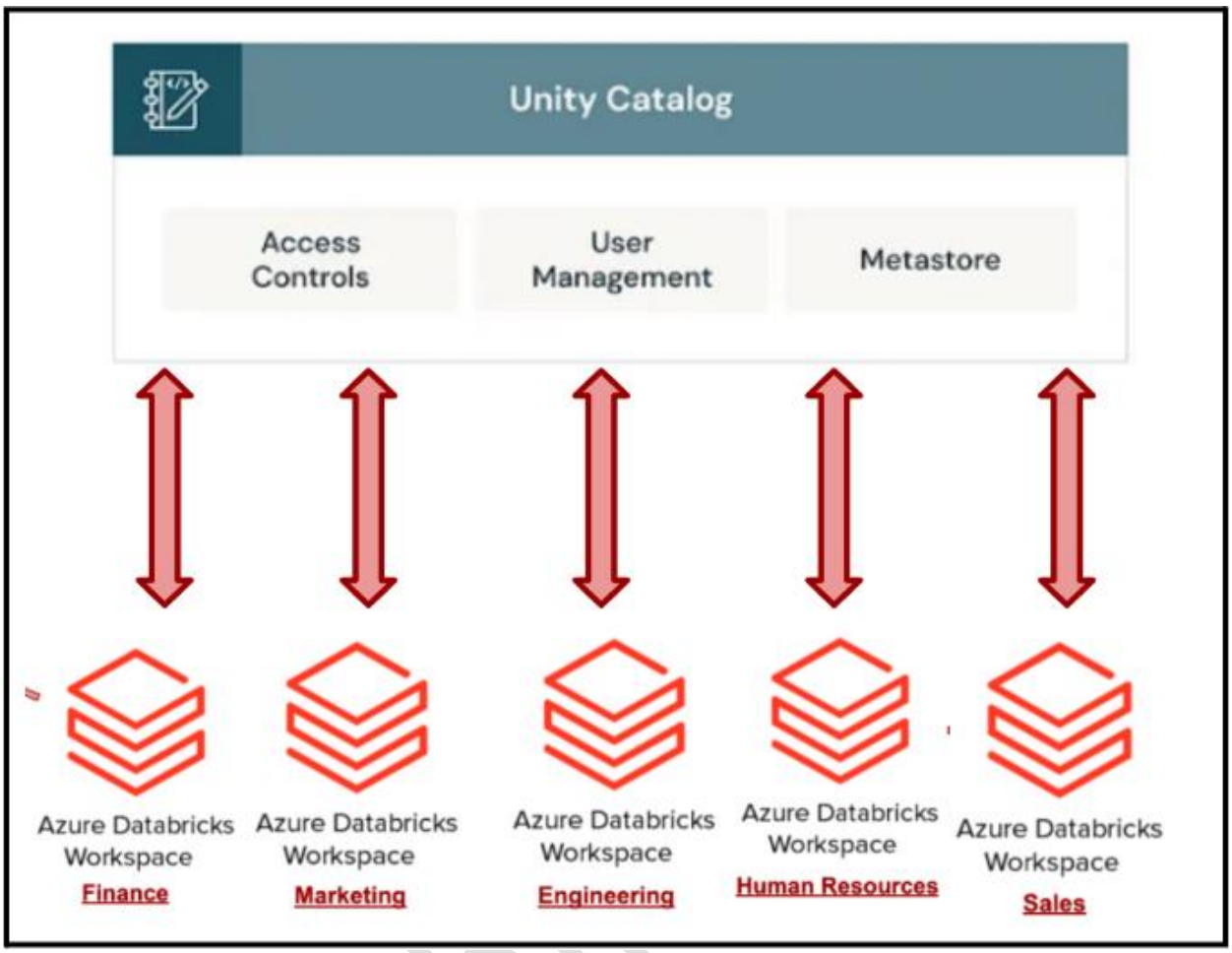
Day 9: What is Unity Catalog: Managed and External Tables in Unity Catalog

Understanding the Problem Unity Catalog Solves

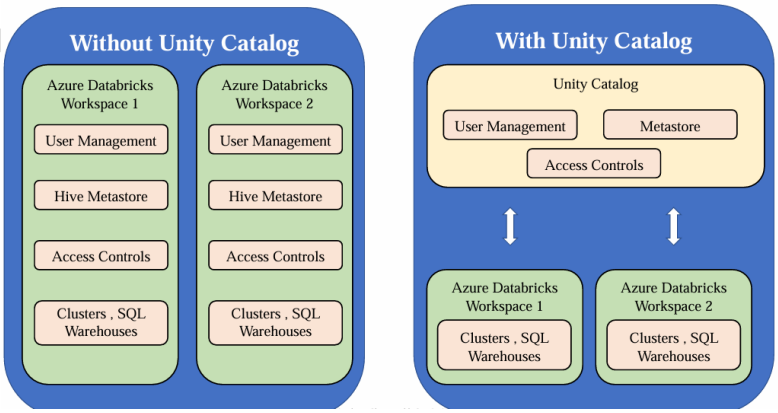


Unity Catalog: bringing order to the chaos of the cloud. It is a data governance tool that provides a centralized way to manage data and AI assets across platforms.

Unity Catalog: a powerful tool designed to centralize and streamline data management within Databricks.

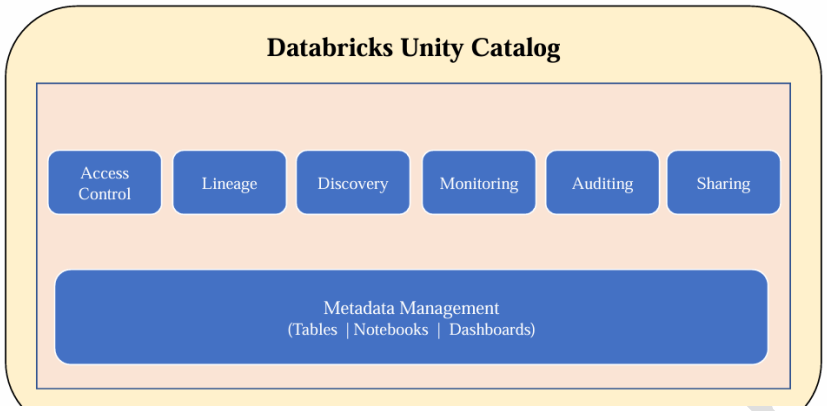


Unity Catalog centralizes all metadata, ensuring that user data definitions are standardized across the organization. This means that the marketing, customer service, and product development teams all have access to a single, consistent source of truth for user data. By providing a unified view of all data assets, **Unity Catalog** makes it easier for teams to access the information they need without having to navigate through multiple systems.

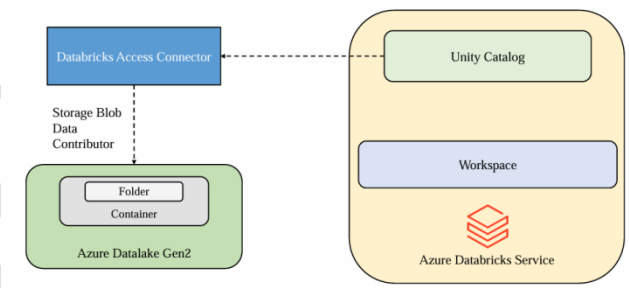


The marketing team can easily access support interaction data, and the product development team can view user engagement metrics, all from a single platform. This unified approach reduces administrative

overhead, enhances data security, and ensures that data is accurate and compliant, supporting better data-driven decision-making and driving business success.



1. **Typical Databricks Setup**
   - Organizations use Databricks workspaces for projects (e.g., Azure Databricks).
   - User management is required for assigning roles (developers, leads, managers).
   - Each workspace has a Hive metastore storing managed tables and defined access controls.



2. **Challenges in Managing Multiple Workspaces**
   - Multiple environments (dev, UAT, prod) and business units/projects require several workspaces.
   - Duplication of user management, access controls, and cluster creation policies across workspaces.
   - Lack of centralized governance for managing users, auditing, and metadata.
   - Difficult to track who has access to specific data across all workspaces.
3. **Need for a Centralized Solution**
   - A central system for governance, audit, and metadata management is required.

How Unity Catalog Solves the Problem

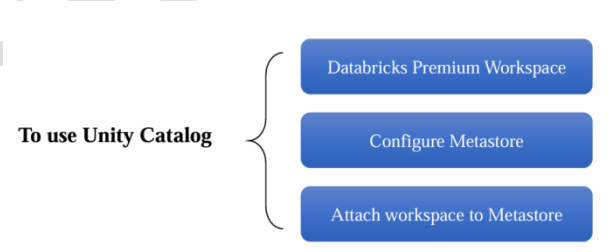
1. **Centralized Governance**
   - Manages user access, metadata, and governance for multiple workspaces centrally.
   - Provides visibility and control over access permissions across all workspaces.
2. **Unified Features**
   - **Access Controls**: Define and enforce who can access what data.
   - **Lineage**: Track how data tables were created and used.
   - **Discovery**: Search for objects like tables, notebooks, and ML models.
   - **Monitoring**: Observe and audit object-level activities.
   - **Delta Sharing**: Share data securely with other systems or users.
   - **Metadata Management**: Centralized management of tables, models, dashboards, and more.

Summary

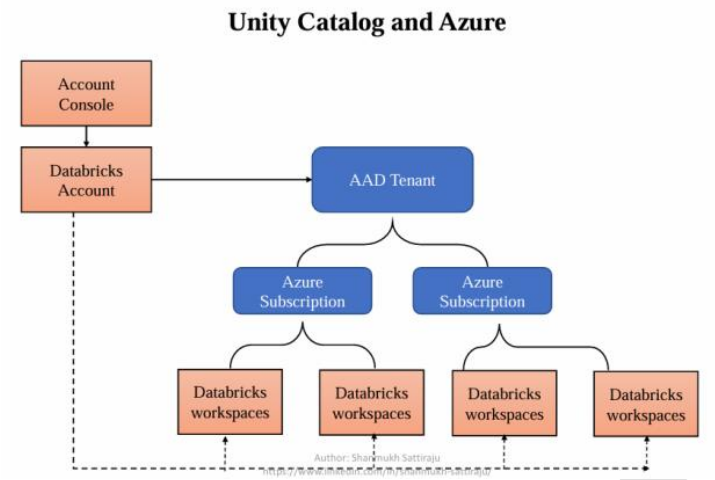
Unity Catalog is a centralized governance layer in Databricks that simplifies user and metadata management across multiple workspaces. It enables unified access control, data lineage, discovery, monitoring, auditing, and sharing, ensuring seamless management and governance in one place.

Hands-on:

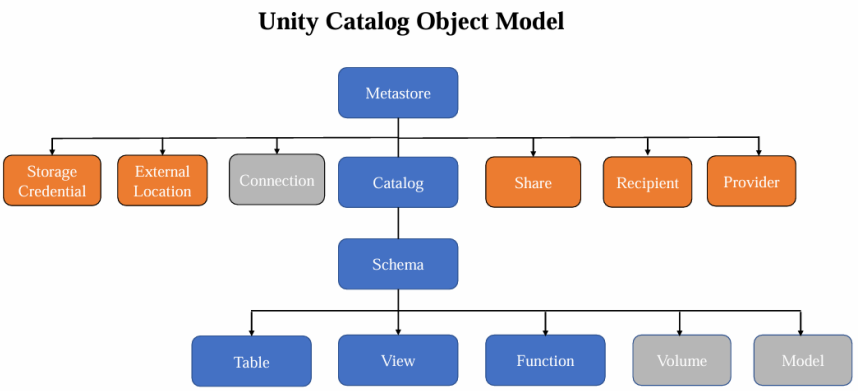
Step 1: In Azure Portal, create a Databricks workspace and ADLS Gen2, add these two Databricks workspace and ADLS Gen2 in "Favorite" section.



Step 2: search for "Access connectors for Azure Databricks", create "New", only give resource group name and Instance name "access-connectors-sachin" here, you need not to change anything here. Click on "Go to Resource". Now in "Overview", "Resource ID", can use this "Resource ID" while creating the Metastore.

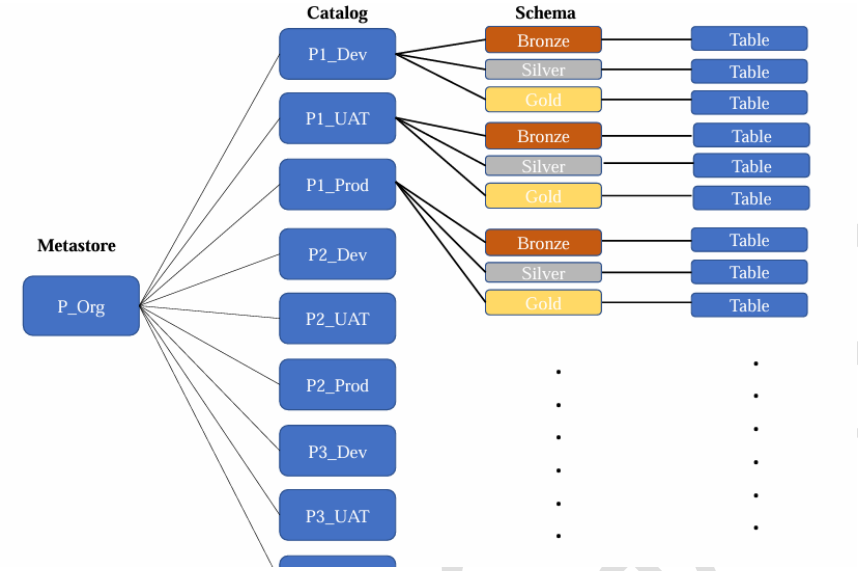


Step 3: Create ADLS Gen2, "deltadbstg"-> "test"-> "files"->"SchemaManagementDelta.csv". Now give access of this Access connectors to ADLS Gen2, go to ADLS Gen2, go to "Access Control IAM" from left pane, click on "Add"-> "Add Role Assignment"-> search for "Storage Blob Data Contributor", in "Members", select "Assign Access to"->"Managed Identity" radio button, "+Select Members"-> select "Access connectors for Azure Databricks" under "Managed identity" drop down menu-> "Select"-> "access-connectors-sachin" -> "Review+Assign".



- Now using this managed identity, our Unity catalog or Metastore can access this particular storage account. And the reason why we are doing is we need to have a container where that is going to be accessed by the unity catalog to store its managed tables, and we will see that in upcoming lectures.

Step 4: To use Unity Catalog: following are pre-requisites:



1. Must have Databricks Premium Workspace
2. Configure Metastore (Done in Step 3)
3. Attach premium workspace to Metastore
4. Note: https://accounts.azuredatabricks.net/ , in case you are not able to access "Manage Account" setting in my Azure Portal, click here:
5. a. https://learn.microsoft.com/en-us/answers/questions/1843839/not-able-to-access-manage-account-setting-in-my-az
6. b. https://learn.microsoft.com/en-us/answers/questions/1344479/unable-to-login-accounts-azuredatabricks-net-and-c
7. c. https://learn.microsoft.com/en-us/azure/databricks/admin/#account-admins

- **Solution:** Try to login using this link: https://accounts.azuredatabricks.net/login, then provide the user who is having "Global Administrator of your subscription", check this link for solution: https://learn.microsoft.com/en-us/answers/questions/1843839/not-able-to-access-manage-account-setting-in-my-az

    - Now go to Databricks, we need to start a creating a meta store, meta store is top level container in the unity catalog, go "Manage Account" under "Sachindatabricks name" from right top -> "Catalog" from left pane, "create meta store", provide "Name" as "metastore-sachin","Region"(can create one meta store in single region), "ADLS Gen2 path" (go to ADLS Gen2-> create container-> "Add Directory", paste <container_name>@<storage_account_name>.dfs.core.windows.net/<directory_Name> In the sample format of test@ deltadbstg.dfs.core.windows.net/files

- Or test@deltadbstg123456.dfs.core.windows.net/files), "Access connector ID" (go to "Access connectors"-> "access-connectors-sachin" -> copy that "resource ID")-> "Create".
- Attach with any workspaces. "Enable Unity Catalog?"-> "Enable".

Step 5: Create the required users to simulate the real time environment: go to "Microsoft Entra ID"-> "Add"-> "users" from left pane-> "new user"-> "create new user"-> give any name to "User Principal Name"-> give any display Name "Sachin_Admin" ->give custom password not "Auto generate password"-> not required to change any "Properties", "Assignments"-> click on "Create".

- Now login from https://accounts.azuredatabricks.net/ using username and password from Step 5. Using unity catalog, we can access to this user. My username is : sa******xgmail.onmicrosoft.com and password is K***r3**#

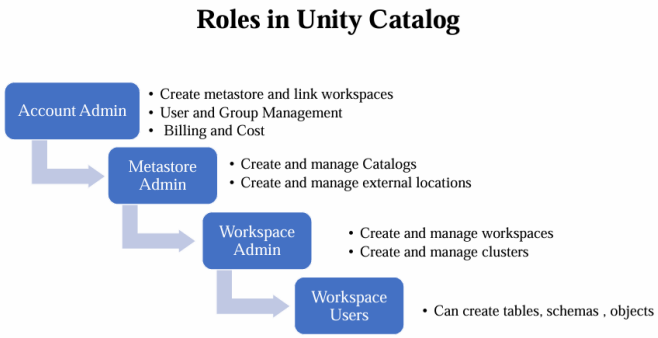
Step 6: Create one more user as in Step 5, now we two new users.

- So these are two new users we have created for this session, where we will try to simulate the real time environment by giving them required access to understand the roles and responsibilities clearly, because user management is something, if you are in a project, generally there will be an admin who can do this, but in real times they will expect you to handle this by your own. A data engineer must also be aware who can access what.
- First user will be Workspace admin and second will be developer.

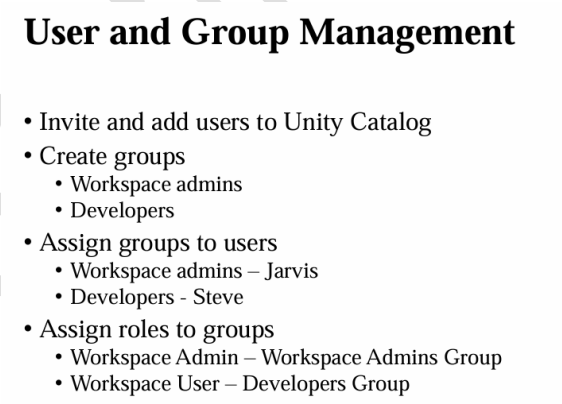
Step 7: Now in Databricks portal, click on "Manage Account", from right top, this Databricks portal is created neither by Workspace admin nor developer, in order to add user, click on "User Management" from left pane, we need to add both Workspace admin and developer.

- Click on "Add User"-> paste email id from "Microsoft Entra ID" -> "User Principal name", can give any "first name" and "last name" as "Workspace admin". Now add developer "User Principal name" in same way.
- Click on "Setting" from left, -> "User Provisioning"-> "Set up user provisioning".
- Open a "incognito window" mode to open https://portal.azure.com/#home with "admin Sachin" and "Developer Sachin" both, it will ask to create new password.

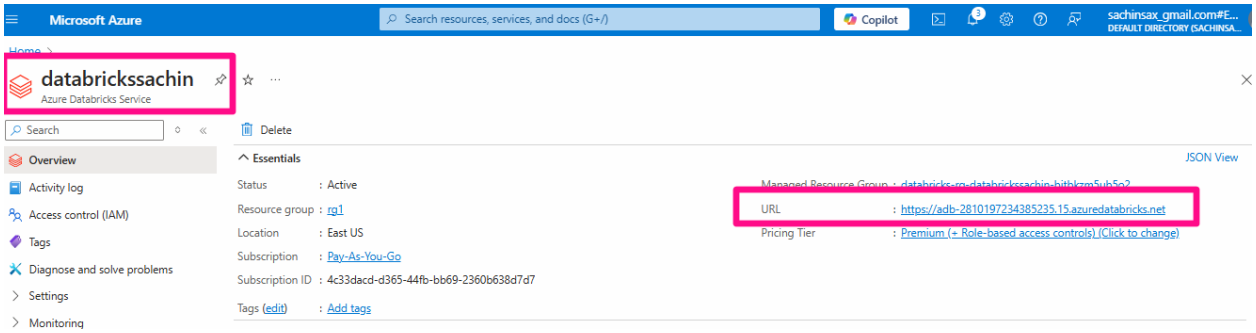
Step 8: to create group, click on “Manage Account”, from right top, this Databricks portal is created neither by Workspace admin nor developer, in order to add user, click on “User Management” from left pane-> “Groups” -> “Add Group”, we are going to create two groups, first group is “Workspace Admins”-> “Add Members” from admin only and second group is “Developer team”-> “Add Members” of developer only.



Step 9: It’s time to give permission, in Databricks portal, click on “Manage Account”, from right top, this Databricks portal is created neither by Workspace admin nor developer, in order to give permission, click on “Workspaces”, click on respective “Workspace” -> inside it “permissions”-> “Add Permissions”-> we need to add groups which we created in Step 8, to admin group assign “Permission” as “Admin” and to developer group assign “Permission” as “User”.



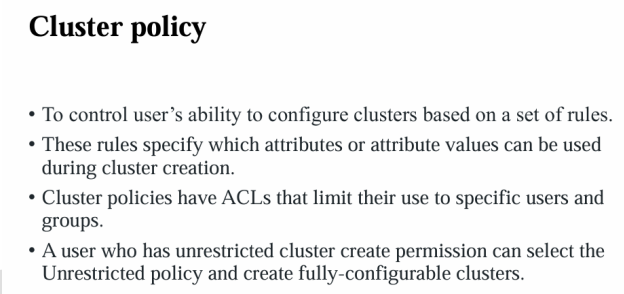
Step 10: Now login from https://portal.azure.com/signin/index/ using username and password, need to paste databricks workspace go to Databricks portal, click on “Manage Account”, from right top, this Databricks portal is created neither by Workspace admin nor developer, go to Azure portal from main where we created first Databricks workspace, copy “Workspace URL” ending with xxx.azuredatabricks.net.



Sign in with admin credentials, in similar way, go to Azure portal from main where we created first Databricks workspace, copy "Workspace URL" ending with xxx.azuredatabricks.net and sign in with user credentials.

- **Just check that in developer portal, we do not have "Manage Account" setting, also in developer portal cannot see any compute resources in "Compute" tab.**

Step 11: Create Cluster Policies: login with admin and move to databricks portal from this "sachin admin" login, go to "admin setting" from right top.



- **Click on "Identity and access" from second left pane. Click on "Manage" from "Management and Permissions" in "Users". Click on "Kumar Developer" right three dots, click on "Entitlements", check on "Unrestricted cluster creation", "Confirm" it.**
- **Now check "Compute" tab of "Kumar Developer" in databricks portal that, this "create compute " resource is now enabled.**
- **We do not want to give all kind of "Compute" resources to "Kumar Developer", so we can restrict by using create policies, otherwise,** it will result in a significantly high bill, subsequently, **and it may incur a substantial expense.**
- **(This step is for disable Compute Resource in developer portal)To create policies, click on "Kumar Developer" right three dots, click on "Entitlements", check off "Unrestricted cluster creation", "Confirm" it. Now check "Compute" tab of "Kumar Developer" in databricks portal that, this "create compute" resource is now disabled again.**

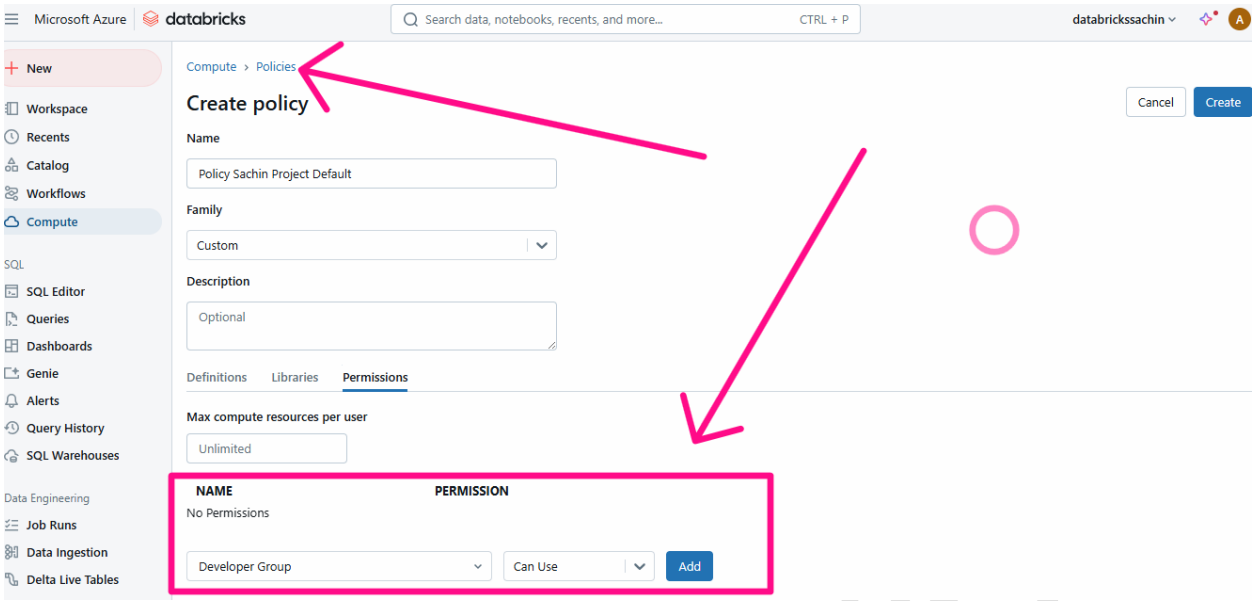
- **Compute policy reference: https://learn.microsoft.com/en-us/azure/databricks/admin/clusters/policy-definition**

- **Jump to “”Sachin Admin” databricks portal, click on “compute”, click on “Policies”: click on “create policy”-> give “Name” as “Sachin Project Default Policy” , select “Family” as “custom”->**

Change the following code:

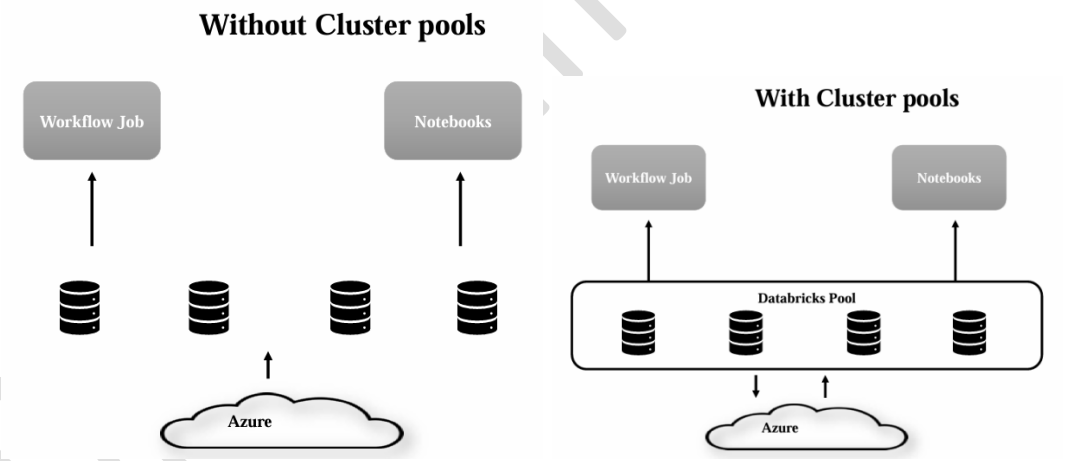
```
{
 "node_type_id": {
  "type": "allowlist",
  "values": [
   "Standard_DS3_v2"
  ]
 },
 "spark_version": {
  "type": "fixed",
  "value": "13.3.x-scala2.12"
 },
 "runtime_engine": {
  "type": "fixed",
  "value": "STANDARD",
  "hidden": true
 },
 "num_workers": {
  "type": "fixed",
  "value": 0,
  "hidden": true
 },
 "data_security_mode": {
  "type": "fixed",
  "value": "SINGLE_USER"
 },
 "cluster_type": {
  "type": "fixed",
  "value": "all-purpose"
 },
 "instance_pool_id": {
  "type": "forbidden",
  "hidden": true
 },
 "azure_attributes.availability": {
  "type": "fixed",
  "value": "ON_DEMAND_AZURE",
  "hidden": true
 },
 "spark_conf.spark.databricks.cluster.profile": {
  "type": "fixed",
  "value": "singleNode",
  "hidden": true
 },
 "autotermination_minutes": {
  "type": "fixed",
  "value": 20
 }
}
```

- **Click on “Permission” in ”Sachin Admin” databricks portal, click on “compute”, click on “Policies”: click on “create policy”-> give “Name” as “Sachin Project Default Policy”, in “Name”, select “Developers” -> “can use”, click on “Create” from right top.**



- **Now, if you jump to as "Sachin Developer" databricks portal, "Compute" tab, one "Sachin Project Default Policy" will appear at right top.**

Step 12: Cluster Pools in Databricks:



Question: What is pool? Why we use pool? How to create pool in Databricks?

- Pool is used to reduce cluster start time while auto scaling, you can attach a cluster to a predefined pool of idle instances.
- When attached to a pool, a cluster allocates its driver and worker nodes from the pool.
- If the pool does not have sufficient idle resources to accommodate the cluster's request, the pool expands by allocating new instances from the instance provider.
- When an attached cluster is terminated, the instances it used are returned to the pool and can be reused by a different cluster.

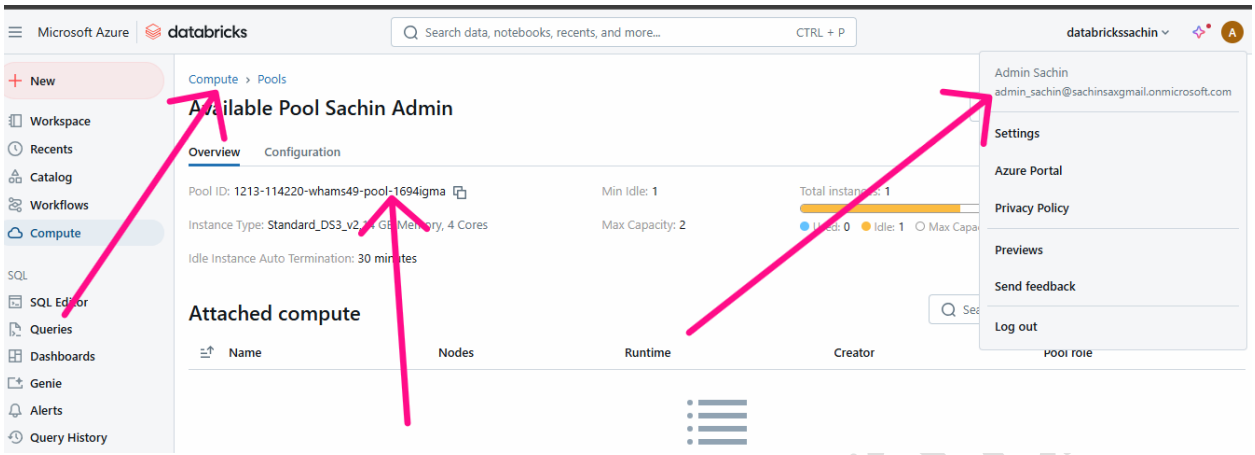
- **Purpose of Cluster Pools**:
  - Reduce cluster creation time for workflows or notebooks.
  - Enable ready-to-use compute resources in mission-critical scenarios.
- **Cluster Creation Process**:
  - Compute resources are required to run workflows or notebooks.
  - Creating a cluster involves requesting virtual machines (VMs) from a cloud service provider.
  - VM initialization takes ~3-5 minutes, which may not be ideal for real-time tasks.
- **Cluster Pool Functionality**:
  - Cluster pools pre-request VMs from the cloud provider based on configurations.
  - Keeps some VMs ready in a running or idle state.
  - Enables faster cluster creation by utilizing these pre-initialized VMs.
- **Performance and Cost Trade-Off**:
  - Performance improves as clusters are ready in reduced time (~half the usual time).
  - Databricks does not charge for idle instances not in use, but cloud provider infrastructure costs still apply.
- **Use Case**:
  - Ideal for workflows or notebooks needing rapid cluster creation.
  - Balances between cost and efficiency by keeping resources ready.
- **Key Takeaway**:
  - Cluster pools enhance performance by maintaining idle VMs for quick allocation, albeit with associated cloud costs.

- **Cluster Pools in Databricks Hands-on: Jump to "Sachin Admin" databricks portal, click on compute, go to "Pool", click on "create pool", name as "Available Pool Sachin Admin", pool will already keep you instances in ready and running state so that we can use them while creating the cluster. And these will access the resources which are readily available.**

- **Also keep "Min Idle" as 1 and "Max Capacity" as 2. Now let me make the minimum idle instance to one and maximum two. This means all the time this one instance will be in ready and in running state. And in case if this one instance is used by any cluster, another will be in the Idle state because minimum one will be idle all the time, irrespective of the one is attached or not. So in maximum of two will be created. So one can be used by cluster and if that is already been occupied, another one will be in the idle state.**
- **Change "terminate instances above minimum tier" to 30 minutes of idle time.**
- **Change "Instance Type" to "Standard_DS3_v2"**
- **Change "On-demand/spot" to "All On-demand" radio button, bcs sometimes Spot instances are not available.**
- **Create it. It will take much time. Copy Pool ID from here.**
- **Now go to Edit Policies under "Policy tab" which was done in Step 11, make changes in:**

```
},
"instance_pool_id": {
  "type": "forbidden",
  "hidden": true
},
```

To Get Pool ID here:




```
"instance_pool_id": {
  "type": "fixed",
  "value": "1211-104550-gybed90-pool-ns7wqs2q<Your Pool ID>"
```

- **Now go to Compute Tab in "Sachin Admin" databricks, click on "Pool"->select given "availavle Pool Sachin Admin" -> click on Permission-> select "Developers group" (not individual developer) to "Can Attach to"-> "+Add", "Save" it.**

Step 13: Creating a Dev Catalogs: go to "Sachin Admin" databricks portal, go to "Catalog" tab, but "Create Catalog" is disabled now because we haven't define this permission, in order to give permission, go to "Main Datbricks" portal (neither Sachin Workspace admin nor Kumar developer), go to databricks portal, go to "Catalog" tab, "Catalog Explror" -> click on "Create Catalog" from right, name "Catalog name" as "DevCatalog", type as "Standard", skip "Storage location". Click on "Create".

Step 14: Unity Catalog Privileges:

Unity Catalog Privileges

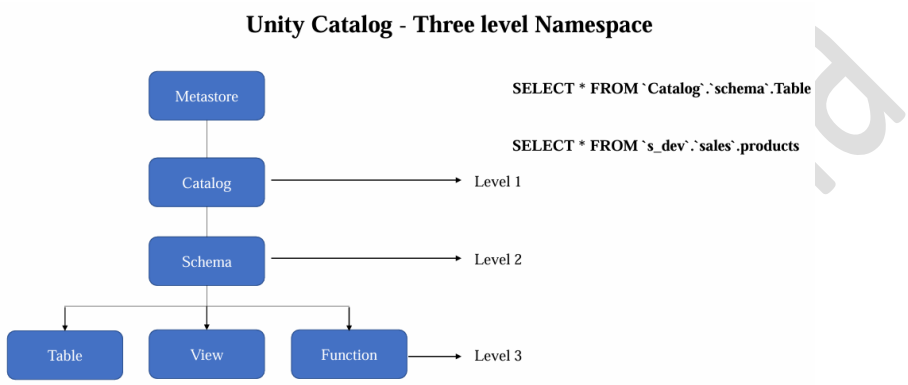
- Privileges are permissions that we assign on objects to users
- Can use SQL command or Unity Catalog UI

Eg:

GRANT **privilege_type** ON **securable_object** TO **principal**

Privilege_Type : Unity Catalog permissions like SELECT, CREATE
Securable_object: Any object like SCHEMA, TABLE , etc
Principal: Can be a user, group, etc.

- Go to “Sachin Admin” databricks portal, but still can’t see “Dev Catalog” because Sachin Admin and Kumar developer both are not having the required privileges or permission to use.
- Go to “Main Databricks” portal who is account admin (neither Sachin Workspace admin nor Kumar developer) go to “Catalog”, Click on “Dev Catalog”, then “Permissions”, then “Grant”, this screen is Unity catalog UI to grant privileges to “Sachin Admin”, then click on “Grant”, select group name “WorkSpace admins” checkbox on “create table”, “USE SCHEMA”, “Use Catalog” and “Select” in “Privileges presets”, do not check anything here. Click on “Grant”. Now, go to “Sachin Admin” databricks portal, “Dev Catalog” is showing here.



- To transfer ownership of “Dev Catalog”, go to “Main Databricks” portal who is account admin (neither Sachin Workspace admin nor Kumar developer), go to “Catalog”, Click on “Dev Catalog”, click on pencil icon from mid top near Owner: sacinsax@gmail.com, “Set Owner for Dev Catalog”, change to “Workspace admins” not to specific user, bcs if one user leave the organization then it creates havoc situations.
- Now, go to “Sachin Admin” databricks portal, “Dev Catalog” is showing here.
- Now, go to “Sachin Admin” databricks portal, create a “notebook” here, to run any cell in this notebook, we need “Compute”, select “Create with Personal compute”, “Project Defaults”.
- Go to “Sachin Admin Databricks” portal go to “Catalog”, Click on “Dev Catalog”, then “Permissions”, then “Grant”, this screen is Unity catalog UI to grant privileges to “Sachin Admin”, then click on “Grant”, select group name “WorkSpace admins” checkbox on “Use Catalog”, “USE SCHEMA”, “Create Table” and “Select” in “Privileges presets”, do not check anything here.
- Now Run SQL command, file is saved in “Day 9” folder with name “Unity Catalog Privileges.sql”. in code GRANT USE_CATALOG ON CATALOG `devcatalog` TO `Developer Group`

Step 15: Creating and accessing External location and storage credentials:

- Step A: Go to “Sachin Admin Databricks” portal go to “Catalog”, we do not find any external data here, to find “External Data”, go to “Main Databricks” portal who is account admin

(neither Sachin Workspace admin nor Kumar developer) go to “Catalog”, in “Catalog Explorer”, there is “External Data” below, click on “Storage Credentials”.

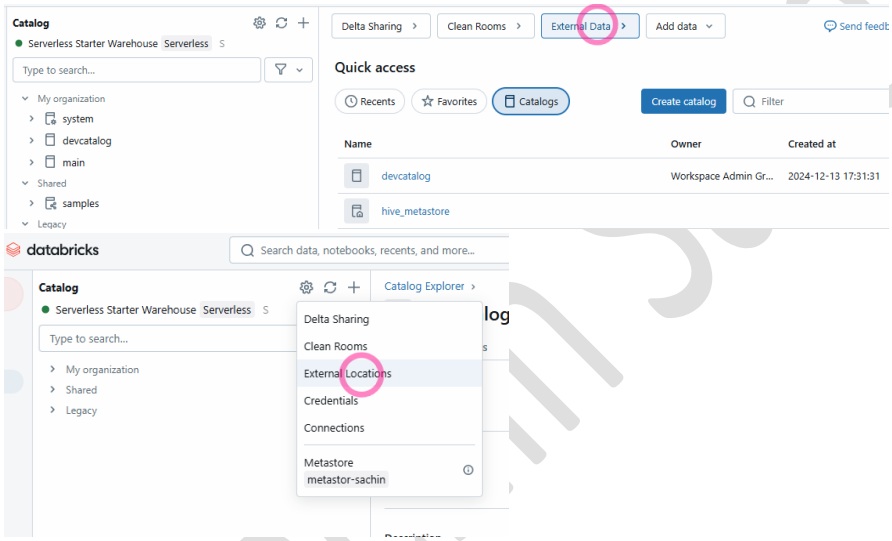
-

➢ **Note: In case following error is being occurred while creating the External Location:**

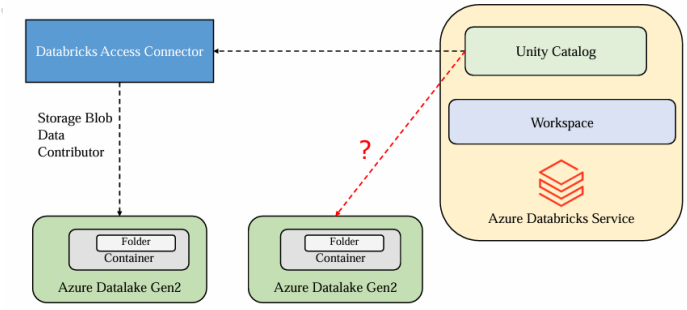
“Input path url 'abfss://landing@deltadbstg1122.dfs.core.windows.net/files' overlaps with an existing external location within 'CreateExternalLocation' call. Conflicting location: metastore_root_location.”

Do change the External location ADLS Gen2 address eg: 'abfss://landing@deltadbstg1122.dfs.core.windows.net/files' to 'abfss://test@deltadbstg1122.dfs.core.windows.net/dir' or change the Metastore location.

Try again to create the External Location instance.

- 

- **Step B: Now in ADLS Gen2, “deltadbstg”-> “test”-> “files”->”SchemaManagementDelta.csv”.**
- **Now give role assignment “Storage blob Data Contributor” to “db-access-connector” from IAM role in Azure Portal of Main admin.**



- **Step C: Now go to Step A Databrick portal, click on “create credential” under “External Data” below, click on “Storage Credentials”, “Storage Credentials Name” as “Deltastorage”, to get “Access connector Id”, go to “db-access-connector” from Azure Portal, will find “Resource ID”, copy this and paste to “Access connector Id”, click on “create”.**

- **Step D: Go to "Main Databricks" portal who is account admin (neither Sachin Workspace admin nor Kumar developer) go to "Catalog", in "Catalog Explorer", there is "External Data" below, click on "External Data", click on "Create external location" -> "Create a new external location"click on "External location name": "DeltaStorageLocation", in "Storage credential", select "Deltastorage" which we created in Step C.**
  - **To find URL: abfss://test@deltadbstg.dfs.core.windows.net/files (go to ADLS Gen2 "deltadbstg"-> "EndPoints"-> "Data Lake Storage"), click on "create".**
- **Click on "Test Connection".**
- **Step E: create a notebook in "Main Databricks" portal who is account admin (neither Sachin Workspace admin nor Kumar developer), create a compute, create with "Unrestricted", "Multi node", create a Access mode "Shared" , uncheck "Use Photon Acceleration", Min workers: 1, Max workers: 2.**
- **Run the following code in notebook in Main Databricks (Neither in Admin nor in Developer):**

```
%sql
CREATE TABLE `devcatalog`.`default`.Person_External
(
    Education_Level STRING,
    Line_Number INT,
    Employed INT,
    Unemployed INT,
    Industry STRING,
    Gender STRING,
    Date_Inserted STRING,
    dense_rank INT)
USING CSV
OPTIONS(
  'header' 'true'
)
LOCATION 'abfss://test3@deltadbstg.dfs.core.windows.net/dir'
```

Note: The LOCATION keyword is used to configure the created Delta Lake tables as external tables. In Databricks, when you create a database and use the LOCATION keyword, you are specifying a custom directory where both the metadata and the data files for the tables within that database will be stored. This overrides the default storage location that would otherwise be used. This feature is particularly useful for organizing data storage in a way that aligns with specific project structures or storage requirements, ensuring that all contents of the database are stored in a designated path.

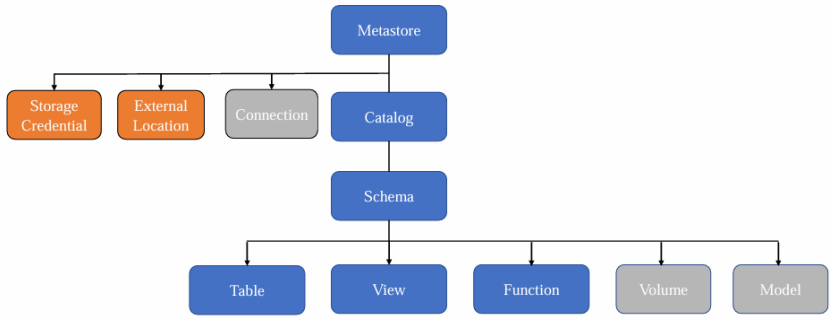
- **Df=(spark.read.format('csv').option('header','true').load('abfss://test@deltadbstg.dfs.core.windows.net/files / '))**
- **Display(Df)**

Step 16: Managed and External Tables in Unity Catalog: Do hands on also.

- Managed Tables
  - These can be defined without a specified location
  - The data files are stored within managed storage in Delta format
  - Dropping the table not only removes its metadata from the catalog, but also deletes the actual data but in Unity Catalog the underlying data will be present for 30 days.

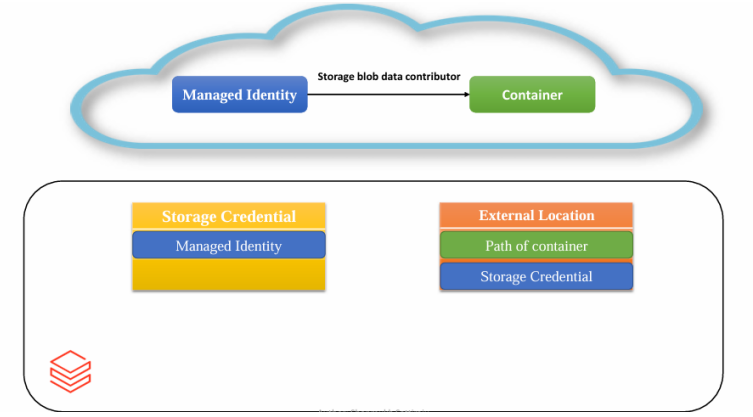
- External Tables
  - You need to have an EXTERNAL LOCATION and STORAGE CREDENTIALS created to access the external storage.
  - These can be defined for a custom file location, other than the managed storage
  - Dropping the table deletes the metadata from the catalog, but doesn't affect the data files.

Unity Catalog Object Model



Question: Which of the following is primary needed to create an external table in an Unity Catalog Enabled workspace?

Answer: You need an external location created primarily pointing out to that location , So you can get access to the path to create external table.



Question: Can managed table use Delta, CSV, JSON, avro format?

Answer: No, Managed table can use only Delta format.

Storage Credential	External Location
An authentication and authorization mechanism for accessing data stored	Serves as a reference point for External storage
Stores the access Credentials to provide access to External Location	Stores the path of the external storages that you want to access.
Credentials can be Managed Identities / Service principles	Makes use of Storage credential to get access to External Storage

Day 10: Spark Structured Streaming – basics

Note: This hands on can be done on Databricks community addition, otherwise, it will result in a significantly high bill.

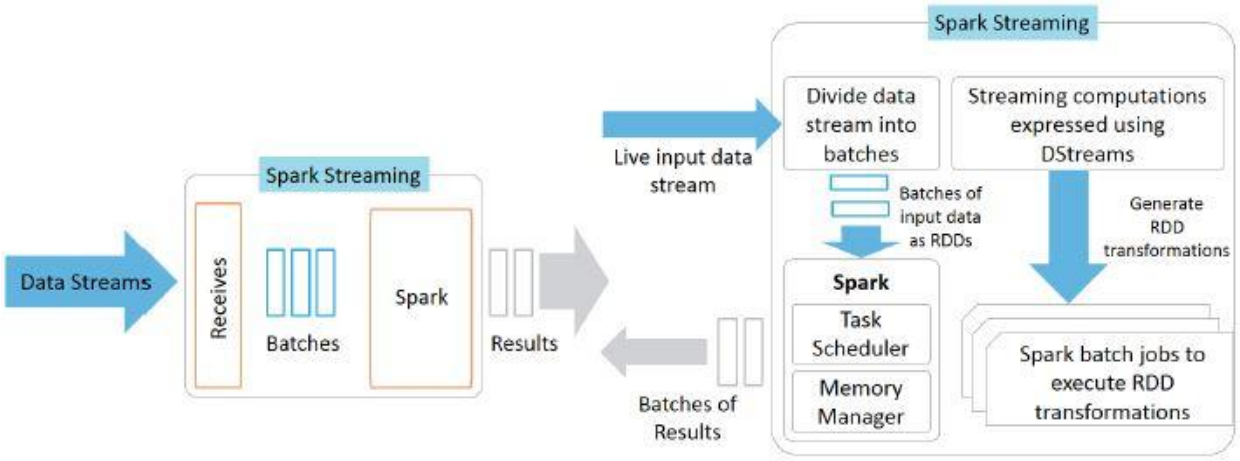
Definition: A data stream is an unbounded sequence of data arriving continuously. Streaming divides continuously flowing input data into discrete units for further processing. Stream processing is low latency processing and analyzing of streaming data.

In Spark Structured Streaming, in order to process data in micro-batches at the user-specified intervals, you can use processingTime keyword. It allows to specify a time duration as a string.

Reference: https://docs.databricks.com/structured-streaming/triggers.html#configure-structured-streaming-trigger-intervals

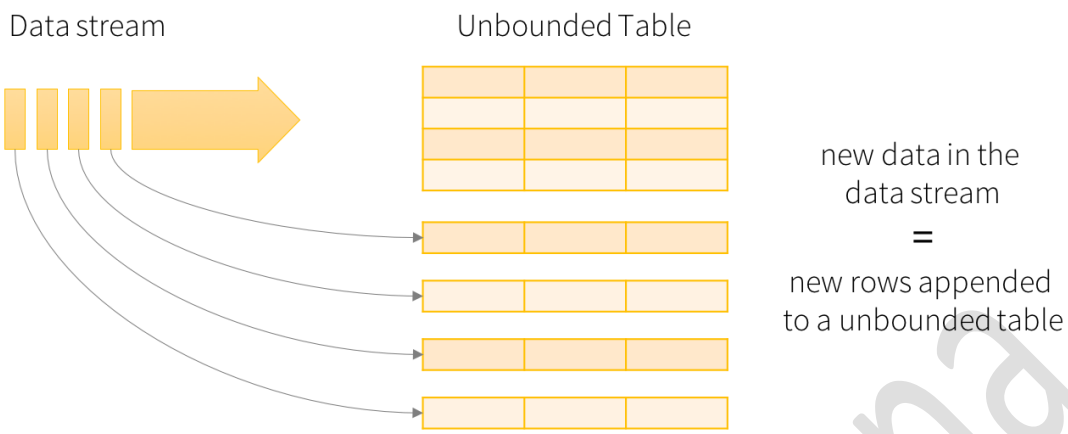
Data ingestion can be done from many sources like Kafka, Apache Flume, Amazon Kinesis or TCP sockets and processing can be done using complex algorithms that are expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases and live dashboards.

Firstly, streaming data is something that will never have a complete data for analysis as data is continuously coming in where there is no stop. To understand this, let's first conceptualize the structured streaming. So let's take a stream source like an IoT device which is collecting details of vehicles travelled on a road. There can be thousands of vehicles that travelled on a road or a log collecting system from an application like social media platform or e-commerce site.



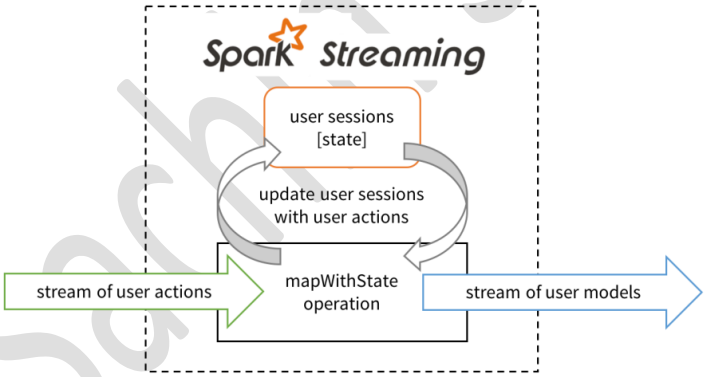
That application can be used by thousands of users, where they can be doing many clicks and you want to collect those click streams. So these are basically the endless incoming data, which is called incoming data stream or streaming data.

Why Streaming in Spark?

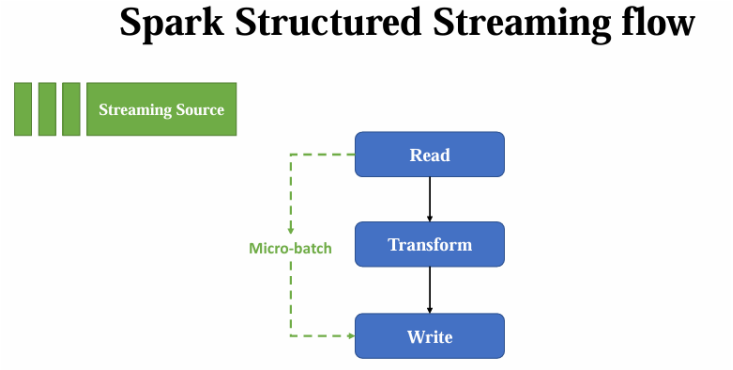


Data stream as an unbounded table

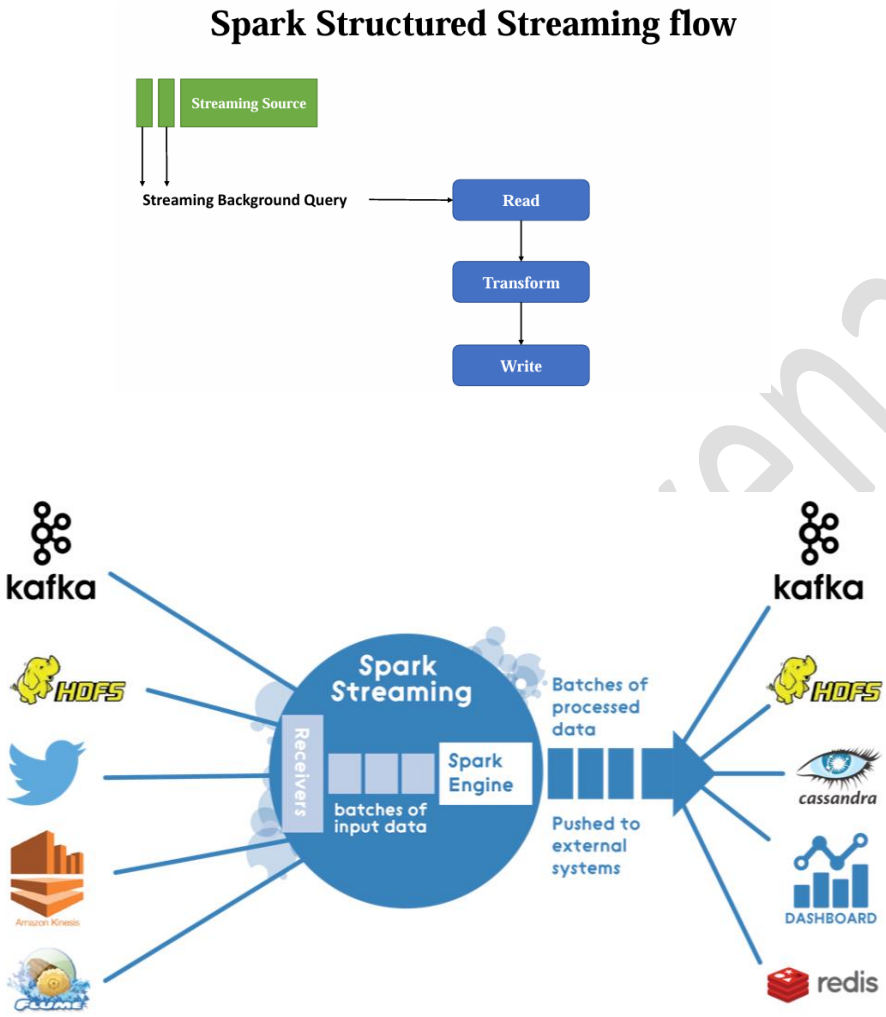
Batch processing systems like Apache Hadoop have high latency that is not suitable for near real time processing requirements.



There is a set of worker nodes, each of which runs one or more continuous operators. Each continuous operator processes the streaming data one record at a time and forwards the records to other operators in the pipeline.



Data is received from ingestion systems via Source operators and given as output to downstream systems via sink operators.



- ✓ **Continuous operators are a simple and natural model. However, this traditional architecture has also met some challenges with today's trend towards larger scale and more complex real-time analytics.**

Advantages:

a) Fast Failure and Straggler Recovery

In real time, the system must be able to fastly and automatically recover from failures and stragglers to provide results which is challenging in traditional systems due to the static allocation of continuous operators to worker nodes.

b) Load Balancing

In a continuous operator system, uneven allocation of the processing load between the workers can cause bottlenecks. The system needs to be able to dynamically adapt the resource allocation based on the workload.

c) Unification of Streaming, Batch and Interactive Workloads

In many use cases, it is also attractive to query the streaming data interactively, or to combine it with static datasets (e.g. pre-computed models). This is hard in continuous operator systems which does not designed to new operators for ad-hoc queries. This requires a single engine that can combine batch, streaming and interactive queries.

d) Advanced Analytics with Machine learning and SQL Queries

Complex workloads require continuously learning and updating data models, or even querying the streaming data with SQL queries. Having a common abstraction across these analytic tasks makes the developer's job much easier.

Step 1: Understanding micro batches and background query: This hands on can be done on Databricks community addition, otherwise, it may incur a substantial expense.

Note: Each unit in streaming is called as micro-batch and it is the fundamental unit of processing.

- **Create a compute resource and create a notebook and run the file named as : "Day 10 Streaming+basics.ipynb".**

- **Upload the file "Countries1.csv" to "FileStore" in "DBFS", create a new directory named "streaming".**
- **Once we have read the data using "readStream" function, let's see what jobs it has initiated, go "Compute" resource from right top, click on "Spark UI".**

- **Databricks File System (DBFS): The Databricks File System (DBFS) is a distributed file system mounted into a Databricks workspace and available on Databricks clusters. DBFS is an abstraction on top of scalable object storage that maps Unix-like filesystem calls to native cloud storage API calls.**

Question: What are the limitations in Jobs?

A. The number of jobs is limited to 1000.

B. A workspace is limited to 150 concurrent (running) job runs.

C. A workspace is limited to 1000 active (running and pending) job runs.

- By observing, we can see in “Saprk UI” no job has been initiated, only jobs are created when we are trying to get some data.
- For streaming data frames, most of the actions are not supported, but transformations are supported.
- If you trying to use show method, it’s not working, “df.show()”.
- Instead, use display method, “display(df)”, streaming data is something it is going to accept the files under the particular directory.
- Now job is still running and it is displaying the data to us, display “dashboards” which is just below “display(df)”, it’s showing statistic graphs.
- Go “Compute” resource from right top, click on “Spark UI”, and see agin, there is “Executor driver added”.
- Upload the second file ”Countries2.csv” to “FileStore” in “DBFS”, in “streaming” directory, .
- Now go to “notebook” again, observe that data is again processing in “Input vs Prcoessing Rate”, there is a spike indicating new data is available.
- In “Spark UI”, there are two jobs, means for each data there is one job it is going to read data.
- In “Spark UI” tab, click on “Structure Streaming” there is something called “Display Query”.
- Upload the third file ”Countries3.csv” to “FileStore” in “DBFS”, in “streaming” directory, see third micro batch, this streaming query in “Structure Streaming” there is something called “Display Query”, acts as a watcher.
- To stop this Streaming Query, you can just click on “cancel” there.
- Several other resources available for Live streaming: File source (DBFS), Kafka, Socket, Rate etc. , Socket, Rate Sources are useful for testing purpose not for real deployment. Several sinks are also available.
- WriteStream : A query on the input will generate the “Result Table”. Every trigger interval (say, every 1 second), new rows get appended to the Input Table, which eventually updates the Result Table. Whenever the result table gets updated, we would want to write the changed result rows to an external sink.


```
WriteStream = ( df.writeStream
       .option('checkpointLocation',f'{source_dir}/AppendCheckpoint')
       .outputMode("append")
       .queryName('AppendQuery')
       .toTable("stream.AppendTable"))
```

- So coming to check pointing it is basically used to store the progress of our stream. Like having the metadata till where the data is copied. Which means if I am just telling some directory, if there is a stream that is available, it is going to read that particular stream, and it is going to write the data to a destination, and it is going to note down till where the data is been copied. It is not going to store the data; it is just going to have the metadata till where the point is copied. And what exactly is the use of this check pointing.

- To make the query executes a micro-batch to process data every 2 minutes:
  trigger(processingTime=”2 minutes")

In Spark Structured Streaming, in order to process data in micro-batches at a user-specified intervals, you can use the processingTime trigger method. This allows you to specify a time duration as a string. By default, it's "500ms".

StreamWriter

```
<StreamingDataframe>.writeStream
                    .option('checkpointLocation',<Location>)
                    .outputMode('append')
                    .toTable('<TableName>')
```

Checkpoint

- To develop fault-tolerant and resilient Spark applications.
- It maintains intermediate state on fault-tolerant compatible file systems like HDFS, ADLS and S3 storage systems to recover from failures.
- Must be **unique** to each stream

- **Regarding checkpointing in Spark Structured Streaming:**
    1. **Checkpoints stores the current state of a streaming job to cloud storage**
    2. **Checkpointing allows the streaming engine to track the progress of a stream processing**
    3. **Checkpoints cannot be shared between separate streams**
    4. **To specify the checkpoint in a streaming query, we use the checkpointLocation option.**
    5. **Checkpointing with write-ahead logs mechanism ensure fault-tolerant stream processing**

- Importance of Checkpoint Files: the importance of checkpoint files in Delta tables, how they are generated, and the types of information they contain. Checkpoint files help us query Delta tables efficiently, as they eliminate the need to scan the entire transaction log, which is created after every DML operation (INSERT, UPDATE, DELETE) on the Delta table. These checkpoint files store information about the latest transaction log files currently being referenced by the Delta table.

  1. State Management:

  Store metadata reflecting the table's state, ensuring consistency and fault tolerance.
  Efficient Querying: Delta Lake uses checkpoints to avoid re-scanning the entire transaction log, improving query performance.
  Fault Tolerance: Checkpoints provide a snapshot for quick recovery in case of system failures.
  Optimized Performance: They keep metadata compact for faster access and minimal overhead.

  2. Querying Flow:

  Checkpoint Files, Transaction Logs, and Parquet Data
  Checkpoint File: Delta checks the latest checkpoint to determine the table's state.
  Transaction Log: Checkpoint files reference the transaction log, which contains metadata pointing to relevant Parquet data files.
  Parquet Files: The transaction log provides pointers to the Parquet files holding the actual data, ensuring efficient data retrieval.

3. Parameters Controlling Checkpoint Creation

Checkpoint Interval: Defines the frequency of checkpoint creation (default is every 10 commits).
Log File Size: Large logs trigger checkpoint creation for better storage management.
Compaction: Delta Lake merges smaller transaction logs, often triggering checkpoints to optimize performance.

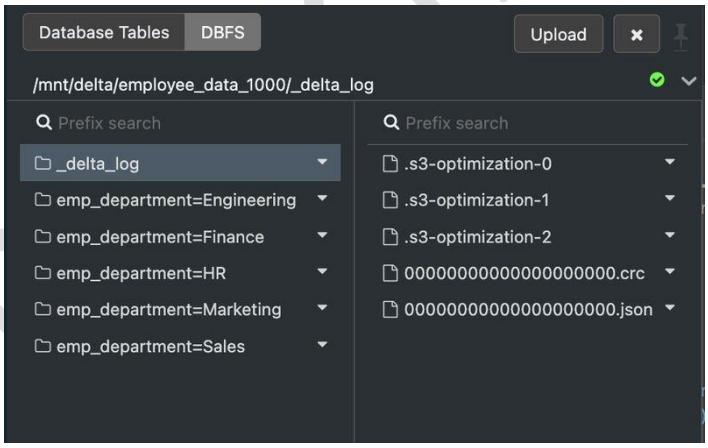
4. Spark Configuration to Control Checkpoint Generation
---
spark.databricks.delta.checkpointInterval: Controls the frequency of checkpoint file creation (default is 10 commits).
spark.databricks.delta.retentionDurationCheck.enabled: Enables retention policy for log and checkpoint file cleanup, ensuring efficient storage management.

5. Conclusion

Optimized Performance: Checkpoints help Delta Lake query efficiently by reducing overhead.

Data Consistency:

They maintain consistency between metadata and data files.
Scalability: Proper management of checkpoints and transaction logs ensures Delta Lake scales effectively.
I have attached a snapshot of checkpoint file also for reference.



- **So it is going to give the fault tolerance and resiliency to our streaming data. So the terms that you are seeing over there, it is to develop the fault tolerant and resilient spark applications.**
- **So to better understand the fault tolerance and the resilient terms, if there is any failure that occurs during the copy of this particular stream, spark is smart enough to start from the point of failure because it is going to store the intermediate metadata in the checkpoints. It will go to the checkpoint location, and it is going to see till where the data is copied, and it is going to begin the data copy from there.**

- So this gives the fault tolerant to this particular spark structure streaming, where the intermediary of the state is copied to particular directory.
- To check "appendtable" files: got to "Database Tables"-> "Stream"-> "appendtable".
- To check parquet files: got to "DBFS"-> "user"->"hive"->"warehouse"-> "stream.db".
- In "Spark UI" tab, click on "Structure Streaming" there is something called "AppendQuery".
- In "DBFS", in "streaming" directory, find "AppendCheckPoint", upload file "Countries4.csv", after executing following code:

```
WriteStream = ( df.writeStream
        .option('checkpointLocation',f'{source_dir}/AppendCheckpoint')
        .outputMode("append")
        .queryName('AppendQuery')
        .toTable("stream.AppendTable"))
```

- Keep in mind that: the community edition was designed in a way if the cluster is been terminated, and if you try to create a new cluster, previous databases will not persist, but folder "stream.db" still exists, but "stream.db" won't show any data when run in sql query. This is not issue with Azure databricks.
- Now run "Day 10 outputModes.ipynb" file.

outputModes

OutputMode	Usage	Description
Append	outputMode('append')	The records from incoming streams will be appended to destination
Complete	outputMode('complete')	All the processed rows will be displayed
Update	outputMode('update')	Spark will output only updated rows. This is valid only if there are aggregation results; otherwise, this would be similar to Append mode.

- OutputMode: The outputMode option in Spark Structured Streaming determines how the streaming results are written to the sink. It specifies whether to append new results, complete results (all data), or update existing results based on changes in the data.

- When defining a streaming source in Spark Structured streaming, what does the term "trigger" refer to?
- Answer: It triggers the start of the streaming application.

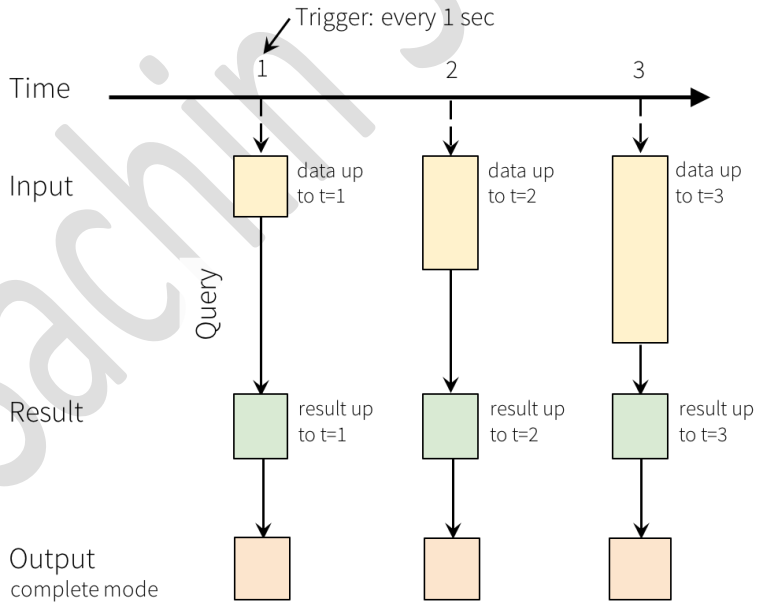
- Also run "Day 10 Triggers.ipynb" file, how do we know that it actually checked the input folder to know that click on click on "Structure Streaming", in "Spark UI" tab, then click on "Run ID" in "Active Streaming Queries".

- **By default, if you don't provide any trigger interval, the data will be processed every half second. This is equivalent to trigger(processingTime="500ms")**

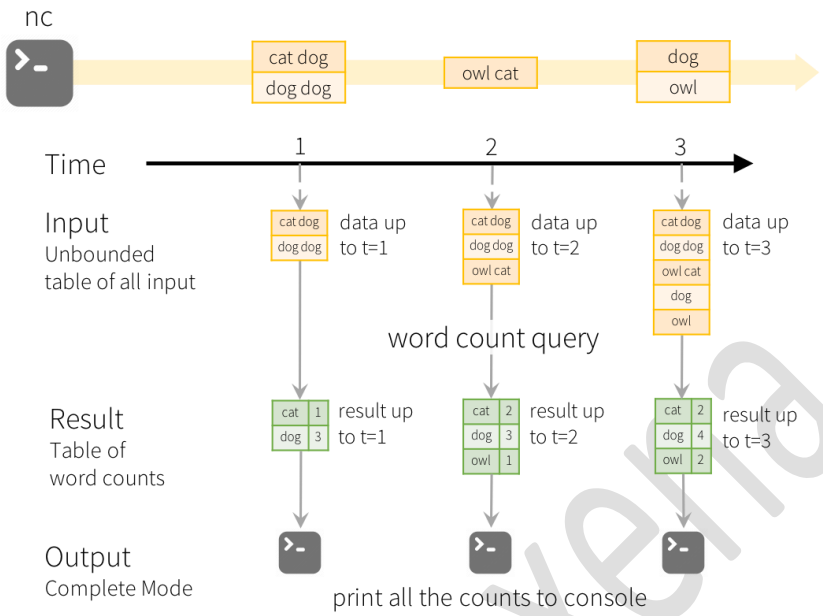
Triggers

Triggers	Usage	Description
Unspecified (default)		will trigger the microbatch for every 500 ms or half a second
processingTime (Fixed Interval)	. trigger(processingTime='2 minutes')	You can set processing time or time interval for each execution .
availableNow (OneTime)	.trigger(availableNow = True)	consumes all available records from previous execution as an incremental batch
Continuous (experimental)	.trigger(continuous = '1 second')	For ~1ms latency

- **Resource: https://spark.apache.org/docs/3.5.3/structured-streaming-programming-guide.html**
- **Resource: https://sparkbyexamples.com/kafka/spark-streaming-checkpoint/**



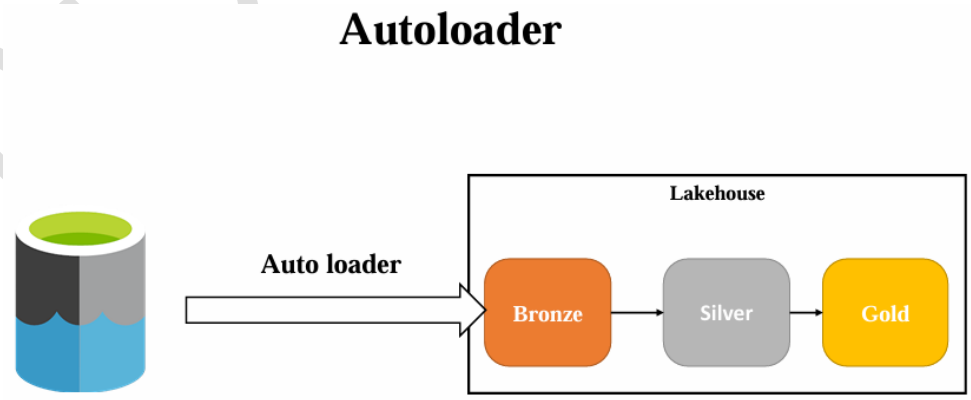
Programming Model for Structured Streaming



Model of the Quick Example

Day 11: Autoloader – Intro, Autoloader - Schema inference: Hands-on

Note: This hands on can be done on Databricks community addition, otherwise, it will result in a significantly high bill.



Why Autoloader?: In this session, let us now see about the auto loader, Let us first understand what exactly is the need of the auto loader before directly going to the definition.

Auto loader monitors a source location, in which files accumulate, to identify and ingest only new arriving files with each command run. While the files that have already been ingested in previous runs are skipped.

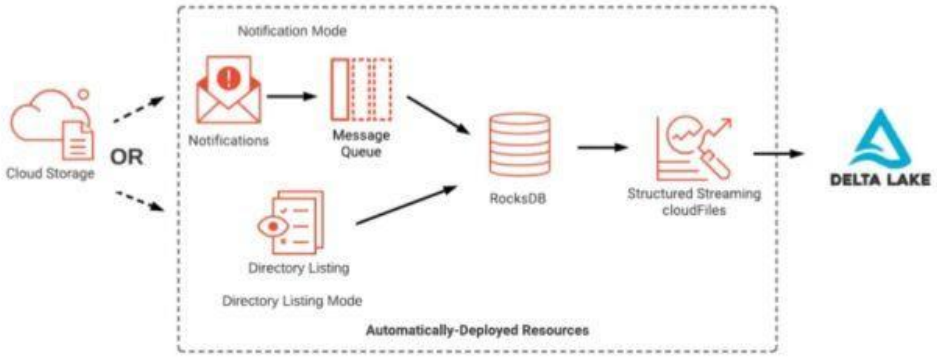
- **So in the real time project we will always have cloud storage where it is going to store our files. So in order to implement medallion architecture or Lakehouse architecture, we will generally read these files from cloud storage to a bronze layer.**

- **Ever Worked with Autoloader??**

Data Handling is one of the crucial segment of any Data related job as proper data planning drives into results which led to efficient and economical storage, retrieval, and disposal of data. When it comes to Data Engineering profile, Data Loading (ETL) plays an equivalent role too.

Data Loading can be done in 2 ways — Full Load or Incremental Load. Databricks provide a great feature with Auto Loader to handle the incremental ETL and taking care of any data that might be malformed and would have been ignored or lost.

Auto Loader supports both Python and SQL in Delta Live Tables and can be used to process billions of files to migrate or backfill a table. Auto Loader scales to support near real-time ingestion of millions of files per hour.



Auto Loader Working

Autoloader

- Autoloader is an **optimized data ingestion tool** that incrementally and efficiently processes new data files as they arrive in the cloud storage built into the Databricks Lakehouse.
- Auto Loader incrementally and efficiently processes new data files as they arrive in cloud storage without any additional setup.
- Auto Loader can load data files from Cloud Storages without being vendor specific (AWS S3 , Azure ADLS , Google Cloud Storage, DBFS).
- Auto Loader can ingest JSON, CSV, PARQUET, AVRO, ORC, TEXT, and BINARYFILE file formats
- This Auto loader is beneficial when you are ingesting data into your lakehouse particularly into bronze layer as a streaming query.

- **Auto Loader is based on Spark Structured Streaming. It provides a Structured Streaming source called cloudFiles.**
- **Reference: https://docs.databricks.com/ingestion/auto-loader/index.html**

- ✓ **Auto Loader Definition: Auto loader monitors a source location, in which files accumulate, to identify and ingest only new arriving files with each command run. While the files that have already been ingested in previous runs are skipped.**
- ✓ **Auto Loader incrementally and efficiently processes new data files as they arrive in cloud storage.**
- ✓ **Reference: https://docs.databricks.com/ingestion/auto-loader/index.html**

- **And from the bronze layer we are going to do the silver and gold and the downstream transformations in a medallion or a lake house project.**
- **Now, in order to get these files from cloud storage, which is like Azure Data Lake in Azure or a lake house project, and we need to ingest the cloud files or the files available in the cloud storage to bronze layer.**
- **So in order to ingest these files, you need to take care of many things. We need to ingest these files incrementally. And there can be billions of files inside the cloud storage. So you need to build a custom logic to handle the incremental loading.**

- **And also this would be quite complex task for any data engineer to set up an incremental load.**
- **Now we also need to handle the bad data. When you are trying to load this to the bronze layer, you need to handle the schema changes and things, etc. all these needs a complex logic to customize and handle these while reading the data from the data lake to bronze layer. So**

all these can be supported without explicitly defining any custom logic by making use of auto loader.

- So auto loader is a feature in the spark streaming, which can handle billions of data incrementally, and it is the best suited auto loader tool to load the data from the files in the cloud storage to bronze layer.

Implementing Autoloader

```
df_str = (spark.readStream
          .format("cloudFiles")     ## This will tell the spark to use AutoLoader.
          .option("cloudFiles.format","csv")  ## Tells Autoloader to expect csv files
          .option('header','true')
          .schema(schema)
          .load(f'{source_dir}')
)
```

- So this is the best beneficial tool when you are trying to ingest the data into your lake house, particularly into the bronze layer as a streaming query, where you can also benefit by making use of triggers. And you can implement this auto loader as a tool, which it can take care of everything for you.

Hands-on: run “Day 11 Autoloader+-+Schema+Evolution.ipynb” file

- Inside this file, .format(‘cloudFiles’), this will tell the spark to use the auto loader here, Cloud files is kind of an API that spark uses to use the auto loader feature.
- Schema evolution is a feature that allows adding new detected fields to the table. It’s activated by adding .option('mergeSchema', 'true') to your .write or .writeStream Spark command.

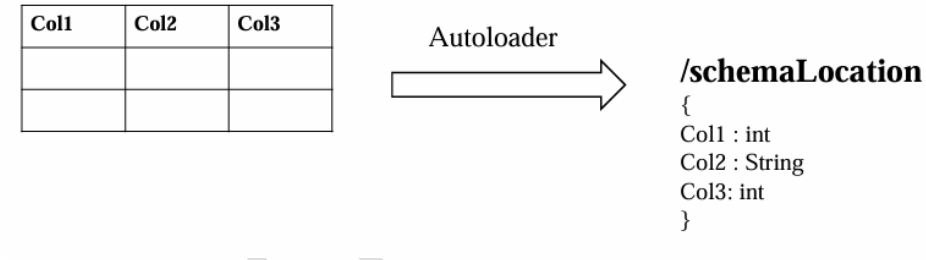
Schema evolution

- Schema evolution is the process of managing changes in data schema as it evolves over time, often due to updates in software or changing business requirements, which can cause schema drift

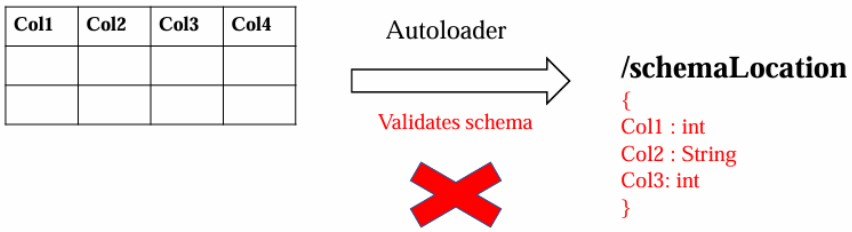
- Ways to handle schema changes
  - Fail the stream
  - Manually change the existing schema
  - Evolve automatically with change in schema

- **Now we have something called schema where we are trying to define the explicit schema. Now auto loader is smart enough to identify the schema of our source.**

Schema validation



Schema validation



- **So you can just feel free to remove this. And all you need to do is you just need to add something called schema location.**
- **So the schema location path is required because first when it is trying to read the file, it is going to understand the schema of this particular data frame.**
- **"schemaInfer": So auto loader will first try to read the 100 files or the first 50 GB files. And it is going to conclude that this is the schema which it is going to expect. Now that schema will be**

written to a path where for the further reading, it is going to refer to that particular schema location.

Schema Evolution: if you are having data ingestion with four columns today and tomorrow, due to some business requirements, there could be a new column to be introduced.

Schema Evolution

- **addNewColumns** = Stream fails. New columns are added to the schema. Existing columns do not evolve data types.
- **failOnNewColumns** = Stream fails. Stream does not restart unless the provided schema is updated, or the offending data file is removed
- **rescue** = Schema is never evolved and stream does not fail due to schema changes. All new columns are recorded in the rescued data column.
- **none** = ignore any new columns (Does not evolve the schema, new columns are ignored, and data is not rescued unless the rescuedDataColumn option is set. Stream does not fail due to schema changes.)

- **This will cause a change in the existing schema where we need to evolve our schema, which is called the schema evolution.**

Change Data Feed (CDF) feature in Databricks:

- **Generally speaking, we use CDF for sending incremental data changes to downstream tables in a multi-hop architecture. So, use CDF when only small fraction of records updated in each batch. Such updates are usually received from external sources in CDC format. If most of the records in the table are updated, or if the table is overwritten in each batch, like in the question, don't use CDF.**
- **Change Data Feed ,or CDF, is a new feature built into Delta Lake that allows it to automatically generate CDC feeds about Delta Lake tables.**
- **CDF records row-level changes for all the data written into a Delta table. This includes the row data along with metadata indicating whether the specified row was inserted, deleted, or updated.**
- **Databricks records change data for UPDATE, DELETE, and MERGE operations in the _change_data folder under the table directory. The files in the _change_data folder follow the retention policy of the table. Therefore, if you run the VACUUM command, change data feed data is also deleted.**

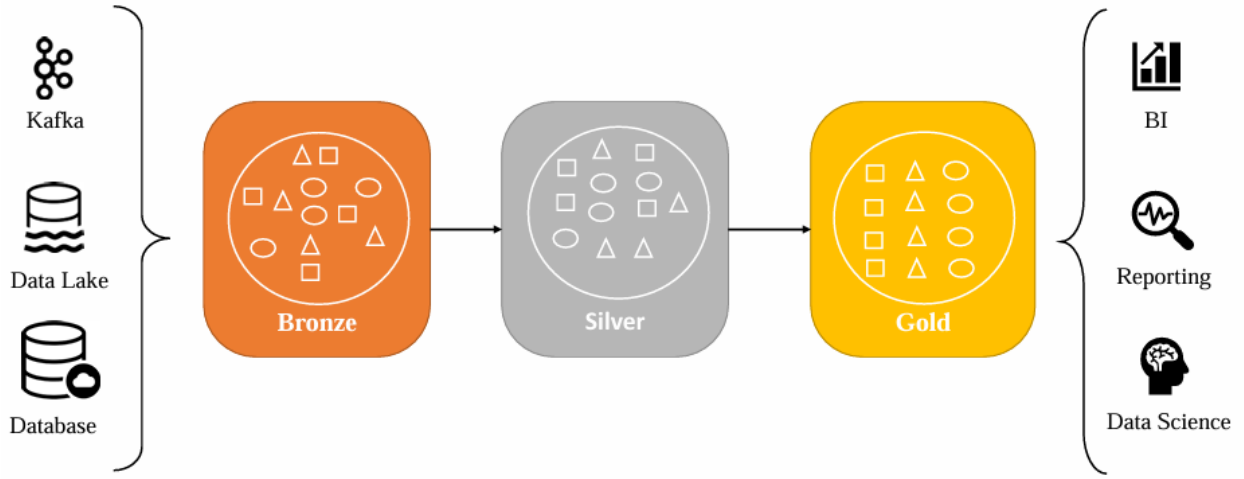
	
Delta changes include updates and/or deletes	Delta changes are append only
Small fraction of records updated in each batch	Most records in the table updated in each batch
Data received from external sources is in CDC format	Data received comprises destructive loads
Send data changes to downstream application	Find and ingest data outside of the Lakehouse

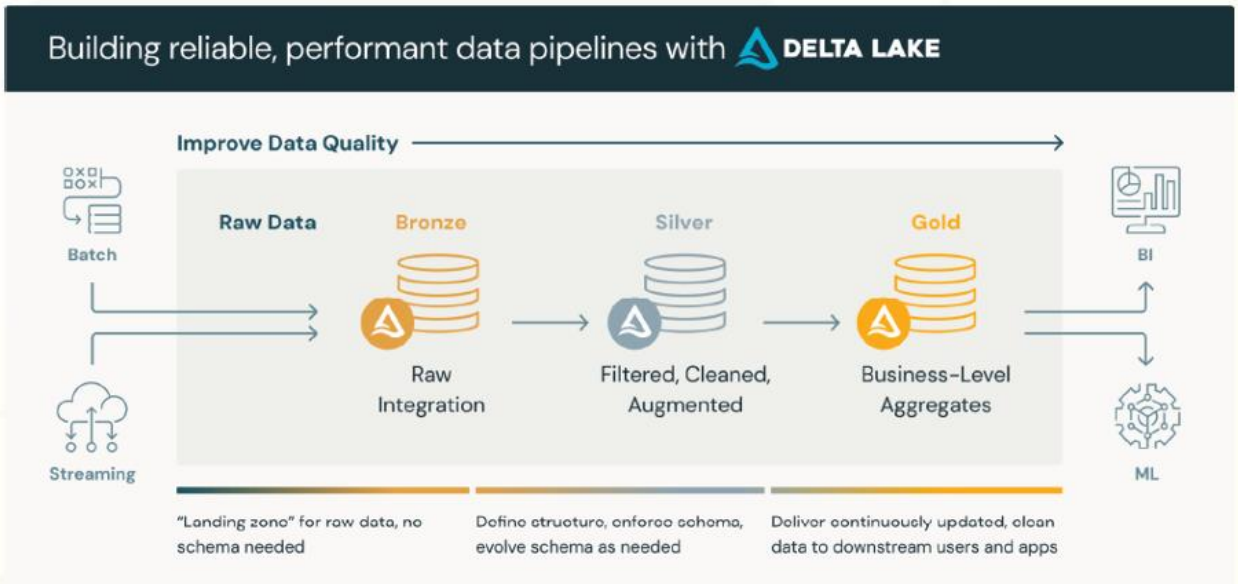
Day 12: Project overview: Creating all schemas dynamically

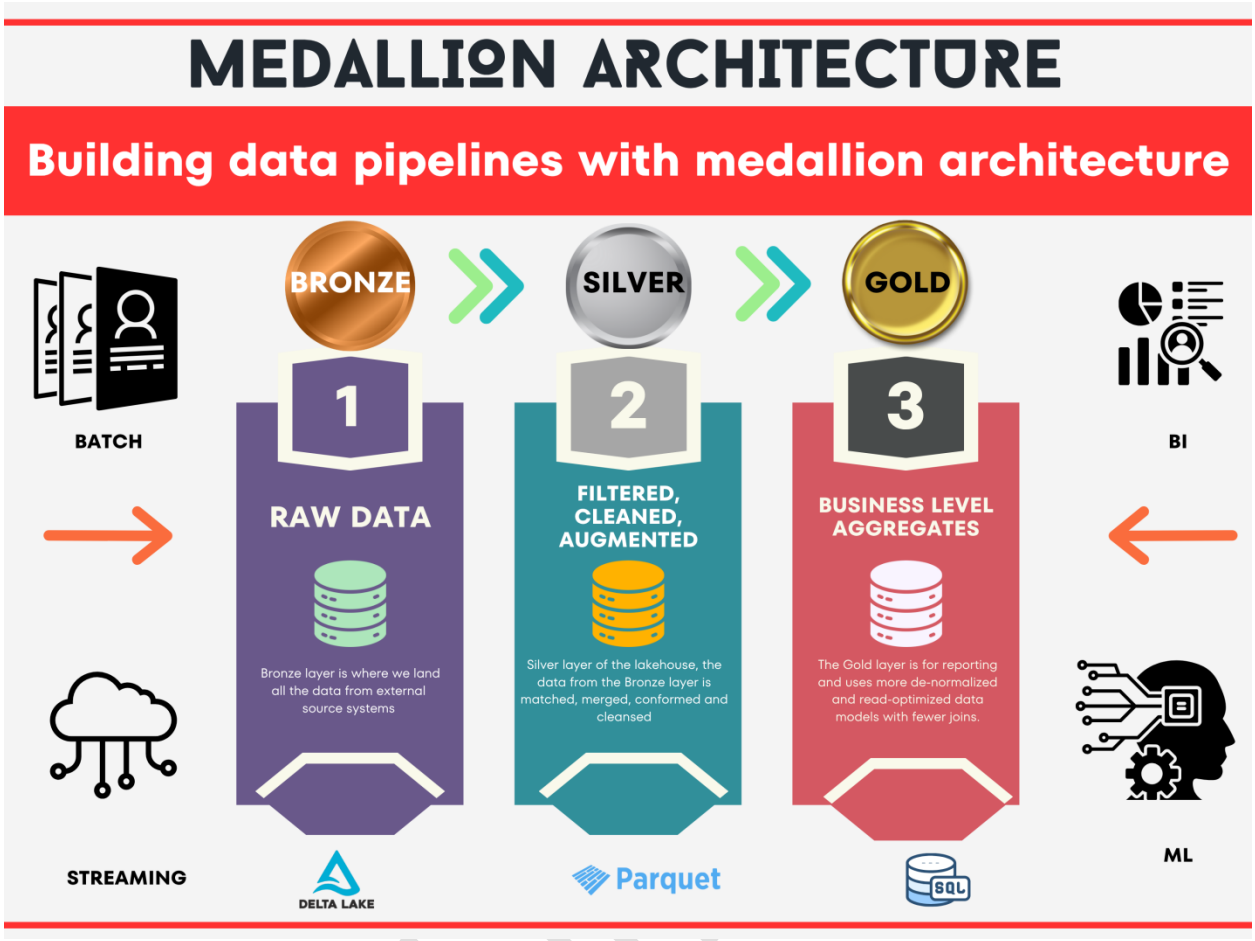
Medallion Architecture: Let us understand about the medallion architecture. So data engineering is all about quality.

- **Reference: https://www.databricks.com/glossary/medallion-architecture**

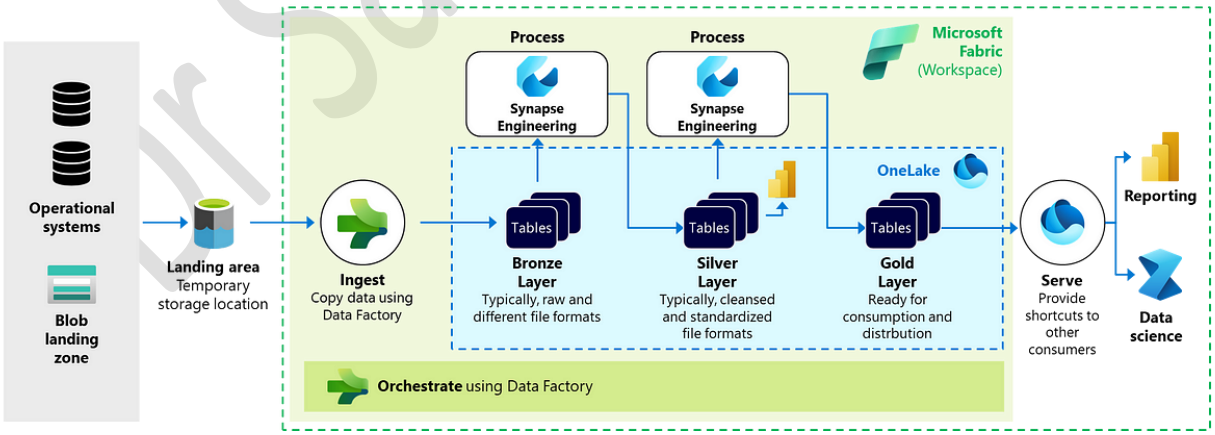
Medallion Architecture

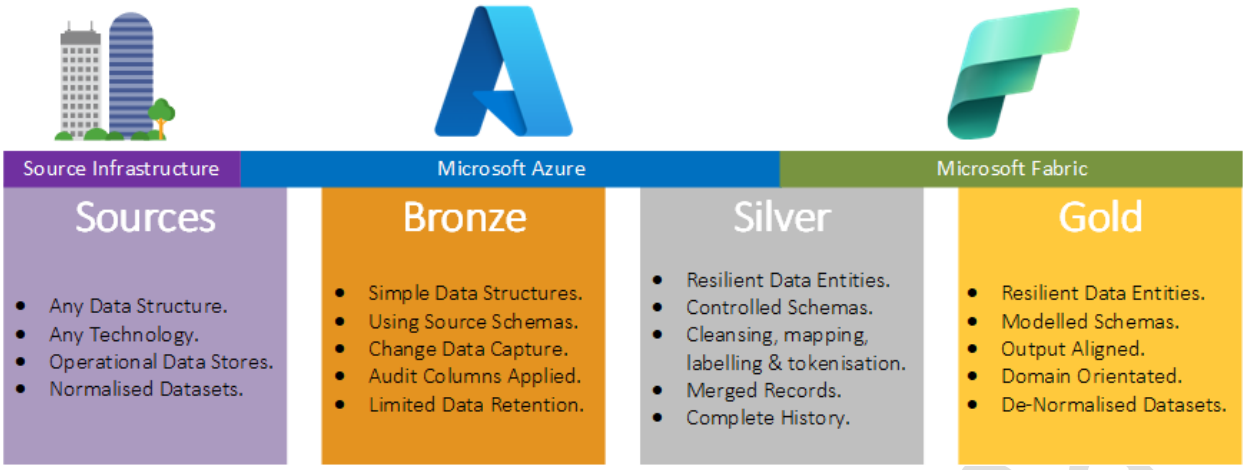


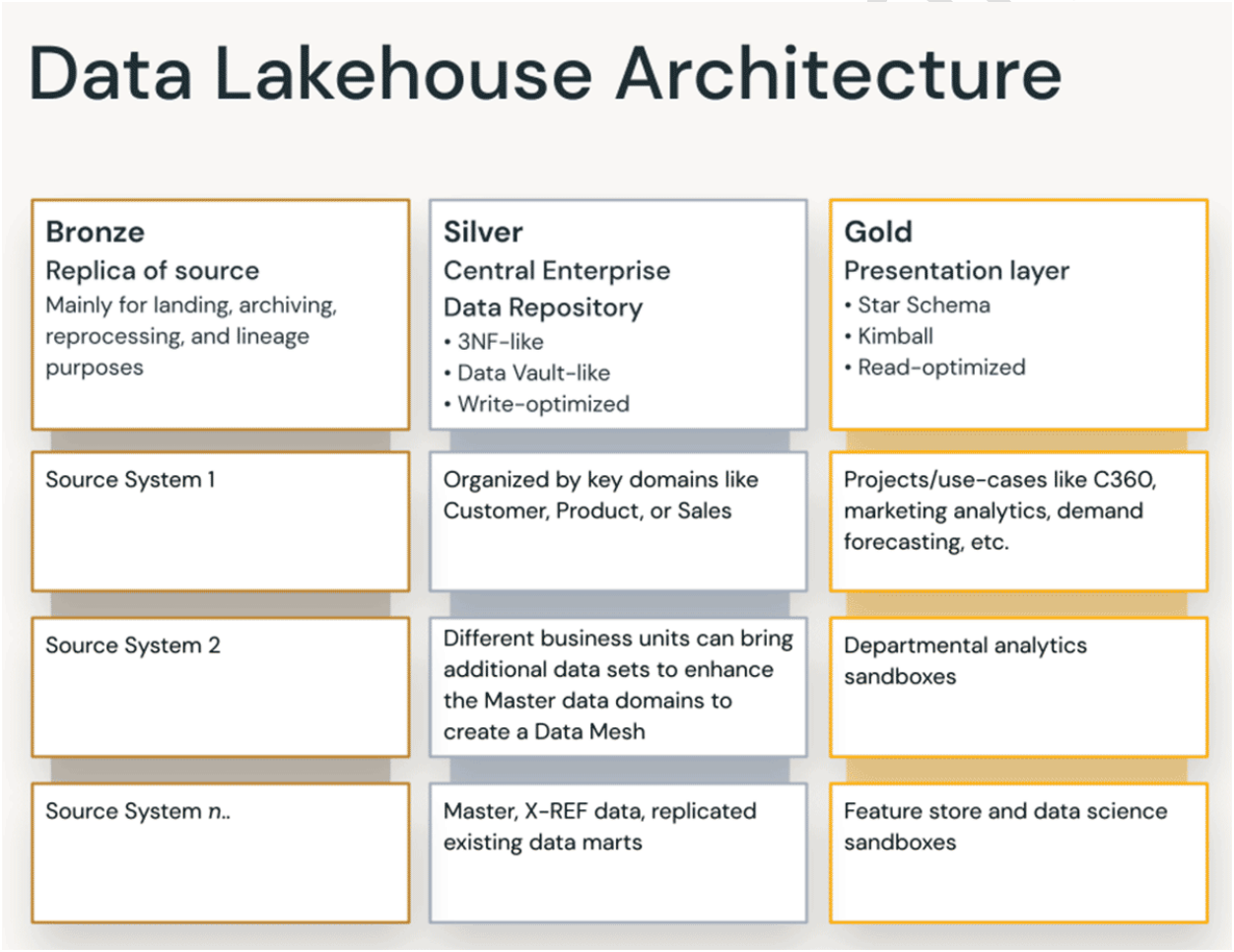




Reference architecture based on Lakehouse







- **The goal of the modern data engineering is to distill data with a quality that is fit for downstream analytics and AI. With the Lakehouse, the data quality is achieved on three different levels.**
- **On a technical level, data quality is guaranteed by enforcing and evolving schemas for data storage and ingestion.**

- On an architectural level, the data quality is often achieved by implementing the medallion architecture, where the data that is flowing on through each and every layer in the medallion architecture increases the quality.

- The Databricks Unity Catalog comes with the robust data quality management, with built in quality controls, testing, monitoring, and enforcement to ensure accurate and useful data is available.
- Now with all this, let's understand and implement the medallion architecture.
- This architecture is often referred as a multi-hop architecture. Medallion architecture is a data design pattern used to logically organize the data in a Lakehouse, with the goal of increasingly and progressively improving the structure and quality of the data as it flows through each layer of the architecture.

- Typically we will have three layers bronze, silver, and gold.
- So let's understand this architecture end to end, starting from the data sources. So the data sources can be Kafka streaming data lakes which we will generally use it.
- Or it can be databases where your data is generally getting ingested. All these will be where the data is coming from.
- You can use any of the ETL tool to get this data ingested from an external system. So this generally sits in a data lake folder which we will use in our project. So being specific with Azure we have an Azure Data Factory which is an ETL tool to ingest the data.
- There are 90 plus sources which are supported. And it is going to do some initial loading or the copying of the data to a certain folder, where we often call it as a landing zone.

- Bronze Layer: Now there comes some medallion architecture and that starts with the bronze layer.
- So this bronze layer typically also called as a raw layer. In this the data is first ingested into the system, as is usually in the bronze layer. The data will be loaded incrementally and this will grow in time. The ingested data into the bronze layer can be a combination of batch and streaming, although the data that is kept here is mostly raw. The data in the bronze layer should be stored in a columnar format, which can be a parquet or delta. The columnar storage is great because it stores the data in columns rather than rows. This can provide more option for compression, and it will allow for more efficient querying of the subset of data.
- So this is the primary zone where we will have the exact same data that we receive from our sources without having any modification.
- So this is going to serve as a single source of truth for the downstream transformations.

- Silver Layer: Now next comes would be the silver layer or the curator layer. So this is the layer where the data from the bronze is matched, merged and conformed, or just cleans just enough so that the silver layer can provide the enterprise view.
- So all the key business entities, concepts and transactions will be applied here. So basically we perform the required transformations to the data which can give some basic business value and a quality data where we apply our data quality rules to bring some trustworthiness to the data.
- Also, few transformations on top of like joining merging the data to bring some sense of it. By the end of this silver layer, we can have multiple tables which are generated in the process of transformation.
- And there comes the business level aggregation.

- Golden layer: And the next level would be having this data in a gold layer or a processed layer of the lake house. This is typically organized in a consumption ready project. Specific databases.
- The golden layer is often used for reporting and uses more denormalized and the read optimized data models with fewer joints. This is where the specific use cases and the business level aggregations are applied. So we mentioned the data will flow through this layer.
- And for each and every layer the quality will be increased.

- Coming to bronze the data can be raw and completely unorganized. Whereas for silver we are giving some structure by applying some business level transformations. And there can be a situation you can have completely transformed and ready available data in the silver. And sometimes gold is just for having the views where we have the exact data in silver, and in cases there are some times where the gold layer will have a minimal transformations where we will have the completely organized data.
- Now this organized data is ready for consumption. So data consumers are the one who use this data to drive the business decisions. It can be like by reporting in the data science. So this is on a typical the medallion architecture where this can be used in the projects like they want with different data sources and the data consumers.

- The basic idea is you will have the data flowing throughout these layers, where each layer will have more quality than the previous layer.
- And in our project, also, we will implement this architecture by making use of the data bricks.

Project Details: Hands-on

Step 1: Create Storage account and directories

Step 2: Create Azure Databricks Resource
Step 3: Access Connector for Azure Databricks
Step 4: Add role assignment in Access Control (IAM)
Step 5: Create Metastore from Databricks Notebook
Step 6: create a dev-catalog
Step 7: Create a Compute Resource
Step 8: Create External Storages
Step 9: Part 0 Medallion Architecture Project overview
Step 10: Part 1 Project set up Creating bronze Tables Dynamically
Step 11: Load to Bronze
Step 12: Silver Traffic Transformations
Step 13: To re-use common functions and variables
Step 14: Silver - Roads Transformation

① create storage acnt in sachinsax-gmail (Ext # sachinsax@gmail.onmicrosoft.com (master) admin) "databricksdevstg"

② create ~~three~~ Four containers "landing", "medallion" and "checkpoints". and "metastoretest".

③ Inside "landing" create two directories "raw_roads" & "raw_traffic".

④ Inside "mediallion" create three directories "bronze", "silver", "gold".

⑤⑦ Go to storage account "databricksdevstg" → "Access Control (IAM)" → "Add" → "Add role assignment" → search for "storge blob data contributor" → "Magaged identity" → "select members" → "Access Connectors for Azure Databricks(2)" → "access controllersachin" → "review + assign".

⑤ Create databricks notebook "data-bricks-dev-ws"

⑥ "Access Connectors for Azure Databricks" create a new with name "accesscontrollersachin"

⑧ Go to "Databricks portal" → "Manage account" from right top → "Catalog" → "Create metastore" → Name → meta-store-sachin (जो की metastore successfully connect होगा और External resource का option दिखाई देगा।)
Make sure you are providing different holder name.

⑨ Add a catelog "Dev-catlog" :- "Databricks portal"

"Catalog" → "Add a Catalog" → Dev Catalog
Catalog Name

→ Type "Standard"

⑩ Inside "Databricks" → "Catalog" → "Catalog Explorer" → "External Location"

⑪ Create 5 External locations :- in URL give path

Ⓐ like :- Landing :-

abfss://landing@databricksdustg.dfs.core.windows.net/

storage credinal :- select that big one (manage identity)

checkpoints :-

Ⓑ abfsss://checkpoints@dd.————————

bronze :-

Ⓒ dbfss://medallion@———— net/bronze

silver

Ⓓ "————————/silver

gold

abfss//medillion@———— /gold

(12) Create a schema → from Catalog tab

Schema Name [bronze]

storage location [bronze] (which we defined in external location)

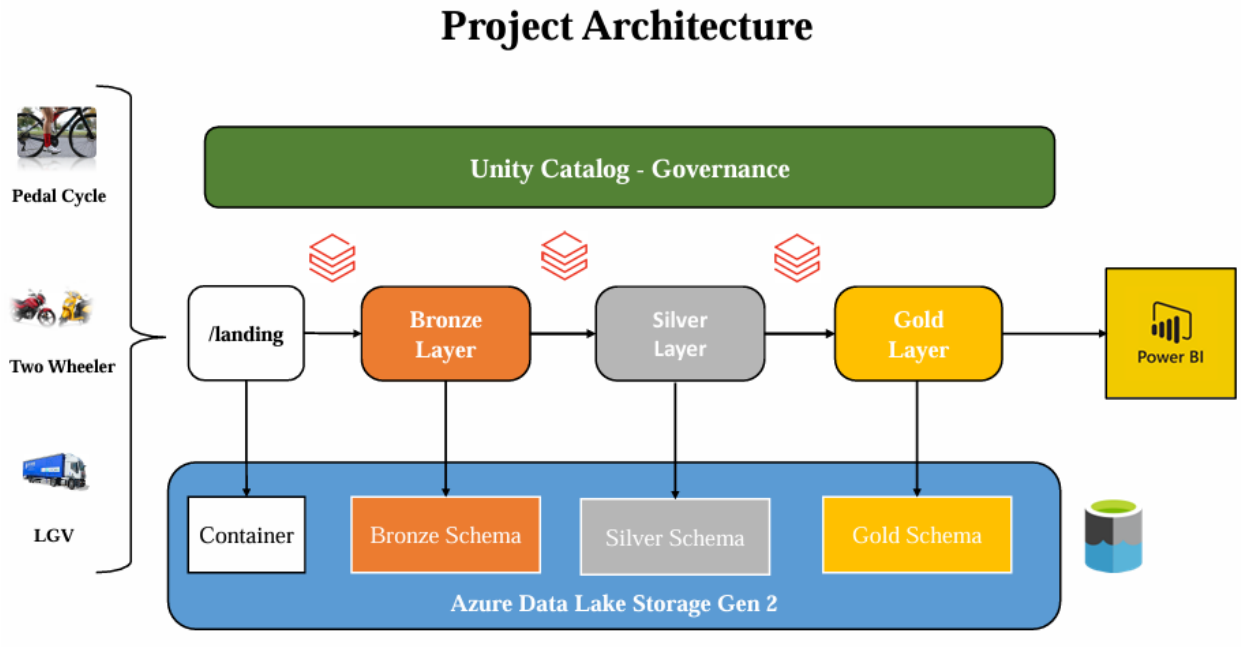
[bronze] → sub path give name here.

→ This is UI based method but we will use code.

(13) create a group and assign this users to this workspace.

(14) give permissions to them.

Project Architecture



Project Architecture Script (Main Points)

1. **Introduction**
   - Recap: Medallion architecture overview from the previous video.
   - Current video focus: Specific project implementation of the architecture.
2. **Data Sources**
   - Use **traffic and roads data** as input.
   - Data will be loaded into a **landing zone** (a container in the data lake).
3. **ETL and Data Ingestion**
   - Typical projects use ETL tools like **Azure Data Factory** for incremental data ingestion.
   - For this course: Manual data input into the landing zone to focus on **Databricks learning**.
   - Multiple approaches exist for ingestion pipelines (not the main focus here).
4. **Landing Zone**
   - Located in **data lake storage** under a specific container.
   - Data manually uploaded for simplicity.
5. **Bronze Layer**
   - Purpose: Store raw data from the **landing zone**.
   - Implementation:
     - Use **Azure Databricks notebooks** to ingest data incrementally.
     - Store data in tables under the **bronze schema** (backed by Azure Data Lake).
   - Transformations: Perform on **newly added records only**.
6. **Silver Layer**
   - Purpose: Perform transformations to refine data.
   - Implementation:
     - Create **silver tables** stored under the **silver schema** in Azure Data Lake.
     - Apply detailed transformations on bronze layer data.
7. **Gold Layer**
   - Purpose: Provide **clean and minimal-transformed data**.
   - Implementation:
     - Create **gold tables** under the **gold schema** in Azure Data Lake.
8. **Data Consumption**

- Final output used by:
  - **Analytics teams**, **data scientists**, and others.
- Data visualization: Import into **Power BI** for insights.

9. **Governance**
   - Govern and back up the entire pipeline with **Unity Catalog**.
10. **Conclusion**
    - Recap of the end-to-end implementation and project focus on Databricks.

Raw Traffic counts dataset

Pedal Cycle

Two Wheeler motor vehicles

Buses and coaches

LGV (Large Goods Vehicle)

HGV (Heavy Goods Vehicle)

Electric Vehicles

Data Dictionary

1. Record ID 2. Count point id 3. Direction of travel 4. Year 5. Count date	Vehicle flow point
6. hour 7. Region id 8. Region name 9. Local authority name 10. Road name 11. Road Category ID 12. Start junction road name 13. End junction road name 14. Latitude 15. Longitude 16. Link length km	Travel info of vehicle
17. Pedal cycles 18. Two wheeled motor vehicles 19. Cars and taxis 20. Buses and coaches 21. LGV Type 22. HGV Type 23. EV Car 24. EV Bike	Count of types of vehicle

Data Dictionary

1. Record ID	=	Uniquely identifies a record
2. Count point id	=	A unique reference for the road link
3. Direction of travel	=	Direction of travel
4. Year	=	Year it happened
5. Count date	=	The date when the actual count took place
6. hour	=	Hour 7 represents from 7am to 8am, and 17 tells from 5pm to 6pm.
7. Region id	=	Website region identifier
8. Region name	=	The name of the Region that travel took place
9. Local authority name	=	Local authority that region
10. Road name	=	This is the road name (for instance M25 or A3).
11. Road Category ID	=	Uniquely identifies road ID
12. Start junction road name	=	The road name of the start junction of the link
13. End junction road name	=	The road name of the end junction of the link
14. Latitude	=	Latitude of the Location
15. Longitude	=	Longitude of the Location
16. Link length km	=	Total length of the network road link
17. Pedal cycles	=	Counts for pedal cycles
18. Two wheeled motor vehicles	=	Counts of Two wheeled motor vehicles
19. Cars and taxis	=	Counts of Cars and taxis
20. Buses and coaches	=	Counts of Buses and coaches
21. LGV Type	=	Counts of LGV Type
22. HGV Type	=	Counts of HGV Type
23. EV Car	=	Counts of EV Car
24. EV Bike	=	Counts of EV Bike

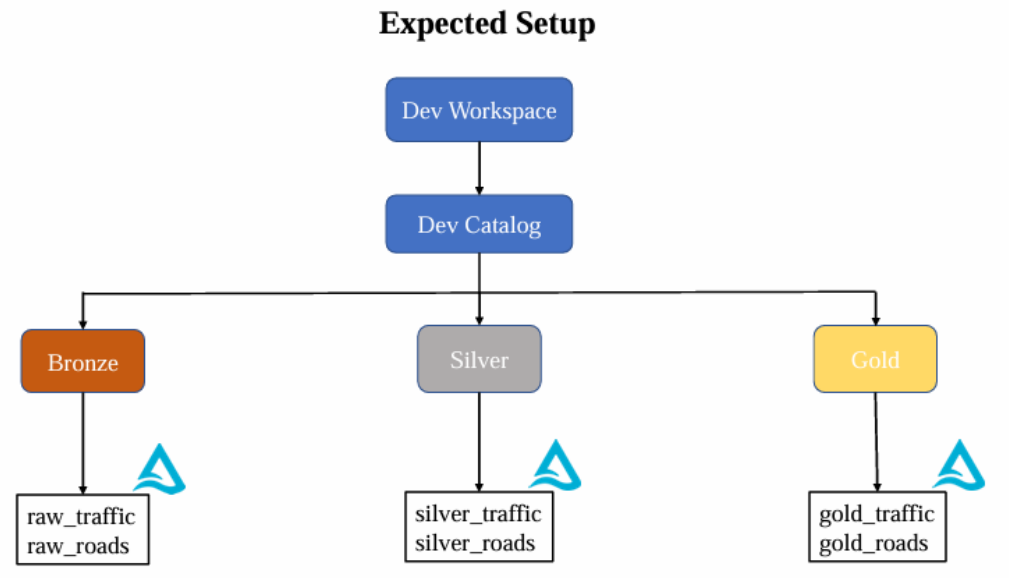
Raw Roads dataset

Road Category

Road Types

Dr Sachin Saxena
Follow on LinkedIn

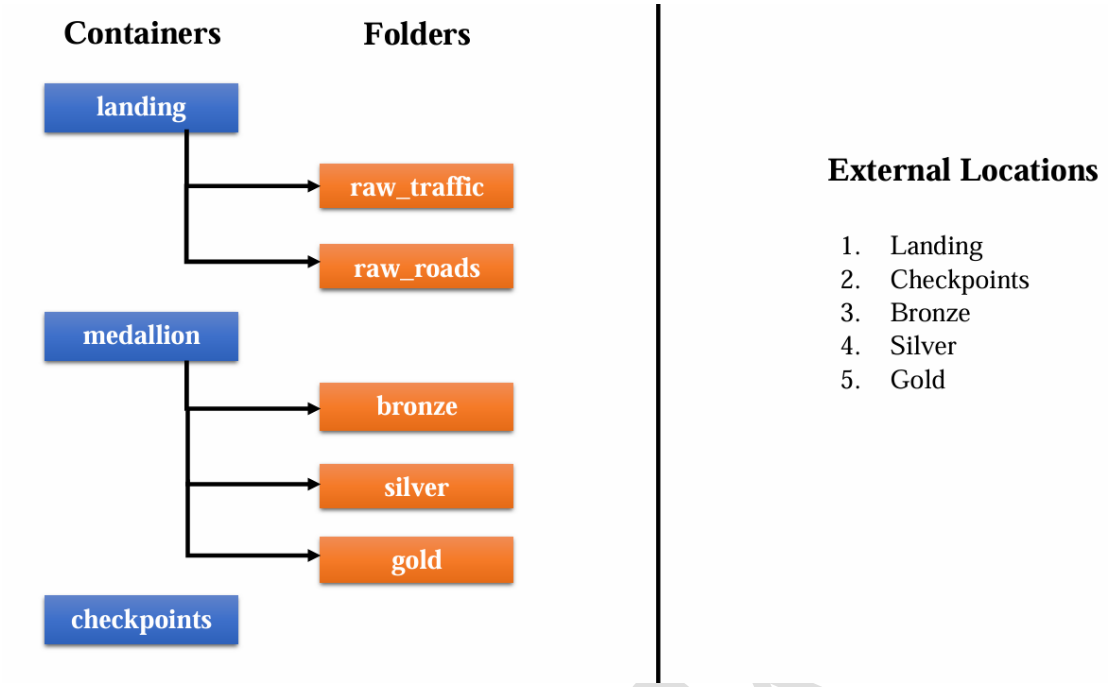
- **Expected Setup pre-requisitive:** **We need to set up a multi-hop architecture setup. In lack house architecture, we have Bronze, Silver and Golden Layers.**



- **And we are going to create the two tables which are raw traffic and the raw roads. But let us now see the complete setup so that you can get an idea on what are the tables we are going to create and what format they should be in.**
- **So once these tables are created and once these are having the data, like taking the data from the landing to bronze, so the raw traffic and the raw Rhodes will have the data and we will perform the required transformations on them.**

⇨

- **Hands-on Activity in Azure portal: Create three containers, first container will be "Landing container", which will consist both "traffic" and "roads" datasets.**



- Hands-on Activity in Azure portal: Create three containers, “landing”, “medallion”, and ”checkpoints”.
- In “landing” containers, create two directories: “raw_roads” and “raw_traffic”.
- Now in “medallion” conatiner, create three directories: “Bronze“,“Silver“ and “Golden“.
- Hands-on Activity in Databricks portal (which is created through super Admin): inside “Catalog Explorer” click on “External Locations” -> check “Storage Credentials” (you need to create the storage credentials here, because we already have the Databricks Access connector that is having the required role in this particular storage account. So the credentials are already stored by the storage credential. Now you just need to create the external locations.).

Create Five “External locations”:

- Create External Locations through “Create Location” button: first “External location name” is “landing”, select “Storage Credential” from default (which is having very big name), provide the link accordingly. “Test Connection”, it must be green for all checks.
- Similarly, Create second External Locations through “Create Location” button: “External location name” is “checkpoints”, select “Storage Credential” from default (which is having very big name), provide the link accordingly. “Test Connection”, it must be green for all checks.
- Similarly, Create third External Locations through “Create Location” button: “External location name” is “bronze”, select “Storage Credential” from default (which is having very big name), provide the link accordingly. “Test Connection”, it must be green for all checks. Give extra path at the end of “URL” “/bronze”, bcs we have extra directory there.
- Similarly, Create forth External Locations through “Create Location” button: “External location name” is “silver”, select “Storage Credential” from default (which is having very big name),

provide the link accordingly. “Test Connection”, it must be green for all checks. Give extra path at the end of “URL” “/silver”, bcs we have extra directory there.

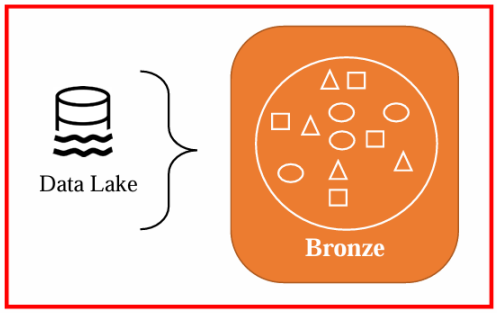
- Similarly, Create fifth External Locations through “Create Location” button: “External location name” is “gold”, select “Storage Credential” from default (which is having very big name), provide the link accordingly. “Test Connection”, it must be green for all checks. Give extra path at the end of “URL” “/gold”, bcs we have extra directory there.
-
-

Day 13: Ingestion to Bronze: raw_roads data to bronze Table

Reference: To be published

Details: to be added

Ingesting data to Bronze Layer



Schema: bronze

Tables:

1. raw_traffic
2. raw_roads

Day 14: Silver Layer Transformations: Transforming Silver Traffic data

Reference:

Renaming Columns

1. Record ID	Record_ID
2. Count point id	Count_point_id
3. Direction of travel	Direction_of_travel
4. Year	Year
5. Count date	Count_date
6. hour	hour
7. Region id	Region_id
8. Region name	Region_name
9. Local authority name	Local_authority_name
10. Road name	Road_name
11. Road Category ID	Road_Category_ID

Details:

Creating Electric_Vehicles_Count

1. Record_ID	1. Record_ID
2. Count_point_id	2. Count_point_id
3. Direction_of_travel	3. Direction_of_travel
4. Year	4. Year
5. Count_date	5. Count_date
6. hour	6. hour
7. Region_id	7. Region_id
.	.
.	.
24. EV_Bike	24. EV_Bike
	25. Electric_Vehicles_Count

Creating Motor_Vehicles_Count

1. Record_ID
2. Count_point_id
3. Direction_of_travel
4. Year
5. Count_date
6. hour
7. Region_id
.
.
25. Electric_Vehicles_Count

1. Record_ID
2. Count_point_id
3. Direction_of_travel
4. Year
5. Count_date
6. hour
7. Region_id
.
.
25. Electric_Vehicles_Count
26. Motor_Vehicles_Count

Two_wheeled_motor_vehicle + Cars_and_taxis + Buses_and_coaches + LGV_Type + HGV_Type + Electric_Vehicle_Count

Transforming Raw Roads dataset:

Renaming Columns

1. Road ID
2. Road category id
3. Road category
4. Region id
5. Region name
6. Total link length km
7. Total link length miles
8. All motor vehicles

1. Record_ID
2. Road_category_id
3. Road_category
4. Region_id
5. Region_name
6. Total_link_length_km
7. Total_link_length_miles
8. All_motor_vehicles

Creating Road_Category_Name

1. Record_ID
2. Road_category_id
3. Road_category
4. Region_id
5. Region_name
6. Total_link_length_km
7. Total_link_length_miles
8. All_motor_vehicles

1. Record_ID
2. Road_category_id
3. **Road_category**
4. Region_id
5. Region_name
6. Total_link_length_km
7. Total_link_length_miles
8. All_motor_vehicles
9. Road_Category_Name

When **Road_Category** = TA THEN Class A Trunk Road
When **Road_Category** = TM THEN Class A Trunk Motor
When **Road_Category** = PA THEN Class A Principal road
When **Road_Category** = PM THEN Class A Principal Motorway
When **Road_Category** = M THEN Class B road

Creating Road_Type

1. Record_ID
2. Road_category_id
3. Road_category
4. Region_id
5. Region_name
6. Total_link_length_km
7. Total_link_length_miles
8. All_motor_vehicles
9. Road_Category_Name

1. Record_ID
2. Road_category_id
3. Road_category
4. Region_id
5. Region_name
6. Total_link_length_km
7. Total_link_length_miles
8. All_motor_vehicles
9. **Road_Category_Name**
10. Road_Type

WHEN **Road_Category_Name** Contains Class A THEN Major
WHEN **Road_Category_Name** Contains Class B THEN Minor

Transforming & Loading Silver datasets:

Creating Vehicle_Intensity

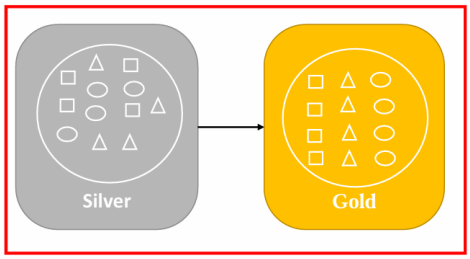
1. Record_ID
2. Count_point_id
3. Direction_of_travel
4. Year
5. Count_date
6. hour
7. Region_id
.
.
26. Motor_Vehicles_Count

1. Record_ID
2. Count_point_id
3. Direction_of_travel
4. Year
5. Count_date
6. hour
7. Region_id
.
.
26. Motor_Vehicles_Count
27. Vehicle_Intensity

Vehicle Intensity = Motor_Vehicles_Count / Link_length_km

Day 15: Golden Layer: Getting data to Gold Layer

Loading data to Gold Layer



Schema: Gold

Tables:

1. gold_traffic
2. gold_roads

Reference:

Details: to be added

Orchestrating with WorkFlows: Adding run for common notebook in all notebooks

- Data workloads will utilize a Bronze table as its source: A job that enriches data by parsing its timestamps into a human-readable format.
- Data workloads will utilize a Silver table as its source: A job that aggregates cleaned data to create standard summary statistics.

Simplify Your Data Pipeline with Databricks Workflows:

Imagine managing complex data workflows, scheduling tasks, and integrating seamlessly across tools—all in one platform. That's Databricks Workflows for you! But when should you use it, and how? Let's break it down.

🛠️**When to Use Databricks Workflows:**
- Automating ETL Pipelines: For transforming large datasets across stages with reliability.
- Orchestrating Machine Learning Models: To train, validate, and deploy models at scale.
- Managing Cross-Team Collaboration: Ideal for teams managing shared resources and dependencies.

✳️ **Key Use Cases & Examples:**

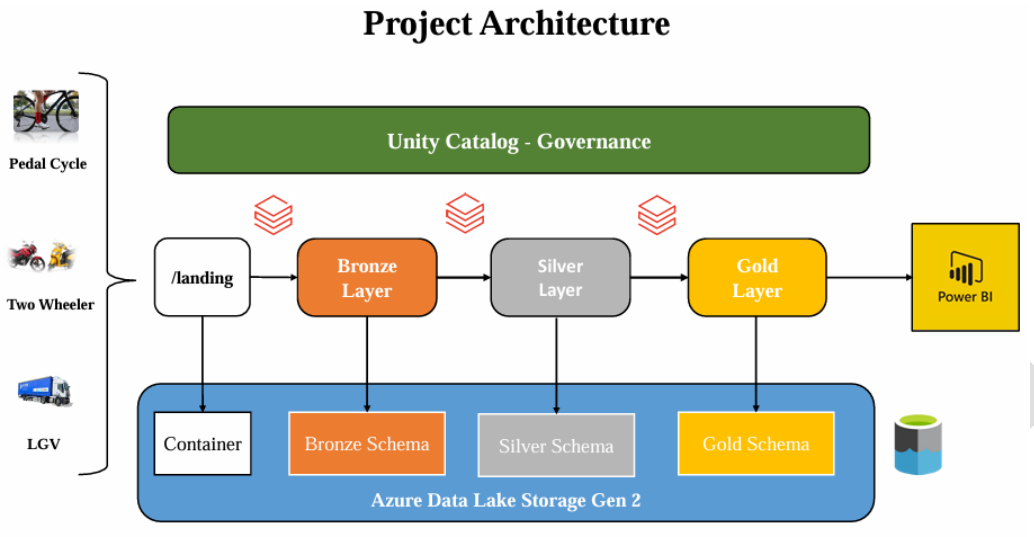
1. Data Ingestion Pipelines
Use case: Automate ingestion of streaming or batch data from multiple sources like Azure Event Hub.
Example: A retail company importing transactional data for real-time analytics.

2. ML Workflow Automation
Use case: Training a customer churn prediction model on a nightly basis.
Example: Trigger preprocessing, training, and model evaluation tasks in sequence.

3. Cost Optimization
Use case: Dynamically manage jobs based on demand.
Example: Scale compute resources for ad-hoc analytics during peak hours.

4. End-to-End Data Ops
Use case: Schedule workflows that combine SQL queries, Python scripts, and Spark jobs.
Example: Finance team running daily reconciliation and compliance checks.
🎯**Why It's a Game-Changer**
-Simplified Orchestration: No need for separate schedulers. Manage everything in Databricks.
- Built-in Monitoring: Get real-time visibility into job runs and logs.
- Flexible Triggers: Schedule, event-based, or manual. You choose.

Day 16: Reporting with PowerBI



Step 1: Login with sachinsax_gmail.com#EXT#@sachinsaxgmail.onmicrosoft.com who is having "Global Administrator" role.

Step 2: Create ADLS Gen 2 instance: Inside ADLS Gen 2, create a ADLS Gen 2 with name "deltadbstg", create a container with name "test", inside this container add a directory with name "sample", upload a csv file name "countires1.csv".

Step 3: Databricks instances and Compute resource: Inside Databricks instances: Create a compute resource with Policy: "Unrestricted", "Single node", uncheck "Use Photon Acceleration", select least node type.

Step 4: "Access connectors for Azure Databricks": search for "Access connectors for Azure Databricks", create "New", only give resource group name and Instance name "access-connectors-sachin" here, you need not to change anything here. Click on "Go to Resource". Now in "Overview", "Resource ID", can use this "Resource ID" while creating the Metastore.

Step 5: Now give access of this Access connectors to ADLS Gen2, go to ADLS Gen2, go to "Access Control IAM" from left pane, click on "Add"-> "Add Role Assignment"-> search for "Storage Blob Data Contributor", in "Members", select "Assign Access to"->"Managed Identity" radio button, "+Select Members"-> select "Access connectors for Azure Databricks" under "Managed identity" drop down menu-> "Select"-> "access-connectors-sachin" -> "Review+Assign".

Step 6: Create a catalog with name 'test-catalog': Go to "Catalog" tab, "Catalog Explorer" -> click on "Create Catalog" from right, name "Catalog name" as "test-catalog", type as "Standard", skip "Storage location". Click on "Create". Create a catalog with name 'test-catalog' inside Databricks instances.

Step 7: It's time to give permission, in Databricks portal, click on "Manage Account", from right top, this Databricks portal is created neither by Workspace admin nor developer, in order to give permission, click on "Workspaces", click on respective "Workspace" -> inside it "permissions"-> "Add Permissions"-> we need to add groups which we created in Step 8, to admin group assign "Permission" as "Admin" and to developer group assign "Permission" as "User".

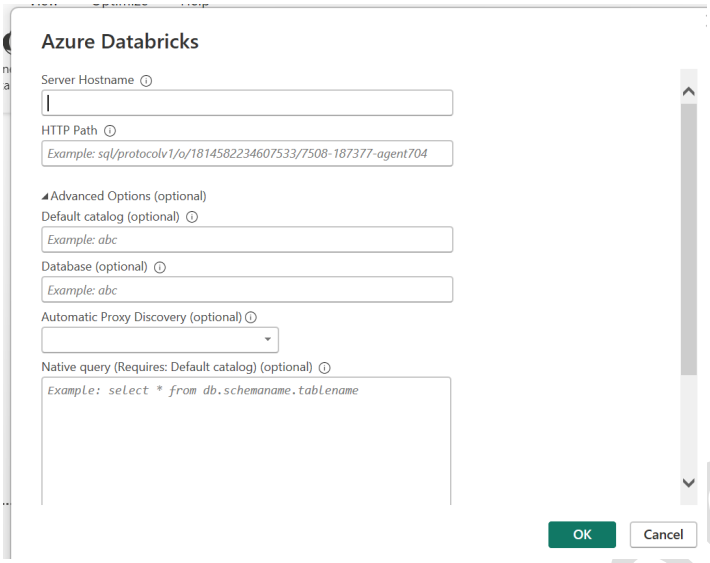
Step 8: Grant Permission: Click on "test-catalog", then "Permissions", then "Grant", this screen is Unity catalog UI to grant privileges to "Sachin Admin", then click on "Grant", select group name "WorkSpace admins" checkbox on "create table", "USE SCHEMA", "Use Catalog" and "Select" in "Privileges presets", do not check anything here. Click on "Grant". Now, go to "Sachin Admin" databricks portal, "test-catalog" is showing here.

Step 9: Enable and creating Metastore: Now go to Databricks, we need to start a creating a meta store, meta store is top level container in the unity catalog, go "Manage Account" under "Sachindatabricks name" from right top -> "Catalog" from left pane, "create meta store", provide "Name" as "metastore-sachin","Region"(can create one meta store in single region), "ADLS Gen2 path" (go to ADLS Gen2-> create container-> "Add Directory", paste <container_name>@<storage_account_name>.dfs.core.windows.net/<directory_Name> In the sample format of test@ deltadbstg.dfs.core.windows.net/files

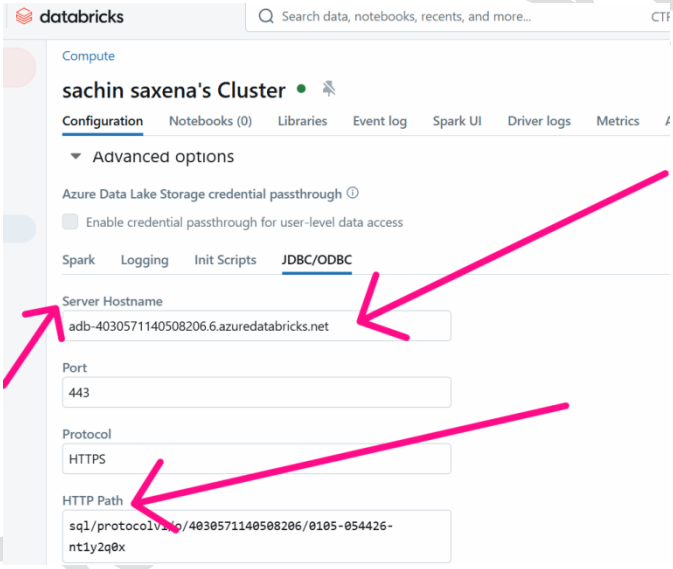
- **Or test@deltadbstg123456.dfs.core.windows.net/files), "Access connector ID" (go to "Access connectors"-> "access-connectors-sachin" -> copy that "resource ID")-> "Create".**
- **Attach with any workspaces. "Enable Unity Catalog?"-> "Enable".**

Step 10: Run "Day 13 Databricks to PowerBI.ipynb" file to create tables using Pyspark code.

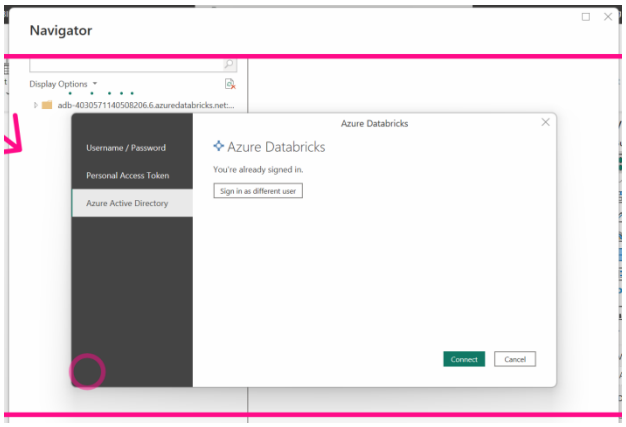
Step 11: Connect to PowerBI, open Power BI dashboard, open "Get Data", search for "Azure Databricks",

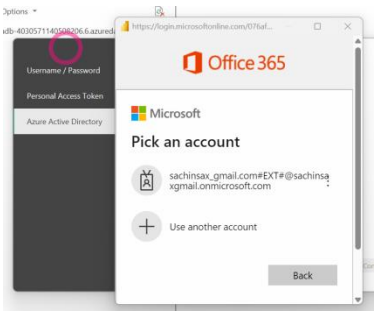


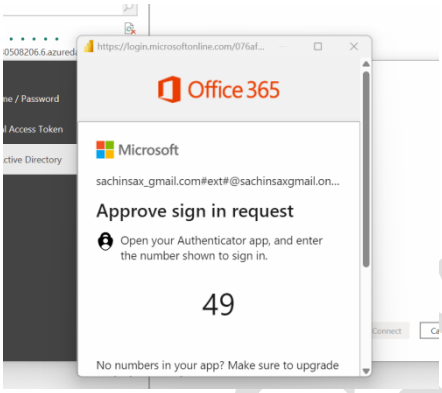
Server Hostname and HTTP path is given in Databricks's compute resource under JDBC/ODBC tab,

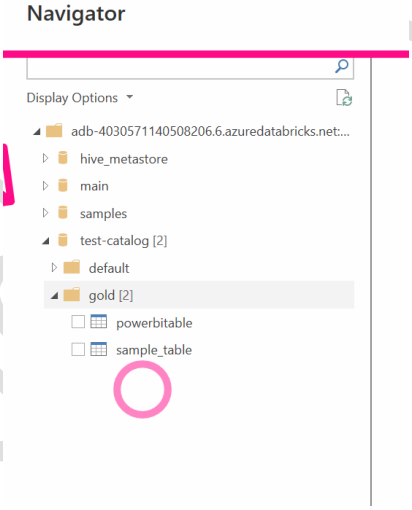


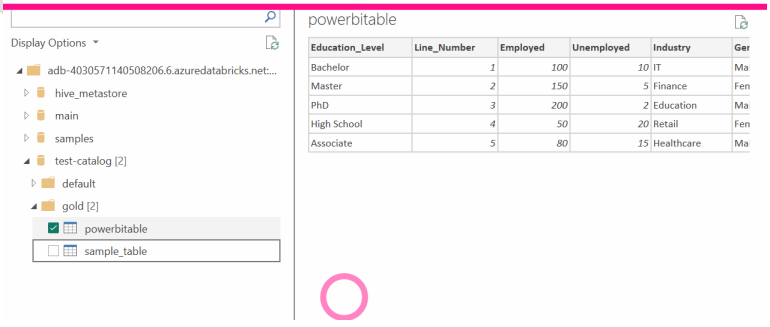
Now, connect using "Azure Active ID", sign-in with your credentials.



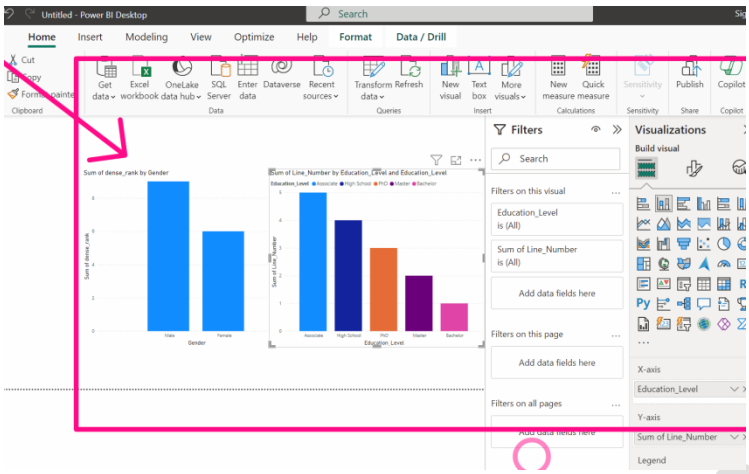








Education_Level	Line_Number	Employed	Unemployed	Industry	Ge
Bachelor	1	100	10	IT	Ma
Master	2	150	5	Finance	Fen
PhD	3	200	2	Education	Ma
High School	4	50	20	Retail	Fen
Associate	5	80	15	Healthcare	Ma



Question: does it necessary to connect every unity catalog with any ADLS gen2?

Answer: Yes, it is necessary to connect Unity Catalog with Azure Data Lake Storage Gen2 (ADLS Gen2) when using Azure Databricks. Unity Catalog requires ADLS Gen2 as the storage service for data processed in Azure Databricks. This setup allows you to leverage the fine-grained access control and governance features provided by Unity Catalog.

Here are the key references: How does Unity Catalog use cloud storage?

https://learn.microsoft.com/en-us/azure/databricks/connect/unity-catalog/

Therefore, to use Unity Catalog effectively with Azure Databricks, you must connect it to ADLS Gen2.

Details:

Day 17: Delta Live Tables: End to end DLT Pipeline

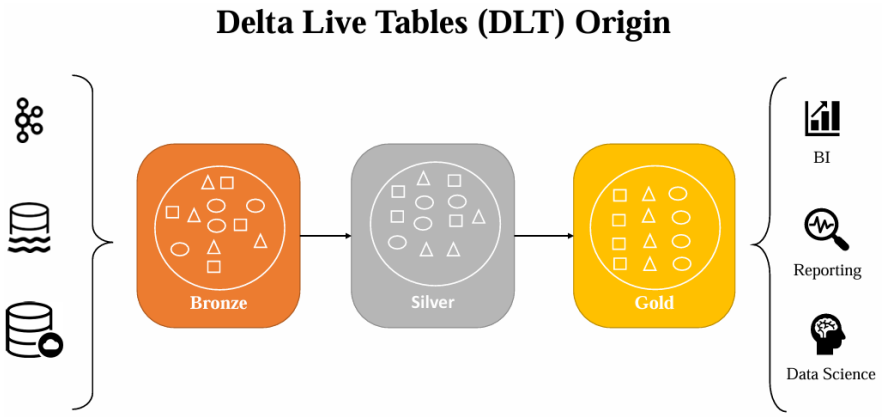
Reference: Unlock the Power of Delta Live Tables (DLT) in Databricks!

Delta Live Tables (DLT) revolutionizes the way you build and manage data pipelines. With a declarative approach, you can define transformations in simple SQL or Python while ensuring data quality and automating operations seamlessly.

What's inside this guide?

✔ Overview of the Databricks Lakehouse platform
✔ Key differences between Data Warehouses and Data Lakes
✔ Building ETL pipelines with DLT
✔ Automating data ingestion using Auto Loader
✔ Implementing Change Data Capture (CDC) with DLT

This comprehensive guide is perfect for anyone looking to streamline their data engineering workflows using Databricks and DLT.



Step 1: Create Databrick Workspace from Azure Portal.

Step 2: Create ADLS Gen 2 Storage-> container with name 'sachinstorage' -> Directory with name 'landing' -> two directory with names 'raw_traffic' and 'raw_roads', upload all 6 csv files in both these directories.

Step 3: search for "Access connectors for Azure Databricks", create "New", only give resource group name and Instance name "access-connectors-sachin" here, you need not to change anything here. Click on "Go to Resource". Now in "Overview", "Resource ID", can use this "Resource ID" while creating the Metastore.

Step 4: Now give access of this Access connectors to ADLS Gen2, go to ADLS Gen2, go to "Access Control IAM" from left pane, click on "Add"-> "Add Role Assignment"-> search for "Storage Blob Data Contributor", in "Members", select "Assign Access to"->"Managed Identity" radio button, "+Select Members"-> select "Access connectors for Azure Databricks" under "Managed identity" drop down menu-> "Select"-> "access-connectors-sachin" -> "Review+Assign".

Step 5: No need of any Unity catalog or External Storage but we require 'Metastore', make path to your metastore as 'landing@ sachinstorage.dfs.core.windows.net' bcs ur data is at same path

Step 6: Create Delta Live table and keep change all the setting, however also switch to JSON view and insert code:

- **"label": "default", //in between this line**
- **"node_type_id": "Standard_DS3_v2", // Insert this code**
- **"num_workers": //in between this line**

Step 7: Insert code file 'Day 14 DLT_Databricks.ipynb' but don't run it.

Step 8: Create 'dev-catlog' from 'catalog' pane also grant relevant permissions to this catalog.

Step 9: First run first two cells and then run DLT Pipeline to initiate first create two tables in dev-catlog and then DLT Pipeline will automatically create relationship or dependencies, Secondly run Third and Forth cells and then run DLT Pipeline to initiate relationship, Thirdly fifth cell in third run of DLT pipeline.

Step 10: Also create 'Job' and instead of 'Notebook', select existing 'Delta Live Table'.

Question: What is DLT in Databricks and How Can It Simplify Your Data Pipelines?

Managing data can be tricky, but Delta Live Tables (DLT) in Databricks is here to help. Let's break down what it is and how it can make your life easier.

What is DLT?

DLT (Delta Live Tables) is a tool in Databricks that helps you easily build and manage data pipelines.

It automates data tasks, ensuring your data is always clean, up-to-date, and ready for analysis.

Whether you're working with large data sets or just a few tables, DLT simplifies the process.

How does DLT work?

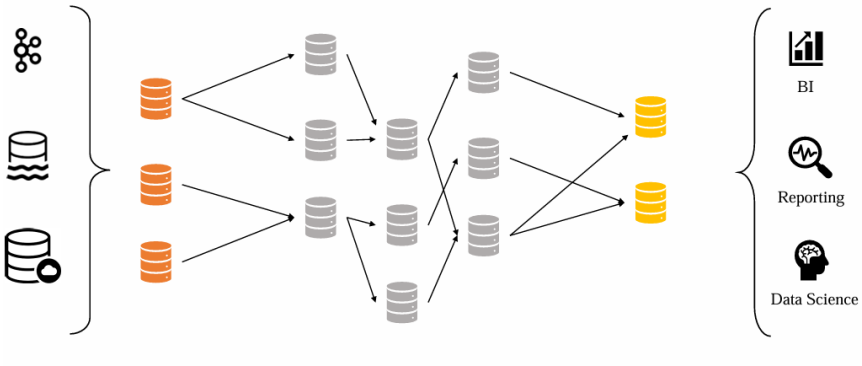
Simple Setup: Use basic SQL or Python to define your data pipelines. No complex coding required.

Automated Data Management: DLT takes care of cleaning, organizing, and updating your data without you having to lift a finger.

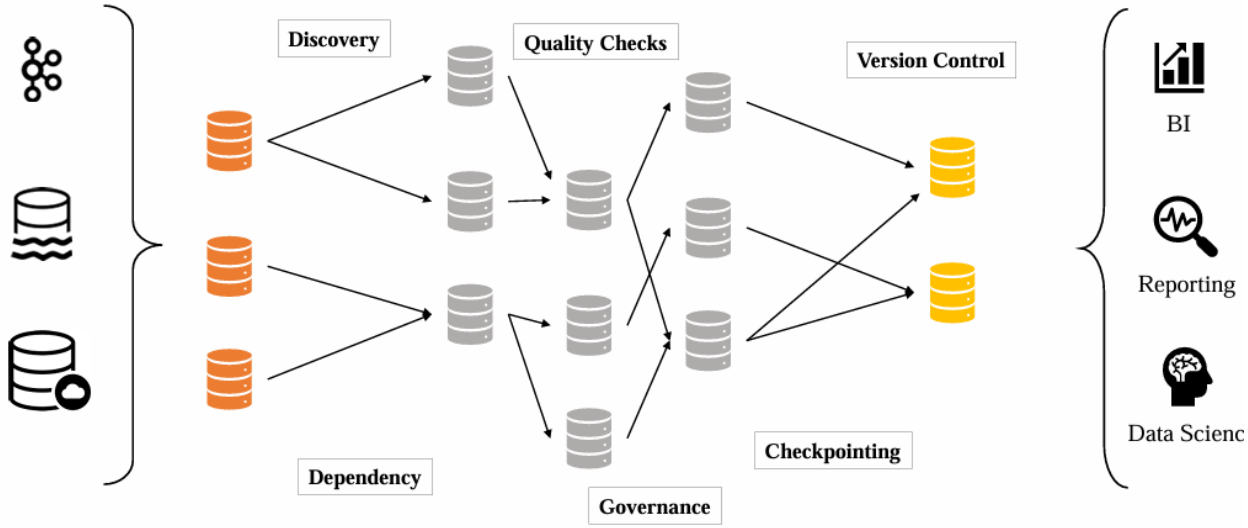
Built-in Data Checks: It ensures your data meets quality standards by running checks automatically.

Data Versioning: Easily track changes in your data and see how it has evolved over time.

Medallion/Lakehouse Architecture Tables



Considerations in Lakehouse Architecture



Declarative programming

Declarative programming say **what should be done ,** not **how to do it**

Procedural programming

```
Numbers = [..]

Sum = 0

For n in numbers:
        sum = sum + n

Print (n)
```

Declarative programming

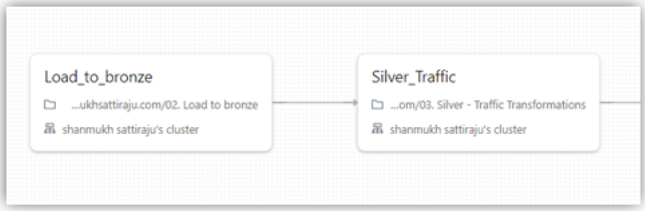
```
SELECT SUM(n)
FROM numbers
```

Declarative ETL with DLT

Declarative programming say **what should be done ,** not **how to do it**

Procedural ETL

- Apache Airflow
- Azure Data Factory



Declarative ETL

Delta live tables

Delta Live Tables (DLT)

Delta Live Tables (DLT) is **a declarative ETL framework** for the Databricks Data Intelligence Platform that helps data teams simplify streaming and batch ETL cost-effectively.

Simply define the transformations to perform on your data and let DLT pipelines automatically manage task orchestration, cluster management, monitoring, data quality and error handling.

Delta Live Table Execution

- Requires premium workspace
- Supports only Python and SQL languages
- Can't run interactively
- No support for magic commands like %run

✓ In DLT pipelines, we use the CREATE LIVE TABLE syntax to create a table with SQL. To query another live table, prepend the LIVE. keyword to the table name.

```
CREATE LIVE TABLE aggregated_sales

AS

SELECT store_id, sum(total)

FROM LIVE.cleaned_sales

GROUP BY store_id
```

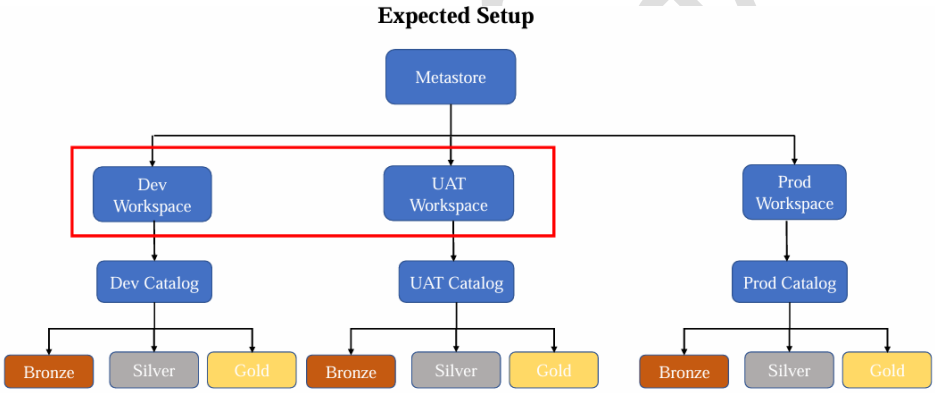
Reference: https://docs.databricks.com/workflows/delta-live-tables/delta-live-tables-sql-ref.html

Expectations in DLT pipeline

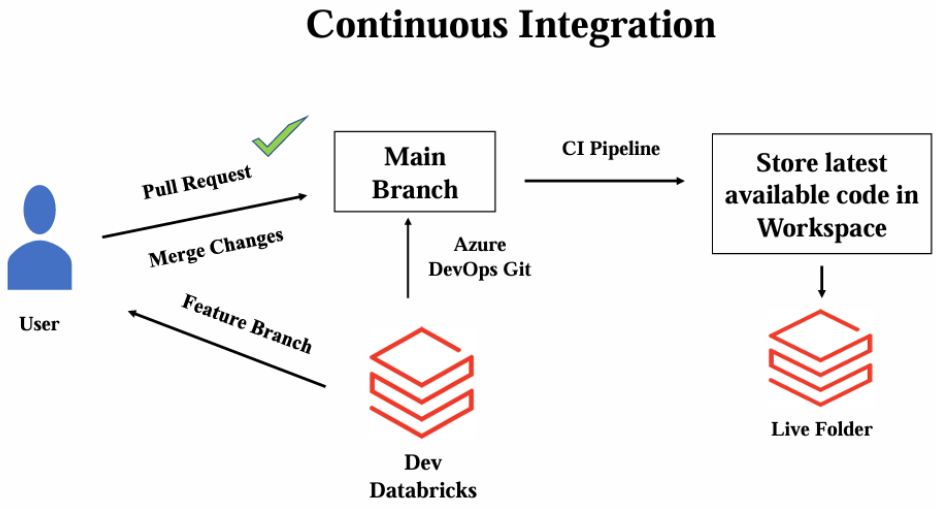
Action	Result	Usage
warn (default)	Invalid records are written to the target; failure is reported as a metric for the dataset.	--
drop	Invalid records are dropped before data is written to the target; failure is reported as a metrics for the dataset.	On Violation Drop Row
fail	Invalid records prevent the update from succeeding. Manual intervention is required before	On Violation Fail Update

Day 18: Capstone Project I

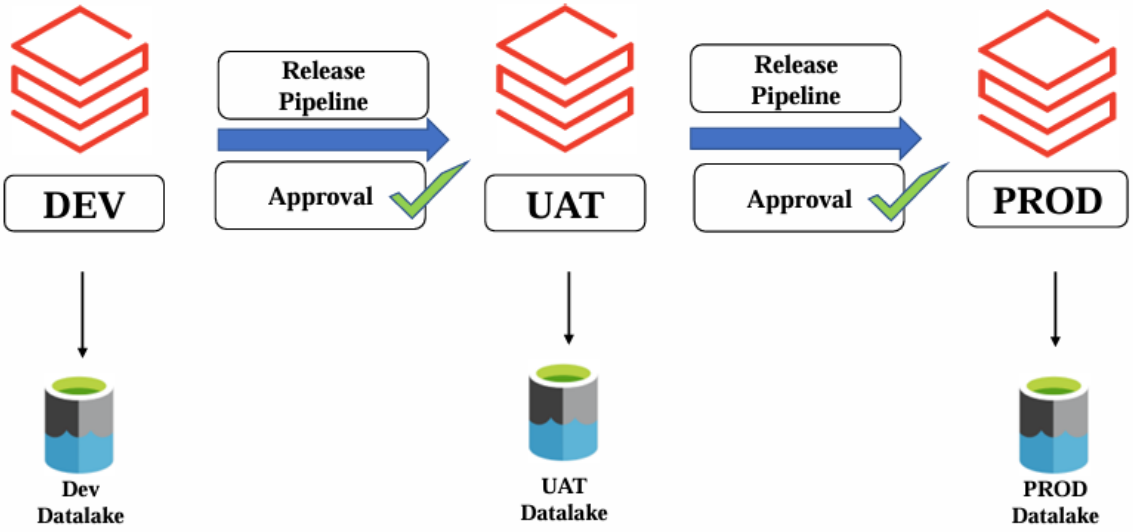
Reference:



Details:



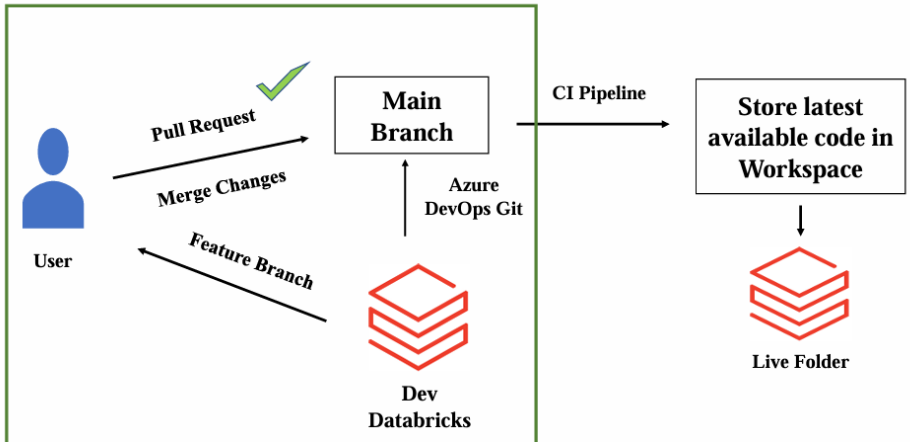
Continuous Deployment



Creating UAT resources in Azure

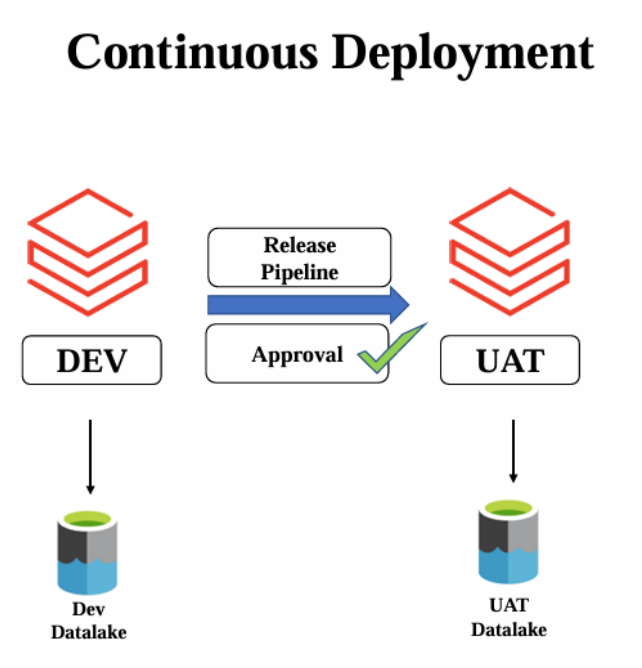
- **Resource Group:** databricks-uat-rg
- **Databricks workspace:** databricks-uat-ws
- **Storage Account:** databricksuatstg

Continuous Integration



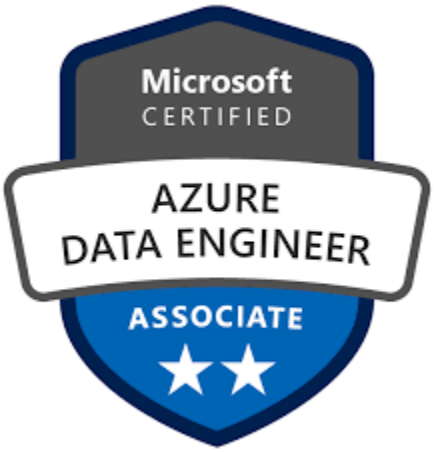
Day 19: Capstone Project II

Reference:



Details:

Azure Data Engineering



Q.1. which of the following commands can a data engineer use to compact small data files of a Delta table into larger ones?

Ans: OPTIMIZE

Overall explanation

Delta Lake can improve the speed of read queries from a table. One way to improve this speed is by compacting small files into larger ones. You trigger compaction by running the OPTIMIZE command

Reference: https://docs.databricks.com/sql/language-manual/delta-optimize.html

Q.2. A data engineer is trying to use Delta time travel to rollback a table to a previous version, but the data engineer received an error that the data files are no longer present.

Which of the following commands was run on the table that caused deleting the data files?

Ans: VACUUM

Overall explanation

Running the VACUUM command on a Delta table deletes the unused data files older than a specified data retention period. As a result, you lose the ability to time travel back to any version older than that retention threshold.

Reference: https://docs.databricks.com/sql/language-manual/delta-vacuum.html

Q.3. In Delta Lake tables, which of the following is the primary format for the data files?

Ans: Parquet

Overall explanation

Delta Lake builds upon standard data formats. Delta lake table gets stored on the storage in one or more data files in Parquet format, along with transaction logs in JSON format.

Reference: https://docs.databricks.com/delta/index.html

Q.4. Which of the following locations hosts the Databricks web application ?

Ans: Control plane

Overall explanation

According to the Databricks Lakehouse architecture, Databricks workspace is deployed in the control plane along with Databricks services like Databricks web application (UI), Cluster manager, workflow service, and notebooks.

Reference: https://docs.databricks.com/getting-started/overview.html

Q.5. In Databricks Repos, which of the following operations a data engineer can use to update the local version of a repo from its remote Git repository ?

Ans: Pull

Overall explanation

The git Pull operation is used to fetch and download content from a remote repository and immediately update the local repository to match that content.

References:

https://docs.databricks.com/repos/index.html

https://github.com/git-guides/git-pull

Q.6. According to the Databricks Lakehouse architecture, which of the following is located in the customer's cloud account?

Ans: Cluster virtual machines

Overall explanation

When the customer sets up a Spark cluster, the cluster virtual machines are deployed in the data plane in the customer's cloud account.

Reference: https://docs.databricks.com/getting-started/overview.html

Q.7. Describe Databricks Lakehouse?

Ans: Single, flexible, high-performance system that supports data, analytics, and machine learning workloads.

Overall explanation

Databricks Lakehouse is a unified analytics platform that combines the best elements of data lakes and data warehouses. So, in the Lakehouse, you can work on data engineering, analytics, and AI, all in one platform.

Reference: https://www.databricks.com/glossary/data-lakehouse

Q.8. If the default notebook language is SQL, which of the following options a data engineer can use to run a Python code in this SQL Notebook ?

Ans: They can add %python at the start of a cell.

Overall explanation

By default, cells use the default language of the notebook. You can override the default language in a cell by using the language magic command at the beginning of a cell. The supported magic commands are: %python, %sql, %scala, and %r.

Reference: https://docs.databricks.com/notebooks/notebooks-code.html

- A Python wheel is a binary distribution format for installing custom Python code packages on Databricks Clusters

Q.9. Which of the following tasks is not supported by Databricks Repos, and must be performed in your Git provider ?

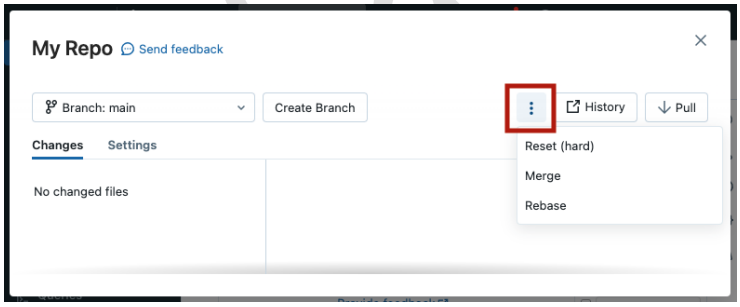
Ans: Delete branches

Overall explanation

The following tasks are not supported by Databricks Repos, and must be performed in your Git provider:

1. Create a pull request
2. Delete branches
3. Merge and rebase branches *

* NOTE: Recently, merge and rebase branches have become supported in Databricks Repos. However, this may still not be updated in the current exam version.



Q.10. Which of the following statements is Not true about Delta Lake ?

Ans: Delta Lake builds upon standard data formats: Parquet + XML

Overall explanation

It is not true that Delta Lake builds upon XML format. It builds upon Parquet and JSON formats

Reference: https://docs.databricks.com/delta/index.html

Q.11. How long is the default retention period of the VACUUM command ?

Ans: 7 days

Overall explanation

By default, the retention threshold of the VACUUM command is 7 days. This means that VACUUM operation will prevent you from deleting files less than 7 days old, just to ensure that no long-running operations are still referencing any of the files to be deleted.

Reference: https://docs.databricks.com/sql/language-manual/delta-vacuum.html

Q.12. The data engineering team has a Delta table called employees that contains the employees personal information including their gross salaries.

Which of the following code blocks will keep in the table only the employees having a salary greater than 3000 ?

Ans: DELETE FROM employees WHERE salary <= 3000;

Overall explanation

In order to keep only the employees having a salary greater than 3000, we must delete the employees having salary less than or equal 3000. To do so, use the DELETE statement:

DELETE FROM table_name WHERE condition;

Reference: https://docs.databricks.com/sql/language-manual/delta-delete-from.html

Q.13. A data engineer wants to create a relational object by pulling data from two tables. The relational object must be used by other data engineers in other sessions on the same cluster only. In order to save on storage costs, the date engineer wants to avoid copying and storing physical data.

Which of the following relational objects should the data engineer create?

Ans: Global Temporary view (not External table or Managed table)

Overall explanation

In order to avoid copying and storing physical data, the data engineer must create a view object. A view in databricks is a virtual table that has no physical data. It's just a saved SQL query against actual tables.

The view type should be Global Temporary view that can be accessed in other sessions on the same cluster. Global Temporary views are tied to a cluster temporary database called global_temp.

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-create-view.html

Q.14. Fill in the below blank to successfully create a table in Databricks using data from an existing PostgreSQL database:

```
CREATE TABLE employees
 USING _____
 OPTIONS (
  url "jdbc:postgresql:dbserver",
  dbtable "employees"
 )
```

Ans: org.apache.spark.sql.jdbc

Overall explanation

Using the JDBC library, Spark SQL can extract data from any existing relational database that supports JDBC. Examples include mysql, postgres, SQLite, and more.

Reference: https://learn.microsoft.com/en-us/azure/databricks/external-data/jdbc

Q.15. Which of the following commands can a data engineer use to create a new table along with a comment ?

Ans: CREATE TABLE payments

COMMENT "This table contains sensitive information"

AS SELECT * FROM bank_transactions

Overall explanation

The CREATE TABLE clause supports adding a descriptive comment for the table. This allows for easier discovery of table contents.

Syntax:

CREATE TABLE table_name

COMMENT "here is a comment"

AS query

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-create-table-using.html

Q.16. A junior data engineer usually uses INSERT INTO command to write data into a Delta table. A senior data engineer suggested using another command that avoids writing of duplicate records.

Which of the following commands is the one suggested by the senior data engineer ?

Ans: MERGE INTO (not APPLY CHANGES INTO or UPDATE or COPY INTO)

MERGE INTO allows to merge a set of updates, insertions, and deletions based on a source table into a target Delta table. With MERGE INTO, you can avoid inserting the duplicate records when writing into Delta tables.

References:

https://docs.databricks.com/sql/language-manual/delta-merge-into.html

https://docs.databricks.com/delta/merge.html#data-deduplication-when-writing-into-delta-tables

- Merge operation cannot be performed if multiple source rows matched and attempted to modify the same target row in the table. The result may be ambiguous as it is unclear which source row should be used to update or delete the matching target row.

- Real scenarios:

1. **Scenario 1: Merge into:** The analyst needs to **update existing records** and **insert new records** into `customer_data` from a staging table, ensuring that duplicates are handled based on a matching condition.
2. **Scenario 2: Copy into:** The analyst has a set of new data files that need to be **bulk-loaded** into `customer_data`, appending the data without modifying any existing records.
3. **Scenario 3: Insert into:** The analyst needs to **append new records** from another table into `customer_data`, with the source table already matching the structure of the target table.

Q.17. A data engineer is designing a Delta Live Tables pipeline. The source system generates files containing changes captured in the source data. Each change event has metadata indicating whether the specified record was inserted, updated, or deleted. In addition to a timestamp column indicating the order in which the changes happened. The data engineer needs to update a target table based on these change events.

Which of the following commands can the data engineer use to best solve this problem?

Ans: APPLY CHANGES INTO

Overall explanation

The events described in the question represent Change Data Capture (CDC) feed. CDC is logged at the source as events that contain both the data of the records along with metadata information:

Operation column indicating whether the specified record was inserted, updated, or deleted

Sequence column that is usually a timestamp indicating the order in which the changes happened

You can use the APPLY CHANGES INTO statement to use Delta Live Tables CDC functionality

Reference: https://docs.databricks.com/workflows/delta-live-tables/delta-live-tables-cdc.html

Q.18. In PySpark, which of the following commands can you use to query the Delta table employees created in Spark SQL?

Ans: spark.table("employees")

Overall explanation

spark.table() function returns the specified Spark SQL table as a PySpark DataFrame

Reference:

https://spark.apache.org/docs/2.4.0/api/python/_modules/pyspark/sql/session.html#SparkSession.table

Q.19. When dropping a Delta table, which of the following explains why only the table's metadata will be deleted, while the data files will be kept in the storage ?

Ans: The table is external

Overall explanation

External (unmanaged) tables are tables whose data is stored in an external storage path by using a LOCATION clause.

When you run DROP TABLE on an external table, only the table's metadata is deleted, while the underlying data files are kept.

Reference: https://docs.databricks.com/lakehouse/data-objects.html#what-is-an-unmanaged-table

- For a managed table, both the metadata and data files are deleted when the table is dropped. However, for an unmanaged (external) table, only the metadata is removed, and the data files remain in their original location.

Q.20. Given the two tables students_course_1 and students_course_2. Which of the following commands can a data engineer use to get all the students from the above two tables without duplicate records ?

Ans: SELECT * FROM students_course_1

UNION

SELECT * FROM students_course_2

Overall explanation

With UNION, you can return the result of subquery1 plus the rows of subquery2

Syntax:

subquery1

UNION [ALL | DISTINCT]

subquery2

If ALL is specified duplicate rows are preserved.

If DISTINCT is specified the result does not contain any duplicate rows. This is the default.

Note that both subqueries must have the same number of columns and share a least common type for each respective column.

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-qry-select-setops.html

Q.21. Given the following command:

CREATE DATABASE IF NOT EXISTS hr_db ;

In which of the following locations will the hr_db database be located?

Ans: dbfs:/user/hive/warehouse

Overall explanation

Since we are creating the database here without specifying a LOCATION clause, the database will be created in the default warehouse directory under dbfs:/user/hive/warehouse

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-create-schema.html

Q.22. Fill in the below blank to get the students enrolled in less than 3 courses from array column students

```
SELECT
 faculty_id,
 students,
 __________ AS few_courses_students
FROM faculties
```

Ans: FILTER (students, i -> i.total_courses < 3)

Overall explanation

filter(input_array, lamda_function) is a higher order function that returns an output array from an input array by extracting elements for which the predicate of a lambda function holds.

Example:

Extracting odd numbers from an input array of integers:

SELECT filter(array(1, 2, 3, 4), i -> i % 2 == 1);

output: [1, 3]

References:

https://docs.databricks.com/sql/language-manual/functions/filter.html

https://docs.databricks.com/optimizations/higher-order-lambda-functions.html

Q.23. The data engineer team has a DLT pipeline that updates all the tables once and then stops. The compute resources of the pipeline continue running to allow for quick testing.

Which of the following best describes the execution modes of this DLT pipeline ?

Ans: The DLT pipeline executes in Triggered Pipeline mode under Development mode.

Overall explanation

Triggered pipelines update each table with whatever data is currently available and then they shut down.

In Development mode, the Delta Live Tables system ease the development process by

- Reusing a cluster to avoid the overhead of restarts. The cluster runs for two hours when development mode is enabled.
- Disabling pipeline retries so you can immediately detect and fix errors.

Reference:

https://docs.databricks.com/workflows/delta-live-tables/delta-live-tables-concepts.html

Q.24. In multi-hop architecture, which of the following statements best describes the Bronze layer ?

Ans: It maintains raw data ingested from various sources

Overall explanation

Bronze tables contain data in its rawest format ingested from various sources (e.g., JSON files, Operational Databaes, Kakfa stream, ...)

Reference:

https://www.databricks.com/glossary/medallion-architecture

Q.25. Which of the following compute resources is available in Databricks SQL ?

Ans: SQL warehouses

Overall explanation

Compute resources are infrastructure resources that provide processing capabilities in the cloud. A SQL warehouse is a compute resource that lets you run SQL commands on data objects within Databricks SQL.

Reference: https://docs.databricks.com/sql/admin/sql-endpoints.html

Q.26. Which of the following is the benefit of using the Auto Stop feature of Databricks SQL warehouses ?

Ans: Minimizes the total running time of the warehouse

Overall explanation

The Auto Stop feature stops the warehouse if it's idle for a specified number of minutes.

Reference: https://docs.databricks.com/sql/admin/sql-endpoints.html

Q.27. A data engineer wants to increase the cluster size of an existing Databricks SQL warehouse.

Which of the following is the benefit of increasing the cluster size of Databricks SQL warehouses ?

Ans: Improves the latency of the queries execution

Overall explanation

Cluster Size represents the number of cluster workers and size of compute resources available to run your queries and dashboards. To reduce query latency, you can increase the cluster size.

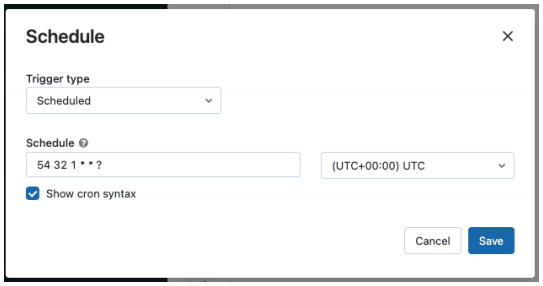
Reference: https://docs.databricks.com/sql/admin/sql-endpoints.html#cluster-size-1

Q.28. Which of the following describes Cron syntax in Databricks Jobs ?

Ans: It's an expression to represent complex job schedule that can be defined programmatically

Overall explanation

To define a schedule for a Databricks job, you can either interactively specify the period and starting time, or write a Cron Syntax expression. The Cron Syntax allows to represent complex job schedule that can be defined programmatically.



Reference: https://docs.databricks.com/workflows/jobs/jobs.html#schedule-a-job

Q.29. The data engineer team has a DLT pipeline that updates all the tables at defined intervals until manually stopped. The compute resources terminate when the pipeline is stopped.

Which of the following best describes the execution modes of this DLT pipeline ?

Ans: The DLT pipeline executes in Continuous Pipeline mode under Production mode.

Overall explanation

Continuous pipelines update tables continuously as input data changes. Once an update is started, it continues to run until the pipeline is shut down.

In Production mode, the Delta Live Tables system:

- Terminates the cluster immediately when the pipeline is stopped.
- Restarts the cluster for recoverable errors (e.g., memory leak or stale credentials).
- Retries execution in case of specific errors (e.g., a failure to start a cluster)

Reference:

https://docs.databricks.com/workflows/delta-live-tables/delta-live-tables-concepts.html

Q.30. Which of the following commands can a data engineer use to purge stale data files of a Delta table?

Ans: VACUUM

Overall explanation

The VACUUM command deletes the unused data files older than a specified data retention period.

Reference: https://docs.databricks.com/sql/language-manual/delta-vacuum.html

Q.31. In Databricks Repos (Git folders), which of the following operations a data engineer can use to save local changes of a repo to its remote repository ?

Ans: Commit & Push

Overall explanation

Commit & Push is used to save the changes on a local repo, and then uploads this local repo content to the remote repository.

References:

https://docs.databricks.com/repos/index.html

https://github.com/git-guides/git-push

Q.32. In Delta Lake tables, which of the following is the primary format for the transaction log files?

Ans: JSON

Overall explanation

Delta Lake builds upon standard data formats. Delta lake table gets stored on the storage in one or more data files in Parquet format, along with transaction logs in JSON format.

Reference: https://docs.databricks.com/delta/index.html

Q.33. Which of the following locations completely hosts the customer data ?

Ans: Customer's cloud account

Overall explanation

According to the Databricks Lakehouse architecture, the storage account hosting the customer data is provisioned in the data plane in the Databricks customer's cloud account.

Reference: https://docs.databricks.com/getting-started/overview.html

Q.34. A junior data engineer uses the built-in Databricks Notebooks versioning for source control. A senior data engineer recommended using Databricks Repos (Git folders) instead.

Which of the following could explain why Databricks Repos is recommended instead of Databricks Notebooks versioning?

Ans: Databricks Repos supports creating and managing branches for development work.

Overall explanation

One advantage of Databricks Repos over the built-in Databricks Notebooks versioning is that Databricks Repos supports creating and managing branches for development work.

Reference: https://docs.databricks.com/repos/index.html

Q.35. Which of the following services provides a data warehousing experience to its users?

Ans: Databricks SQL

Overall explanation

Databricks SQL (DB SQL) is a data warehouse on the Databricks Lakehouse Platform that lets you run all your SQL and BI applications at scale.

Reference: https://www.databricks.com/product/databricks-sql

Q.36. A data engineer noticed that there are unused data files in the directory of a Delta table. They executed the VACUUM command on this table; however, only some of those unused data files have been deleted.

Which of the following could explain why only some of the unused data files have been deleted after running the VACUUM command ?

Ans: The deleted data files were older than the default retention threshold. While the remaining files are newer than the default retention threshold and can not be deleted.

Overall explanation

Running the VACUUM command on a Delta table deletes the unused data files older than a specified data retention period. Unused files newer than the default retention threshold are kept untouched.

Reference: https://docs.databricks.com/sql/language-manual/delta-vacuum.html

Q.37. The data engineering team has a Delta table called products that contains products' details including the net price.

Which of the following code blocks will apply a 50% discount on all the products where the price is greater than 1000 and save the new price to the table?

Ans: UPDATE products SET price = price * 0.5 WHERE price > 1000;

Overall explanation

The UPDATE statement is used to modify the existing records in a table that match the WHERE condition. In this case, we are updating the products where the price is strictly greater than 1000.

Syntax:

```
UPDATE table_name
SET column_name = expr
WHERE condition
```

Reference:

https://docs.databricks.com/sql/language-manual/delta-update.html

Q.38. A data engineer wants to create a relational object by pulling data from two tables. The relational object will only be used in the current session. In order to save on storage costs, the date engineer wants to avoid copying and storing physical data.

Which of the following relational objects should the data engineer create?

Ans: Temporary view

Overall explanation

In order to avoid copying and storing physical data, the data engineer must create a view object. A view in databricks is a virtual table that has no physical data. It's just a saved SQL query against actual tables.

The view type should be Temporary view since it's tied to a Spark session and dropped when the session ends.

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-create-view.html

Q.39. A data engineer has a database named db_hr, and they want to know where this database was created in the underlying storage.

Which of the following commands can the data engineer use to complete this task?

Ans: DESCRIBE DATABASE db_hr

Overall explanation

The DESCRIBE DATABASE or DESCRIBE SCHEMA returns the metadata of an existing database (schema). The metadata information includes the database's name, comment, and location on the filesystem. If the optional EXTENDED option is specified, database properties are also returned.

Syntax:

DESCRIBE DATABASE [EXTENDED] database_name

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-aux-describe-schema.html

Q.40. Which of the following commands a data engineer can use to register the table orders from an existing SQLite database ?

Ans:

```
CREATE TABLE orders
 USING org.apache.spark.sql.jdbc
 OPTIONS (
  url "jdbc:sqlite:/bookstore.db",
  dbtable "orders"
 )
```

Overall explanation

Using the JDBC library, Spark SQL can extract data from any existing relational database that supports JDBC. Examples include mysql, postgres, SQLite, and more.

Reference: https://learn.microsoft.com/en-us/azure/databricks/external-data/jdbc

Q.41. When dropping a Delta table, which of the following explains why both the table's metadata and the data files will be deleted ?

Ans: The table is managed

Overall explanation

Managed tables are tables whose metadata and the data are managed by Databricks.

When you run DROP TABLE on a managed table, both the metadata and the underlying data files are deleted.

Reference: https://docs.databricks.com/lakehouse/data-objects.html#what-is-a-managed-table

In Databricks:

- **Managed Tables:** These tables are fully managed by Databricks, meaning that both their data and metadata are stored in the Databricks File System (DBFS). They persist across sessions and are not automatically deleted unless explicitly dropped.
- **Unmanaged (External) Tables:** These tables store their metadata in DBFS but store their data externally (e.g., in cloud storage like AWS S3 or Azure Blob Storage). They also persist across sessions.
- **Temporary Tables (or Temp Views):** These are session-scoped, meaning they only exist for the duration of the session in which they were created. They are automatically dropped when the session ends.
  This understanding is crucial when working with data in Databricks, as it affects how data is managed and accessed across different user sessions.

Q.42. Given the following commands:

```
CREATE DATABASE db_hr;

USE db_hr;

CREATE TABLE employees;
```

In which of the following locations will the employees table be located?

Ans: dbfs:/user/hive/warehouse/db_hr.db

Overall explanation

Since we are creating the database here without specifying a LOCATION clause, the database will be created in the default warehouse directory under dbfs:/user/hive/warehouse. The database folder have the extension (.db)

And since we are creating the table also without specifying a LOCATION clause, the table becomes a managed table created under the database directory (in db_hr.db folder)

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-create-schema.html

Q.43. Which of the following statements best describes the usage of CREATE SCHEMA command ?

Ans: It's used to create a database

Overall explanation

CREATE SCHEMA is an alias for CREATE DATABASE statement. While usage of SCHEMA and DATABASE is interchangeable, SCHEMA is preferred.

Reference: https://docs.databricks.com/sql/language-manual/sql-ref-syntax-ddl-create-database.html

Q.44. Given the following Structured Streaming query:

```
(spark.table("orders")
    .withColumn("total_after_tax", col("total")+col("tax"))
  .writeStream
    .option("checkpointLocation", checkpointPath)
    .outputMode("append")
    .__________
    .table("new_orders") )
```

Fill in the blank to make the query executes multiple micro-batches to process all available data, then stops the trigger.

Ans: trigger(availableNow=True)

Overall explanation

In Spark Structured Streaming, we use trigger(availableNow=True) to run the stream in batch mode where it processes all available data in multiple micro-batches. The trigger will stop on its own once it finishes processing the available data.

Reference: https://docs.databricks.com/structured-streaming/triggers.html#configuring-incremental-batch-processing

Q.45. In multi-hop architecture, which of the following statements best describes the Silver layer tables?

Ans: They provide a more refined view of raw data, where it's filtered, cleaned, and enriched.

Overall explanation

Silver tables provide a more refined view of the raw data. For example, data can be cleaned and filtered at this level. And we can also join fields from various bronze tables to enrich our silver records

Reference:

https://www.databricks.com/glossary/medallion-architecture

Q.46. The data engineer team has a DLT pipeline that updates all the tables at defined intervals until manually stopped. The compute resources of the pipeline continue running to allow for quick testing.

Which of the following best describes the execution modes of this DLT pipeline ?

Ans: The DLT pipeline executes in Continuous Pipeline mode under Development mode.

Overall explanation

Continuous pipelines update tables continuously as input data changes. Once an update is started, it continues to run until the pipeline is shut down.

In Development mode, the Delta Live Tables system ease the development process by Reusing a cluster to avoid the overhead of restarts. The cluster runs for two hours when development mode is enabled.

Disabling pipeline retries so you can immediately detect and fix errors.

Reference:

https://docs.databricks.com/workflows/delta-live-tables/delta-live-tables-concepts.html

Q.47. Given the following Structured Streaming query:

```
(spark.readStream
    .table("cleanedOrders")
    .groupBy("productCategory")
```

```
        .agg(sum("totalWithTax"))
    .writeStream
        .option("checkpointLocation", checkpointPath)
        .outputMode("complete")
        .table("aggregatedOrders")
)
```

Which of the following best describe the purpose of this query in a multi-hop architecture?

Ans: The query is performing a hop from Silver layer to a Gold table

Overall explanation

The above Structured Streaming query creates business-level aggregates from clean orders data in the silver table cleanedOrders, and loads them in the gold table aggregatedOrders.

Reference:

https://www.databricks.com/glossary/medallion-architecture

Q.48. Given the following Structured Streaming query:

```
(spark.readStream
    .table("orders")
    .writeStream
    .option("checkpointLocation", checkpointPath)
    .table("Output_Table")
)
```

Which of the following is the trigger Interval for this query ?

Ans: Every half second

Overall explanation

By default, if you don't provide any trigger interval, the data will be processed every half second. This is equivalent to trigger (processingTime="500ms")

Reference: https://docs.databricks.com/structured-streaming/triggers.html#what-is-the-default-trigger-interval

Q.49. In multi-hop architecture, which of the following statements best describes the Gold layer tables?

Ans: They provide business-level aggregations that power analytics, machine learning, and production applications

Overall explanation

Gold layer is the final layer in the multi-hop architecture, where tables provide business level aggregates often used for reporting and dashboarding, or even for Machine learning.

The gold layer represents the final, refined data layer that is most commonly used by data analysts for querying and generating insights. This layer contains data that has been cleaned, enriched, and aggregated, making it ideal for reporting, dashboards, and advanced analytics. The gold layer is optimized for performance, ensuring that queries can be run quickly and efficiently. Unlike the bronze layer, which stores raw data, or the silver layer, which is used for data preparation and transformation, the gold layer provides analysts with ready-to-use data for their analysis tasks.

Reference:

https://www.databricks.com/glossary/medallion-architecture

Q.50. The data engineer team has a DLT pipeline that updates all the tables once and then stops. The compute resources of the pipeline terminate when the pipeline is stopped.

Which of the following best describes the execution modes of this DLT pipeline ?

Ans: The DLT pipeline executes in Triggered Pipeline mode under Production mode.

Overall explanation

Triggered pipelines update each table with whatever data is currently available and then they shut down.

In Production mode, the Delta Live Tables system:

- Terminates the cluster immediately when the pipeline is stopped.
- Restarts the cluster for recoverable errors (e.g., memory leak or stale credentials).
- Retries execution in case of specific errors (e.g., a failure to start a cluster)

Reference:

https://docs.databricks.com/workflows/delta-live-tables/delta-live-tables-concepts.html

Q.51. A data engineer needs to determine whether to use Auto Loader or COPY INTO command in order to load input data files incrementally.

In which of the following scenarios should the data engineer use Auto Loader over COPY INTO command ?

Ans: If they are going to ingest files in the order of millions or more over time

Overall explanation

Here are a few things to consider when choosing between Auto Loader and COPY INTO command:

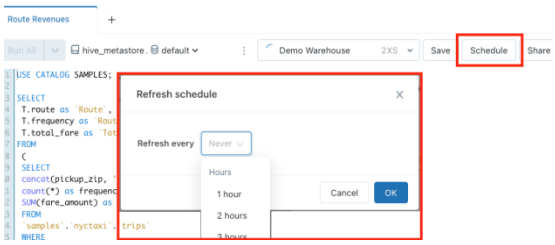
- If you're going to ingest files in the order of thousands, you can use COPY INTO. If you are expecting files in the order of millions or more over time, use Auto Loader.
- If your data schema is going to evolve frequently, Auto Loader provides better primitives around schema inference and evolution.

Reference: https://docs.databricks.com/ingestion/index.html#when-to-use-copy-into-and-when-to-use-auto-loader

- To import data from object storage into Databricks SQL, the most effective method is to use the `COPY INTO` command. This command allows you to load data directly from external object storage (such as AWS S3, Azure Blob Storage, or Google Cloud Storage) into a Databricks SQL table. It is designed to be straightforward and efficient, enabling users to specify the source file location and target table within Databricks SQL, automating the process of importing data. Other methods, like manual uploads or custom scripts, are less efficient and more error-prone compared to the built-in capabilities provided by Databricks SQL with the `COPY INTO` command.

Q.52. From which of the following locations can a data engineer set a schedule to automatically refresh a Databricks SQL query ?

Ans: From the query's page in Databricks SQL



Overall explanation

In Databricks SQL, you can set a schedule to automatically refresh a query from the query's page.

Reference: https://docs.databricks.com/sql/user/queries/schedule-query.html

Q.53. Databricks provides a declarative ETL framework for building reliable and maintainable data processing pipelines, while maintaining table dependencies and data quality.

Which of the following technologies is being described above?

Ans: Delta Live Tables

Overall explanation

Delta Live Tables is a framework for building reliable, maintainable, and testable data processing pipelines. You define the transformations to perform on your data, and Delta Live Tables manages task orchestration, cluster management, monitoring, data quality, and error handling.

Reference: https://docs.databricks.com/workflows/delta-live-tables/index.html

Q.54. Which of the following services can a data engineer use for orchestration purposes in Databricks platform ?

Ans: Databricks Jobs

Overall explanation

Databricks Jobs allow orchestrating data processing tasks. This means the ability to run and manage multiple tasks as a directed acyclic graph (DAG) in a job.

Reference: https://docs.databricks.com/workflows/jobs/jobs.html

Q.55. A data engineer has a Job with multiple tasks that takes more than 2 hours to complete. In the last run, the final task unexpectedly failed.

Which of the following actions can the data engineer perform to complete this Job Run while minimizing the execution time ?

Ans: They can repair this Job Run so only the failed tasks will be re-executed

Overall explanation

You can repair failed multi-task jobs by running only the subset of unsuccessful tasks and any dependent tasks. Because successful tasks are not re-run, this feature reduces the time and resources required to recover from unsuccessful job runs.

Reference: https://docs.databricks.com/workflows/jobs/repair-job-failures.html

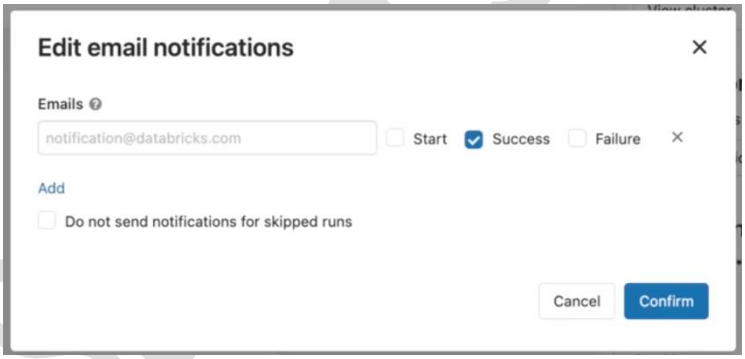
Q.56. A data engineering team has a multi-tasks Job in production. The team members need to be notified in the case of job failure.

Which of the following approaches can be used to send emails to the team members in the case of job failure ?

Ans: They can configure email notifications settings in the job page

Overall explanation

Databricks Jobs support email notifications to be notified in the case of job start, success, or failure. Simply, click Edit email notifications from the details panel in the Job page. From there, you can add one or more email addresses.



Q.57. For production jobs, which of the following cluster types is recommended to use?

Ans: Job clusters

Overall explanation

Job Clusters are dedicated clusters for a job or task run. A job cluster auto terminates once the job is completed, which saves cost compared to all-purpose clusters.

In addition, Databricks recommends using job clusters in production so that each job runs in a fully isolated environment.

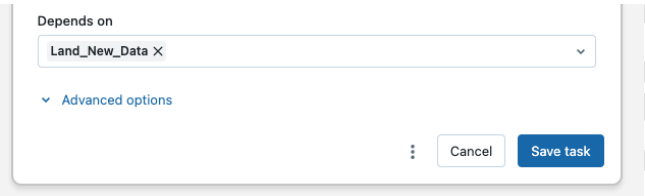
Reference: https://docs.databricks.com/workflows/jobs/jobs.html#choose-the-correct-cluster-type-for-your-job

Q.58. In Databricks Jobs, which of the following approaches can a data engineer use to configure a linear dependency between Task A and Task B ?

Ans: They can select the Task A in the Depends On field of the Task B configuration

Overall explanation

You can define the order of execution of tasks in a job using the Depends on dropdown menu. You can set this field to one or more tasks in the job.



Q.59. Which part of the Databricks Platform can a data engineer use to revoke permissions from users on tables ?

Ans: Data Explorer

Overall explanation

Data Explorer in Databricks SQL allows you to manage data object permissions. This includes revoking privileges on tables and databases from users or groups of users.

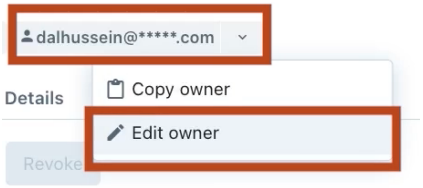
Reference: https://docs.databricks.com/security/access-control/data-acl.html#data-explorer

Q.60. In which of the following locations can a data engineer change the owner of a table?

Ans: In Data Explorer, from the Owner field in the table's page

Overall explanation

From Data Explorer in Databricks SQL, you can navigate to the table's page to review and change the owner of the table. Simply, click on the Owner field, then Edit owner to set the new owner.



Reference: https://docs.databricks.com/security/access-control/data-acl.html#manage-data-object-ownership

Q.61. **What is Broadcasting in PySpark, and Why Is It Useful?**

→ Explain how broadcasting optimizes joins by reducing data shuffling.

Ans: Broadcasting is a technique used in PySpark to optimize the performance of operations involving small DataFrames. When a DataFrame is broadcasted, it is sent to all worker nodes and cached, ensuring that each node has a full copy of the data. This eliminates the need to shuffle and exchange data between nodes during operations, such as joins, significantly reducing the communication overhead and improving performance.

- **When to Use Broadcasting:**

Broadcasting should be used when you have a small DataFrame that is used multiple times in your processing pipeline, especially in join operations. Broadcasting the small DataFrame can significantly improve performance by reducing the amount of data that needs to be exchanged between worker nodes.

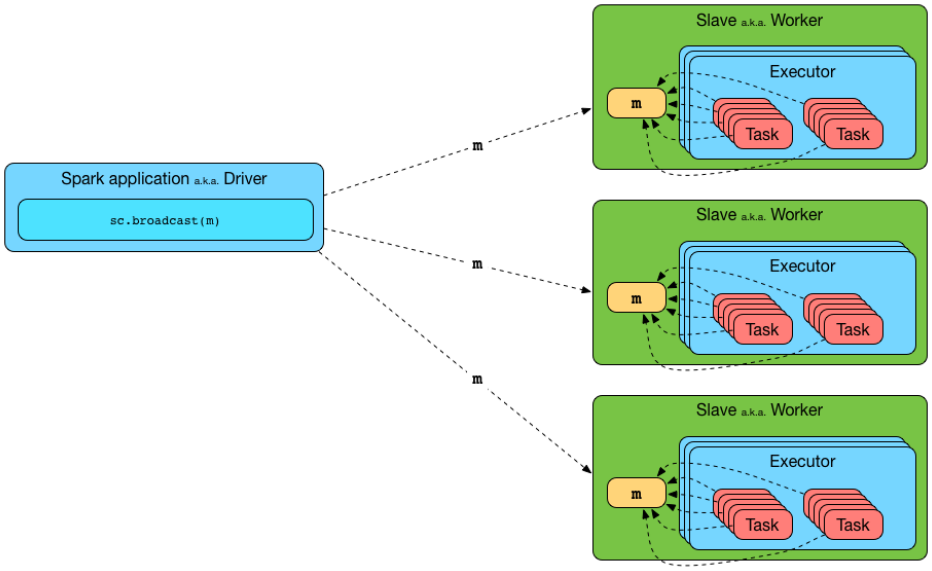
- Resource: https://www.sparkcodehub.com/broadcasting-dataframes-in-pyspark

When working with large datasets in PySpark, optimizing performance is crucial. Two powerful optimization techniques are broadcast variables and broadcast joins. Let's dive into what they are, when to use them, and how they help improve performance with clear examples.

- **Broadcast Variables:**

What is a Broadcast Variable?

A broadcast variable allows you to efficiently share a small, read-only dataset across all executors in a cluster. Instead of sending this data with every task, it is sent once from the driver to each executor, minimizing network I/O and allowing tasks to access it locally.



When to Use a Broadcast Variable?

- Scenario: When you need to share small lookup data or configuration settings with all tasks in the cluster.

- Optimization: Minimizes network I/O by sending the data once and caching it locally on each executor.

Example Code

Let's say we have a small dictionary of country codes and names that we need to use in our transformations.

[1]:
```
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName('Broadcast Variable Example').getOrCreate()

# Create a dictionary to broadcast
country_codes = {"US": "United States", "IN": "India", "AU": "Australia"}

# Broadcast the dictionary
broadcastCountries = spark.sparkContext.broadcast(country_codes)

# Sample data
data = [("Alice", "Brown", "US"),
        ("Bob", "Smith", "IN"),
        ("Charlie", "Williams", "AU")]

columns = ["firstname", "lastname", "country_code"]
df = spark.createDataFrame(data, columns)

# Function to convert country codes using the broadcast variable
def country_convert(code):
    return broadcastCountries.value.get(code, "Unknown")

# Apply the function to the DataFrame
result = df.rdd.map(lambda x: (x[0], x[1], country_convert(x[2]))).toDF(["firstname", "lastname", "country"])
result.show(truncate=False)
```

```
+---------+--------+-------------+
|firstname|lastname|country      |
+---------+--------+-------------+
|Alice    |Brown   |United States|
|Bob      |Smith   |India        |
|Charlie  |Williams|Australia    |
+---------+--------+-------------+
```

In above example:

- We create and broadcast a dictionary of country codes.

- Each task accesses the broadcasted dictionary locally to convert country codes.

- **Broadcast Joins:**

What is a Broadcast Join?

A broadcast join optimizes join operations by broadcasting a small dataset to all executor nodes. This allows each node to perform the join locally, reducing the need for shuffling large datasets across the network.

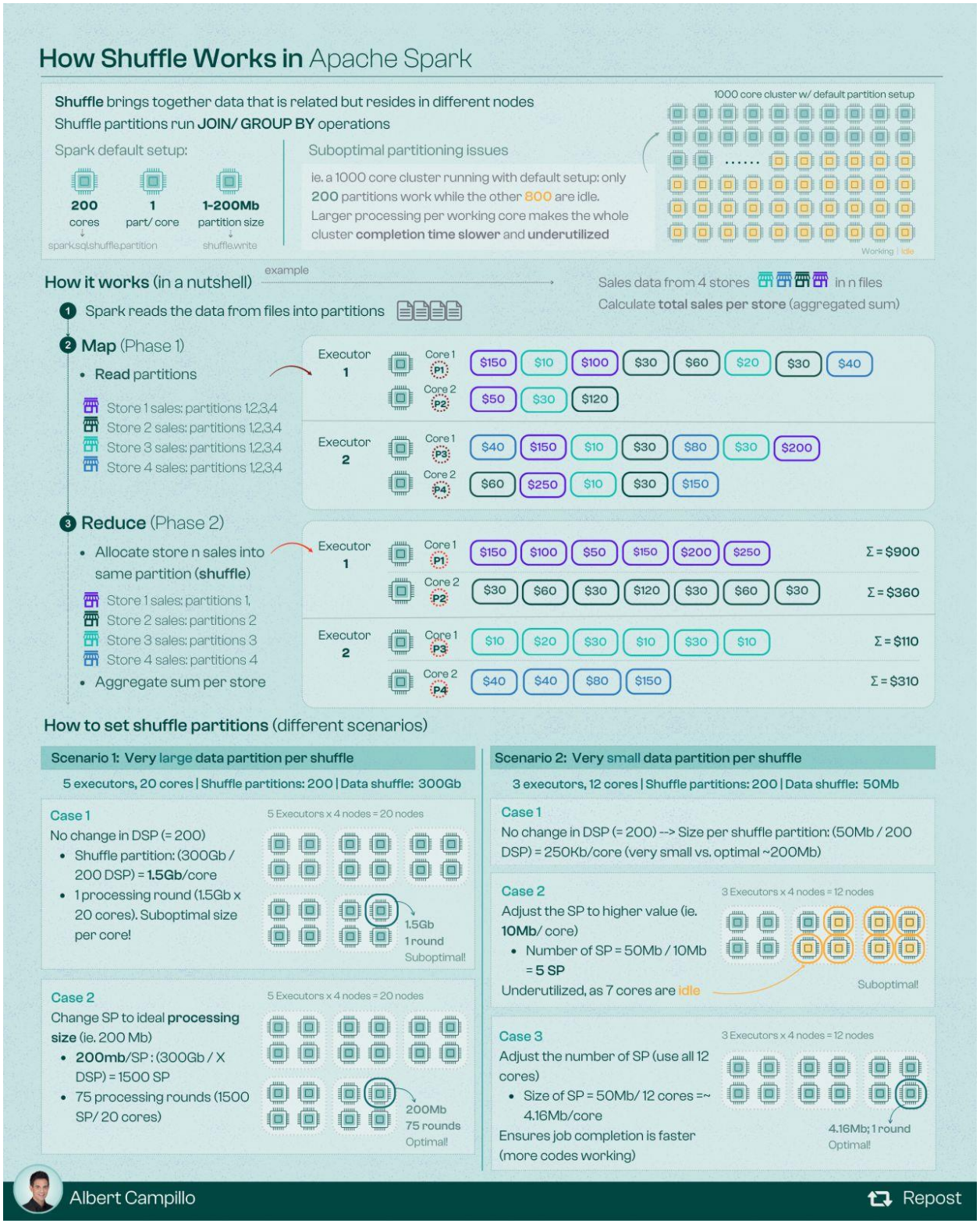
When to Use a Broadcast Join?

- Scenario: When performing joins and one of the datasets is small enough to fit in memory.

- Optimization: Reduces shuffling and network I/O, making joins more efficient by enabling local join operations.

Best Practices for broadcasting:

- **Only broadcast small DataFrames:** Broadcasting large DataFrames can cause performance issues and consume a significant amount of memory on worker nodes. Make sure to only broadcast DataFrames that are small enough to fit in the memory of each worker node.
- **Monitor the performance:** Keep an eye on the performance of your PySpark applications to ensure that broadcasting is improving performance as expected. If you notice any performance issues or memory problems, consider adjusting your broadcasting strategy or revisiting your data processing pipeline.
- **Consider alternative techniques:** Broadcasting is not always the best solution for optimizing performance. In some cases, you may achieve better results by repartitioning your DataFrames or using other optimization techniques, such as bucketing or caching. Evaluate your specific use case and choose the most appropriate technique for your needs.
- **Be cautious with broadcasting in iterative algorithms:** If you're using iterative algorithms, be careful when broadcasting DataFrames, as the memory used by the broadcasted DataFrame may not be released until the end of the application. This could lead to memory issues and performance problems over time.



Q.62. What Makes Spark Superior to MapReduce?

→ Discuss Spark's in-memory processing and speed advantages over Hadoop MapReduce.

Ans: Anser

Q.63. Differentiate Between RDDs, DataFrames, and Datasets.

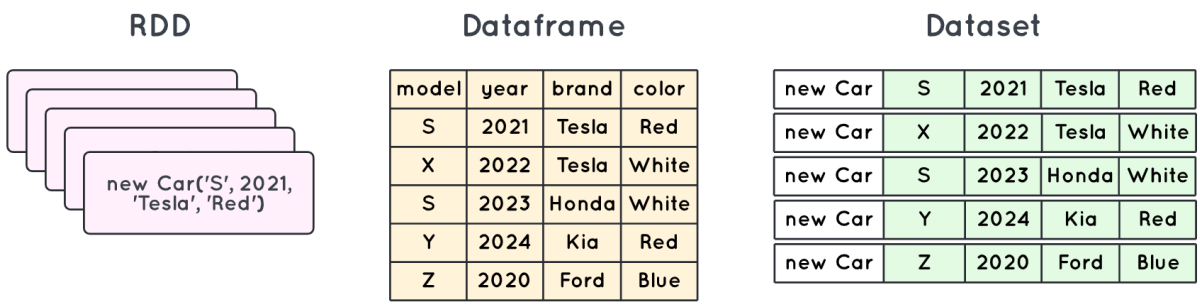
→ Compare their flexibility, performance, and optimization capabilities.

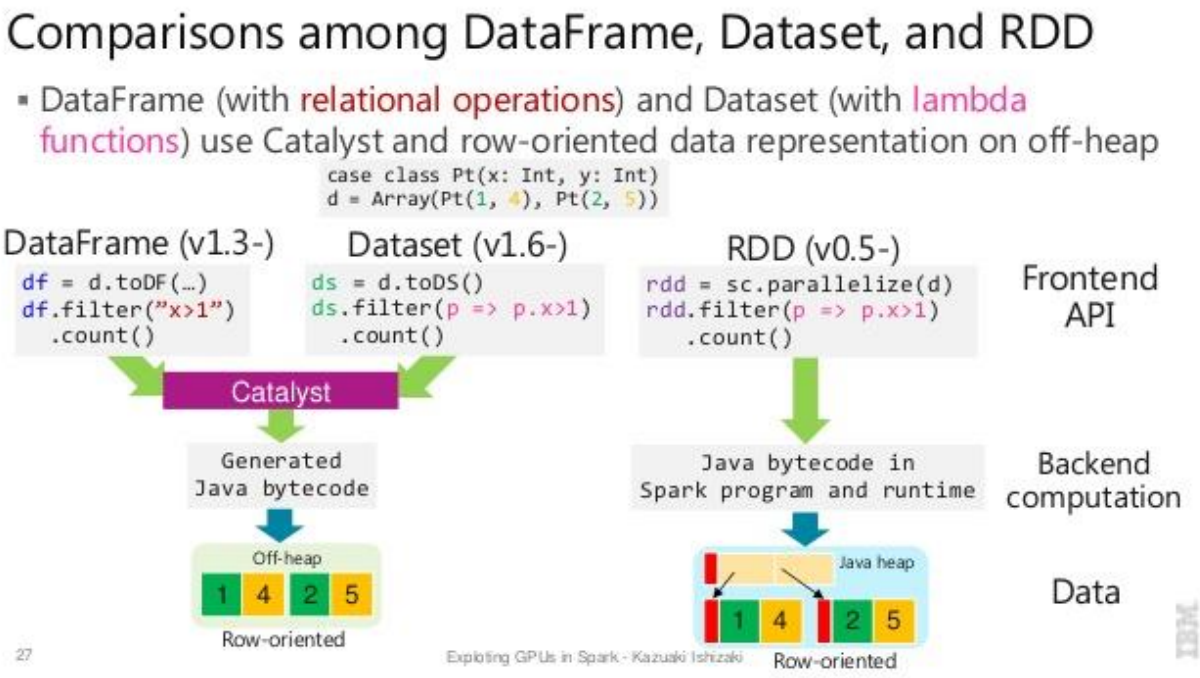
Answer:

RDD	DataFrame	DataSet
RDD is the foundational data structure in Spark and represents a distributed collection of data, typically unstructured	DataFrame is a higher-level abstraction that represents distributed data in a tabular form with rows and columns.	Dataset is a hybrid abstraction introduced in Spark that combines the best features of RDDs and DataFrames
RDDs are inherently distributed and partitioned across the cluster.	DataFrames have a schema, which defines the data types and names of columns.	Datasets are strongly typed like RDDs but also have a well-defined schema like DataFrames.
immutable, and transformations on RDDs result in new RDD	DataFrames are designed for structured data processing and optimization.	Datasets can work with both structured and unstructured data
RDDs are low-level and do not have a schema	DataFrame operations can be expressed using SQL-like syntax	Datasets support both safe and unsafe type conversions, allowing you to handle data of different types flexibly.
RDDs offer full control and can handle complex data processing scenarios.	DataFrames can read from and write to various data sources	Datasets offer good interoperability with both Java and Scala, allowing developers to choose the language that best suits their needs

# Creating an RDD from a list 	
rdd = sc.parallelize([1, 2, 3, 4, 5])
	data = [(1, "Alice"), (2, "Bob"), (3, "Charlie")]
	columns = ["ID", "Name"]
	df = spark.createDataFrame(data, columns)
		data = [Person(1, "Alice"), Person(2, "Bob"), Person(3, "Charlie")]
		schema = StructType([StructField("ID", IntegerType(), True), StructField("Name", StringType(), True)])
		ds = spark.createDataFrame(data, schema).as[Person]

Data Structure	RDD	DataFrame	Dataset
Data Formats	Structured and Unstructured	Structured and Semi Structured	Structured and Unstructured
Schema Projection	Schemas need to be defined manually.	Auto-discovery of file schemas.	Auto-discovery of file schemas.
Performance	Slower	Faster	Similar to DataFrame
API	Low-level (functional)	Higher-level (SQL-like, DataFrame API)	Object-oriented (type safety)
Optimization	No built-in optimization engine. Each RDD is optimized individually.	Query optimization through the Catalyst optimizer.	Query optimization through the Catalyst optimizer, like DataFrames.
Fault Tolerance	Yes (through lineage information)	Yes (through lineage information)	Yes (through lineage information)
Programming Language	Java, Scala, Python	Java, Scala, Python, R	Scala, Java
Type System	Dynamically typed	Dynamically typed	Statically typed
Expressiveness	Lower	Higher	Higher
Use Cases	Flexibility, control	Structured data, analysis, balance	Performance, type safety, OOP





Ans:

Q.64. How Can You Create a DataFrame in PySpark?

→ Provide examples using spark.read and createDataFrame().

Ans:

Q.65. **Why Is Partitioning Important in Spark?**

→ Explain data distribution and its effect on parallelism and performance.

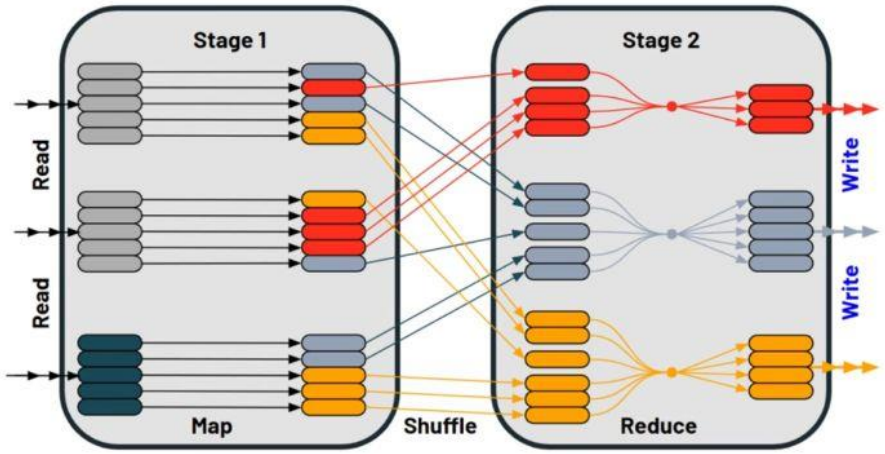
Ans: Efficient data processing in Apache Spark hinges on shuffle partitions, which directly impact performance and cluster utilization.

Here's a quick breakdown:
Shuffling: Occurs during wide transformations (e.g., groupBy, join) to group related data across nodes.

Key Scenarios:
1 Large Data per Partition: Increase partitions to ensure each core handles an optimal workload (1MB–200MB).
2 Small Data per Partition: Reduce partitions or match them to the available cores for better utilization.

Common Challenges:
Data Skew: Uneven data distribution slows jobs.
Solutions: Use Adaptive Query Execution (AQE) or salting to balance partitions.

🎯 Why it Matters: Properly tuning shuffle partitions ensures faster job completion and optimal resource usage, unlocking the true power of Spark!



Q.66. Why Is Caching Crucial in Spark? Share how caching improves performance by storing intermediate results.

Ans: cache() is an Apache Spark transformation that can be used on a DataFrame, Dataset, or RDD when you want to perform more than one action. cache() caches the specified DataFrame, Dataset, or RDD in the memory of your cluster's workers. Since cache() is a transformation, the caching operation

takes place only when a Spark action (for example, count(), show(), take(), or write()) is also used on the same DataFrame, Dataset, or RDD in a single action.

Under what scenarios caching is an optimized solution —

- **Reusing Data: Caching is optimal when you need to perform multiple operations on the same dataset to avoid reading from storage repeatedly.**
- **Frequent Subset Access: Useful for frequently accessing small subsets of a large dataset, reducing the need to load the entire dataset repeatedly.**

In Spark, caching is a mechanism for storing data in memory to speed up access to that data. When you cache a dataset, Spark keeps the data in memory so that it can be quickly retrieved the next time it is needed. Caching is especially useful when you need to perform multiple operations on the same dataset, as it eliminates the need to read the data from a disk each time.

Understanding the concept of caching and how to use it effectively is crucial for optimizing the performance of your Spark applications. By caching the right data at the right time, you can significantly speed up your applications and make the most out of your Spark cluster.

To cache a dataset in Spark, you simply call the cache() method on the RDD or DataFrame. For example, if you have an RDD called myRDD, you can cache it like this:

- **myRDD.cache()**

- **Databricks uses disk caching to accelerate data reads by creating copies of remote Parquet data files in nodes' local storage using a fast intermediate data format. The data is cached automatically whenever a file has to be fetched from a remote location. Successive reads of the same data are then performed locally, which results in significantly improved reading speed. The cache works for all Parquet data files (including Delta Lake tables).**

- **Disk cache vs. Spark cache: The Databricks disk cache differs from Apache Spark caching. Databricks recommends using automatic disk caching.**

- **The following table summarizes the key differences between disk and Apache Spark caching so that you can choose the best tool for your workflow:**

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

Resource: https://docs.databricks.com/en/optimizations/disk-cache.html

- **Correct Approach:** Enabling **result caching** in Databricks is an effective way to reduce query latency and development time. When a query is run and the results are cached, subsequent executions of the same or similar queries can retrieve the results directly from memory, avoiding the need to reprocess the data. This significantly speeds up development cycles, especially when working with large datasets or complex queries.
- **Query History Utilization:** While query history is useful for tracking what queries have been run, it doesn't directly reduce query latency. Query history can be helpful in identifying frequently run queries that might benefit from caching, but it's the caching itself that reduces latency.
- **Manual Result Management:** Manually exporting and importing past query results, or copying queries from history into new notebooks, can be cumbersome and doesn't optimize performance as effectively as caching does.
- **Disabling Caching:** Disabling caching can lead to longer execution times, as each query would have to recompute results from the raw data. While this ensures data freshness, it's counterproductive when the goal is to reduce latency and speed up development. Using **query history** to identify frequently run queries and combining this with **caching** is the most effective strategy for reducing both development time and query latency in a Databricks environment.

Q.67. What is Persistence in spark?

Ans: Alternate to cache(), you can use the persist() method to cache a dataset. The persist() method allows you to specify the level of storage for the cached data, such as memory-only or disk-only storage. For example, to cache an RDD in memory only, you can use the following code:

- myRDD.persist (StorageLevel.MEMORY_ONLY)

When you cache a dataset in Spark, you should be aware that it will occupy memory on the worker nodes. If you have limited memory available, you may need to prioritize which datasets to cache based on their importance to your processing workflow.

- Persistence is a related concept to caching in Spark. When you persist a dataset, you are telling Spark to store the data on disk or in memory, or a combination of the two, so that it can be retrieved quickly the next time it is needed.

The persist() method can be used to specify the level of storage for the persisted data. The available storage levels include MEMORY_ONLY, MEMORY_ONLY_SER, MEMORY_AND_DISK, MEMORY_AND_DISK_SER, DISK_ONLY, and OFF_HEAP. The MEMORY_ONLY and MEMORY_ONLY_SER levels store the data in memory, while the MEMORY_AND_DISK and MEMORY_AND_DISK_SER levels store the data in memory and on disk. The DISK_ONLY level stores the data on disk only, while the OFF_HEAP level stores the data in off-heap memory.

To persist a dataset in Spark, you can use the persist() method on the RDD or DataFrame. For example, if you have an RDD called myRDD, you can persist it in memory using the following code:

- myRDD.persist(StorageLevel.MEMORY_ONLY)

If you want to persist the data in memory and on disk, you can use the following code:

- myRDD.persist(StorageLevel.MEMORY_AND_DISK)

When you persist a dataset in Spark, the data will be stored in the specified storage level until you explicitly remove it from memory or disk. You can remove a persisted dataset using the unpersist() method. For example, to remove the myRDD dataset from memory, you can use the following code:

- myRDD.unpersist()

Difference between cache() and persist() methods:

- Using cache() and persist() methods, Spark provides an optimization mechanism to store the intermediate computation of an RDD, DataFrame, and Dataset so they can be reused in subsequent actions(reusing the RDD, Dataframe, and Dataset computation results).
- Both caching and persisting are used to save the Spark RDD, Dataframe, and Datasets. But, the difference is, RDD cache() method default saves it to memory (MEMORY_ONLY) and, DataFrame cache() method default saves it to memory (MEMORY_AND_DISK), whereas persist() method is used to store it to the user-defined storage level.

- When you persist a dataset, each node stores its partitioned data in memory and reuses them in other actions on that dataset. And Spark's persisted data on nodes are fault-tolerant meaning if any partition of a Dataset is lost, it will automatically be recomputed using the original transformations that created it.

Caching vs. Persistence in PySpark — Which to Use and When?

Caching and persisting data in PySpark are techniques to store intermediate results, enabling faster access and efficient processing. Knowing when to use each can optimize performance, especially for iterative tasks or reuse of DataFrames within a job.

Why Cache or Persist Data?

1. Speeds Up Reuse of DataFrames : When a DataFrame is cached or persisted, it remains in memory (or specified storage) for quicker access, saving the time it takes to recompute the data.

2. Improves Iterative Operations : In machine learning or data transformations, where the same data is needed multiple times, caching/persisting minimizes redundant calculations.

3. Minimizes I/O Operations : By keeping data in memory, you avoid repeated disk I/O operations, which are costly in terms of time.

Difference Between Caching and Persistence

- Caching is a simplified form of persistence that defaults to storing data in memory (`MEMORY_ONLY`). It's often used when memory is not a constraint and you only need the data during the current Spark session.

- Persistence allows you to specify storage levels (e.g., `MEMORY_AND_DISK`, `DISK_ONLY`) and provides more control over where and how data is stored. This is useful for data too large to fit in memory.

When to Use Caching vs. Persistence

When to Cache?

- Use Caching for Quick Reuse : If you need to repeatedly access a DataFrame within the same job without changing its storage level, caching is efficient and straightforward.

When to Persist?

- Use Persistence for Storage Control : When the DataFrame is large or memory is limited, persisting allows you to specify storage levels like `MEMORY_AND_DISK`, which offloads part of the data to disk if memory is full.

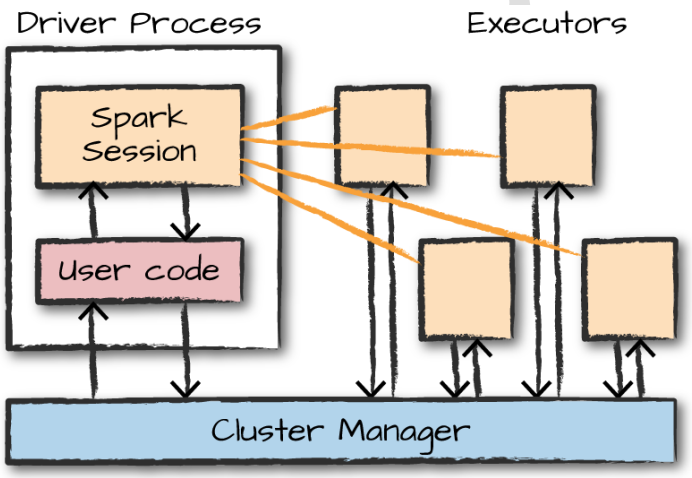
Advantages of Caching and Persistence:

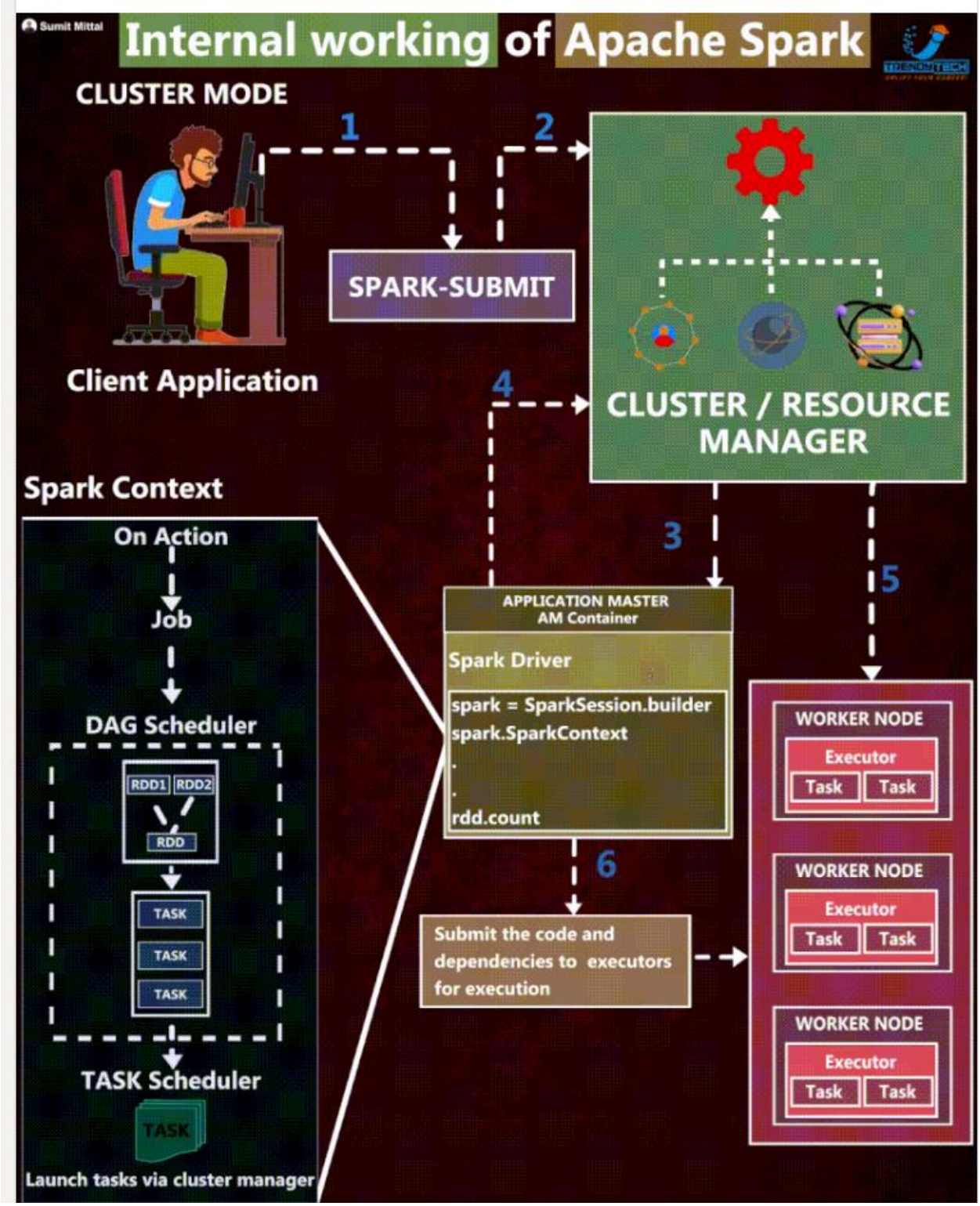
Below are the advantages of using Spark Cache and Persist methods.

- **Cost efficient –** Spark computations are very expensive; hence, reusing the computations are used to save cost.
- **Time efficient –** Reusing repeated computations saves lots of time.
- **Execution time –** Saves execution time of the job and we can perform more jobs on the same cluster.

Q.68. What do you understand by Spark Context, SQL Context and Spark Session?

Ans:





Spark context, Sql Context & Spark session:

In Apache Spark, Spark Context and SQL Context are essential components for interacting with Spark. Each has its specific role in setting up the environment, managing resources, and enabling

functionalities like querying, transforming data, and working with different APIs (RDD, DataFrame, Dataset).

Here's a detailed explanation of each:

1. Spark Context:

- **Role: SparkContext is the entry point for using the RDD (Resilient Distributed Dataset) API and the core abstraction in Spark's distributed computation. It allows for the creation of RDDs and provides methods to access Spark's capabilities, like resource allocation and job execution across the cluster.**

- **Key Functions:**

1. **Resource Allocation: When a Spark job is submitted, SparkContext connects to the cluster manager (like YARN, Mesos, or Kubernetes), which allocates resources like executors and tasks.**
2. **RDD Operations: SparkContext is responsible for creating RDDs, distributing data, and managing job execution on the distributed cluster.**
3. **Job Execution: Through SparkContext, transformations and actions applied to RDDs are scheduled and executed across the cluster.**
4. **Limitations: SparkContext primarily supports RDDs, which are low level, making it difficult to perform SQL-like operations and manipulate structured data easily.**

2. SQL Context:

- **Role: SQLContext was the original class introduced to work with structured data and to run SQL queries on Spark. It allows users to interact with Spark DataFrames and execute SQL-like queries on structured data.**

- **Key Functions:**

1. **DataFrame Creation: SQLContext allows for creating DataFrames from various data sources like JSON, CSV, Parquet, etc.**
2. **SQL Queries: Users can run SQL queries on DataFrames using SQLContext. This gives Spark SQL capabilities for querying structured and semi-structured data.**
3. **Integration with Hive: With HiveContext (a subclass of SQLContext), users could interact with Hive tables and perform more complex SQL operations.**

3. Spark Session:

- **Role:**

1. **SparkSession is the new unified entry point for using all the features of Spark, including RDD, DataFrame, and Dataset APIs. It consolidates different contexts like SparkContext, SQLContext, and HiveContext into a single, more user-friendly object.**

2. **Introduced in Spark 2.0, SparkSession simplifies the user interface by managing SparkContext internally. It is the primary point of entry to run Spark applications involving DataFrames and SQL queries.**

- **Key Features:**

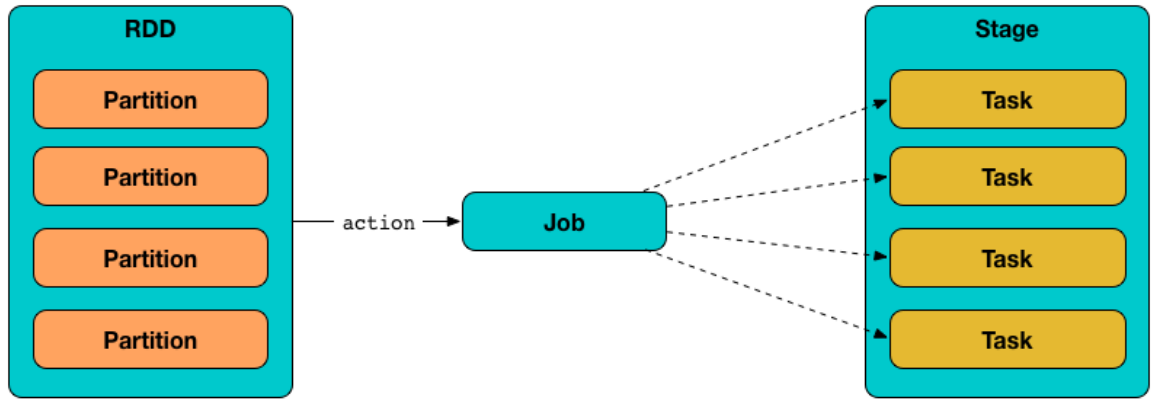
1. **Unified API: SparkSession combines the capabilities of SparkContext, SQLContext, and HiveContext, allowing users to create DataFrames, run SQL queries, and access all Spark functionalities through a single object.**
2. **DataFrame Operations: SparkSession provides an easy-to-use interface for working with DataFrames and Datasets, which are more efficient and easier to use than RDDs for structured and semi structured data.**
3. **Configuring Spark Properties: You can configure settings (like Spark configurations, execution properties, etc.) within the SparkSession object.**

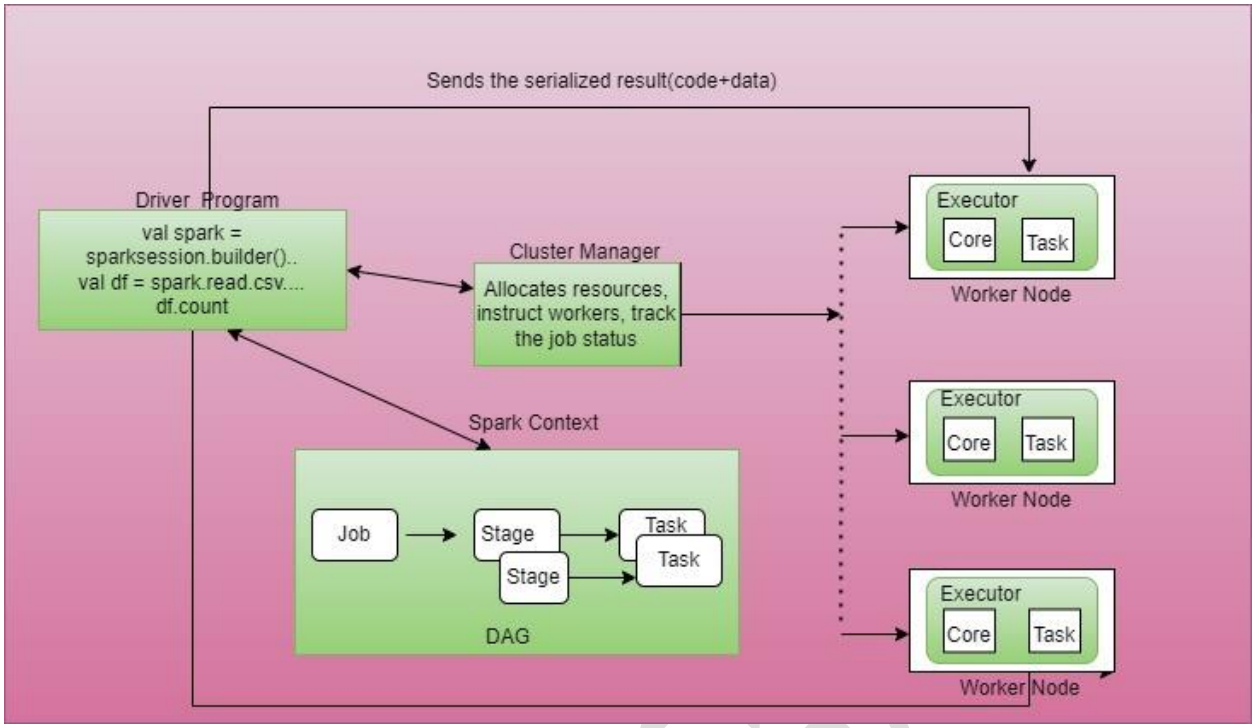
- **Advantages:**

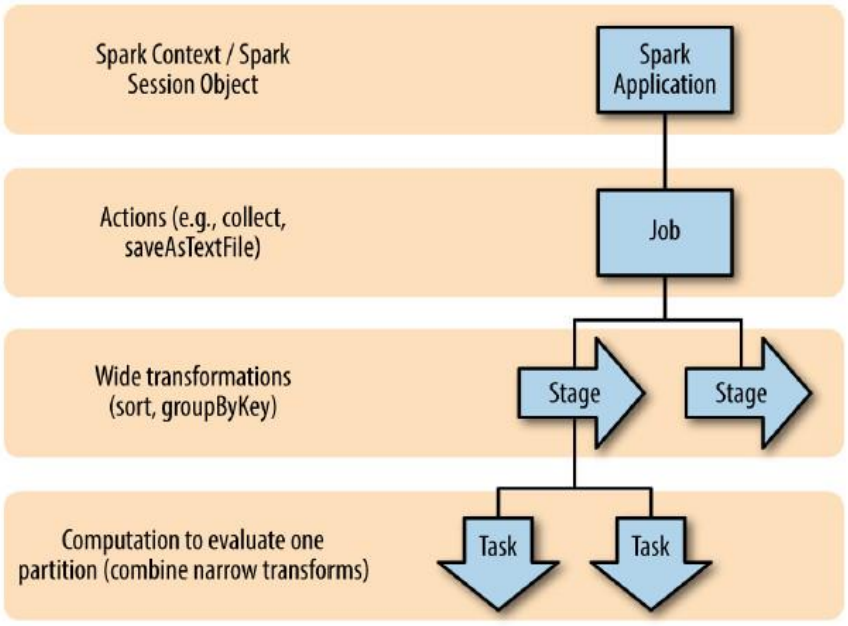
1. **With SparkSession, you no longer need to instantiate separate objects for SQLContext or HiveContext.**
2. **It simplifies the user experience and reduces the need for manually managing multiple context objects.**

Q.69. What are DAG, Jobs, Stages and Task in Spark Databricks?

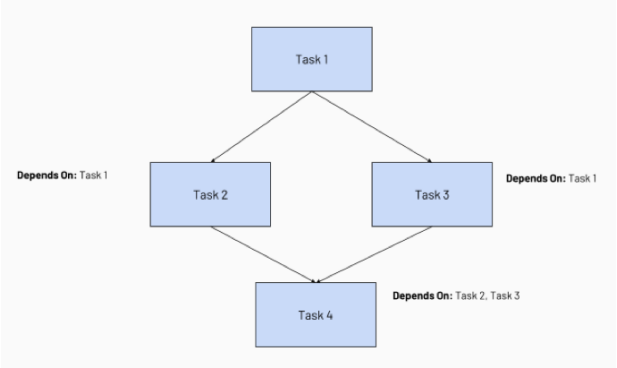
Ans:

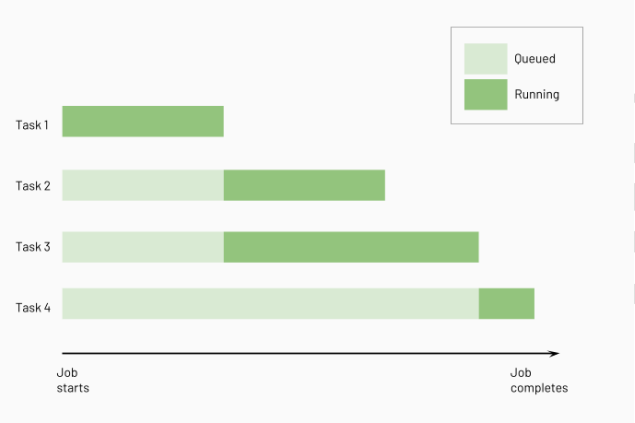




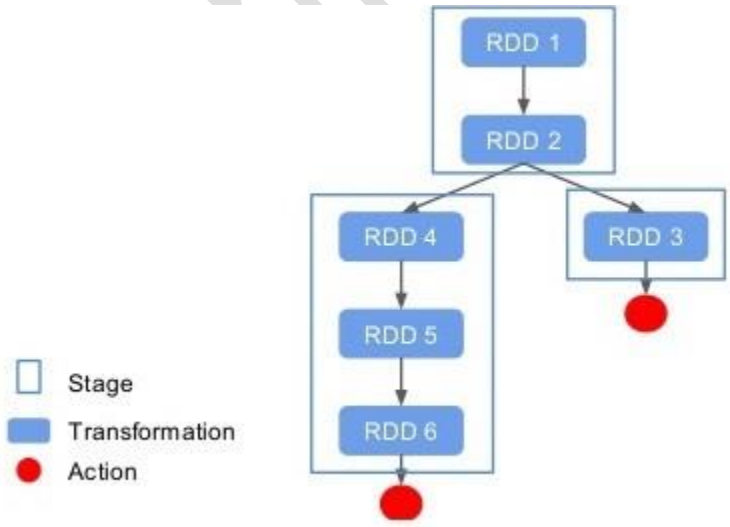


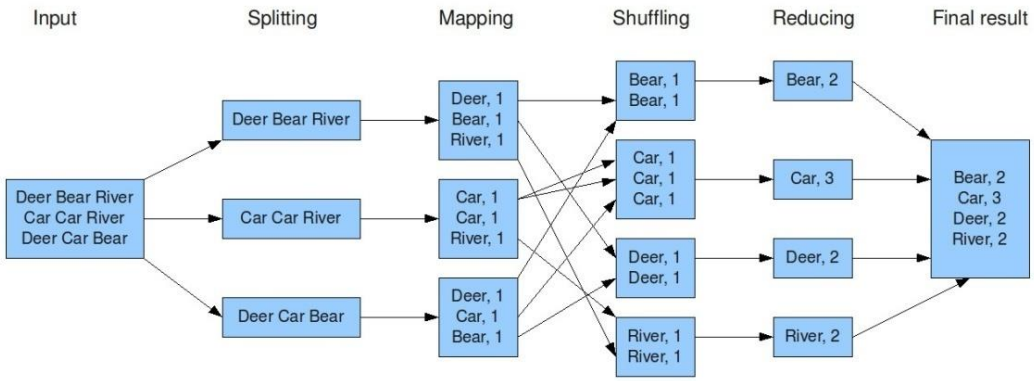
- **DAG:** Spark uses a DAG (Directed Acyclic Graph) scheduler, which schedules stages of tasks.





- Task 1 is the root task and does not depend on any other task.
- Task 2 and Task 3 depend on Task 1 completing first.
- Finally, Task 4 depends on Task 2 and Task 3 completing successfully.





Each DAG consists of stages and each stage consists of transformations applied on RDD. Each transformation generates tasks executed in parallel on each cluster nodes. Once this DAG is generated the Spark scheduler is responsible for execution of both transformation and action across the cluster.

Directed Acyclic Graph (DAG): Represents the logical execution plan, showing task dependencies. The DAG scheduler optimizes execution by breaking operations into stages and minimizing data shuffling.

- **Jobs:** A job in Spark refers to a sequence of transformations on data.

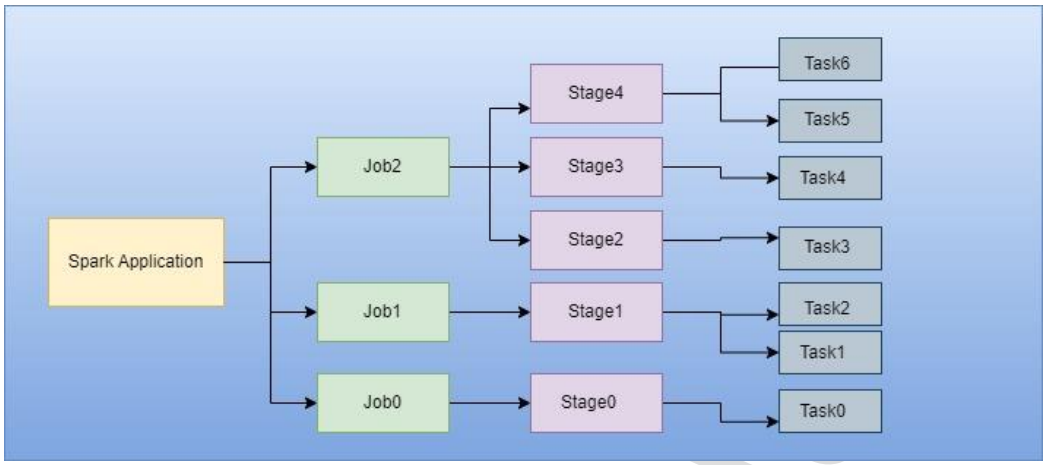
Whenever an action like count(), first(), collect(), and save() is called on RDD (Resilient Distributed Datasets), a job is created. A job could be thought of as the total work that your Spark application needs to perform, broken down into a series of steps.

Consider a scenario where you're executing a Spark program, and you call the action count() to get the number of elements. This will create a Spark job. If further in your program, you call collect(), another job will be created. So, a Spark application could have multiple jobs, depending upon the number of actions.

Each action is creating a job. Jobs in Azure Databricks are used to schedule and run automated tasks. These tasks can be notebook runs, Spark jobs, or arbitrary code executions. Jobs can be triggered on a schedule or run in response to certain events, making it easy to automate workflows and periodic data processing tasks.

- **Stages:** A stage in Spark represents a sequence of transformations that can be executed in a single pass, i.e., without any shuffling of data. When a job is divided, it is split into stages. Each stage comprises tasks, and all the tasks within a stage perform the same computation.
  - A Stage is a collection of tasks that share the same shuffle dependencies, meaning that they must exchange data with one another during execution.

- Stages are executed sequentially, with the output of one stage becoming the input to the next stage.
- **Stages are where wide transformations occur (e.g.,groupBy(), repartition(), join())**

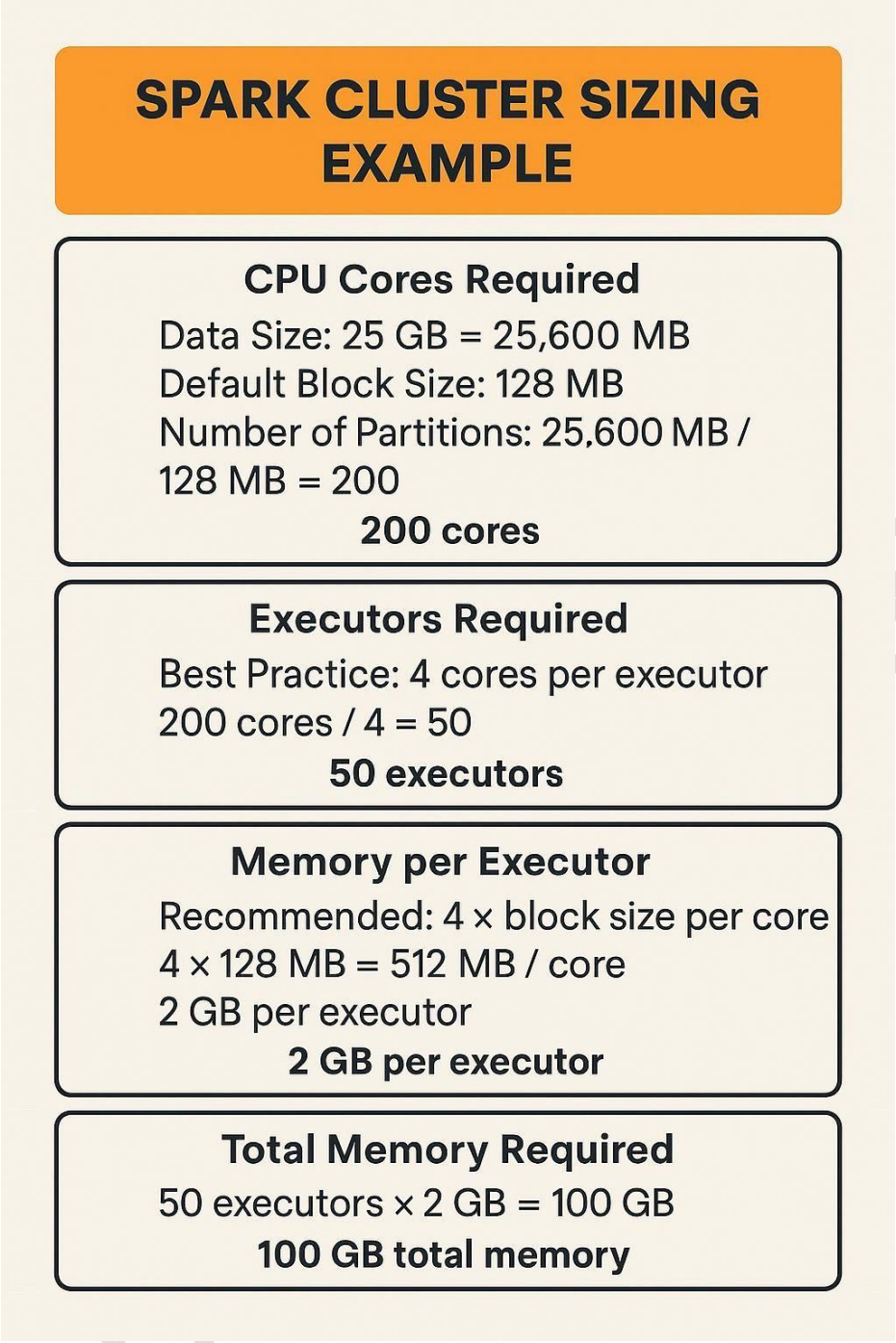


The boundary between two stages is drawn when transformations cause data shuffling across partitions. Transformations in Spark are categorized into two types: narrow and wide. Narrow transformations, like map(), filter(), and union(), can be done within a single partition. But for wide transformations like groupByKey(), reduceByKey(), or join(), data from all partitions may need to be combined, thus necessitating shuffling and marking the start of a new stage.

Each transformation is creating a stage. Two wide transformations are including three stages.

- **Task:** Each partition or core is equivalent to number of task. Tasks are where narrow transformations occur (e.g.,union(), map(), filter())

Each time there Spark needs to perform a shuffle of the data it will decide and change how many partitions the shuffle RDD will have. The default value is 200. Therefore, after using groupBy() which requires a full data shuffle, the number of tasks will have increased to 200 (as seen in your second INFO print).



It is quite common to see 200 tasks in one of your stages and more specifically at a stage which requires wide transformation. The reason for this is, wide transformations in Spark requires a shuffle. Operations like join, group by etc. are wide transform operations and they trigger a shuffle.

By default, Spark creates 200 partitions whenever there is a need for shuffle. Each partition will be processed by a task. So, you will end up with 200 tasks during execution.

==========

Question: How many Spark Jobs & Stages are involved in this activity?👇
In Databricks say you are reading a csv file from a source, doing a filter transformation, counting number of rows and then writing the result DataFrame to storage in parquet format.

🔍**Answer**:
Lets divide the question asked here in steps:

✔**Step-1** - ***Reading the File***:
The file is read as an input, and no computation occurs at this stage.

✔**Step-2**- ***Applying the Filter***:
This is a **lazy transformation** and does not execute until an action triggers the computation.

✔**Step-3** - ***Counting the Rows***:
The count() **action** triggers a Spark job to process the filter and compute the row count.

✔**Step-4** - ***Writing to Parquet***:
Writing to Parquet is another **action**, which triggers a separate job to execute the transformations and save the output.

JOBS:
Here we have 2 actions:
One for the count() action.
One for the write() action.

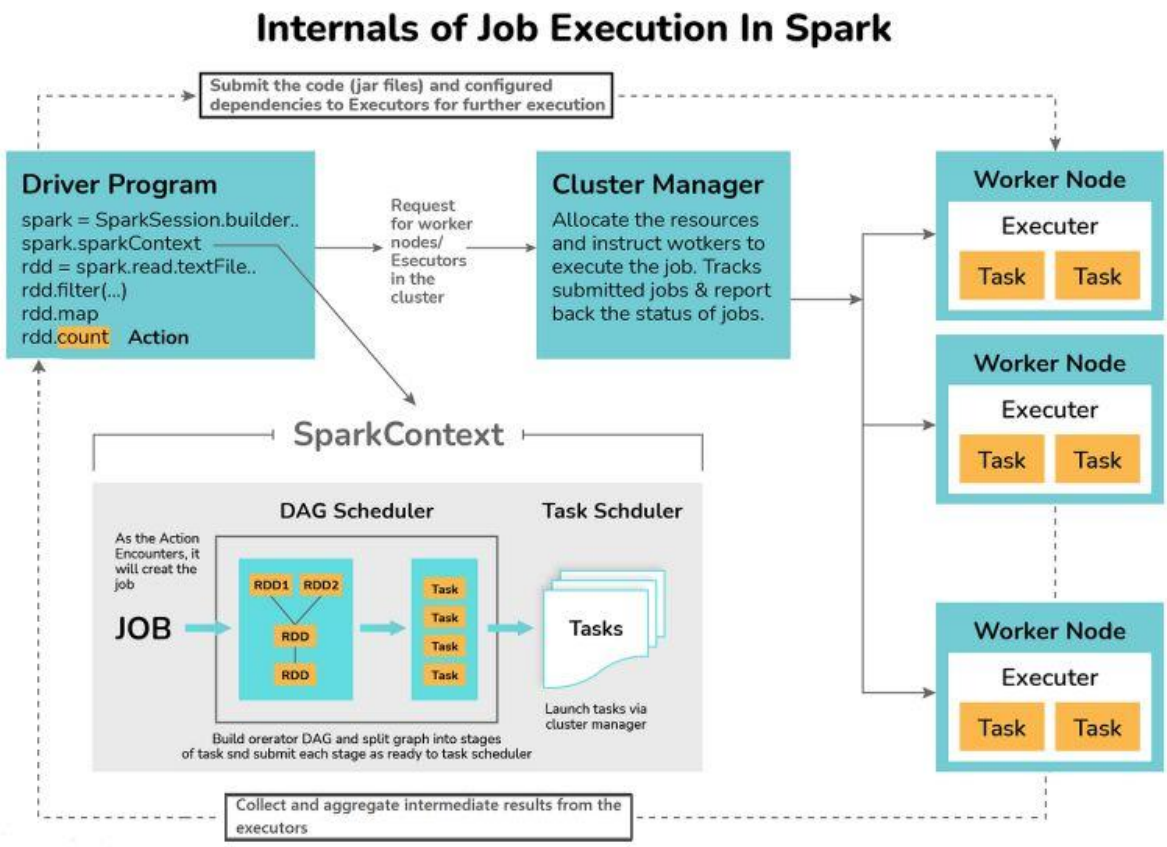
Number of actions= Number of jobs
Hence 2 Jobs.

🎯**STAGES**:
Each job consists of one or more stages, depending on whether shuffle operations are involved.
Job 1 (Count): Typically one stage if there's no shuffle.
Job 2 (Write): At least one stage for writing the output, but more stages if re-partitioning is required.

Spark Architecture:

In the context of Spark, the Driver Program is responsible for coordinating and executing jobs within the application. These are the main components in the Spark Driver Program.

1. Spark Context - Entry point to the Spark Application, connects to Cluster Manager
2. DAG Scheduler - converts Jobs → Stages
3. Task Scheduler - converts Stages → Tasks
4. Block Manager - In the driver, it handles the data shuffle by maintaining the metadata of the blocks in the cluster. In executors, it is responsible for caching, broadcasting and shuffling the data.



==========

Resources:

- https://mayankoracledba.wordpress.com/2022/10/05/apache-spark-understanding-spark-job-stages-and-tasks/
- Debugging with the Apache Spark UI: https://docs.databricks.com/en/compute/troubleshooting/debugging-spark-ui.html
- https://docs.databricks.com/en/jobs/run-if.html
- https://sparkbyexamples.com/spark/what-is-spark-stage/
- https://blog.det.life/decoding-jobs-stages-tasks-in-apache-spark-05c8b2b16114

Q.70. What are the Difference Between Number of Cores and Number of Executors?

Ans: When running a Spark job two terms can often be confusing

- **Number of Cores and Number of Executors. Let's break it down:**

- **Number of Executors**

1. **Executors are Spark's workhorses.**
2. **Each executor is a JVM instance is responsible for executing tasks**
3. **Executors handle parallelism at the cluster level.**

- **Number of Cores**

1. **Cores determine the number of tasks an executor can run in parallel.**
2. **It represents CPU power allocated to the executor.**
3. **It controls parallelism within an executor.**

- **In simple terms:**

1. **Executors = How many workers you have.**
2. **Cores = How many hands each worker has to complete tasks.**

Q.71. What are the differences between Partitioning and Bucketing in Big Data with PySpark for performance optimization?

Ans:

The common confusion among Data engineers is, ever wondered when to use partitioning and when to bucket your data? Here's a quick breakdown, with an example and PySpark code to help you differentiate between these two essential techniques for optimizing query performance in large datasets.

Partitioning:

Partitioning divides data into separate directories based on column values. It improves query performance by pruning unnecessary partitions but can lead to too many small files if not managed properly.

Use Case: Suppose you're analyzing transaction data by region. You can partition the data by the region column to limit the data scanned for region-specific queries.

PySpark Example:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("PartitioningExample").getOrCreate()


# Sample data

data = [("John", "North", 1000), ("Doe", "South", 1500), ("Jane", "East", 1200)]

columns = ["name", "region", "sales"]

df = spark.createDataFrame(data, columns)

# Write partitioned data

df.write.partitionBy("region").parquet("partitioned_data")
```

Querying for region='North' scans only the relevant partition.

Bucketing:

Bucketing divides data into fixed buckets based on a hash function applied to column values. Unlike partitioning, bucketing doesn't create physical directories; it stores data in files organized logically within each partition.

Feature	Bucketing	Liquid Clustering (Z-Ordering)
Compatibility	Works with Parquet, ORC, non-Delta	Primarily for Delta Lake
Flexibility	Cross-platform, vendor-neutral	Best in Databricks environments
Advanced Features	Query-level optimization	Seamless with Delta optimizations

Use Case: For evenly distributing data, such as customer IDs in a transaction table, bucketing ensures better load balancing and reduces skewness during joins.

PySpark Example:

```
# Enable bucketing
```

df.write.bucketBy(4, "region").sortBy("sales").saveAsTable("bucketed_data")

 # This creates 4 buckets for the `region` column

Note: Joins and aggregations on the region column will now benefit from bucketing, leading to faster execution.

Key Difference

Partitioning: Organizes data into physical directories; great for pruning irrelevant data.

Bucketing: Groups data into logical buckets; ideal for improving join performance.

Both techniques are powerful tools for optimizing query execution in large-scale data pipelines. Combine them strategically based on your data distribution and use case!

Q.72. : List of transformations and actions used in Apache Spark DataFrames for a Data Engineering role.

Ans: List of transformations and actions used in Apache Spark DataFrames for a Data Engineering role:

Transformations:

Transformations are operations on DataFrames that return a new DataFrame. They are lazily evaluated, meaning they do not execute immediately but build a logical plan that is executed when an action is performed.

Transformations: Operations like map, filter, join, and groupBy that create new RDDs. They are lazily evaluated, defining the computation plan.

1. Basic Transformations:

select(): Select specific columns.

filter(): Filter rows based on a condition.

withColumn():Add or replace a column.

drop(): Remove columns.

where(**condition**): Equivalent to filter(condition).

drop(***cols**): Returns a new DataFrame with columns dropped.

distinct():Remove duplicate rows.

sort(): Sort the DataFrame by columns.

orderBy(): Order the DataFrame by columns.

2. Aggregation and Grouping:

groupBy(): Group rows by column values.

agg(): Aggregate data using functions.

count(): Count rows.

sum(***cols**):Computes the sum for each numeric column.

avg(***cols**): Computes the average for each numeric column.

min(***cols**):Computes the minimum value for each column.

max(***cols**): Computes the maximum value for each column.

3. Joining DataFrames:

join(**other**, **on**=**None**, **how**=**None**): Joins with another DataFrame using the given join expression.

union(): Combine two DataFrames with the same schema.

intersect(): Return common rows between DataFrames.

4. Advanced Transformations:

withColumnRenamed(): Rename a column.

dropDuplicates(): Drop duplicate rows based on columns.

sample(): Sample a fraction of rows.

limit(): Limit the number of rows.

5. Window Functions:

over(**windowSpec**): Defines a window specification for window functions.

row_number().**over**(**windowSpec**): Assigns a row number starting at 1 within a window partition.

rank().over(windowSpec): Provides the rank of rows within a window partition.

Actions:

Actions trigger the execution of the transformations and return a result to the driver program or write data to an external storage system.

Actions: Operations like count, collect, save, and reduce that trigger computation and return results to the driver or write them to an output.

1. Basic Actions:

show(): Display the top rows of the DataFrame.

collect(): Return all rows as an array.

count(): Count the number of rows.

take(): Return the first N rows as an array.

first(): Return the first row.

head(): Return the first N rows.

2. Writing Data:

write(): Write the DataFrame to external storage.

write.mode(): Specify save mode (e.g., overwrite, append).

save(): Save the DataFrame to a specified path.

toJSON(): Convert the DataFrame to a JSON dataset.

3. Other Actions:

foreach(): Apply a function to each row.

foreachPartition(): Apply a function to each partition.

Q.73. What is Data Skew in Azure Databricks, how to solve it?

Ans: **Solving Data Skew in Azure Databricks**: **Causes**, **Detection**, **and Fixes** ?

Handling data skew is one of the most common challenges faced by data engineers when working with distributed computing systems like **Databricks**. Skew can severely degrade performance, leading to longer job runtimes, increased costs, and potential failures.

🔍 **What is Data Skew**?

Data skew occurs when some partitions in a dataset are significantly larger than others, causing uneven workload distribution across worker nodes. The result? A few nodes get overwhelmed while others remain idle—leading to inefficient processing.

⚠️ **Common Causes of Data Skew**:

1️⃣ **Imbalanced Keys**: When specific keys appear much more frequently in the dataset.

2️⃣ **Large Join Imbalances**: When one side of a join is heavily skewed.

3️⃣ **Wide Transformations**: Operations like groupByKey or reduceByKey aggregate data by keys, leading to uneven partition sizes.

🛠️ **How to Detect Skew in Databricks**:

1️⃣ **Task Metrics**: Use the Spark UI to identify stages where some tasks take significantly longer.

2️⃣ **Partition Size**: Monitor data partition sizes and look for disproportionate partitioning.

3️⃣ **Job Metrics**: Set up monitoring tools like Azure Monitor or Ganglia to identify performance bottlenecks.

💡 **Fixes for Data Skew**:

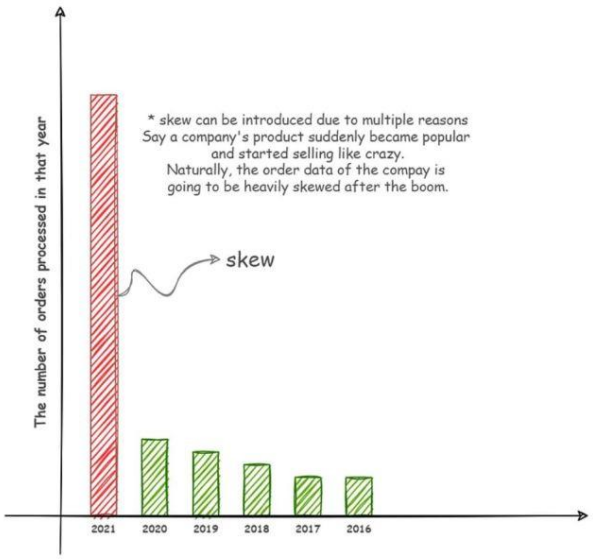
✅ **Salting Keys**: Append a random “salt” value to the keys during transformations (e.g., key_1, key_2) to spread data evenly across partitions.

✅ **Broadcast Joins**: For highly skewed joins, broadcast the smaller dataset to all nodes.

✅ **Custom Partitioning**: Define a custom partitioner to redistribute data more evenly.

✅ **Reduce Partition Size**: Avoid default partition numbers—optimize it based on data size.

✅ **Optimize Data Layout**: Use file formats like Delta Lake with features like Z-order clustering to improve query performance.



An example of a data skew

🧐 **Key Takeaway**

In distributed computing, addressing data skew is essential for optimizing job performance, reducing costs, and ensuring the reliability of your workloads. As a data engineer, mastering these techniques will set you apart in solving real-world scalability challenges.

Q.74. Do you know the difference between the coalesce() method and repartition() in Spark?

Ans:

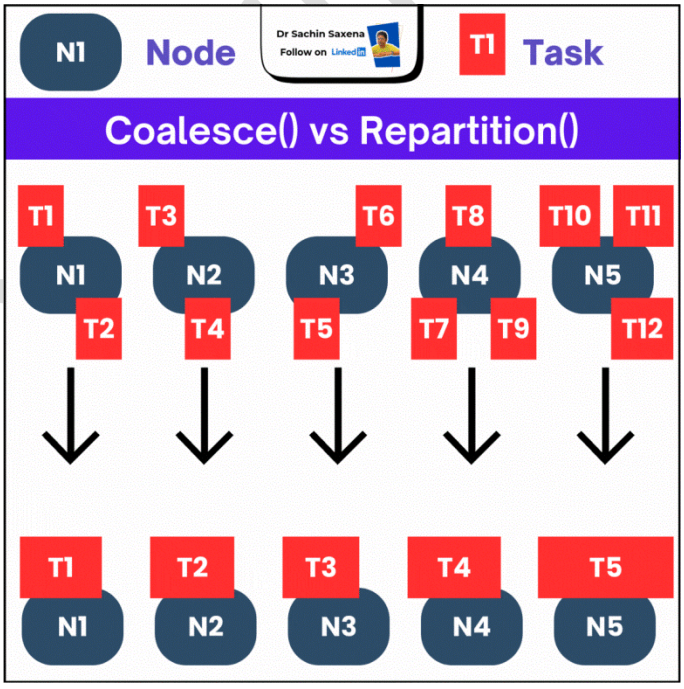
☑ The coalesce() method takes the target number of partitions and combines the local partitions available on the same worker node to achieve the target.
☑ For example, let's assume you have a five-node cluster (in figure). And you have a dataframe of 12 partitions.
✅ Those 12 partitions are spread on these ten worker nodes.

✅ Now you want to reduce the number of partitions to 5.
✅ So you executed coalesce(5) on your dataframe.
✅ So Spark will try to collapse the local partitions and reduce your partition count to 5.
✅ The final state might look like this.

🔰 You must learn the following things about the coalesce.
1️⃣ Coalesce doesn't cause a shuffle/sort.
2️⃣ It will combine local partitions only.
3️⃣ So you should use coalesce when you want to reduce the number of partitions.
4️⃣ If you try to increase the number of partitions using coalesce, it will do nothing.
5️⃣ You must use repartition() to increasing the number of partitions.
6️⃣ Coalesce can cause skewed partitions.
7️⃣ So try to avoid drastically decreasing the number of partitions.

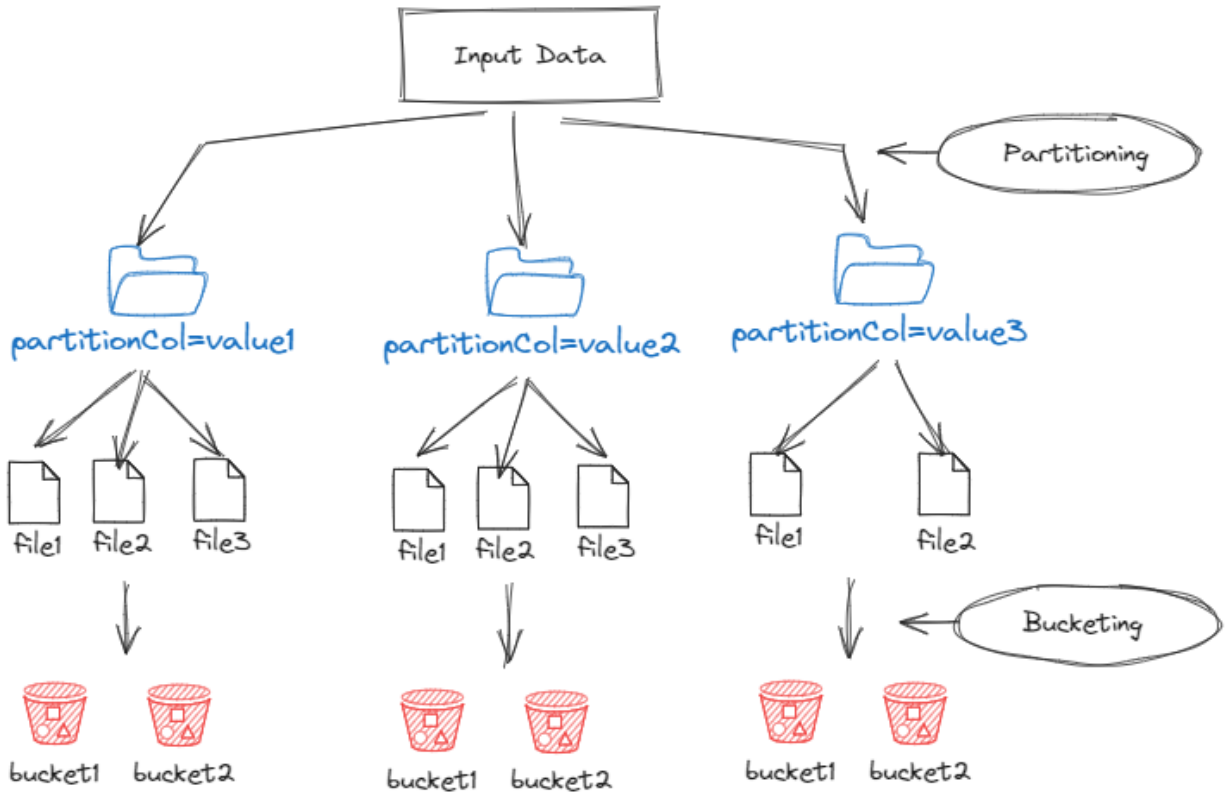
❌ You should also avoid using repartition to reduce the number of partitions.

❓ Why?
🔴 Because you can reduce your partitions using coalesce without doing a shuffle?
✅ You can also reduce your partition count using repartition(), but it will cost you a shuffle operation.



Q.75. What is Partitioning vs. Bucketing in Big Data with PySpark?

Ans: Understanding Partitioning vs. Bucketing in Big Data with PySpark for performance optimization



The common confusion among Data engineers is, ever wondered when to use partitioning and when to bucket your data? Here's a quick breakdown, with an example and PySpark code to help you differentiate between these two essential techniques for optimizing query performance in large datasets.

Partitioning:

Partitioning divides data into separate directories based on column values. It improves query performance by pruning unnecessary partitions but can lead to too many small files if not managed properly.

Partitioning (creates folders): When cardinality or category values (eg, Country, City, Transport Mode) are low and not used when cardinality or category values (eg, EmpID, Aadhaar Card Numbers, Contact Numbers, Date) are high.

- eg: low-cardinality ORDER_STATUS

- benefit: partition pruning (filtering) by skipping some partitions

Use Case: Suppose you're analyzing transaction data by region. You can partition the data by the region column to limit the data scanned for region-specific queries.

PySpark Example:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("PartitioningExample").getOrCreate()

# Sample data
```

```
data = [("John", "North", 1000), ("Doe", "South", 1500), ("Jane", "East", 1200)]
columns = ["name", "region", "sales"]

df = spark.createDataFrame(data, columns)

# Write partitioned data
df.write.partitionBy("region").parquet("partitioned_data")
```

Querying for region='North' scans only the relevant partition.

Bucketing (creates Files): Used when cardinality or category values (eg, EmpID, Aadhaar Card Numbers, Contact Numbers, Date) are high and not used when cardinality or category values (eg, Country, City, Transport Mode) are low.

Bucketing divides data into fixed buckets based on a hash function applied to column values. Unlike partitioning, bucketing doesn't create physical directories; it stores data in files organized logically within each partition.

bucketing

- eg: **high-cardinality** ORDER_id

- benefit: join optimization of 2 big DFs

- number of buckets fixed, a deterministic **HASH** function sends records with same keys to same bucket

Use Case: For evenly distributing data, such as customer IDs in a transaction table, bucketing ensures better load balancing and reduces skewness during joins.

Bucketing NOT supported on delta tables due to limitations, alternative is ZORDER

PySpark Example:

```
# Enable bucketing
df.write.bucketBy(4, "region").sortBy("sales").saveAsTable("bucketed_data")

# This creates 4 buckets for the `region` column
```

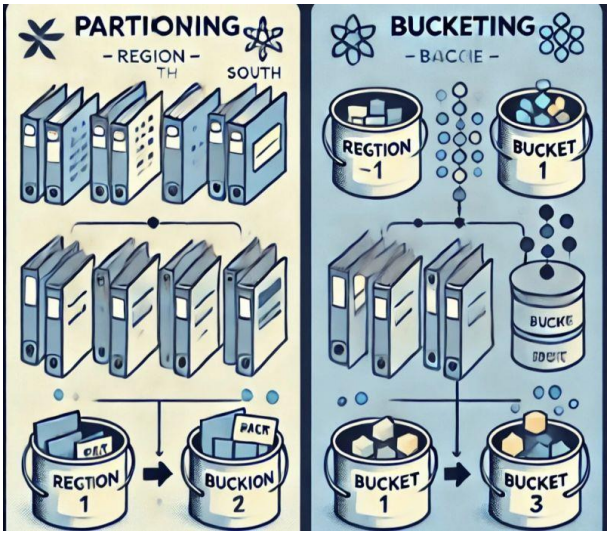
Note: Joins and aggregations on the region column will now benefit from bucketing, leading to faster execution.

Key Difference

Partitioning: Organizes data into physical directories; great for pruning irrelevant data.

Bucketing: Groups data into logical buckets; ideal for improving join performance.

Both techniques are powerful tools for optimizing query execution in large-scale data pipelines. Combine them strategically based on your data distribution and use case!



Q.76. What is z-ordering in Databricks?

Ans: → If you're working with **large datasets in Delta Lake**, optimizing query performance is crucial. Here's how **Z-Ordering** can help:

What is Z-Ordering?

Z-Ordering reorganizes your data files by clustering similar data together based on columns frequently used in queries. It leverages a **space-filling curve** (like the Z-order curve) to store related data closer on disk, reducing unnecessary file scans.

How It Works:

Imagine you frequently query sales data in a retail analytics application:

- Query by store:

```sql
SELECT * FROM sales_transactions WHERE store_id = 'S1';
```

- Query by store and product:

```sql
SELECT * FROM sales_transactions WHERE store_id = 'S1' AND product_id = 'P1';
```

```

Without optimization, these queries scan the entire table, slowing down performance.

### Enter Z-Ordering:

Optimize your table by clustering the data with Z-Ordering:

```sql
OPTIMIZE sales_transactions
ZORDER BY (store_id, product_id);
```

Why Use Z-Ordering?

1**Improved Data Skipping:** Only scan relevant files for the query.

2**Faster Query Performance:** Scan fewer files for filtered queries (e.g., `store_id = 'S1'`).

3**Resource Efficiency:** Save on compute by reducing I/O.

Real Results:

- **Before Z-Ordering:** 1000 files scanned, 2 minutes query time.
- **After Z-Ordering:** 100 files scanned, 20 seconds query time.

Best Practices:

- Use Z-Ordering for columns in **WHERE**, **JOIN**, or **GROUP BY** clauses.
- Avoid Z-Ordering high-cardinality columns like unique IDs.
- Re-run Z-Ordering periodically after major data ingestion.

Real-world Applications: From sales reporting to inventory tracking, Z-Ordering is a game-changer for retail analytics and beyond.

💡 Start using Z-Ordering today and experience faster, more efficient queries in Databricks!
```

Q.77. What are different Execution Plans in Apache Spark?

Ans: Apache Spark generates a plan of execution, how internally things will work whenever we fire a query. It is basically what steps Spark is going to take while running the query. Spark tries to optimize best for better performance and efficient resource management.

There are 4 types of plan Spark generates:

1. Unresolved Logical Plan/ Parsed Logical Plan: In this phase Spark checks the syntax of the query whether it's syntactically correct or not. Apart from syntax it doesn't check anything . If syntax is not correct it throws an error: ParseException

2. Resolved Logical Plan: In this phase Spark tries to resolve all the objects present in the query like databases, tables, views, columns etc. Spark has a metadata called catalog which stores the details about all the objects and verifies these objects when we run queries. If the objects are not present it throws an error: AnalysisException(UnresolvedRelation)

3. Optimised logical Plan: Once query passes first two phases it enters this phase. Spark tries to create optimized plan .Query goes through a set of pre configured and/or custom defined rules in this layer to optimize the query. In this phase spark combines all the projection ,works on filter ,aggregation optimization. Predicate pushdown takes place.

4. Physical Plan : This is the plan that describes how the query will be physically executed on the cluster. Catalyst optimizer will generate multiple physical plans .Each of these plans will be estimated based on execution time and resource consumption projection .One of the best cost optimized plan will be selected finally .This will be used to generate RDD code and then it runs on the machine.

Q.78. What's the Difference SortBy vs OrderBy in Apache Spark?

Ans:

🔷 **sortBy() – Faster but Partition-Based**

✅ Works at the RDD level

✅ Each partition sorts its own data (not globally sorted)

✅ Faster, as it avoids expensive shuffle operations

✅ Best for pre-sorting before transformations

🔷 **orderBy() – Globally Sorted, but Expensive**

✅ Works at the DataFrame level
✅ Sorts data across all partitions (global sort)
✅ Uses shuffle operations, making it slower but more accurate
✅ Best for final sorting before saving/reporting
💡 When to Use What?
✔ Use sortBy() for performance optimization when exact sorting isn't critical.
✔ Use orderBy() when precise global sorting is required.

Q.79. What is shallow Clone?

Ans: With Shallow Clone, you create a copy of a table by just copying the Delta transaction logs.

That means that there is no data moving during Shallow Cloning.

Running the VACUUM command on the source table may purge data files referenced in the transaction log of the clone. In this case, you will get an error when querying the clone indicating that some data files are no longer present.

In case, deep or shallow cloning, data modifications applied to the cloned version of the table will be tracked and stored separately from the source, so it will not affect the source table.

Q.80. What is the role of Databricks SQL Dashboards?

Ans: Databricks SQL Dashboards allow the analyst to display the results of multiple SQL queries in a single, cohesive view. This feature enables the creation of interactive and shareable dashboards that can combine various visualizations and query results, providing a comprehensive analysis in one place.

Q.81. When handling Personally Identifiable Information (PII) data within an organization, what step are most crucial to ensure compliance and data protection?

Ans: Handling PII data requires careful consideration to protect individuals' privacy and comply with data protection regulations such as GDPR, CCPA, or HIPAA. Key organizational considerations include:

- **Access Control:** Limiting access to PII based on roles ensures that only authorized personnel can view or manipulate sensitive data.
- **Regular Audits:** Conducting regular audits helps monitor and enforce compliance with data protection regulations.
- **Data Anonymization:** In many cases, it's crucial to anonymize or pseudonymize PII data to reduce risk, especially when sharing data outside the organization.

These practices help organizations mitigate the risks associated with handling sensitive data and ensure compliance with legal and regulatory requirements.

Q.82. What do you mean by **Last-mile ETL?**

Ans: The term **"Last-mile ETL"** refers to the final stage of processing that occurs just before analysis, after the core ETL (Extract, Transform, Load) processes are completed. It involves performing additional transformations and cleaning that are specific to the analysis at hand, ensuring the data is fully prepared for final use.

Q.83. What is **Data enhancement** ?

Ans: **Data enhancement** refers to the process of enriching a dataset by adding external or supplementary information that provides more context or detail. In this case, adding customer demographic information such as age, income level, and location would allow the marketing team to create more targeted campaigns, which is a prime example of how data enhancement can provide valuable insights.
Other options like removing duplicates or splitting columns improve data quality and structure but don't involve adding new, enriching information. Converting currency is helpful for reporting but does not enhance the data in the same way that adding demographic or behavioral information does.

Eg: A data analyst is working with a customer transaction dataset that includes basic information such as customer ID, purchase amount, and transaction date. The marketing team wants to run more targeted campaigns based on customer demographics and purchasing behavior patterns. In which of the following scenarios would **data enhancement** be most beneficial?

Correct Ans: To add additional columns with customer age, income level, and location to create more personalized marketing campaigns.

Q.84. What is Serverless Databricks SQL warehouse?

Ans: Serverless Databricks SQL warehouses are designed to start quickly, allowing users to execute SQL queries with minimal delay, making them an ideal option for quick-starting query execution environments.

Q.85. A data analyst is analyzing a dataset and wants to summarize the central tendency and spread of the data. They decide to calculate the **mean**, **median**, **standard deviation**, and **range**. Which of the following statements best compares and contrasts these key statistical measures?

Ans:

- **Mean and Median**: Both measure **central tendency**, but the mean can be influenced by outliers, whereas the median is resistant to them. The median gives the middle value when the data is ordered.
- **Standard Deviation**: This measures the **spread** of the data around the mean, indicating how much the values deviate from the average.
- **Range**: The range represents the difference between the **highest** and **lowest** values in the dataset, giving a sense of the overall spread of the data but without detailing how the data is distributed in between.

Q.86. What is Data Blending?

Ans: Take a scenario: A data analyst is tasked with combining data from two different source applications: a customer relationship management (CRM) system and an e-commerce platform. The goal is to create a unified view of customer transactions, where the CRM

provides customer details and the e-commerce platform provides purchase history. What is the term used to describe this process of combining and integrating data from these two sources?

Data blending is correct answer.

Data blending refers to the process of combining data from multiple sources, like a CRM and an e-commerce platform, to create a unified view. In this scenario, data from the CRM (customer details) is blended with data from the e-commerce platform (purchase history) to create a comprehensive dataset that provides more insights than either dataset alone.

- **Data deduplication** involves removing duplicate records.
- **Data migration** refers to moving data from one system to another.
- **Data partitioning** involves splitting data for storage or performance optimization.
- **Data archiving** is the process of storing data for long-term retention.
Data blending allows analysts to bring together different datasets for a more complete analysis.

Q.87. What is Data Skipping?

Ans: Data skipping information is collected automatically when you write data into a Delta Lake table. Delta Lake takes advantage of this information (minimum and maximum values for each column) at query time to provide faster queries. You do not need to configure data skipping; the feature is activated whenever applicable. However, its effectiveness depends on the layout of your data. For best results, apply Z-Ordering.

Q.43. What is Liquid clustering?

Ans: Liquid clustering improves the existing partitioning and ZORDER techniques by simplifying data layout decisions in order to optimize query performance. Liquid clustering provides flexibility to redefine clustering columns without rewriting existing data, allowing data layout to evolve alongside analytic needs over time.

The following are examples of scenarios that benefit from clustering:

- Tables often filtered by high cardinality columns.
- Tables with significant skew in data distribution.
- Tables that grow quickly and require maintenance and tuning effort.
- Tables with access patterns that change over time.
- Tables where a typical partition column could leave the table with too many or too few partitions.

Q.41.

Ans:

Q.42.

Ans:

Q.43.

Ans:

Q.41.

Ans:

Q.42.

Ans:

Q.43.

Ans:

3How Does Spark Achieve Fault Tolerance?

→ Dive into lineage, RDD immutability, and data recovery.

💾 Data Processing & Optimization

6️⃣ What Role Does Spark SQL Play in Data Processing?

→ Discuss structured data handling and SQL querying capabilities.

7️⃣ How Does Spark Manage Memory Efficiently?

→ Talk about unified memory management and partitioning.

🔄 Advanced PySpark Concepts

11 How Do You Handle Schema Evolution in PySpark?

→ Discuss evolving schemas in structured streaming and data ingestion.

12 Explain Window Functions in PySpark.

→ Describe how to perform operations like ranking and aggregation over partitions.

14 Which Version Control Tools Do You Use?

→ Highlight experience with Git, Bitbucket, or similar platforms.

15 How Do You Test Your Spark Code?

→ Mention unit testing with PyTest, mocking data, and integration testing.

⚙️ Performance & Optimization

16 What Is Shuffling, and How Can You Minimize It?

→ Discuss the impact of shuffling and strategies to reduce it (partitioning, broadcasting).

17 How Does Spark Ensure Fault Tolerance? (Repeated for Emphasis)

→ Dive deeper into RDD lineage and data recovery.

18 Why Is Caching in Spark So Significant? (Repeated for Emphasis)

→ Reinforce understanding of memory optimization.

19 Explain Broadcast Variables in Detail. (Repeated for Emphasis)

→ Provide real-world use cases for broadcasting.

20 How Does Spark SQL Enhance Data Processing? (Repeated for Emphasis)

→ Discuss schema inference and performance tuning.

22.