Explore temporary views and global temporary views in Databricks

1. **Explore the differences between global temp and local temp**

In **Databricks**, both **global temporary views** and **local temporary views** are used to store intermediate query results and allow users to reference them without persisting them in a database. However, they have key differences in terms of **scope, visibility, and persistence**.

Differences between Global and Local Temporary Views in Databricks

Feature	**Global Temporary View (CREATE GLOBAL TEMP VIEW)**	**Local Temporary View (CREATE TEMP VIEW)**
Scope	Available across all sessions within the same cluster	Available only within the session that created it
Visibility	Accessible by all users connected to the same cluster	Only accessible by the user who created it
Persistence	Removed when the cluster is terminated	Removed when the session ends
Database Namespace	Stored in global_temp schema	Stored in the session's default schema
Usage Example	SELECT * FROM global_temp.view_name;	SELECT * FROM temp_view_name;
Best Use Case	Sharing temporary results across notebooks and users	Storing temporary data for single-session processing

2. Create and explore temporary views and global temporary views in Databricks using SQL queries.

*** Set up a Hive database, create sample tables, and insert test data.**

*** Investigate filter conditions and Null conditions**

To create Temporary views and Global Views we need to first create a data frame

Creating a mount to ADLS storage using access key

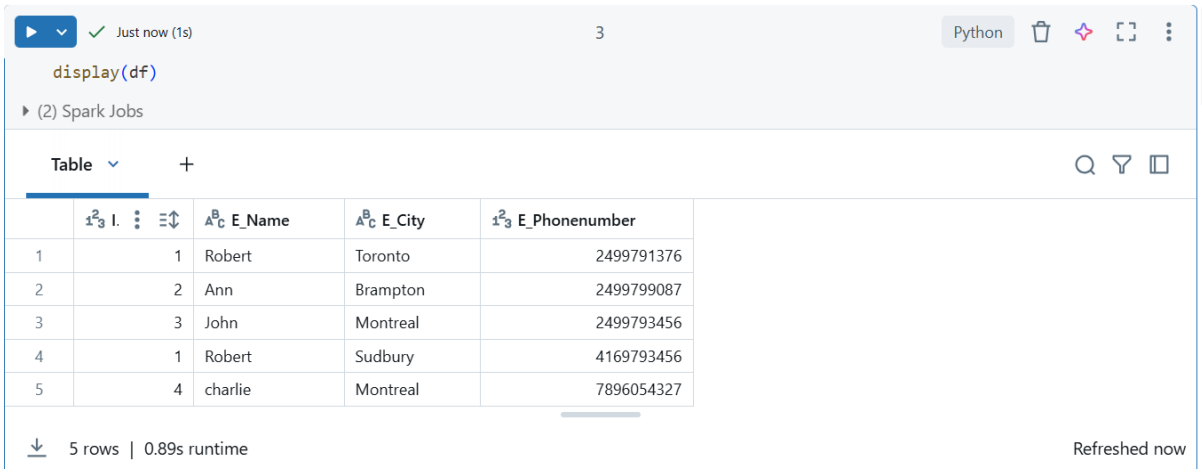
```
dbutils.fs.mount(
  source = "wasbs://container1@adlsdeepthik.blob.core.windows.net",
  mount_point = "/mnt/containers1",
```

extra_configs = {"fs.azure.account.key.adlsdeepthik.blob.core.windows.net":"aG5rHQ12QMCkYf0qfqpes8NpAXASqqY/JvHi/I3zrqUy6GObRWji+5yYna0BXCReGeQeIX09JDHU+AStdzKfrQ=="})

Now need to read path from storage account and assign it to a data frame df=spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/container1/CSV_Data/*.csv")

To display data of data frame we need to use

Display(df)



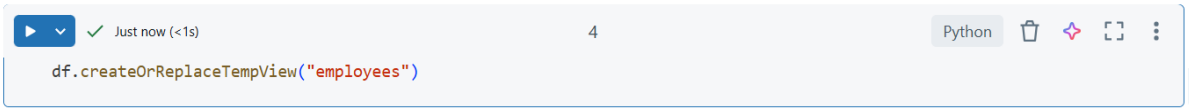
	I.	E_Name	E_City	E_Phonenumber
1	1	Robert	Toronto	2499791376
2	2	Ann	Brampton	2499799087
3	3	John	Montreal	2499793456
4	1	Robert	Sudbury	4169793456
5	4	charlie	Montreal	7896054327

As of now we are using python language in data bricks, to use different language we need to use magic commands

For SQL we need to use %sql and write SQL query

Here we are creating Temp view for this data frame, using

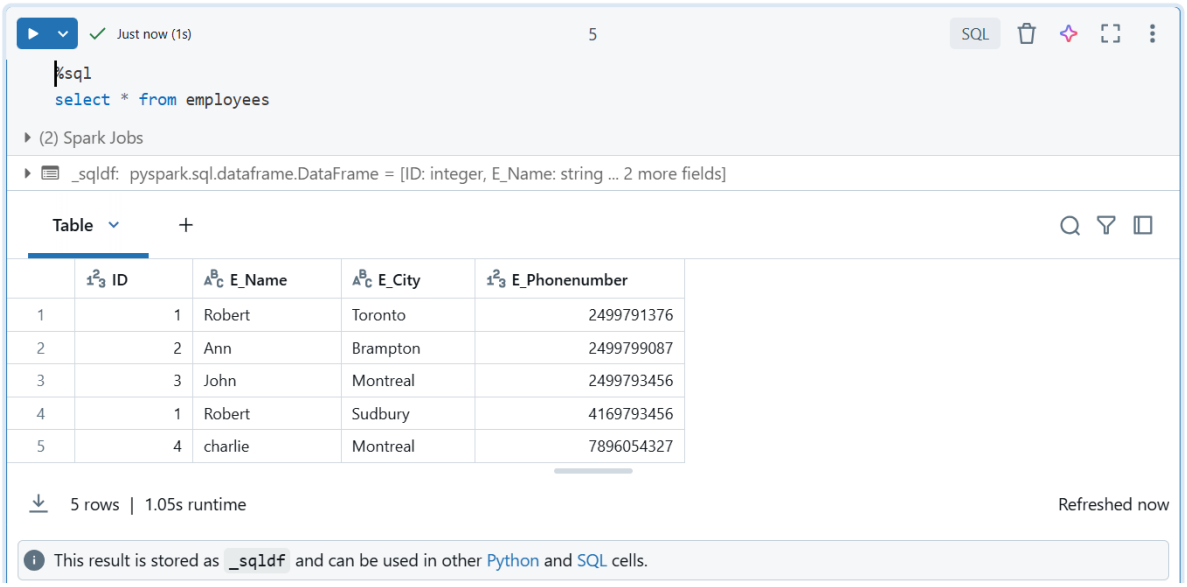
df.createOrReplaceTempView("employees")



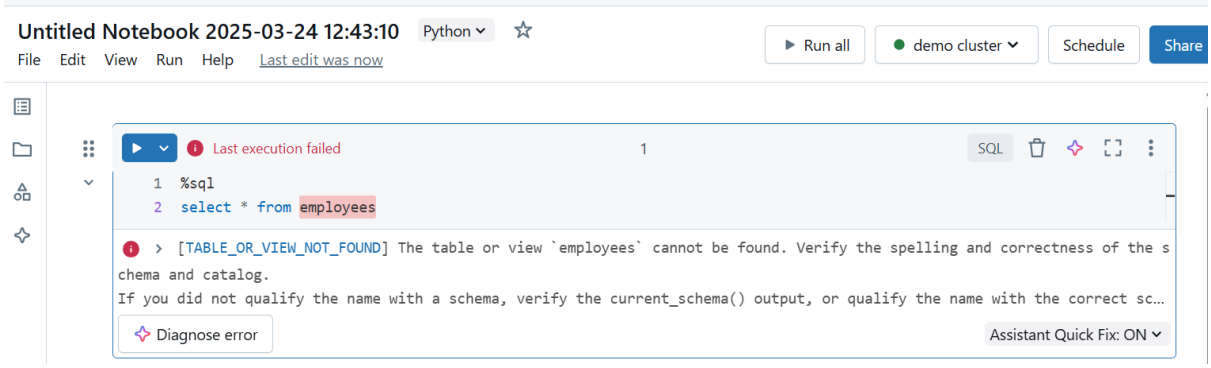
To check if this employees view is created, we will use SQL command

%sql

select * from employees

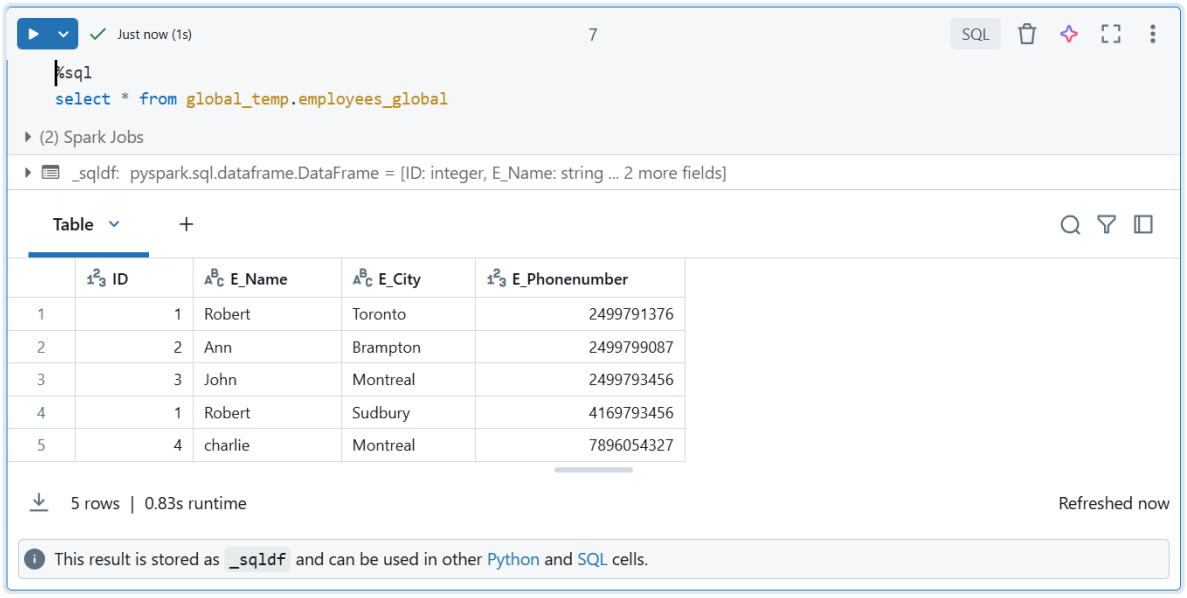


This view is available only within this session if I create new note and want to use this view I will not be able to use it



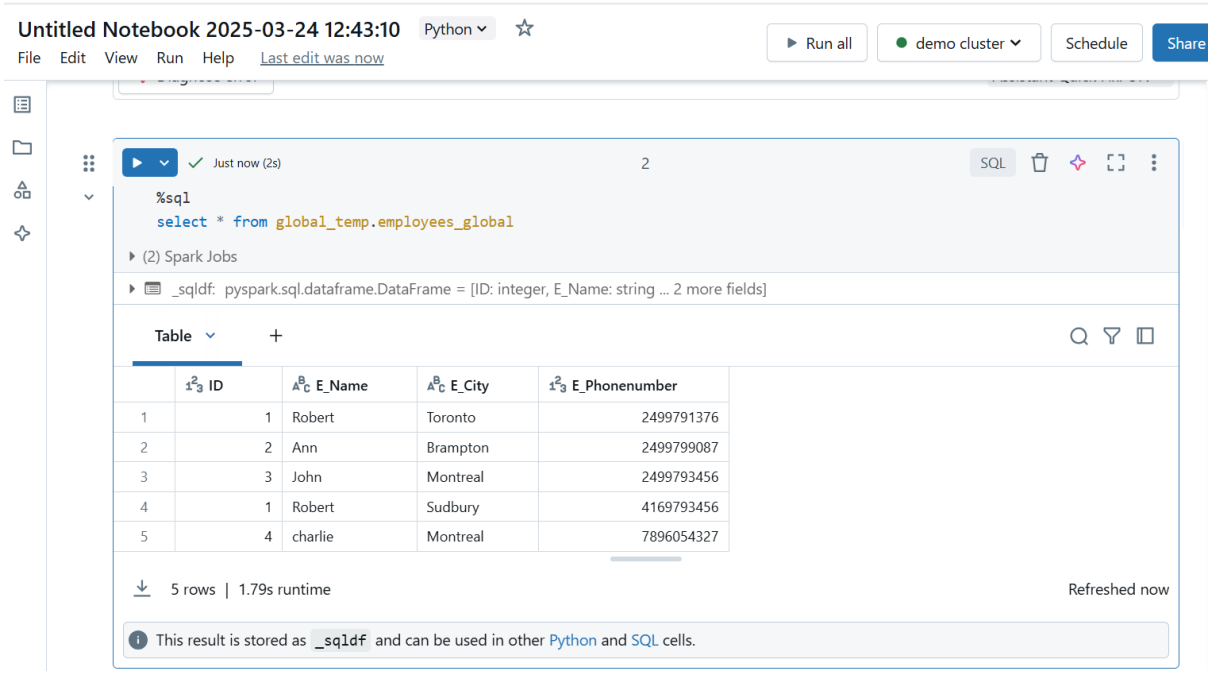
This is Temporary view, its scope is only within the session
Now we will create a global view
df.createOrReplaceGlobalTempView("employees_global")

To check if the Global view is created, we will use SQL command
%sql
select * from global_temp.employees_global



It is created successfully.

Now we can access this Global view in different notebook with in the same cluster



As global view scope is within the entire cluster

Hive Database:

Apache Hive is a **data warehouse system** built on top of **Apache Hadoop** that enables querying and managing large datasets stored in **HDFS (Hadoop Distributed File System)**. It provides an SQL-like interface called **HiveQL (Hive Query Language)** to interact with structured data, making it easier for users familiar with SQL to work with big data.

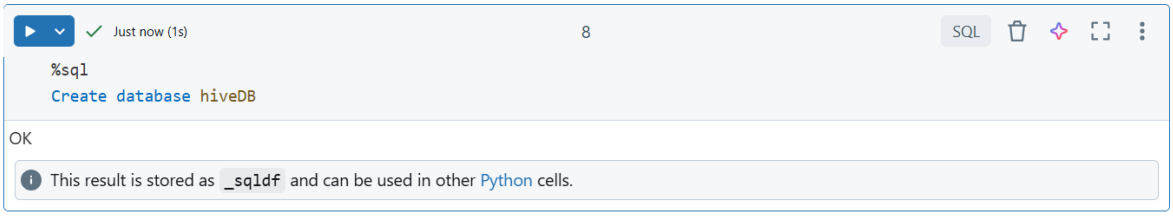
Uses of Hive Database:

1. **Data Warehousing** – Stores and processes large-scale structured and semi-structured data.
2. **SQL-like Querying** – Provides an easy-to-use query language (HiveQL) similar to SQL.
3. **Big Data Processing** – Handles petabytes of data efficiently.
4. **ETL (Extract, Transform, Load)** – Used for data transformation and aggregation in analytics workflows.
5. **Integration with Hadoop Ecosystem** – Works with Spark, Tez, HDFS, and other big data tools.
6. **Scalability** – Designed to scale horizontally across distributed clusters.
7. **Batch Processing** – Ideal for running large-scale batch jobs.
8. **Data Partitioning & Bucketing** – Improves query performance by dividing data into smaller chunks.

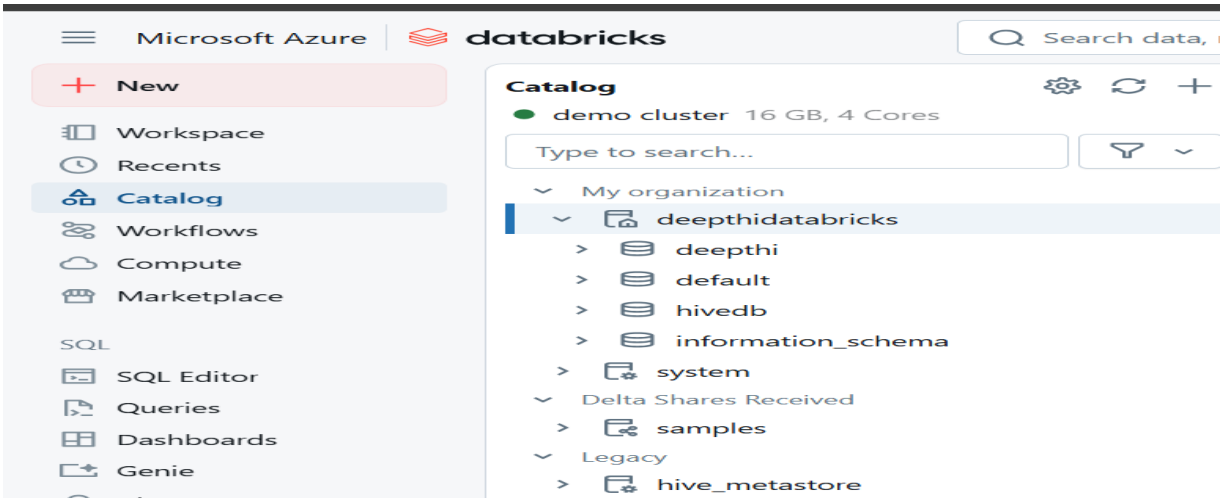
To create Hive Database, we will use SQL command

%sql

Create database hiveDB



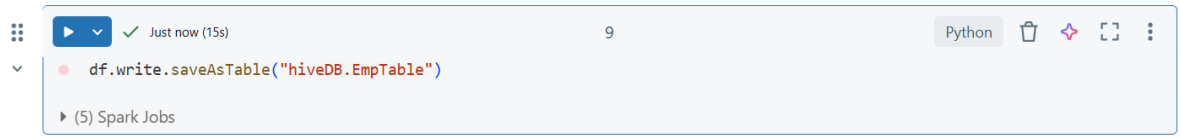
To check if it is created to workspace-> catalog



hiveDB is created

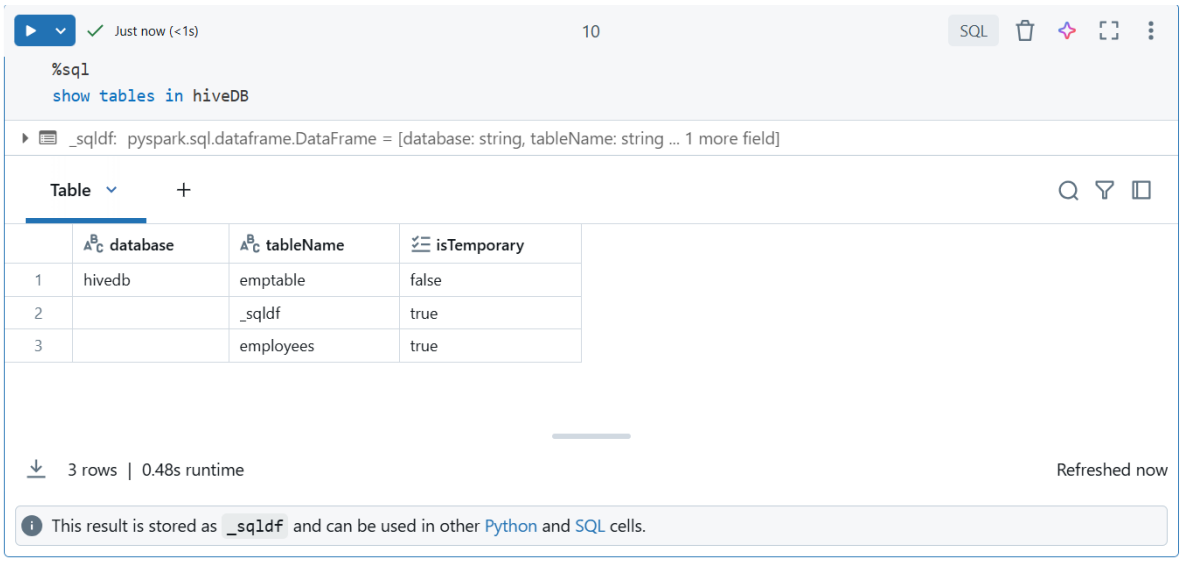
To create a table in this database we can use the data frame so we can directly create a table and store the data from the data frame into that table

df.write.saveAsTable("hiveDB.EmpTable")

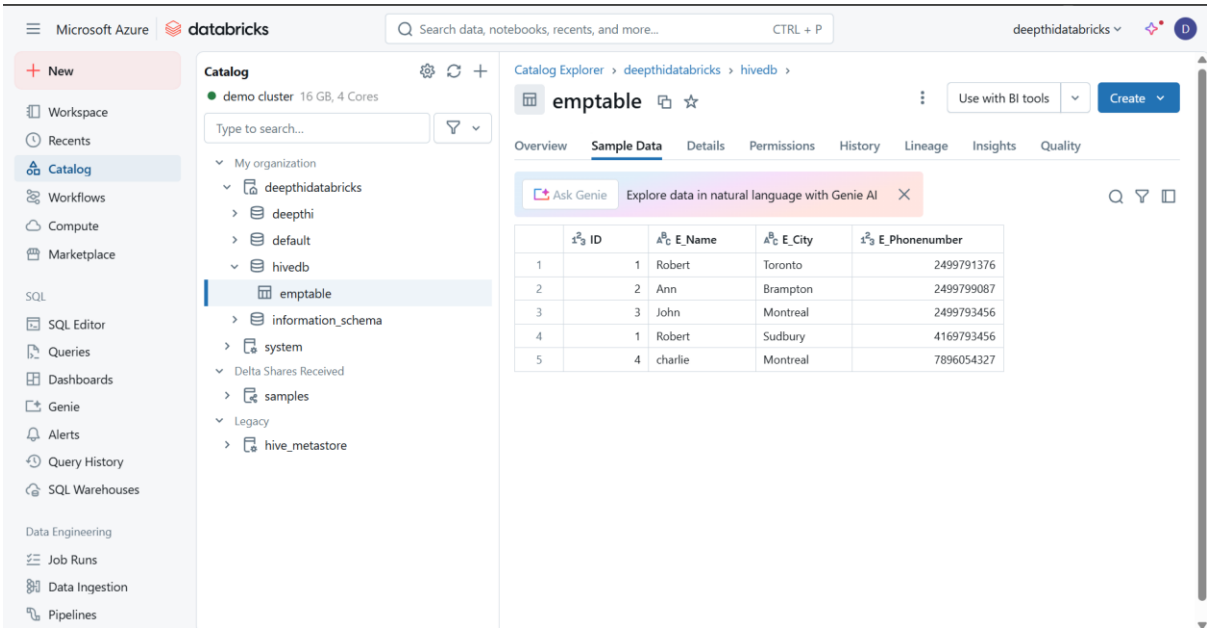


To check if the table is created in data base, we will use SQL command

%sql
show tables in hiveDB



If we see in hiveDB emptable is created



*** Investigate filter conditions and Null conditions**

Filter Conditions

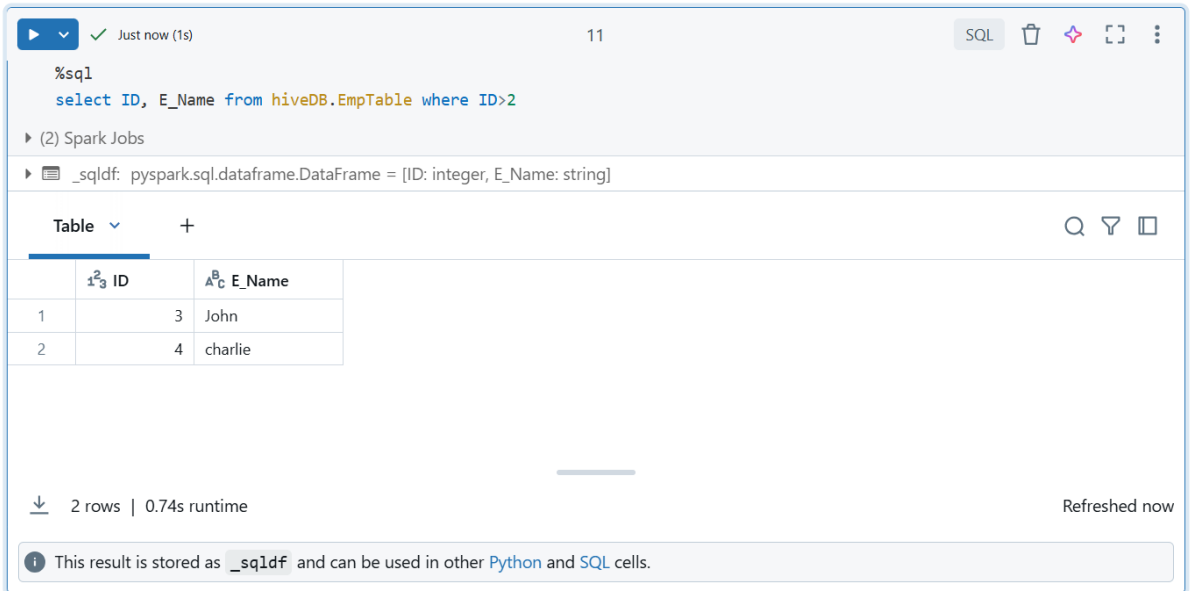
Filter conditions are used in the WHERE clause to retrieve specific records that meet certain criteria.

Common Operators for Filtering

Operator	Description
=	Equals
!= or <>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
BETWEEN	Selects values within a range
LIKE	Pattern matching with wildcards
IN	Matches a set of values

For example:

%sql
select ID, E_Name from hiveDB.EmpTable where ID>2



NULL Conditions in SQL & Hive

Understanding NULL in SQL & Hive

- NULL represents **missing or unknown values**.
- NULL is **not equal to any value**, even another NULL.
- Normal comparison operators (=, !=, etc.) **do not work** with NULL.

- **Checking for NULL Values**

Condition	Description
`IS NULL`	Checks if a column contains `NULL` values
`IS NOT NULL`	Checks if a column does not contain `NULL` values

For example,
%sql
select ID, E_Name from hiveDB.EmpTable where ID is not null