04-spark_table

February 22, 2025

[2]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
        .appName('Spark Table') \
        .getOrCreate()
```

```
25/02/22 04:49:10 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```

[3]:
```
spark
```

[3]:
```
<pyspark.sql.session.SparkSession at 0x7f995ca856c0>
```

[13]:
```
hdfs_path = '/ecommerce_data/ecommerce_data/300MB/customers.csv'
```

[14]:
```
df = spark.read \
     .format('csv') \
     .option('header', 'True') \
     .option('inferschema', 'True') \
     .load(hdfs_path)
```

[15]:
```
df.show()
```

```
+-----------+-----------+---------+-----------+-------+-------------------+-----
----+
|customer_id|       name|     city|      state|country|
registration_date|is_active|
+-----------+-----------+---------+-----------+-------+-------------------+-----
----+
|          0| Customer_0|     Pune|Maharashtra|  India|2023-01-19 00:00:00|
true|
|          1| Customer_1|     Pune|West Bengal|  India|2023-08-10 00:00:00|
true|
|          2| Customer_2|    Delhi|Maharashtra|  India|2023-08-05 00:00:00|
true|
|          3| Customer_3|   Mumbai|  Telangana|  India|2023-06-04 00:00:00|
true|
|          4| Customer_4|    Delhi|  Karnataka|  India|2023-03-15 00:00:00|
```

```
false|
|          5| Customer_5|  Kolkata|West Bengal|  India|2023-08-19 00:00:00|
true|
|          6| Customer_6|  Kolkata| Tamil Nadu|  India|2023-04-21 00:00:00|
false|
|          7| Customer_7|   Mumbai|  Telangana|  India|2023-05-23 00:00:00|
true|
|          8| Customer_8|     Pune| Tamil Nadu|  India|2023-07-17 00:00:00|
true|
|          9| Customer_9|    Delhi|  Karnataka|  India|2023-06-02 00:00:00|
true|
|         10|Customer_10|Hyderabad|      Delhi|  India|2023-02-23 00:00:00|
true|
|         11|Customer_11|    Delhi|West Bengal|  India|2023-11-08 00:00:00|
true|
|         12|Customer_12|    Delhi|      Delhi|  India|2023-06-27 00:00:00|
false|
|         13|Customer_13|     Pune|Maharashtra|  India|2023-02-03 00:00:00|
true|
|         14|Customer_14|  Chennai|  Karnataka|  India|2023-04-06 00:00:00|
true|
|         15|Customer_15|Hyderabad|West Bengal|  India|2023-03-31 00:00:00|
true|
|         16|Customer_16|  Chennai|Maharashtra|  India|2023-04-26 00:00:00|
true|
|         17|Customer_17|     Pune|      Delhi|  India|2023-04-14 00:00:00|
false|
|         18|Customer_18|  Chennai|Maharashtra|  India|2023-02-04 00:00:00|
false|
|         19|Customer_19|  Chennai|  Karnataka|  India|2023-01-22 00:00:00|
true|
+-----------+-----------+---------+-----------+-------+-------------------+-----
----+
only showing top 20 rows
```

0.0.1 Creating Temporary View (Session Based) in Apache Spark

In Apache Spark, we can create temporary views in two ways:

1. `createTempView()`
2. `createOrReplaceTempView()`

1. `createTempView()`:

- **Purpose**: It creates a temporary view with the specified name.
- **Behavior**: If a temporary view with the same name already exists, this method will fail with an error. It does not overwrite or replace the existing view.

- **Syntax**:

  ```
  df.createTempView('customer')
  ```

- **Purpose**: It creates a temporary view with the specified name, but it will replace the existing view if one already exists with the same name.
- **Behavior**: If a temporary view with the same name exists, it replaces the existing view without any errors. If the view does not exist, it simply creates the new one.
- **Syntax**:

  ```
  df.createOrReplaceTempView('customer')
  ```

In Apache Spark, there isn't a direct method like `dropTempView()` to delete a temporary view. However, you can "drop" a temporary view by using the `spark.catalog.dropTempView()` method.

0.0.2 Syntax to delete (drop) a temporary view:

```
spark.catalog.dropTempView('view_name')
```

[25]:
```
df.createTempView('customer')
```

[26]:
```
spark.sql('select * from customer limit 5').show()
```

```
+-----------+----------+------+-----------+-------+------------------+---------
+
|customer_id|      name|  city|      state|country|
registration_date|is_active|
+-----------+----------+------+-----------+-------+------------------+---------
+
|          0|Customer_0|  Pune|Maharashtra|  India|2023-01-19 00:00:00|
true|
|          1|Customer_1|  Pune|West Bengal|  India|2023-08-10 00:00:00|
true|
|          2|Customer_2| Delhi|Maharashtra|  India|2023-08-05 00:00:00|
true|
|          3|Customer_3|Mumbai|  Telangana|  India|2023-06-04 00:00:00|
true|
|          4|Customer_4| Delhi|  Karnataka|  India|2023-03-15 00:00:00|
false|
+-----------+----------+------+-----------+-------+------------------+---------
+
```

[27]:
```
df.createOrReplaceTempView('customer')
```

[29]:
```
spark.sql('select * from customer limit 10').show()
```

```
+-----------+----------+-------+-----------+-------+------------------+--------
-+
|customer_id|      name|   city|      state|country|
registration_date|is_active|
+-----------+----------+-------+-----------+-------+------------------+--------
-+
|          0|Customer_0|   Pune|Maharashtra|  India|2023-01-19 00:00:00|
true|
|          1|Customer_1|   Pune|West Bengal|  India|2023-08-10 00:00:00|
true|
|          2|Customer_2|  Delhi|Maharashtra|  India|2023-08-05 00:00:00|
true|
|          3|Customer_3| Mumbai|  Telangana|  India|2023-06-04 00:00:00|
true|
|          4|Customer_4|  Delhi|  Karnataka|  India|2023-03-15 00:00:00|
false|
|          5|Customer_5|Kolkata|West Bengal|  India|2023-08-19 00:00:00|
true|
|          6|Customer_6|Kolkata| Tamil Nadu|  India|2023-04-21 00:00:00|
false|
|          7|Customer_7| Mumbai|  Telangana|  India|2023-05-23 00:00:00|
true|
|          8|Customer_8|   Pune| Tamil Nadu|  India|2023-07-17 00:00:00|
true|
|          9|Customer_9|  Delhi|  Karnataka|  India|2023-06-02 00:00:00|
true|
+-----------+----------+-------+-----------+-------+------------------+--------
-+
```

[44]:
```
spark.catalog.dropTempView('customer')
```

[44]: False

0.0.3 `createOrReplaceGlobalTempView` (Accessible across sessions)

In Apache Spark, `createOrReplaceGlobalTempView` is similar to `createOrReplaceTempView`, but it has a key difference: global temporary views are available across all sessions in the Spark application, while temporary views are only available within the session that created them.

Purpose: It creates a global temporary view with the specified name, or replaces it if a global view with the same name already exists. Global views are accessible across all Spark sessions in the same application.

Behavior: If a global temporary view with the same name exists, it will be replaced. If it doesn't exist, it will be created.

Scope: Global temporary views are available across all Spark sessions in the application. They are stored in a system database called `global_temp`, and their lifetime is tied to the Spark application, not the session.

Syntax

```
df.createOrReplaceGlobalTempView('global_customer')
```

[30]:
```
df.createOrReplaceGlobalTempView('global_customer')
```

```
ivysettings.xml file not found in HIVE_HOME or
HIVE_CONF_DIR,/etc/hive/conf.dist/ivysettings.xml will be used
```

[32]:
```
spark.sql('select * from global_temp.global_customer limit 10').show()
```

```
+-----------+----------+-------+-----------+-------+------------------+--------
-+
|customer_id|      name|   city|      state|country|
registration_date|is_active|
+-----------+----------+-------+-----------+-------+------------------+--------
-+
|          0|Customer_0|   Pune|Maharashtra|  India|2023-01-19 00:00:00|
true|
|          1|Customer_1|   Pune|West Bengal|  India|2023-08-10 00:00:00|
true|
|          2|Customer_2|  Delhi|Maharashtra|  India|2023-08-05 00:00:00|
true|
|          3|Customer_3| Mumbai|  Telangana|  India|2023-06-04 00:00:00|
true|
|          4|Customer_4|  Delhi|  Karnataka|  India|2023-03-15 00:00:00|
false|
|          5|Customer_5|Kolkata|West Bengal|  India|2023-08-19 00:00:00|
true|
|          6|Customer_6|Kolkata| Tamil Nadu|  India|2023-04-21 00:00:00|
false|
|          7|Customer_7| Mumbai|  Telangana|  India|2023-05-23 00:00:00|
true|
|          8|Customer_8|   Pune| Tamil Nadu|  India|2023-07-17 00:00:00|
true|
|          9|Customer_9|  Delhi|  Karnataka|  India|2023-06-02 00:00:00|
true|
+-----------+----------+-------+-----------+-------+------------------+--------
```

```
-+
```

0.0.4 Step 3: Create a Persistent Table (Stored in Hive Metastore)

In this step, we are working with a **persistent table** in Apache Spark, which is stored in the **Hive Metastore**. This type of table persists across Spark sessions and applications, meaning it remains accessible even after the session ends. The data is stored in an external storage system (such as HDFS or S3), and it can be queried at any time.

Key Concepts:

1. **Creating the Persistent Table**:
   - A **persistent table** is created using the `saveAsTable()` method, which writes the DataFrame into a table format and registers it in the Hive Metastore.
   - By default, the table is **managed**, meaning Spark manages both the **data** and **metadata**.
   - The data for the table is stored externally (e.g., on HDFS, S3), and the metadata (such as schema) is stored in the Hive Metastore.
   - Symtax
     `df.write.mode("write").saveAsTable("customers_persistent")`
2. **Querying the Persistent Table**:
   - Once the table is created, it can be queried using SQL, and the data can be accessed even after restarting the Spark session.
   - This demonstrates that the table is **persistent**, meaning it survives across sessions and is accessible by multiple Spark applications.
   - Syntax
     `spark.sql("SELECT * FROM customers_persistent LIMIT 5").show()`

[48]:
```
spark.sql('show databases').show()
```

```
+---------+
|namespace|
+---------+
|  default|
+---------+
```

[16]:
```
spark.sql('use default')
```

[16]:
```
DataFrame[]
```

[17]:
```
df.write.format('csv').saveAsTable('default.customer')
```

```
25/02/22 06:20:57 WARN HiveExternalCatalog: Couldn't find corresponding Hive
SerDe for data source provider csv. Persisting data source table
`default`.`customer` into Hive metastore in Spark SQL specific format, which is
NOT compatible with Hive.
```

[18]:
```
spark.sql('select * from default.customer limit 10').show()
```

```
+-----------+----------+-------+-----------+-------+------------------+--------
-+
|customer_id|      name|   city|      state|country|
registration_date|is_active|
+-----------+----------+-------+-----------+-------+------------------+--------
-+
|          0|Customer_0|   Pune|Maharashtra|  India|2023-01-19 00:00:00|
true|
|          1|Customer_1|   Pune|West Bengal|  India|2023-08-10 00:00:00|
true|
|          2|Customer_2|  Delhi|Maharashtra|  India|2023-08-05 00:00:00|
true|
|          3|Customer_3| Mumbai|  Telangana|  India|2023-06-04 00:00:00|
true|
|          4|Customer_4|  Delhi|  Karnataka|  India|2023-03-15 00:00:00|
false|
|          5|Customer_5|Kolkata|West Bengal|  India|2023-08-19 00:00:00|
true|
|          6|Customer_6|Kolkata| Tamil Nadu|  India|2023-04-21 00:00:00|
false|
|          7|Customer_7| Mumbai|  Telangana|  India|2023-05-23 00:00:00|
true|
|          8|Customer_8|   Pune| Tamil Nadu|  India|2023-07-17 00:00:00|
true|
|          9|Customer_9|  Delhi|  Karnataka|  India|2023-06-02 00:00:00|
true|
+-----------+----------+-------+-----------+-------+------------------+--------
-+
```

1 Persistent Table in Hive Metastore

1.1 Overview

A **Persistent Table** in Apache Spark refers to a table whose **data** and **metadata** are stored in the **Hive Metastore**. The table persists beyond the Spark session, meaning it remains available until explicitly dropped. These tables can be accessed across multiple Spark sessions and even across different applications.

1.2 Key Concepts

1.2.1 Data and Metadata:

- Both the **data** and **metadata** (schema) are stored in the **Hive Metastore**.

- The table persists even after the Spark session ends, making it accessible beyond a single session.
- The metadata, including the schema, is stored in the Hive Metastore, while the data is typically stored in an external storage system (e.g., HDFS, S3).

1.3 Managed vs External Tables

1.3.1 1. Managed Table:

- **Data & Metadata**: Spark manages both the **metadata** and the **data**.
- **Behavior on Drop**: If you drop a managed table, both the metadata and the data are deleted.
- **Example**: A managed table is typically created when you use `saveAsTable()` without specifying a storage location.

1.3.2 2. External Table:

- **Metadata Only**: In an external table, only the **metadata** is stored in the Hive Metastore, while the **data** is stored externally (e.g., in HDFS, S3).
- **Behavior on Drop**: Dropping an external table only removes the metadata from the Hive Metastore. The actual data in the external storage is not deleted.
- **Example**: An external table is created when you specify a `LOCATION` for the table data.

1.4 Key Points

- **Persistence**: Persistent tables are available across sessions and Spark applications, ensuring that they remain accessible for long-term use.
- **Flexibility**: You can create either **managed** or **external** tables depending on how you want to handle the table's data and metadata.
- **Hive Metastore**: The metadata of the table is stored in the Hive Metastore, making it accessible to other tools and Spark applications that can read from the Metastore.

1.5 Summary

- **Persistent Tables** are stored in the Hive Metastore and can be queried across multiple Spark sessions.
- **Managed Tables**: Both data and metadata are managed by Spark. Dropping a managed table removes both the data and metadata.
- **External Tables**: Only metadata is managed by Spark, while the data is stored externally. Dropping an external table removes only the metadata, leaving the data intact.

1.5.1 Managed Table

[53]:
```
spark.sql("DROP TABLE IF EXISTS customer")
```

[53]:
```
DataFrame[]
```

[19]:
```
spark.sql('describe extended customer').show(truncate=False)
```

```
+----------------------------+--------------------------------------------------
+-------+
|col_name                    |data_type
|comment|
+----------------------------+--------------------------------------------------
+-------+
|customer_id                 |int
|null   |
|name                        |string
|null   |
|city                        |string
|null   |
|state                       |string
|null   |
|country                     |string
|null   |
|registration_date           |timestamp
|null   |
|is_active                   |boolean
|null   |
|                            |
|       |
|# Detailed Table Information|
|       |
|Database                    |default
|       |
|Table                       |customer
|       |
|Owner                       |root
|       |
|Created Time                |Sat Feb 22 06:20:57 UTC 2025
|       |
|Last Access                 |UNKNOWN
|       |
|Created By                  |Spark 3.3.2
|       |
|Type                        |MANAGED
|       |
|Provider                    |csv
|       |
```

```
|Statistics                    |417334215 bytes
|       |
|Location
|hdfs://cluster-f3f6-m/user/hive/warehouse/customer|       |
|Serde Library
|org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe|       |
+--------------------------+--------------------------------------------------
+-------+
only showing top 20 rows
```

2 Before dropping the Managed table

Path: `/user/hive/warehouse/customer`

2.0.1 Directory Content

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r–r–	root	hadoop	0 B	Feb 22 11:50	2	128 MB	_SUCCESS
-rw-r–r–	root	hadoop	155.87 MB	Feb 22 11:50	2	128 MB	part-00000-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv
-rw-r–r–	root	hadoop	155.42 MB	Feb 22 11:50	2	128 MB	part-00001-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv
-rw-r–r–	root	hadoop	86.71 MB	Feb 22 11:50	2	128 MB	part-00002-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv

Note : Please image section for complete details refer to screenshots

[ ]:
```
spark.sql("DROP TABLE IF EXISTS customer")
spark.sql('show tables').show()
```

3 After dropping Managed table

Path: `/user/hive/warehouse/customer`

3.0.1 Directory Content

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name

Note : Please image section for complete details refer to screenshots

4 External Table

[1]:
```
from pyspark.sql import SparkSession


spark = SparkSession.builder \
    .appName("External Table Example") \
    .enableHiveSupport() \
    .getOrCreate()
```

```
25/02/22 06:13:02 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```

[25]:
```
spark.sql("""
CREATE EXTERNAL TABLE IF NOT EXISTS customers_external (
    customer_id INT,
    name STRING,
    city STRING,
    state STRING,
    country STRING,
    registration_date TIMESTAMP,
    is_active BOOLEAN
)
USING CSV
LOCATION '/user/hive/warehouse/customer/
 ↪part-00000-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv'
""")
```

```
25/02/22 06:24:04 WARN HiveExternalCatalog: Couldn't find corresponding Hive
SerDe for data source provider CSV. Persisting data source table
`default`.`customers_external` into Hive metastore in Spark SQL specific format,
which is NOT compatible with Hive.
```

[25]:
```
DataFrame[]
```

[26]:
```
spark.sql('describe extended customers_external').show(truncate=False)
```

```
+----------------------------+----------------------------------------------------
----------------------------------------------------------+-------+
|col_name                    |data_type
|comment|
```

```
+---------------------------+--------------------------------------------------
--------------------------------------------------------+-------+
|customer_id                |int
|null   |
|name                       |string
|null   |
|city                       |string
|null   |
|state                      |string
|null   |
|country                    |string
|null   |
|registration_date          |timestamp
|null   |
|is_active                  |boolean
|null   |
|                           |
|       |
|# Detailed Table Information|
|       |
|Database                   |default
|       |
|Table                      |customers_external
|       |
|Owner                      |root
|       |
|Created Time               |Sat Feb 22 06:24:04 UTC 2025
|       |
|Last Access                |UNKNOWN
|       |
|Created By                 |Spark 3.3.2
|       |
|Type                       |EXTERNAL
|       |
|Provider                   |CSV
|       |
|Location                   |hdfs://cluster-f3f6-
m/user/hive/warehouse/customer/part-00000-41449e73-7692-40cc-986d-729e1a7a662f-
c000.csv|       |
|Serde Library              |org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
|       |
|InputFormat                |org.apache.hadoop.mapred.SequenceFileInputFormat
|       |
+---------------------------+--------------------------------------------------
--------------------------------------------------------+-------+
only showing top 20 rows
```

5 Before dropping the external table

Path: `/user/hive/warehouse/customer/part-00000-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv`

5.0.1 Directory Content

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r–r–	root	hadoop	0 B	Feb 22 11:50	2	128 MB	_SUCCESS
-rw-r–r–	root	hadoop	155.87 MB	Feb 22 11:50	2	128 MB	part-00000-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv
-rw-r–r–	root	hadoop	155.42 MB	Feb 22 11:50	2	128 MB	part-00001-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv
-rw-r–r–	root	hadoop	86.71 MB	Feb 22 11:50	2	128 MB	part-00002-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv

Note : Please image section for complete details refer to screenshots

[27]:
```
spark.sql("DROP TABLE IF EXISTS customers_external")
spark.sql('show tables').show()
```

```
+---------+---------+-----------+
|namespace|tableName|isTemporary|
+---------+---------+-----------+
+---------+---------+-----------+
```

6 After dropping the external table

Path: `/user/hive/warehouse/customer/part-00000-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv`

6.0.1 Directory Content

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r–r–	root	hadoop	0 B	Feb 22 11:50	2	128 MB	_SUCCESS

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r–r–	root	hadoop	155.87 MB	Feb 22 11:50	2	128 MB	part-00000-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv
-rw-r–r–	root	hadoop	155.42 MB	Feb 22 11:50	2	128 MB	part-00001-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv
-rw-r–r–	root	hadoop	86.71 MB	Feb 22 11:50	2	128 MB	part-00002-41449e73-7692-40cc-986d-729e1a7a662f-c000.csv

Note : Please image section for complete details refer to screenshots

[28]:
```
spark.stop()
```