How can I read JDBC data in parallel using PySpark?

Authored by

stats writer

June 24, 2024

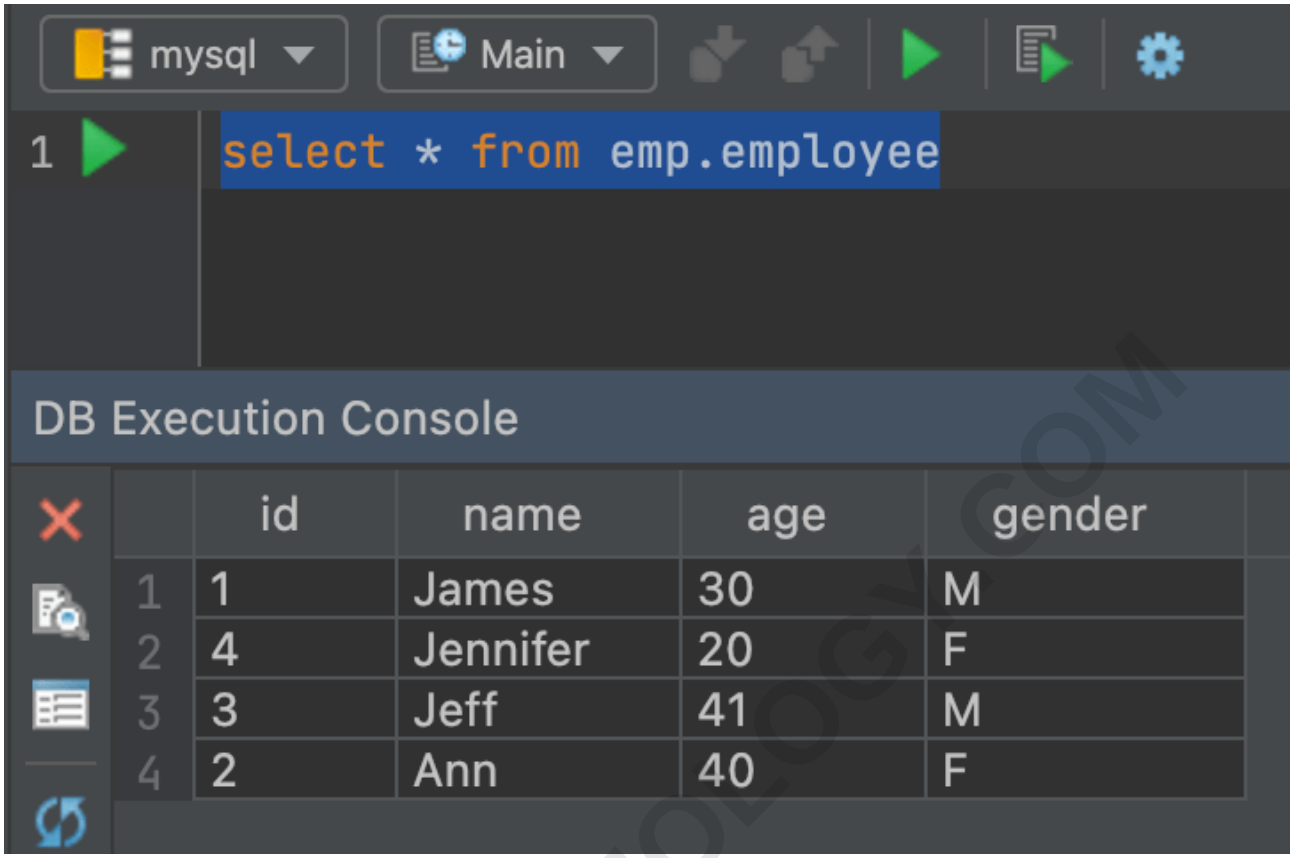
The process of reading JDBC data in parallel using PySpark involves utilizing the parallel processing capabilities of PySpark to retrieve data from a JDBC database. This can be achieved by partitioning the data into smaller chunks and then using multiple worker nodes to simultaneously read and process the data. This allows for efficient and faster retrieval of data from the database, making it ideal for handling large datasets. Additionally, PySpark also provides the ability to optimize the parallel processing by utilizing the underlying parallel execution engine, providing further improvements in performance.

How to read the JDBC in parallel by using PySpark? PySpark jdbc() method with the option numPartitions you can read the database table in parallel. This option is used with both reading and writing. The Apache Spark document describes the option `numPartitions` as follows.

The maximum number of partitions that can be used for parallelism in table reading and writing. This also determines the maximum number of concurrent JDBC connections. If the number of partitions to write exceeds this limit, we decrease it to this limit by calling `coalesce(numPartitions)` before writing.

spark.apache.org

In my previous article, I explained different options with PySpark Read JDBC. In this article, I will explain how to load the JDBC table in parallel by connecting to the MySQL database. I have a database `emp` and table `employee` with columns `id`, `name`, `age` and `gender` and I will use this to explain. For a complete example with MySQL refer to how to use MySQL to Read and Write PySpark DataFrame



1. Read JDBC in Parallel

I will use the PySpark jdbc() method and option `numPartitions` to read this table in parallel into DataFrame. This property also determines the maximum number of concurrent JDBC connections to use. The below example creates the DataFrame with 5 partitions.

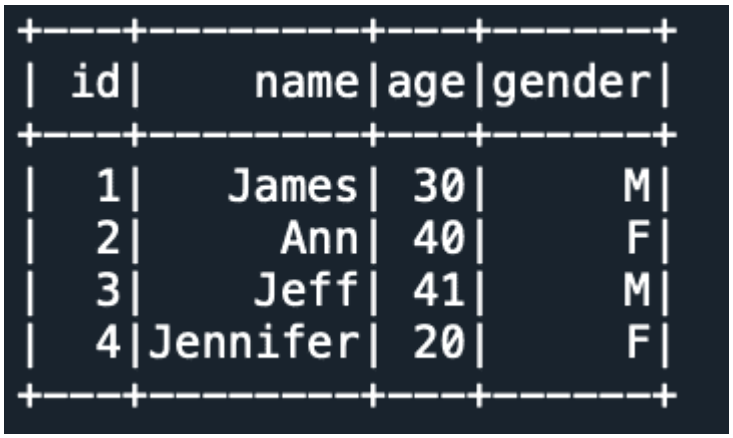
```
# Imports
from pyspark.sql import SparkSession

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

# Query table using jdbc()
df = spark.read
.jdbc("jdbc:mysql://localhost:3306/emp", "employee",
properties={"user": "root", "password": "root", "driver":"com.mysql.cj.jdbc.Driver"})
```

```
# show DataFrame
df.show()
```

Yields below output.



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

2. Read JDBC in Parallel using format()

To read the JDBC table, you can also use the `format("jdbc").load()`. When you use this, you should specify the database details with the option() method and use the `format("jdbc")` method to specify JDBC as the data source.

```
# Read Table in Parallel
df = spark.read
.format("jdbc")
.option("driver","com.mysql.cj.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/emp")
.option("dbtable","employee")
.option("numPartitions",5)
.option("user", "root")
.option("password", "root")
.load()
```

You can also select the specific columns with where condition by using the pyspark `jdbc` query option to read in parallel.

Please note that you can utilize either the "`dbtable`" or "`query`" option, but not both simultaneously. Additionally, when opting for the "`query`" option, the "`partitionColumn`" option cannot be used.

```
# Select columns with where clause
df = spark.read
.format("jdbc")
.option("driver","com.mysql.cj.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/emp")
.option("query","select id,age from employee where gender='M'")
.option("numPartitions",5)
.option("user", "root")
.option("password", "root")
.load()
```

3. Using fetchsize with numPartitions to Read

The `fetchsize` is another option which is used to specify how many rows to fetch at a time, by default it is set to 10. The JDBC fetch size determines how many rows to retrieve per round trip which helps the performance of JDBC drivers. Do not set this to very large number as you might see issues.

```
# Using fetchsize
df = spark.read
.format("jdbc")
.option("driver","com.mysql.cj.jdbc.Driver")
.option("url", "jdbc:mysql://localhost:3306/emp")
.option("query","select id,age from employee where gender='M'")
.option("numPartitions",5)
.option("fetchsize", 20)
.option("user", "root")
.option("password", "root")
.load()
```

4. Other JDBC Options to Read in Parallel

Note that when one option from the below table is specified you need to specify all of them along with numPartitions.

They describe how to partition the table when reading in parallel from multiple workers. `partitionColumn` must be a numeric, date, or timestamp column from the table in question

JDBC Option	Description
partitionColumn	Partition column to specify
lowerBound	Lower Bound
upperBound	Upper Bound

Conclusion

In conclusion, reading data from JDBC sources in parallel using PySpark offers significant advantages in terms of efficiency and performance. By leveraging parallel processing capabilities, PySpark can effectively distribute the workload across multiple executor nodes, allowing for faster data retrieval from relational databases.

This parallelized approach enables seamless integration with a wide range of JDBC-compliant databases while ensuring optimal utilization of computing resources. However, it's essential to be mindful of certain limitations, such as the inability to use both "dbtable" and "query" options simultaneously, as well as restrictions on the usage of certain options like "partitionColumn" in conjunction with the "query" option.

Related Articles