How can I use PySpark to save a DataFrame into a Hive table?

Authored by

stats writer

June 24, 2024

PySpark, a powerful Python-based framework for big data processing, can be used to save a DataFrame into a Hive table. Hive is a popular data warehouse tool that allows for the storage and analysis of large datasets. To save a DataFrame into a Hive table using PySpark, the following steps can be followed:
1. Import the necessary libraries and create a SparkSession.
2. Load the DataFrame that needs to be saved into a Hive table.
3. Specify the Hive table name and database where the table will be stored.
4. Use the PySpark "saveAsTable" function to save the DataFrame into the specified Hive table.
5. The DataFrame will then be converted into a Hive table and can be accessed and queried using Hive SQL or other tools. This allows for efficient data storage and analysis using PySpark and Hive. Overall, using PySpark to save a DataFrame into a Hive table provides a seamless and efficient way to manage and analyze large datasets.

To save a PySpark DataFrame to Hive table use saveAsTable() function or use SQL CREATE statement on top of the temporary view. To save DataFrame as a Hive table in PySpark, you should use `enableHiveSupport()` at the time of creating a SparkSession.

The enableHiveSupport() function in PySpark is used to enable integration with the Apache Hive metastore when creating a SparkSession. When you enable Hive support, PySpark will use the Hive implementation of the metastore client to connect to the Hive metastore.

Enabling Hive support allows PySpark to access and query tables stored in the Hive metastore using SQL syntax. It also enables PySpark to use Hive's SerDe (Serializer/Deserializer) for reading and writing data in various formats, such as Parquet, Avro, ORC, etc., which are commonly used in the Hive ecosystem.

Following are the Steps to Save PySpark DataFrame to Hive Table.

1. Spark Session with Hive Enabled

To enable Hive support while creating a SparkSession in PySpark, you need to use the enableHiveSupport() method. You can also set the custom location of Hive warehouse directory to `spark.sql.warehouse.dir` property. It's essential to ensure that Spark has read and write permissions to this directory.

```
from os.path import abspath
from pyspark.sql import SparkSession
```

# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')

```
# Create spark session with hive enabled
spark = SparkSession
.builder
.appName(arabpsychology.com)
.config("spark.sql.warehouse.dir", warehouse_location)
.enableHiveSupport()
.getOrCreate()
```

2. PySpark Save DataFrame to Hive Table

By using saveAsTable() from `DataFrameWriter` you can save or write a PySpark DataFrame to a Hive table.
Provide the table name you wish to save as an argument to this function, ensuring that the table name adheres to the format `database.tablename`. If the database doesn't exist, you will get an error. To start with you can also try just the table name without a database.

PySpark automatically writes the data to the default Hive warehouse location, typically `/user/hive/warehouse` for Hive clusters, and the current directory for local setups.

When you run this program from Spyder IDE, it creates a `metastore_db` and `spark-warehouse` under the current directory.

What is Hive Metastore and Data Warehouse Location?

2.1 Save DataFrame as Internal Table from PySpark

an internal table is a type of table where the table data and metadata are managed by Hive itself. Internal tables are stored in a Hive warehouse directory, and their data files are managed by Hive's storage handler. When you drop an internal table, it drops the data and also drops the metadata of the table.

Use saveAsTable() method from DataFrameWriter to save PySpark DataFrame as a internal Hive table

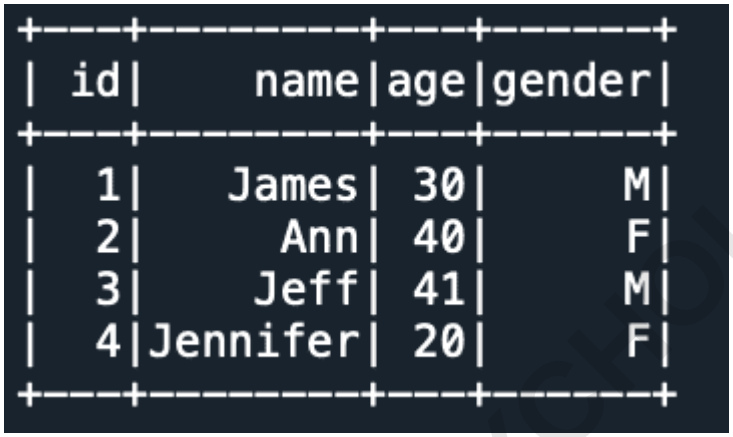
```
columns =

# Create DataFrame
data =
sampleDF = spark.sparkContext.parallelize(data).toDF(columns)
```

```
# Create Hive Internal table
sampleDF.write.mode('overwrite')
.saveAsTable("employee")

# Read Hive table
df = spark.read.table("employee")
df.show()
```

After you run the above program, notice that the Hive metastore `metastore_db` and Hive warehouse location `spark-warehouse` are created at the current directory. Within warehouse directory, you should see `employee` table created.



```
+---+--------+---+------+
| id|    name|age|gender|
+---+--------+---+------+
|  1|   James| 30|     M|
|  2|     Ann| 40|     F|
|  3|    Jeff| 41|     M|
|  4|Jennifer| 20|     F|
+---+--------+---+------+
```

When you save DataFrame to a table, it creates files in parquet format and the default compression would be snappy.

PySpark Read & Write Parquet Files

If you want to create a table within a Database, use the prefix database name. If you don't have the database, you can create one.

```
# Create database
spark.sql("CREATE DATABASE IF NOT EXISTS emp")

# Create Hive Internal table
sampleDF.write.mode('overwrite')
.saveAsTable("emp.employee")
```

In the above example, `emp` is a database and `employee` is a table. Since we use mode("overwite"),

it overwrites the table if it already exists.

2.2 Save as External Table

An external table in Apache Hive is a type of table where the table metadata is managed by Hive, but the table data is stored outside of the Hive warehouse directory. Instead of managing the data files, Hive simply references the location of the external table's data files, which can be located in any user-specified directory or external storage system.

Using `option()` to specify the location of the external table. When you drop an external table, it just drops the metadata but not the actual file. The actual file is still accessible outside of Hive.

```
# Create Hive External table
sampleDF.write.mode(SaveMode.Overwrite)
.option("path", "/path/to/external/table")
.saveAsTable("emp.employee")
```

3. Using PySpark SQL Temporary View to Save Hive Table

Use `SparkSession.sql()` method and `CREATE TABLE` statement to create a table in Hive from PySpark temporary view. Above we have created a temporary view "`sampleView`". Now we shall create a Database and Table using SQL in Hive Metastore and insert data into the Hive table using the view we created above.

Use createOrReplaceTempView() to create a temporary view.

```
# Create temporary view
sampleDF.createOrReplaceTempView("sampleView")
```

```
# Create a Database CT
spark.sql("CREATE DATABASE IF NOT EXISTS ct")

# Create a Table naming as sampleTable under CT database.
spark.sql("CREATE TABLE ct.sampleTable (id Int, name String, age Int, gender String)")

# Insert into sampleTable using the sampleView.
spark.sql("INSERT INTO TABLE ct.sampleTable SELECT * FROM sampleView")

# Lets view the data in the table
spark.sql("SELECT * FROM ct.sampleTable").show()
```

The database `ct` and table `sampletable` is created.

4. Complete Example

Following is a complete example of how to write PySpark DataFrame to Hive table.

```
#!/usr/bin/env python3

from os.path import abspath
from pyspark.sql import SparkSession

# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')

# Create spark session with hive enabled
spark = SparkSession
.builder
.appName(arabpsychology.com)
.config("spark.sql.warehouse.dir", warehouse_location)
.config("spark.sql.catalogImplementation", "hive")
.enableHiveSupport()
.getOrCreate()

columns =

# Create DataFrame
data =
sampleDF = spark.sparkContext.parallelize(data).toDF(columns)

# Create database
spark.sql("CREATE DATABASE IF NOT EXISTS emp")

# Create Hive Internal table
sampleDF.write.mode('overwrite')
.saveAsTable("emp.employee")

# Spark read Hive table
df = spark.read.table("emp.employee")
df.show()
```

6. Conclusion

In conclusion, saving a PySpark DataFrame to a Hive table persist data within the Hive cluster. By utilizing PySpark's DataFrame API and SQL capabilities, users can easily create, manipulate, and save data to Hive tables, enabling a wide range of data analytics and processing tasks. Whether working with internal or external tables, PySpark provides the flexibility to define schema, partition data, and optimize storage formats, ensuring efficient data storage and retrieval.

metastore_db is used by Hive to store the metastore database, while spark-warehouse is used by Spark with Hive support to store table data and metadata.

You can find the complete working example at GitHub PySpark Hive Example

Related Articles

References