**NOTES :**

**https://luminous-manuscript-743.notion.site/Databricks-Certification-Preparation-Associate-d69bb92ce74c48d8b3e84ec46cf0b9ee**

**Databricks Certified Data Engineer Associate – Preparation**

If you save your file in DBFS, it will be stored in underlying cloud storage. So even after termination of cluster, you will be able to retrieve that file.

If you create a databricks workspace , it will create 3 additional resources :

1. Storage account. : underlying storage for DBFS.
2. Network security group.
3. Virtual network.

**Compute :**

Single node cluster : as the name suggests, there will be only 1 node that will act as driver. Your spark job will be executed on driver node.

Policy : Unrestricted : to fully configure the cluster.

Access Mode : Single user and Shared(Python and SQL only)

DBU : databricks runtime unit, unit of processing capability per hour. (More nodes will increase the DBU)

You can edit the cluster, changing a cluster configuration requires restarts.

Event logs in Cluster : Tell us about when the cluster started or terminated or edited etc.

Driver Logs : Tell us about logs generated with in cluster notebook and libraries.

**Notebook :**

%run , it will run the provided notebook and variables of that notebook, you will be able to use in calling notebook after the successful run in next cells.

dbutils.help()

dbutils.fs.help()

Note : by using % or magic command , you will not be able to store the output in variable but by using dbutils , you can store the output in variable.

You can restore the version of databricks notebook as well.

**Repos :**

To connect your databricks notebook with GIT, you need to create token on Github.

**STEPS ON GITHUB:**

1. Go to github.
2. Click on your avatar then “setting” then “developer settings”.
3. Then “personal Access Token” then Token “Classic”.
4. Click “generate new token” then “ classic” one.
5. Choose notes, expiry date and “purpose” as repo.
6. Click “generate token”.

**STEPS ON DATABRICKS:**

1. Go to user settings.
2. Go to git integration tab.
3. Choose provider as “github” and provide username and newly generated token.

Now, create a new repository on GitHub, copy the URL and move to databricks workspace. Click on “Repo” tab and click on “add repo”. Paste the URL and submit it.

**Delta Lake :**

Open-source storage framework that brings reliability to data lakes.

Delta lake is a component that deployed on the clusters as a part of databricks runtime.

When we store the data in delta table, it creates parquet files a transaction log file in format of JSON.

Delta Log : Ordered records of every transaction performed on Delta Table since its creation. It serve as single source of truth, whenever we query on this table, spark checks the transaction logs to get the latest version of the table.

Each committed transaction is stored in JSON file that includes what type of operation it performed , predictive filters and files that are affected from this transaction.

**Scenario and Delta Processing :**

1. Writing process : Lets assume we are writing 2 files in delta lake . file 1 and file2. It will write to the storage and a transaction log 000.json will be created.
2. Reading process : when we read this delta table, process will read the delta logs first, then get the information about file 1 and file 2 . then read it.
3. Write with Modification : lets assume we modified the records that is in file1. Delta process will not alter file 1 , instead of that it will create file3 with updated data and also create a new transaction log file 001.json with updated details of file2 and file3.
4. Read Again : Now for reading purposes, the process will read the latest transaction log file which is 001.json and get the information about file 2 and file3.
5. Failure : Lets assume, during the write process of file 4, we get an error and the process failed. In that case your file 4 will be at storage but your transaction log file will not be generated and hence for next reading process, spark will not read this corrupt file.
6. Read and write at same time : if you write file 5 and read the delta table at same time, delta table will not read file 5 as its transaction log 002.json is not completed yet. Read process will read the latest transaction log which is 001.json in that case.

Json files contains information related to adding or removing of the files related to delta table.-

Meta Data : Describe detail table\_name;

History : Describe History Table\_Name;

**Advance Feature Of Delta Table :**

**Time Travel :**

2 Ways:

1. Select \* from delta\_table timestamp as of “2023-01-01”;
2. Select \* from delta\_table version as of version\_number(like 8);

Select \* from delta\_table@v8 ;

You can also restore the current version to older versions using **RESTORE** command:

1. RESTORE table delta\_table to TO TIMESTAMP AS OF “2023-01-01”;
2. RESTORE table delta\_table to TO VERSION AS OF 8;

**OPTIMIZE** : compacting small files into larger one, increase the optimization during read process.

OPTIMIZE delta\_table;

**ZORDER** : co-locate column information.

**VACCUM** : Its kind of garbage collection for Delta Tables.

VACCUM delta\_table [retention period];

Default retention period is 7 days; if you want to remove files in delta\_table less that 7 days retention period, you need to change the setting “SET spark.databricks.delta.retentionDuretionCheck.enabled = False”;

VACCUM delta\_table RETAIN 172 HOURS;

DROP TABLE delta\_table;

**Databricks Managed and External Tables :**

A diagram of a diagram

Description automatically generated

If you are creating a table without using “LOCATION” keyword, it will be created as managed table.

You can also create databases by using CREATE DATABASE or CREATE SCHEMA command.

By default , your database location will be “dbfs:/user/hive/warehouse” but you can also provide the location as well while creating the database itself.

**Set Up Delta Tables :**

You can create a delta table by using “CTAS” as well.

CREATE TABLE delta\_table\_1 AS SELECT \* FROM delta\_table\_2;

or

CREATE TABLE new\_table

COMMENT "Contains PII”

PARTITIONED BY (city, birth\_date)

LOCATION ‘/some/path’

AS SELECT id, name, email, birth\_date, city FROM delta\_table\_1;

**Table Constraints :**

1. **NOT NULL constraints**
2. **CHECK constraints.**

ALTER TABLE table\_name ADD CONSTRAINT constraint\_name constraint\_details.

ALTER TABLE orders ADD CONSTRAINT valid\_date CHECK (date > '2020-01-01').

**Cloning Delta Lake Tables**

DEEP CLONE

SHALLOW CLONE

**Deep Cloning**

Fully copies data + metadata from a source table to a target.

CREATE TABLE table\_clone DEEP CLONE source\_table.

1. Can sync changes.
2. Take quite a while for large datasets.

**Shallow Cloning**

Quickly create a copy of a table

Just copy the Delta transaction logs.

CREATE TABLE table\_clone SHALLOW CLONE source\_table.

**Cloning Delta Lake Tables**

Useful to set up tables for testing in development.

In either case, data modifications will not affect the source.

**VIEWS :**

Types of views

1. (Stored) Views

2. Temporary views

3. Global Temporary views

**1- (Stored) Views**

Persisted objects.

CREATE VIEW view\_name AS query.

**2- Temporary view**

Session-scoped view (only available for same spark session)

CREATE TEMP VIEW view\_name AS query.

A new Spark Session will be created when :

1. Opening a new notebook
2. Detaching and reattaching to a cluster
3. Installing a python package
4. Restarting a cluster

**3- Global Temporary views**

Cluster-scoped view

CREATE GLOBAL TEMP VIEW view\_name AS query.

SELECT \* FROM global\_temp.view\_name.

**Module 2:**

You can directly use SQL query from file using file\_format.’path\_of\_file’.

Select \* from parquet.’dbfs:/user/hive/warehouse/emp.parquet’.

**DATA STREAM :**

**Data Stream :**

Any data source that grows over time

New files land in cloud storage.

Updates to a database captured in a CDC feed.

Events queued in a pub/sub messaging feed.

**Processing Data Stream :**

2 approaches:

1. Reprocess the entire source dataset each time

2. Only process those new data added since last update

Structured Streaming

**Read Data from Input Table :**

streamDF = spark.readStream.table("Input\_Table")

**Write Data to output Table :**

streamDF.writeStream.

trigger(processingTime="2 minutes")

.outputMode("append")

.option("checkpointLocation", "/path")

.table("Output\_Table")

**Trigger Intervals**

**Trigger**

**Unspecified :** Default: processingTime="500ms"

**Fixed interval :**  trigger(processingTime=”5 minutes") : Process data in micro-batches at the user-specified intervals

**Triggered (batch) :** trigger(once=True) :Process all available data in a single batch, then stop

**Triggered(micro-batches) :** trigger(availableNow=True) : Process all available data in multiple micro-batches, then stop

**Output Modes :**

Append(Default) : outputMode("append") : Only newly appended rows are incrementally.

appended to the target table with each batch.

Complete : outputMode("complete") : The target table is overwritten with each batch

**Checkpointing**

Store stream state

Track the progress of your stream processing.

Can Not be shared between separate streams.

**Guarantees**

1. Fault Tolerance

Checkpointing + Write-ahead logs.

record the offset range of data being processed during each trigger interval.

2. Exactly-once guarantee

Idempotent sinks

**Unsupported Operations**

Some operations are not supported by streaming DataFrame.

Sorting

Deduplication

**Advanced methods**

Windowing

Watermarking

**INCREMENTAL DATA INGESTION :**

Loading new data files encountered since the last ingestion.

Reduces redundant processing.

**2 mechanisms:**

1. COPY INTO
2. Auto loader

**COPY INTO :**

SQL command

Idempotently and incrementally load new data files

Files that have already been loaded are skipped.

**COPY INTO :**

COPY INTO my\_table

FROM '/path/to/files’

FILEFORMAT = <format>

FORMAT\_OPTIONS (<format options>)

COPY\_OPTIONS (<copy options>);

**Example :**

COPY INTO my\_table

FROM '/path/to/files’

FILEFORMAT = CSV

FORMAT\_OPTIONS ('delimiter' = '|’,

'header' = 'true')

COPY\_OPTIONS ('mergeSchema' = 'true’)

**Auto loader :**

Structured Streaming

Can process billions of files.

Support near real-time ingestion of millions of files per hour.

**Auto loader Checkpointing :**

Store metadata of the discovered files

Exactly-once guarantees.

Fault tolerance

**Auto Loader in PySpark API :**

spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", <source\_format>)

.load('/path/to/files’)

.writeStream

.option("checkpointLocation", <checkpoint\_directory>)

.table(<table\_name>)

**Auto Loader + Schema :**

spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", <source\_format>)

.option("cloudFiles.schemaLocation", <schema\_directory>)

.load('/path/to/files’)

.writeStream

.option("checkpointLocation", <checkpoint\_directory>)

.option("mergeSchema", “true”)

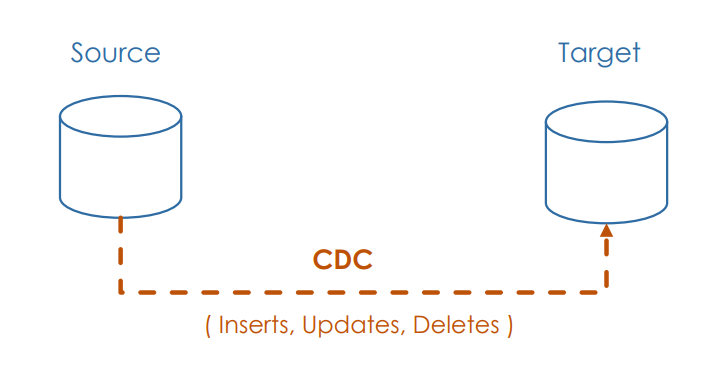
.table(<table\_name>)

PRODUCTION PIPELINES :

Delta Live Tables :

CHANGE DATA CAPTURE :

CDC: Process of identifying changes made to data in the source and delivering those changes to the target



Row-Level changes :

Inserting new records

Updating existing records

Deleting existing records

CDC with DLT :

APPLY CHANGES INTO LIVE.target\_table

FROM STREAM(LIVE.cdc\_feed\_table)

KEYS (key\_field)

APPLY AS DELETE WHEN operation\_field = "DELETE"

SEQUENCE BY sequence\_field

COLUMNS \* ;

APPLY CHANGES INTO: Pros

Orders late-arriving records using the sequencing key

Default assumption is that rows will contain inserts and updates

Can optionally apply deletes (APPLY AS DELETE WHEN condition)

Specify one or many fields as the primary key for a table

Specify columns to ignore with the EXCEPT keyword

Support applying changes as SCD Type 1 (default) or SCD Type 2

APPLY CHANGES INTO: Cons

Breaks the append-only requirements for streaming table sources.

Can Not perform streaming queries against the table.

**Data governance model**

Programmatically grant, deny, and revoke access to data objects.

GRANT SELECT ON TABLE my\_table TO user\_1@company.com

GRANT Privilege ON Object <object-name> TO <user or group>

**Data objects**

GRANT Privilege ON Object <object-name> TO <user or group>;

**Object**  **Scope**

CATALOG controls access to the entire data catalog.

SCHEMA controls access to a database.

TABLE controls access to a managed or external table.

VIEW controls access to SQL views.

FUNCTION controls access to a named function.

ANY FILE controls access to the underlying filesystem.

**Privileges**

GRANT Privilege ON Object <object-name> TO <user or group>;

Privilege Ability

SELECT read access to an object.

MODIFY add, delete, and modify data to or from an object.

CREATE create an object.

READ\_METADATA view an object and its metadata.

USAGE No effect! required to perform any action on a database object.

ALL PRIVILEGES gives all privileges

**Granting Privileges by Role**

Role Can grant access privileges for

Databricks administrator All objects in the catalog and the underlying filesystem.

Catalog owner All objects in the catalog.

Database owner All objects in the database.

Table owner Only the table

**More operations**

Grant

DENY

REVOKE

SHOW GRANTS

**Unity Catalog**

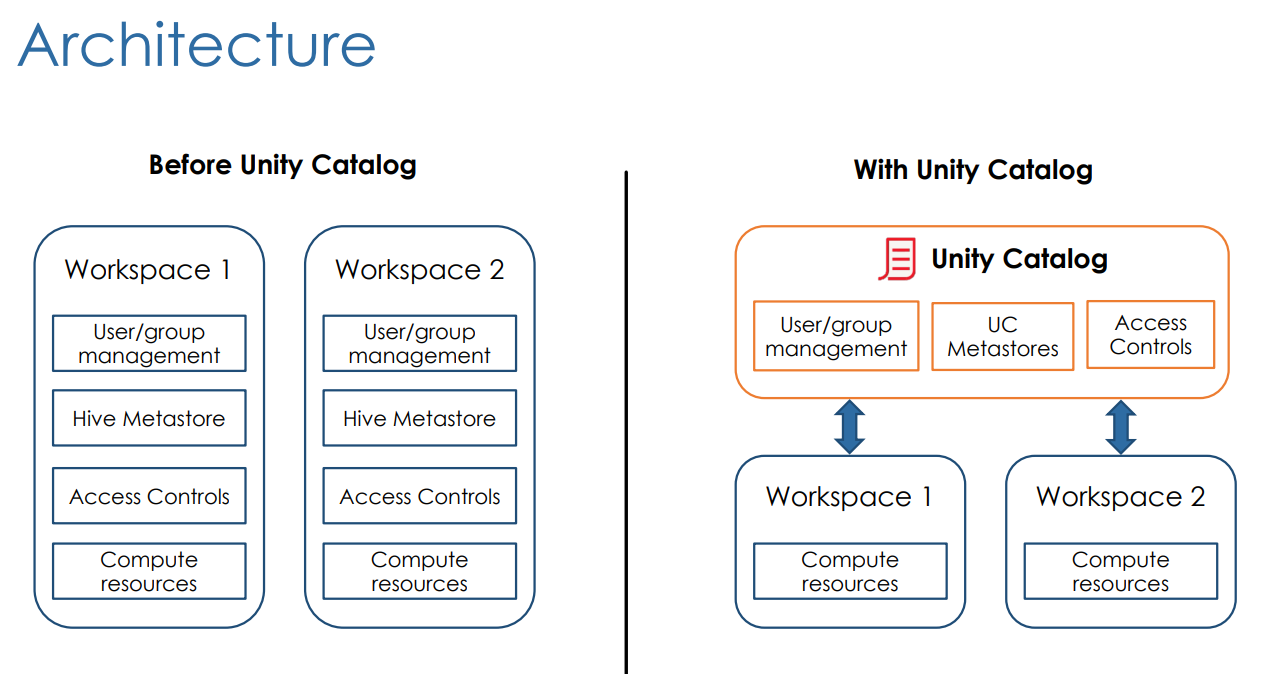
Centralized governance solution across all your workspaces on any

cloud.

Unify governance for all data and AI assets

files, tables, machine learning models and dashboards

based on SQL



UC 3-level namespace

SELECT \* FROM schema.table -----🡪 SELECT \* FROM catalog.schema.table.

A diagram of a company structure

Description automatically generated

**Identities**

Users: identified by e-mail addresses

Account administrator

Service Principles: identified by Application IDs

Service Principles with administrative privilege

Groups: grouping Users and Service Principles

Nested groups

A screenshot of a computer

Description automatically generated

Privileges

CREATE

USAGE

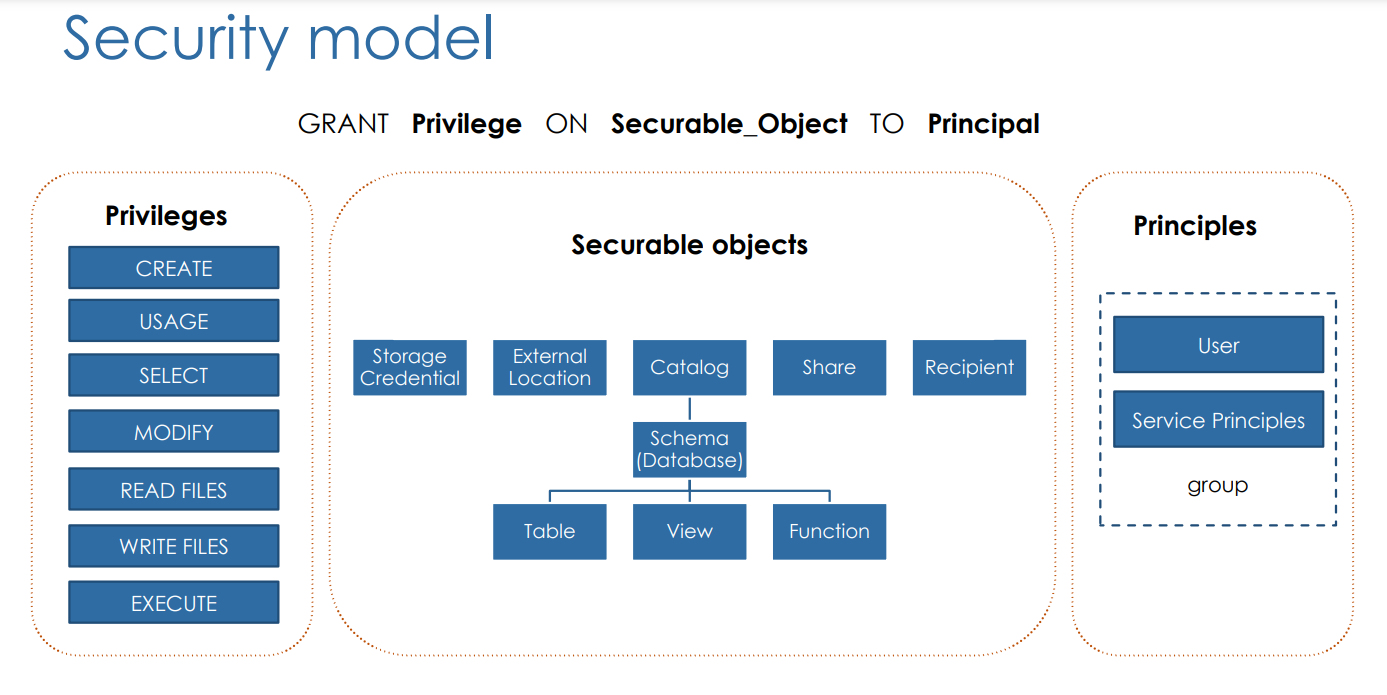
SELECT

MODIFY

READ FILES

WRITE FILES

EXECUTE



Practice Paper :

1. Auto loader supports both directory listing and file notification but COPY INTO only supports directory listing.

Auto Loader incrementally and efficiently processes new data files as they arrive in cloud storage without any additional setup. Auto Loader provides a new Structured Streaming source called cloudFiles. Given an input directory path on the cloud file storage, the cloudFiles source automatically processes new files as they arrive, with the option of also processing existing files in that directory.

1. Databricks Notebooks support real-time coauthoring on a single notebook.
2. Spark.readStream.table("sales").createOrReplaceTempView("streaming\_vw")

When you load a Delta table as a stream source and use it in a streaming query, the query processes all of the data present in the table as well as any new data that arrives after the stream is started.

You can load both paths and tables as a stream, you also have the ability to ignore deletes and changes(updates, Merge, overwrites) on the delta table.