How can I use PySpark to read and write data from a MySQL database table?

Authored by

stats writer

June 24, 2024

PySpark is a powerful tool for processing and analyzing large datasets. It also offers the capability to read and write data from various sources, including MySQL database tables. To use PySpark for this purpose, you can first establish a connection to the MySQL database using the PySpark SQL module. Then, you can use the "read" function to retrieve the data from the desired table and "write" function to save the data to a new table or overwrite an existing one. This allows for seamless integration of MySQL data with PySpark, making it easier to perform complex data analysis and manipulation tasks.

In today's world, where data is everywhere, data engineers and analysts need to work well with different kinds of data sources. PySpark makes it easy to connect with many data storage systems, including MySQL databases.

Using PySpark, you can read data from MySQL tables and write data back to them. This means you can pull data from a MySQL database into your PySpark application, process it, and then save the results back to MySQL. This ability to read and write data between PySpark and MySQL helps in handling big data tasks smoothly and efficiently.

In this article, I will cover step-by-step instructions on how to connect to the MySQL database, read the table into a PySpark/Spark DataFrame, and write the DataFrame back to the MySQL table. To connect to the MySQL server from PySpark, you would need the following details: Ensure you have these details before reading or writing to the MySQL server.

1. MySQL Connector for PySpark

You'll need the MySQL connector to work with the MySQL database; hence, first download the connector. also, you would need database details such as the driver, server IP, port, table name, user, password, and database name.

PySpark interacts with MySQL database using JDBC driver, JDBC driver provides the necessary interface and protocols to communicate between the PySpark application (written in Python) and the MySQL database (which uses the MySQL-specific protocol).

In this article, I'm utilizing the `mysql-connector-java.jar` connector and the `com.mysql.jdbc.Driver` driver. MySQL offers connectors specific to a server version, so it's crucial to select the appropriate version corresponding to your server. You need to add this jar to PySpark.

For learning and doing prototypes, you can set the jar via config(), and for production applications, you have to add the jar to PySpark using spark-submit.

First, create a SparkSession by specifying the MySQL connector jar file with config(). By specifying

the JAR file, Spark ensures its availability across cluster nodes, enabling seamless connectivity to MySQL databases.

Download the jar file from the mysql website, select platform independent jar option to download, and use `SparkSession.builder` to create a Spark session, setting the application name and including the path to the MySQL JDBC driver with `.config("spark.jars", "/path/to/mysql-connector-java-8.0.13.jar")`.

```
# Imports
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder
.appName('SparkByExamples.com')
.config("spark.jars", "mysql-connector-java-8.0.13.jar")
.getOrCreate()

# Create DataFrame
columns =
data =

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

3. Write PySpark DataFrame to MySQL Database Table

PySpark enables seamless data transfer from Spark DataFrames into MySQL tables. Whether you're performing data transformations, aggregations, or analyses, By specifying the target MySQL table, mode of operation (e.g., append, overwrite), and connection properties, PySpark handles the data insertion process smoothly.

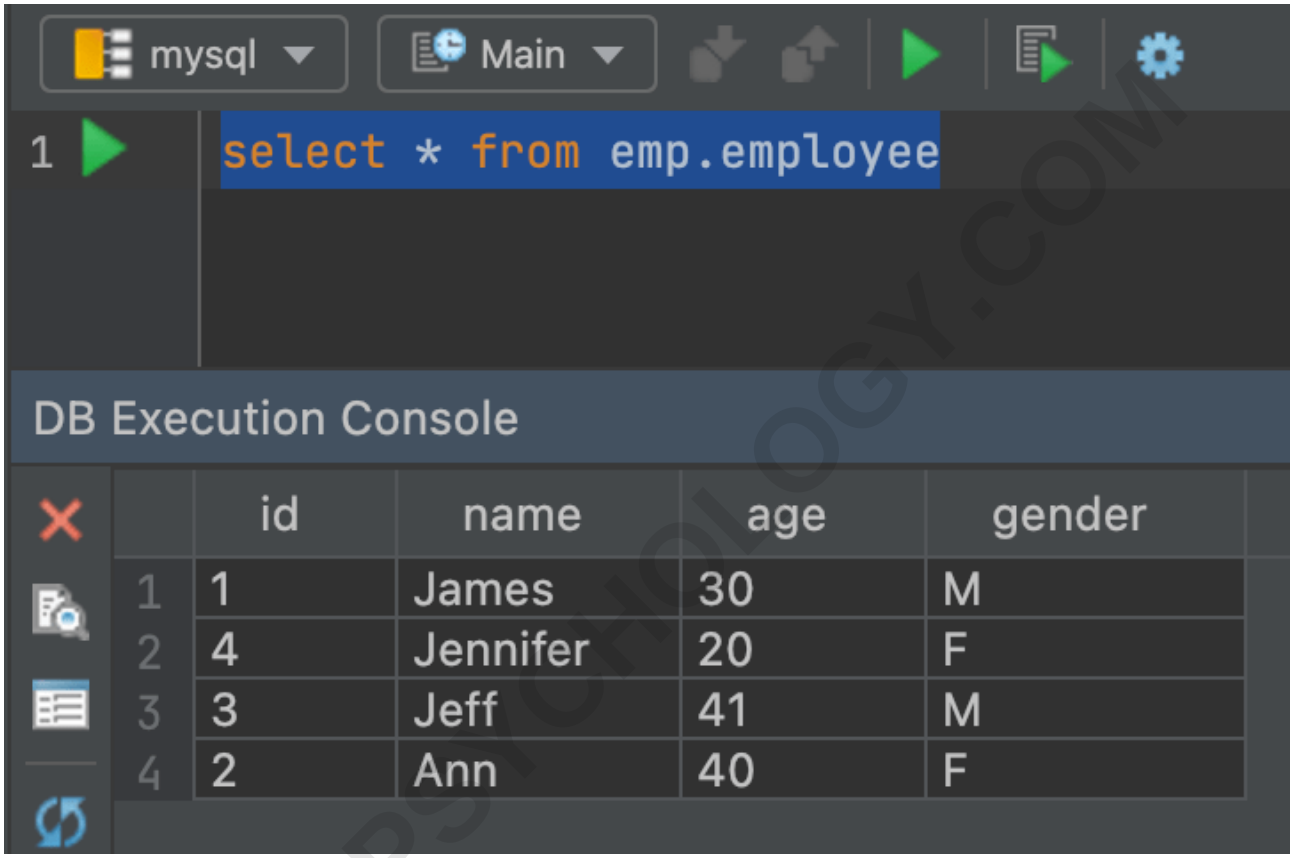
Use the DataFrameWriter.format() to write the DataFrame to MySQL table.

Some points to note while writing

```
# Write to MySQL Table
sampleDF.write
.format("jdbc")
.option("driver","com.mysql.cj.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/emp")
.option("dbtable", "employee")
```

```
.option("user", "root")
.option("password", "root")
.save()
```

As specified above, the `overwrite` mode first drops the table if it already exists in the database.



4. Read MySQL Database Table to PySpark DataFrame

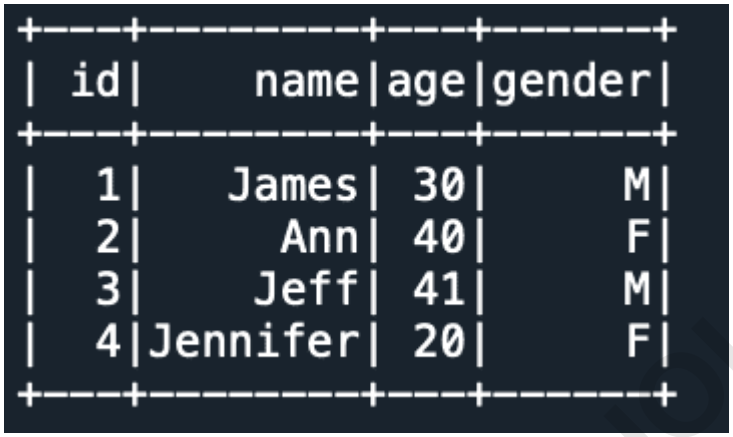
Using PySpark's JDBC connector, you can easily fetch data from MySQL tables into Spark DataFrames. This allows for efficient parallelized processing of large datasets residing in MySQL databases. By specifying the JDBC URL, table name, and appropriate connection properties, PySpark can establish a connection to the MySQL server and ingest data with ease.

In the below example, I am reading a table `employee` from the database `emp` to the DataFrame. The `dbtable` option is used to specify the name of the table you want to read from the MySQL database. This is straightforward and suitable when you want to read the entire table.

```
# Read from MySQL Table
val df = spark.read
```

```
.format("jdbc")
.option("driver","com.mysql.cj.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/emp")
.option("dbtable", "employee")
.option("user", "root")
.option("password", "root")
.load()
```

Yields below output. Alternatively, you can also use spark.read.jdbc() in PySpark



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

5. Select Specific Columns to Read

To read only specific columns from the table, specify the SQL query to the `dbtable` option.

```
# Read from SQL Table
df = spark.read
.format("jdbc")
.option("driver","com.mysql.cj.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/emp")
.option("dbtable", "select id,age from employee where gender='M'")
.option("user", "root")
.option("password", "root")
.load()
```

df.show()

Similarly, you can also use the `option("query","sql query")`, the `query` option allows you to specify a custom SQL query to fetch data. This is useful when you need to read a subset of the

table, join multiple tables, or apply some SQL logic/filtering before loading the data into PySpark.

```
#Using query
df = spark.read
.format("jdbc")
......
......
.option("query", "select id,age from employee where gender='M'")
.......
.......
.load()
```

6. Read MySQL Table in Parallel

Use the `numPartitions` option to parallelize reading from a MySQL table, which also determines the maximum number of concurrent JDBC connections. In the example below, a DataFrame is created with 4 partitions. Additionally, the `fetchsize` option is used to specify the number of rows to fetch per iteration, with a default value of 10.

```
# Using numPartitions
df = spark.read
.format("jdbc")
.option("query", "select id,age from employee where gender='M'")
.option("numPartitions",4)
.option("fetchsize", 20)
.......
.......
.load()
```

7. Append Table

To append rows to the existing MySQL database table, use the `spark.write.mode("append")`

```
# Write to SQL Table
sampleDF.write
.format("jdbc")
.option("driver","com.mysql.cj.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/emp")
```

```
.option("dbtable", "employee")
.option("user", "root")
.option("password", "root")
.save()
```

8. PySpark Shell MySQL Connector

Sometimes you may be required to connect to MySQL from the PySpark shell interactive mode to test some queries, you can achieve this by providing MySQL connector library to spark-shell. once the shell started, you can run your queries.

```
bin/pyspark
--master local
--jars /path/to/mysql/connector/mysql-connector-java-8.0.13.jar
```

9. Spark Submit MySQL Connector

Similarly, you also need to add the MySQL connector jar to the -jar with spark-submit. If you have multiple jars refer to how to add multiple jars to spark-submit. With this command

```
bin/spark-submit
--master yarn
--jars /path/to/mysql/connector/mysql-connector-java-8.0.13.jar
.........
.........
.........
```

9. Complete Example

Following is the complete example of reading and writing DataFrame to MySQL table.

```
# Imports
from pyspark.sql import SparkSession
```

```
# Create SparkSession
spark = SparkSession.builder
.appName('SparkByExamples.com')
```

```
.config("spark.jars", "mysql-connector-java-8.0.13.jar")
.getOrCreate()

# Create DataFrame
columns =
data =

sampleDF = spark.sparkContext.parallelize(data).toDF(columns)

# Write to MySQL Table
sampleDF.write
.format("jdbc")
.option("driver","com.mysql.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/database_name")
.option("dbtable", "employee")
.option("user", "root")
.option("password", "root")
.save()

# Read from MySQL Table
val df = spark.read
.format("jdbc")
.option("driver","com.mysql.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/database_name")
.option("dbtable", "employee")
.option("user", "root")
.option("password", "root")
.load()

df.show()
```


Conclusion

In conclusion, PySpark provides robust capabilities for seamlessly interacting with MySQL databases, offering efficient mechanisms for both reading from and writing to MySQL tables. Through the JDBC connector, PySpark facilitates parallelized data retrieval, enabling scalable and high-performance data processing.

Similarly, PySpark's DataFrame API simplifies the process of writing data back to MySQL tables, offering flexibility and ease of use. With the ability to leverage parallel processing and distributed computing, PySpark empowers data engineers and analysts to harness the full potential of MySQL

data within Spark-powered analytics workflows, facilitating informed decision-making and unlocking insights from diverse datasets.

Related Articles

References