⇒ We will see Creating Tables from External Data Sources

This helps to extracts data from diverse external systems.

⇒ These tables will not be stored in Delta Lake format. Original data will remain stored in original native format (CSV, JSON, etc.)

⇒ Registering Tables on External Data with Read Options :-

⇒ While reading data from external sources, for many formats, we need to define schema, so that data is parsed / read into correct format :-

table schema

```
CREATE TABLE <table_name> (col1, col2, ...)
USING <data_source>
OPTIONS (key1 = val1, key2 = val2, ....)
LOCATION = path
```

format of the data files

Keys should be without quotes

values should be inside quotes

Ex:-

```
CREATE TABLE IF NOT EXISTS sales_tbl
(
  order_id LONG,
  email STRING,
  qty INTEGER,
  Revenue DOUBLE
)
USING CSV
OPTIONS (
  header = "true",
  delimiter = "|"
)
LOCATION "${DA.paths.sales_csv}"
```

schema (column names & datatypes)

CSV → data source

header = "true" → 1st row should not be taken as data row

delimiter = "|" → symbol that seperates data values in the file

LOCATION → unmanaged location of the csv file.

This is actually the path where table is created & the data files are stored but in this case, as this path is unmanaged and already contains the data files, that will actually serve as data location backing the table.

Corresponding pyspark python code :-

```
spark.sql(f"""
  CREATE TABLE IF NOT EXISTS sales_tbl
  (
    order_id LONG,
    email STRING,
    qty INTEGER,
    Revenue DOUBLE
  )
  USING CSV
  OPTIONS (
    header = "true",
    delimiter = "|"
  )
  LOCATION "{DA.paths.sales_csv}"
""")
```

CSV → data source

header = "true" → 1st row should not be taken as data row

delimiter = "|" → symbol that seperates data values in the file

≥ Above query did not move any data (as file). The created table actually points to the data files in the external unmanaged location.

This can now be queried as normal table :-

```
SELECT * FROM sales_tbl;
```

this can be thought of as a pointer, that has its schema stored in Databricks managed metastore, by data files stored in actual external location.

≥ The schema related details stored in metastores ensures data is always read in the mentioned schema way.

NOTE :-

For CSV files, more data records addition must happen with same column order, and if not, the former order of columns will be applied to latest data reads to leading to corruption of data due to mismatching types.

≥ To see table metadata :-

In this case

```
DESCRIBE EXTENDED sales_tbl;
```

Table type will be External

"Storage Properties" attribute holds the options that we passed during table schema declaration.

⇒ This table is not a Delta Lake table, and thus not has the features that Delta Lake Tables provide (Ex:- Querying it always may not fetch the most recent version, and will show older cached version of data).

⇒ Spark caches external table into local storage while queried, to provide performance during subsequent queries.

To refresh the cache with the most updated version of data (in external source), we need to manually run the following command :-

```
REFRESH TABLE sales_tbl
```

This will fill the cache with the recent version of data.

Extracting Data from SQL Databases :-

SQL databases are an extremely common data source, and Databricks has a standard JDBC driver for connecting with many flavors of SQL.

The general syntax for creating these connections is:

```
CREATE TABLE
USING JDBC
OPTIONS (
    url = "jdbc:{databaseServerType}://{jdbcHostname}:{jdbcPort}",
    dbtable = "{jdbcDatabase}.table",
    user = "{jdbcUsername}",
    password = "{jdbcPassword}"
)
```

In the code sample below, we'll connect with SQLite.

NOTE: SQLite uses a local file to store a database, and doesn't require a port, username, or password.

⚠ **WARNING**: The backend-configuration of the JDBC server assume you are running this notebook on a single-node cluster. If you are running on a cluster with multiple workers, the client running in the executors will not be able to connect to the driver.

Ex:-

```
CREATE TABLE users_tbl
USING JDBC  → data source (can also be JSON, PARQUET, ...)
OPTIONS (
    url = "...."
    dbtable = "<table-name_in_database>"
)
```

In case of local file table, username & password is not needed.

⇒ The schema information of the table is automatically captured from the external system, and can be viewed as :-

```
DESCRIBE EXTENDED users_tbl
```

⇒ In this case, table will be shown as "Managed",

but no data is persisted locally.

in the place of JDBC.

Note that some SQL systems such as data warehouses will have custom drivers. Spark will interact with various external databases differently, but the two basic approaches can be summarized as either:

1. Moving the entire source table(s) to Databricks and then executing logic on the currently active cluster
2. Pushing down the query to the external SQL database and only transferring the results back to Databricks

In either case, working with very large datasets in external SQL databases can incur significant overhead because of either:

1. Network transfer latency associated with moving all data over the public internet
2. Execution of query logic in source systems not optimized for big data queries

SUMMARY :-

(1) For files stored in managed locations, in JSON, parquet, delta formats, we load them into table, by creating view, as :-

```
CREATE OR REPLACE VIEW <view_name>
AS SELECT * FROM <json>.`<location>`
```

(2) For files in unmanaged location, to load data into table, use CREATE TABLE command using data source, specifying schema and options.
This table will be called External Table, as data source is external.

Imp:-

①
```
CREATE TABLE <tbl_name>
(
  columns
)
USING <data_source>
LOCATION <path>
```
CSV / JSON / delta, etc.

The path / directory mentioned here is the path where Databricks will create the table, i.e., where Databricks will store the main data files of the table.

⇒ If this path is empty initially, then Databricks stores here the files in file format mentioned in the Data Source.

For ex:-
```
CREATE TABLE <name>
LOCATION <path>
AS
SELECT * FROM parquet.`<path_2>`
```
This path is considered to be empty having no files

Here, data files, will read from here, converted to Delta format, and will be stored here.

But, in such case:-

```
CREATE TABLE <name>
(
    columns - - -
)
USING CSV
OPTIONS (
    - - -
)
LOCATION = <path>
```

Here, this path will be used for storing the data files backing the table.

Bcz, as this directory already contains data files (say, in CSV format), Databricks will not overwrite or write any data here, rather, will only point to these data files, and will use the schema provided, while reading the data.

Thus, Data files here is unmanaged, and is only pointed by Databricks, thus, External Table.

The metadata i.e, the Schema, and the Options (here), gets stored in the Metastore, which ensures data in the external location is always read with those options

Generally,

LOCATION option is used (while creating table) to maked External Tables.

For managed table creation, specifying LOCATION is not needed.

CSV common options :-

```
USING CSV
OPTIONS (
    header =
    inferSchema =
    delimiter =
    path =
)
```

useful while creating view or temporary view, and then using the view to create managed table.