Dong Zhang Follow
May 10, 2021 · 8 min read · Listen

Connecting PySpark to MySQL, PostgreSQL and IBM DB2 for Data Science: Tutorial for Beginners

Introduction

PySpark is an interface for Apache Spark in Python. It is a great tool for data scientists who stick with Python to manipulate data, build machine learning pipelines and deploy models in a distributed environment. Compared with Python and its libraries such as **pandas** and **scikit-learn**, PySpark has better scaling capabilities to handle really huge data sets.

MySQL, PostgreSQL are two database management systems. MySQL is an open-source relational database management system (RDBMS), while PostgreSQL, also know as Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. MySQL has been famous for its ease of use and speed, while PostgreSQL has many more advanced features. You can find the interesting comparison between MySQL vs PostgreSQL, also the history of two systems.

IBM Db2, on the other hand, is a RDBMS produced by IBM. IBM had initially developed DB2 for their own platform, and later IBM developed the family of Db2 common products. In 2018 the IBM SQL product was renamed and is now know as IBM Db2 Big SQL (Big SQL).

People use various tools to manage MySQL, PostgreSQL and IBM Db2, but it is great to integrate multiple databases into a unified platform. Python can be used in database




This blog post is a tutorial about how to set up local PySpark environment and connect to MySQL, PostgreSQL and IBMDB2 for data science modeling.

The outline of this blog is as follows:

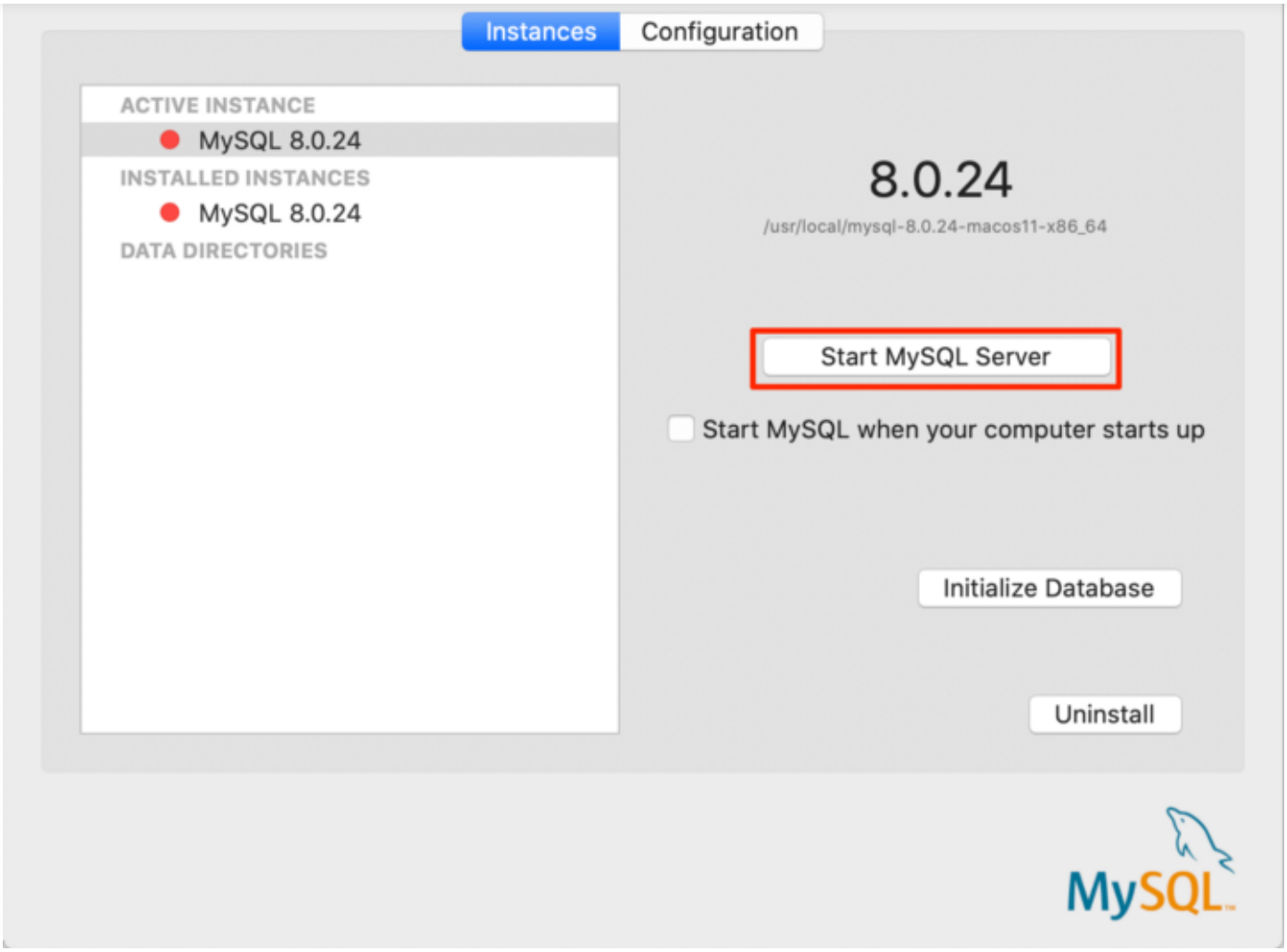
- MySQL
- Connecting PySpark to MySQL
- Building Machine Learning Model in PySpark
- PostgreSQL
- Connecting PySpark to PostgreSQL and Writing Data
- IBM DB2 and Connection with PySpark
- Conclusion

1. MySQL

MySQL Server

I use MacOS Catalina. The first step is to install **MySQL Server** on MacOS, following the official instruction on the mysql website. The newest version of MySQL Community Server can be found here. You need to create a password for the "root" user when the database is initialized.

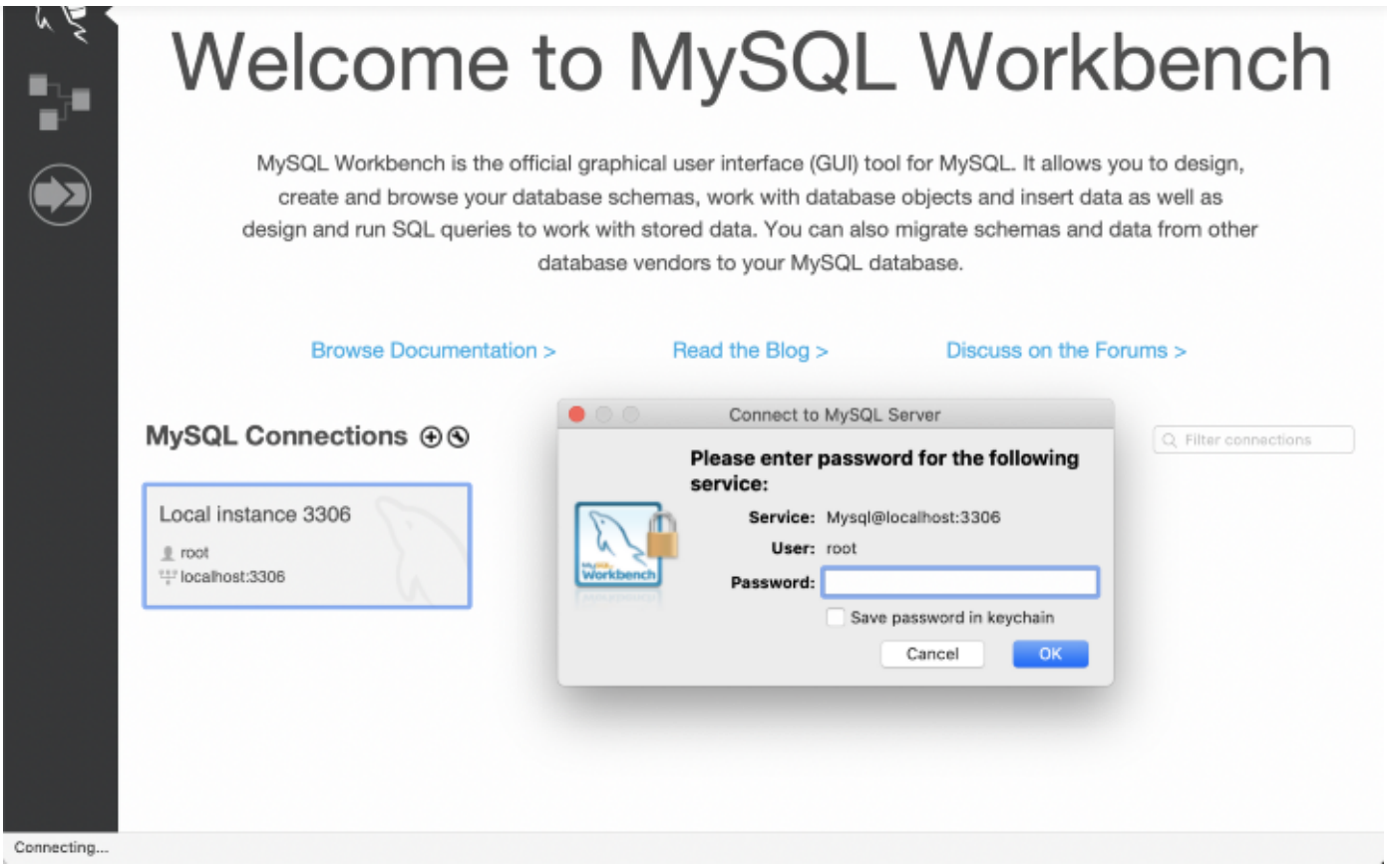


Once the installation is done, you can start the **MySQL Server** from your **System Preferences**, click the "Start MySQL Server" button on the right, the active and installed instance bubbles will go from red to green. You can also select "Start MySQL when your computer stars up" to connect to local MySQL Server automatically.

MySQL Workbench

The second step is to install a SQL database management tool. **MySQL Workbench** is highly recommended for beginners. MySQL Workbench can be downloaded here. It is straightforward to install it, then open it from Application or Launchpad. You will find a local instance (**localhost**) in your MySQL Workbench dashboard, with a default port **3306**.



Use the root password when you installed the MySQL Server to connect the local instance to Workbench, then you should be good to go. More workbench tutorials for beginners can be found here, or in this link.

2. Connecting PySpark to MySQL

I am not going to talk about how to use MySQL Workbench. Instead, I move forward to use Python to create and access MySQL (local) databases, and connect PySpark to MySQL.

The first step is always to install PySpark. You can use **pip install** to install PySpark and MySQL driver written in Python:

I use a sample data to demonstrate how to write data into MySQL from python, and connect MySQL to PySpark for data science modeling. The Iris flower classification problem is probably the best-known example of supervised machine learning classification problem for beginners. The Iris dataset contains a set of 150 data points with four features, and three species as labels. Using pandas in Python to load data, we have

```python
import pandas as pd

# load data using pandas
df_iris = pd.read_csv("data/iris.csv")

df_iris
```

python_pandas_load_iris_data.py hosted with ❤ by GitHub view raw




0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

150 rows × 5 columns

Note that you may need to change the last column a little bit in order to convert it to a MySQL table:

```
df_iris['variety'] = df_iris.variety.apply(lambda x: "'"+x+"'")
```

so that **Setosa** becomes **'Setosa'** and so on.

In Python, one can use **mysql.connector** to access MySQL, **getpass** to input the password for the localhost root user, and create new schema:

```
1  import mysql.connector
2  import getpass
```




```
db_connection = mysql.connector.connect(host = 'localhost',port = 3306, user = 'root', pas
# create and use a new database/schema TestDB
db_cursor = db_connection.cursor()
db_cursor.execute("CREATE DATABASE TestDB;")
db_cursor.execute("USE TestDB;")
```

pyspark_mysql_create_mysql_schema.py hosted with ❤ by GitHub view raw

Now a new schema/database called **TestDB** has bee created. Next, create a new table called **iris** in **TestDB** and write data into it:

```
db_cursor.execute("CREATE TABLE iris(sepal_length DECIMAL(2,1) NOT NULL, \
                  sepal_width DECIMAL(2,1) NOT NULL, \
                  petal_length DECIMAL(2,1) NOT NULL, \
                  petal_width DECIMAL(2,1), species INT);")

iris_tuples = list(df_iris.itertuples(index=False, name=None))
iris_tuples_string = ",".join(["(" + ",".join([str(w) for w in wt]) + ")" for wt in iris_t

db_cursor.execute("INSERT INTO iris(sepal_length, sepal_width, petal_length, petal_width,
                  + iris_tuples_string + ";")
db_cursor.execute("FLUSH TABLES;")
```

python_mysql_create_sql_table.py hosted with ❤ by GitHub view raw

The new table **iris** has been created in **TestDB**. You can check the new table via **SQL Workbench**, or directly read the iris table in Python:

```
sql = "SELECT * FROM iris"
db_cursor.execute(sql)

tuples = db_cursor.fetchall()

cols = list(df_iris.columns)
df_temp = pd.DataFrame(tuples,columns=cols)
```

In order to start a Spark session, you need to load the local Java driver for MySQL (**mysql-connector-java-8.0.22.jar** for me) from local directory. Then you can use **spark.read** to load the external table from MySQL to PySpark:

```
from pyspark.sql import SparkSession

spark = SparkSession \
        .builder.config("spark.jars", "/usr/share/java/mysql-connector-java-8.0.22.jar")\
        .master("local").appName("PySpark MySQL basic example")
        .getOrCreate()

iris_df = spark.read \
          .format("jdbc") \
          .option("url", "jdbc:mysql://localhost:3306/TestDB") \
          .option("driver", "com.mysql.jdbc.Driver") \
          .option("dbtable", "iris") \
          .option("user", "root") \
          .option("password", password) \
          .load()
```

pyapark_mysql_load_spark.py hosted with ❤ by **GitHub** **view raw**

3. Building Machine Learning Model in PySpark

Once the iris data from MySQL has been loaded into PySpark, you can play around with machine learning models for this classification problem. First encode three species **Setosa**, **Versicolor** and **Virginica** to numbers:

```
from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol="species", outputCol="species_numeric").fit(iris_df)
indexed_df = indexer.transform(iris_df)
indexed_df.drop("bar").show()
```



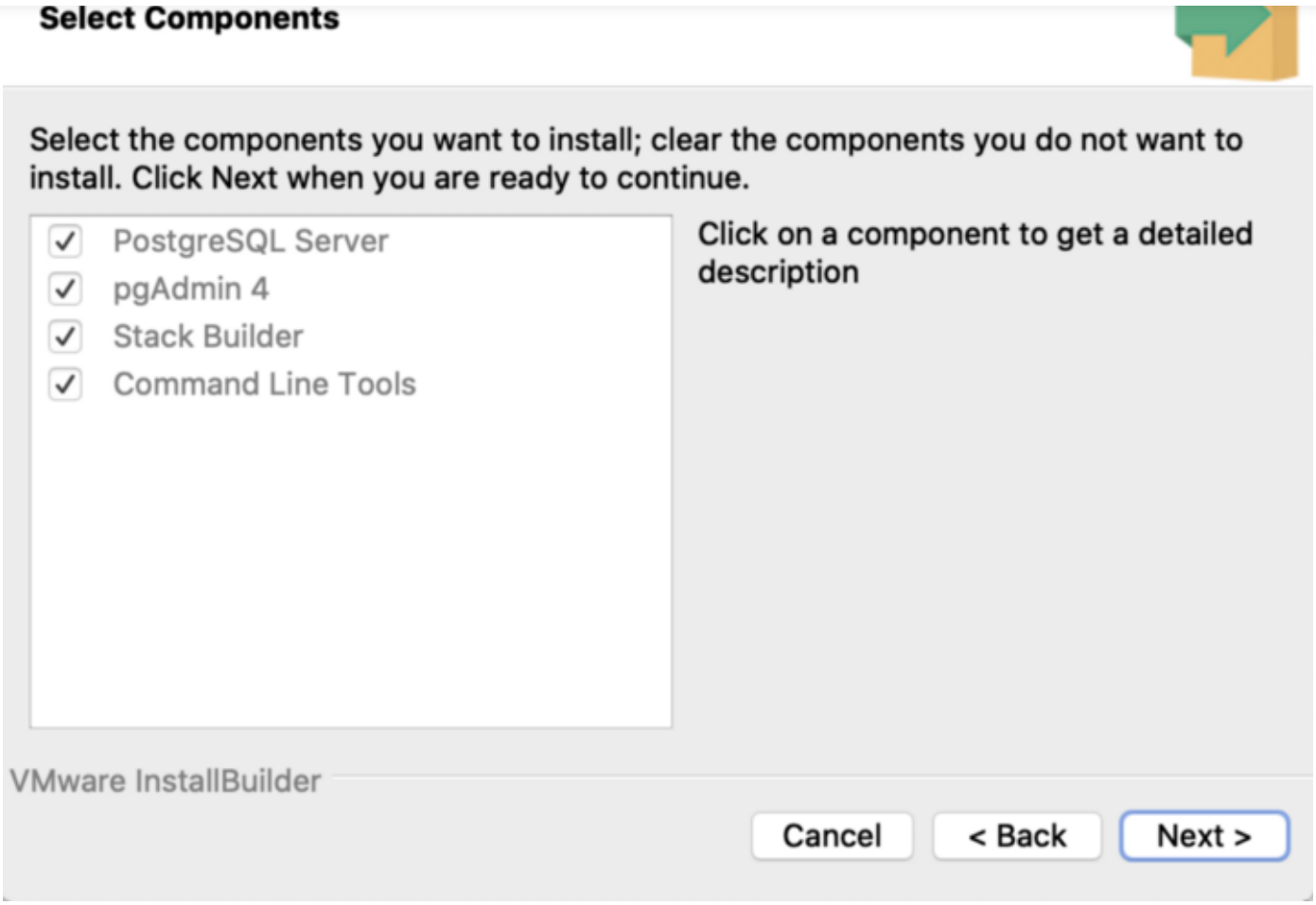

```
#|          4.9|          3.0|          1.4|          0.2|Setosa     |              0.0|
#|          4.7|          3.2|          1.3|          0.2|Setosa     |              0.0|
#|          4.6|          3.1|          1.5|          0.2|Setosa     |              0.0|
#|          5.0|          3.6|          1.4|          0.2|Setosa     |              0.0|
#|          5.4|          3.9|          1.7|          0.4|Setosa     |              0.0|
#|          4.6|          3.4|          1.4|          0.3|Setosa     |              0.0|
#|          5.0|          3.4|          1.5|          0.2|Setosa     |              0.0|
#|          4.4|          2.9|          1.4|          0.2|Setosa     |              0.0|
#|          4.9|          3.1|          1.5|          0.1|Setosa     |              0.0|
#|          5.4|          3.7|          1.5|          0.2|Setosa     |              0.0|
#|          4.8|          3.4|          1.6|          0.2|Setosa     |              0.0|
#|          4.8|          3.0|          1.4|          0.1|Setosa     |              0.0|
#|          4.3|          3.0|          1.1|          0.1|Setosa     |              0.0|
#|          5.8|          4.0|          1.2|          0.2|Setosa     |              0.0|
#|          5.7|          4.4|          1.5|          0.4|Setosa     |              0.0|
#|          5.4|          3.9|          1.3|          0.4|Setosa     |              0.0|
#|          5.1|          3.5|          1.4|          0.3|Setosa     |              0.0|
#|          5.7|          3.8|          1.7|          0.3|Setosa     |              0.0|
#|          5.1|          3.8|          1.5|          0.3|Setosa     |              0.0|
#+-------------+-------------+-------------+-------------+-----------+-----------------+
#only showing top 20 rows
```

pyapark_mysql_encoder.py hosted with ❤ 5 | 1 | ••• **view raw**

It is straightforward to build the supervised machine learning classification model with the encoded labels (**species_numeric**). Below is the code, you may want to compare with the pandas/scikit-learn version if you are more familiar with scikit-learn:

4. PostgreSQL

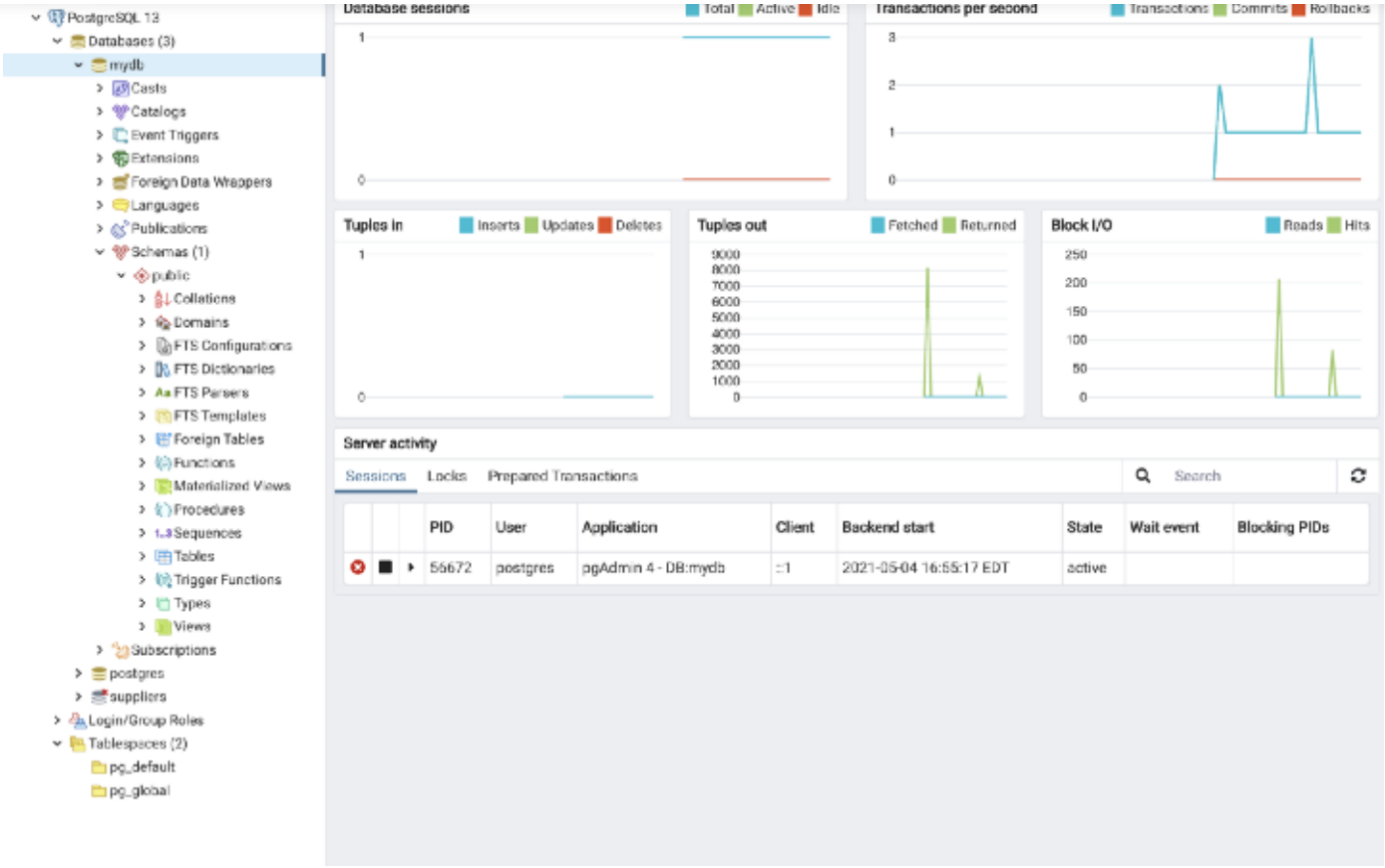
As mentioned in the introduction, PostgreSQL is a RDBMS. There are multiple ways to install PostgreSQL. Perhaps the most straightforward way is to download the **EDB PostgreSQL installer**. Find the installer for your system. I use MacOS so I installed **postgresql-13.2–2-osx.dmg**, which package includes **PostgreSQL Sever**, **pdAdmin 4**, Stack Builder and Command Line Tools:



EDB PostgreSQL installer

Follow the steps in this link to install the PostgreSQL package. Similar to MySQL installation, you need to set up a password for root user, and the default port for PostgreSQL Server is **5432**.

pdAdmin is the most popular PostgreSQL management tool. You can find more tutorials on its website (https://www.pgadmin.org/). Open pdAdmin and input the password, you are connected to the local server and good to go!



pdAdmin

Let us create a new database called “**mydb**” in pgAdmin.

5. Connecting PySpark to PostgreSQL and Writing Data

Still, we start from loading the iris flower data (**iris.csv**) to a pandas dataframe by

Connect Python to PostgreSQL. Before doing that you need to install Python library **psycopg2.** Sometimes **pip install psycopg2** does not work and you need to do:

```
pip install psycopg2-binary
```

The Python code for connecting to PostgreSQL is




The following steps show how to create a table in PostgreSQL using Python connection, and how to load the table in Python. These steps are similar to the above discussion for MySQL connection with Python.

We want to create a table also called **iris** in PostgreSQL localhost database **mydb**:

Let us double check if the iris table is already in **mydb.** You can go to pdAdmin to review the data, or in Python you can connect to the database, run a SQL query and convert the loaded data to pandas dataframe:




Now we want to connect PySpark to PostgreSQL. You need to download a PostgreSQL JDBC Driver jar and do the configuration. I used **postgresql-42.2.20.jar**, but the driver is up-to-date.

Start the Spark and load data from PostgreSQL by:

Note that for configuration you need to direct **spark.jars** to the right directory. Instead of using **com.mysql.jdbc.Driver** for **PySpark + MySQL** connection, you should use **org.postgresql.Driver** as the driver.

Once the dataframe is ready in PySpark, you can follow the exact same steps in Section 3 (**Build Machine Learning Model in PySpark**) to build a baseline machine learning model in PySpark.

6. IBM DB2 and Connection with PySpark

IBM had initially developed DB2 product for their own platform. Since 1990, IBM decided to deploy a universal database DB2 Server, and produced a Db2 common product.

SQL, the Structured Query Language, was initially developed by IBM in 1974. In 2018 the IBM SQL product was renamed and is now know as IBM Db2 Big SQL (Big SQL). Db2 Big SQL now can be integrated with Cloudera Data Platform, or available as a cloud-native service on the IBM Cloud Pak for Data platform.

In order to connect PySpark with IBM DB2 databases, you need to have the permit to access the databases, such as a w3 ID and a Big SQL account. You also need to know the database url and port. This is a template of all information, including username, password, bigsql account, bigsql url and port:




The next thing is to download a IBM DB2 JDBC Driver to your local machine. You can get the download information from the IBM website to download the most recent **db2jcc4.jar** file, or you can directly download **db2jcc4.jar** from this link.

Once you start a Spark session, direct spark.jars to the **db2jcc4.jar** file in your local machine, and input the Big SQL information:




Note that **sql_script** is an example of Big SQL query to get the relevant data:

```
sql_script = """(SELECT *
                 FROM name_of_the_table
                 LIMIT 10)"""
```

Then you can read Big SQL data via **spark.read**. I recommend to use PySpark to build models if your data has a fixed schema (i.e. no new columns are added too often), but if you are more familiar with pandas and other python libraries, you can always convert a Spark dataframe to pandas dataframe by

```
df.select("*").toPandas()
```




with Python to manipulate data, build machine learning pipelines and deploy models in a distributed environment. I recommend to use PySpark to integrate various databases into a unified platform for data science work.

In this blog I showed how to install MySQL and PostgreSQL Servers locally, set up PySpark environment and connect to MySQL and PostgreSQL databases. Db2 is widely used inside IBM, I also provide the method to connect Db2 Big SQL to PySpark — so multiple databases can be integrated to one platform: PySpark. Once the databases are connected and loaded to PySpark, we can create models and build machine learning pipelines in PySpark.

Please let me know if you have any question.

Find me on LinkedIn.