

# Spark DataSources JDBC PostgreSQL

---

[https://github.com/luiscoco/Spark\\_DataSources\\_JDBC\\_PostgreSQL](https://github.com/luiscoco/Spark_DataSources_JDBC_PostgreSQL)

NOTE: if you have any doubt about how to run and initialize the PostgreSQL docker container, see the following youtube video.

[https://www.youtube.com/watch?v=S25oz7uDn\\_g](https://www.youtube.com/watch?v=S25oz7uDn_g)

## 1. Previous steps: install IntelliJ Community + Scala plugin, Java 11, Spark and winutils in your computer

---

### 1.1. Install IntelliJ Community + Scala plugin

---

<https://www.jetbrains.com/idea/download/?section=windows>

### 1.2. Install Java 11 (JDK) and set JAVA\_HOME environmental variable

---

<https://www.oracle.com/es/java/technologies/javase/jdk11-archive-downloads.html>

### 1.3. Install Spark and set SPARK\_HOME environmental variable or directly add the bin folder path in the PATH environmental variable

---

<https://spark.apache.org/downloads.html>

After unzipping the file "spark-3.5.0-bin-hadoop3" place the folder in your C: hard disk root.

There are two options for setting the Spark environmental variables:

a) Create a new variable SPARK\_HOME and set the following value: C:\spark-3.5.0-bin-hadoop3

Then we set the bin folder path in the PATH environmental variable.

b) Add the bin folder path to the PATH environmental variable.

## 1.4. Install winutils and set HADOOP\_HOME environmental variable

---

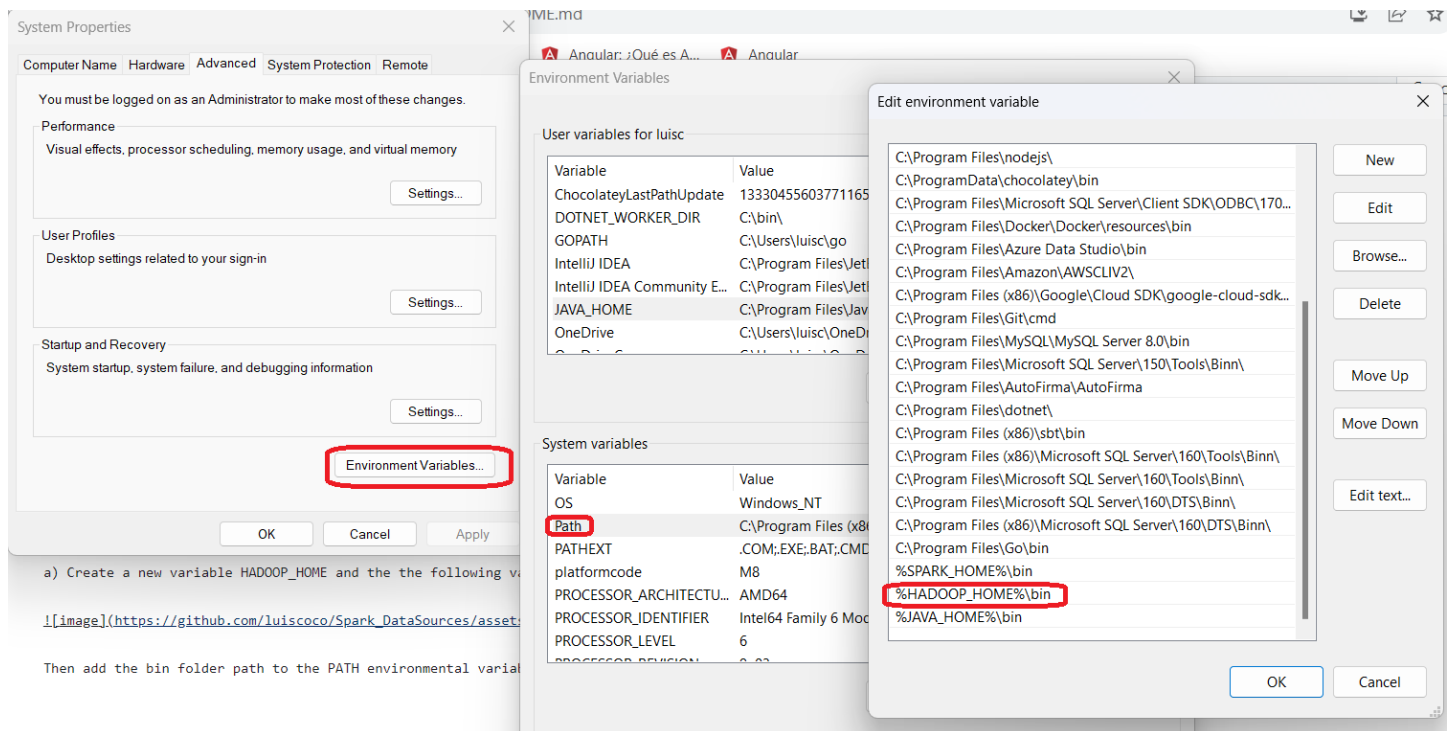
We download or clone this git repository: <https://github.com/kontext-tech/winutils>

We place the winutils folder in C:

Set HADOOP\_HOME environmental variable. There are two options:

a) Create a new variable HADOOP\_HOME and the the following value

Then add the bin folder path to the PATH environmental variable



b) Add the bin folder path to the PATH environmental variable.

## 1.5. Install PostgreSQL and pgAdmin in you local laptop

How to Install PostgreSQL 15 on Windows 10 [ 2023 Update ] Complete guide | pgAdmin 4

<https://www.youtube.com/watch?v=0n41UTkOBb0>

In the following URL you can downlaod postgresQL

<https://www.postgresql.org/download/>

In the following URL you can download pgAdmin

<https://www.pgadmin.org/download/pgadmin-4-windows/>

pgadmin.org/download/pgadmin-4-windows/

Gmail YouTube Maps Noticias Traducir RxJS v6.6.7 Angular: ¿Qué es A... Angular

pgAdmin Home Development Documentation Download Support

### Quick Links

- Download
- FAQ
- Latest Docs
- Get Help
- Screenshots

## pgAdmin 4 (Windows)

### Download

**Maintainer:** pgAdmin Development Team





pgAdmin is available for 64 bit Windows™ 7 SP1 (desktop) or 2008R2 (server) and above, up to v4.30.

v5.0 and later are supported on Windows 8 (desktop) or 2012 (server) and above.

v7.0 and later are supported on Windows 10 (desktop) or 2016 (server) and above.

32 bit Windows support is available for versions up to v4.29.

The packages below include both the Desktop Runtime and Web Application:

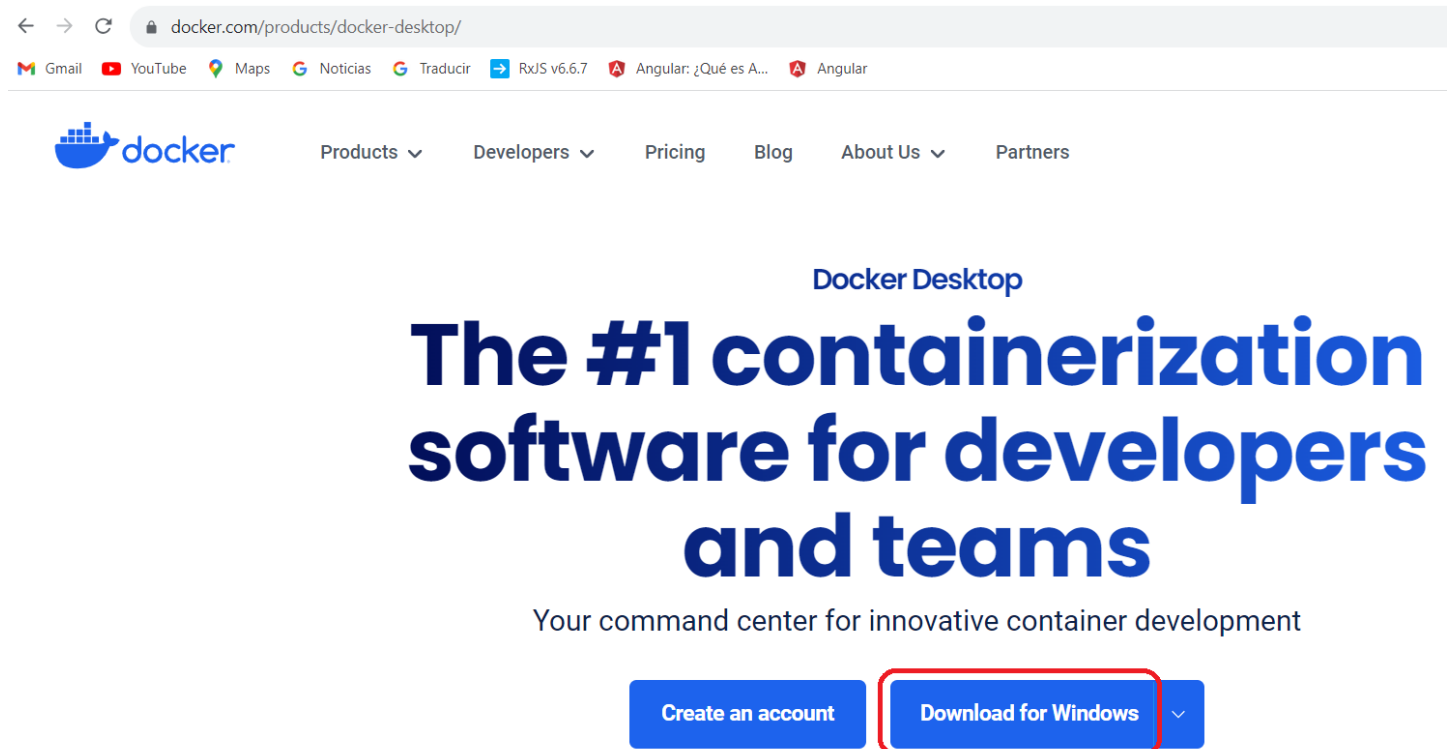
-  [pgAdmin 4 v7.8](#) (released Oct. 19, 2023)
-  [pgAdmin 4 v7.7](#) (released Sept. 21, 2023)
-  [pgAdmin 4 v7.6](#) (released Aug. 24, 2023)
-  [pgAdmin 4 v6.21](#) (released March 9, 2023)

### Info

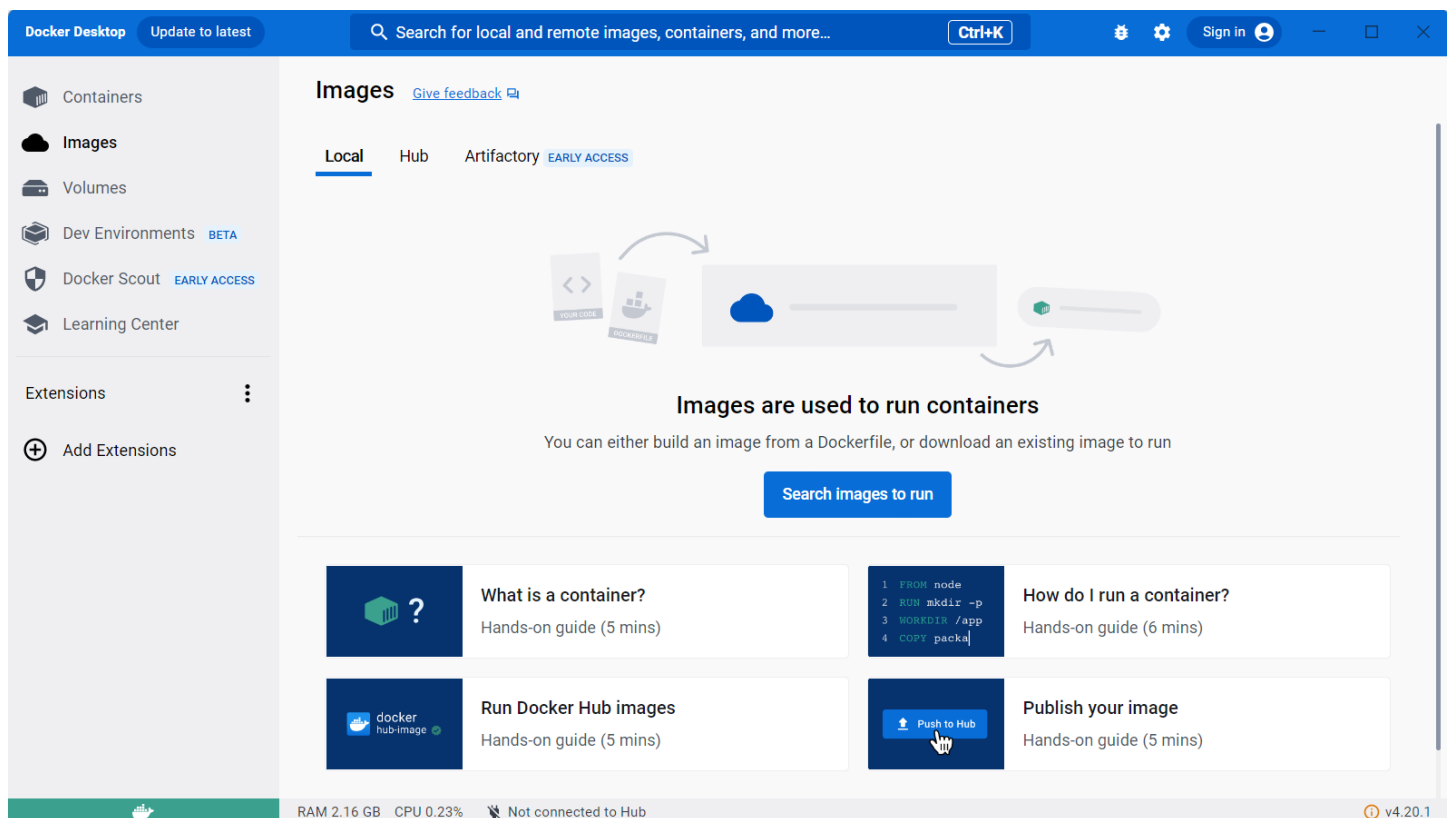
## 1.6. Run PostgreSQL in Docker container and create a database and populate a table

1. Download, install Docker Desktop

<https://www.docker.com/products/docker-desktop/>



## Run Docker Desktop



## 2. Pull and run the PostgreSQL docker container.

For details see: [https://hub.docker.com/\\_/postgres](https://hub.docker.com/_/postgres)

Open command prompt as administrator and run the following command to run the PostgreSQL docker container.

```
docker run --name mypostgres -e POSTGRES_PASSWORD=password -p 5432:5432 -d postgres
```

This is a command to run a Docker container using the official PostgreSQL image from the Docker Hub. Let me break it down for you:

**docker run:** This is the command to run a Docker container.

**--name mypostgres:** This flag sets the name of the container to "mypostgres". You can use this name to refer to the container in other Docker commands.

**-e POSTGRES\_PASSWORD=password:** This sets an environment variable within the container. In this case, it's setting the password for the PostgreSQL user to "password". You can change "password" to whatever you prefer.

**-p 5432:5432:** This flag maps the container's port 5432 (PostgreSQL's default port) to the host machine's port 5432. This means you can connect to the PostgreSQL database on the host machine using port 5432.

**-d postgres:** This specifies the Docker image to use. In this case, it's using the official PostgreSQL image from Docker Hub.

So, in summary, this Docker command is creating and running a PostgreSQL container named "mypostgres" with a specified password,

mapping the container's PostgreSQL port to the host machine's port, and running it in the background (-d flag).

3. We check the PostgreSQL docker container is running. We also copy the ContainerID to execute it later.

```
docker ps -a
```

4. We execute the PostgreSQL container.

```
docker start dockerContainerID
```

```
C:\>docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
014275b34d81	postgres	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:5432->5432/tcp	mypostgres

```
C:\>docker start 014275b34d81
```

We enter in the command bash in the running PostgreSQL docker container

```
docker exec -it dockerContainerID bash
```

5. We run this command

```
psql -U postgres -W
```

This is a command-line instruction for interacting with PostgreSQL, a popular open-source relational database management system. Let's break it down:

**psql:** This is the command-line client for PostgreSQL. It allows you to interact with the database using SQL queries and commands.

**-U postgres:** This specifies the username to connect to the database. In this case, it's set to "postgres." You're connecting as the user "postgres."

**-W:** This option prompts for the password. After entering the command, you'll be asked to enter the password for the specified user ("postgres" in this case).

It's a security measure to ensure that only authorized users can access the database.

So, when you run this command, it initiates a connection to a PostgreSQL database as the user "postgres" and prompts you for the password before allowing access.

6. In Password enter the password we set when running the docker container

```
Password: password
```

7. We create a new database called mydb

```
create database mydb;
```

```
postgres=# create database mydb;  
CREATE DATABASE
```

8. For listing all the databases

```
\l
```

9. Now we create a new table called t1 inside the mydb database

```
create table t1(id int);
```

10. We select all rows and we check there is still no rows in the table

```
select * from t1;
```

11. We insert a row in the table

```
insert into t1 values(1);
```

12. Again we run the select to see the rows items

```
select * from t1;
```

13. We create a new user "myuser" and set the password "mypass" for that user

```
create user myuser with encrypted password 'mypass';
```

14. We grant all privileges to the user for using the mydb database

```
grant all privileges on database mydb to myuser;
```

```
postgres=# grant all privileges on database mydb to myuser;  
GRANT
```

15. We exit. Now we are in the root user

```
exit
```



```
postgres=# exit
root@014275b34d81:/#
```

16. We clear the screen

```
clear
```

17. Connect to the database with the superuser

```
psql -U postgres -h localhost -p 5432 -d mydb
```

Try to create a table and insert a row running these commands:

18. If you cannot create the table then follow these steps:

Grant necessary privileges to the myuser on the public schema

```
GRANT USAGE, CREATE ON SCHEMA public TO myuser;
```

Connect as myuser

```
psql -U myuser -h localhost -p 5432 -d mydb
```

19. Now try creating the table again

```
create table t1(id int);
```

We insert a row in the table

```
insert into t1 values(1);
```

20. Now we check with pgAdmin 4 that the database mydb and the table exist with values

First we have to start the PostgreSQL server. Run this command to start the server:

```
C:\Program Files\PostgreSQL\15\bin>pg_ctl start -D "C:\Program Files\PostgreSQL\15\data" -o "-
```



If you need to stop the server, you can use the following command:

```
C:\Program Files\PostgreSQL\15\bin>pg_ctl stop -D "C:\Program Files\PostgreSQL\15\data"
```

If you need to know the status

```
C:\Program Files\PostgreSQL\15\bin>pg_ctl status -D "C:\Program Files\PostgreSQL\15\data"
```

21. Now we run the application "pg Admin 4"

22. We right click on "Servers" and select the menu option "Register->Server..."

23. Set the server connection values

We enter the server name "localhost", the server port "5432", the database name "mydb", the username "myuser" and the user password "mypass".

We also have to input the connection name "mypostgres"

If you expand the tree you can see the database "mydb" and the table "t1"

You can run the select statement to see the first table rows

pgAdmin 4

File Object Tools Help

Object Explorer

- Publications
  - Schemas (1)
    - public
      - Aggregates
      - Collations
      - Domains
      - FTS Configurations
      - FTS Dictionaries
      - FTS Parsers
      - FTS Templates
      - Foreign Tables
      - Functions
      - Materialized Views
      - Operators
      - Procedures
      - Sequences
      - Tables (1)
        - t1
          - Columns
          - Constraints
          - Indexes
          - RLS Policies

Count Rows

Create

Delete/Drop

Refresh...

Restore...

Backup...

Drop Cascade

Import/Export Data...

Reset Statistics

ERD For Table

Maintenance...

Scripts

Truncate

View/Edit Data

Search Objects...

PSQL Tool

Query Tool

Properties...

Dashboard Properties SQL Statistics Dependence

public.t1/mydb/myuser@mypostgres

No limit

Query Query History

Show queries generated internally by pgAdmin?

Remove Remove All

Today - 29/10/2023

10:54:03

27/10/2023

SELECT \* FROM public.t1 LIMIT 100

14:40:36

14:31:20

public.t1;

ERROR: permission denied for table t1

SQL state: 42501

25. If any problem accessing to table t1 from PostgreSQL then:

pgAdmin 4

File Object Tools Help

Object Explorer

public.t1/mydb/myuser@mypostgres

No limit

Query Query History

Show queries generated internally by pgAdmin?

Remove Remove All

Today - 29/10/2023

10:54:03

27/10/2023

SELECT \* FROM public.t1 LIMIT 100

14:40:36

14:31:20

SELECT id FROM public.t1;

14:30:31

27/10/2023 1 350 msec

14:40:36

Date Rows affected Duration

Copy Copy to Query Editor

SELECT \* FROM public.t1 LIMIT 100

Messages

Successfully run. Total query runtime: 350 msec. 1 rows affected.

ERROR: permission denied for table t1

SQL state: 42501

It seems like the user myuser might not have the necessary privileges on the table t1.

Let's make sure myuser has the right permissions:

Connect to the database as the superuser:

```
psql -U postgres -h localhost -p 5432 -d mydb
```

```
C:\>docker exec -it 014275b34d81 bash
root@014275b34d81:/# psql -U postgres -h localhost -p 5432 -d mydb
psql (16.0 (Debian 16.0-1.pgdg120+1))
Type "help" for help.
```

Grant necessary privileges to myuser on the t1 table:

```
GRANT ALL PRIVILEGES ON TABLE t1 TO myuser;
```

Exit the PostgreSQL prompt:

```
\q
```

Now, connect to the database as myuser:

```
psql -U myuser -h localhost -p 5432 -d mydb
```

Try running the SELECT query again:

```
SELECT id FROM public.t1;
```

```
C:\>docker exec -it 014275b34d81 bash
root@014275b34d81:/# psql -U postgres -h localhost -p 5432 -d mydb
psql (16.0 (Debian 16.0-1.pgdg120+1))
Type "help" for help.

mydb=# GRANT ALL PRIVILEGES ON TABLE t1 TO myuser;
GRANT
mydb=# \q
root@014275b34d81:/# psql -U myuser -h localhost -p 5432 -d mydb
psql (16.0 (Debian 16.0-1.pgdg120+1))
Type "help" for help.

mydb=> SELECT id FROM public.t1;
 id
----
  1
(1 row)
```

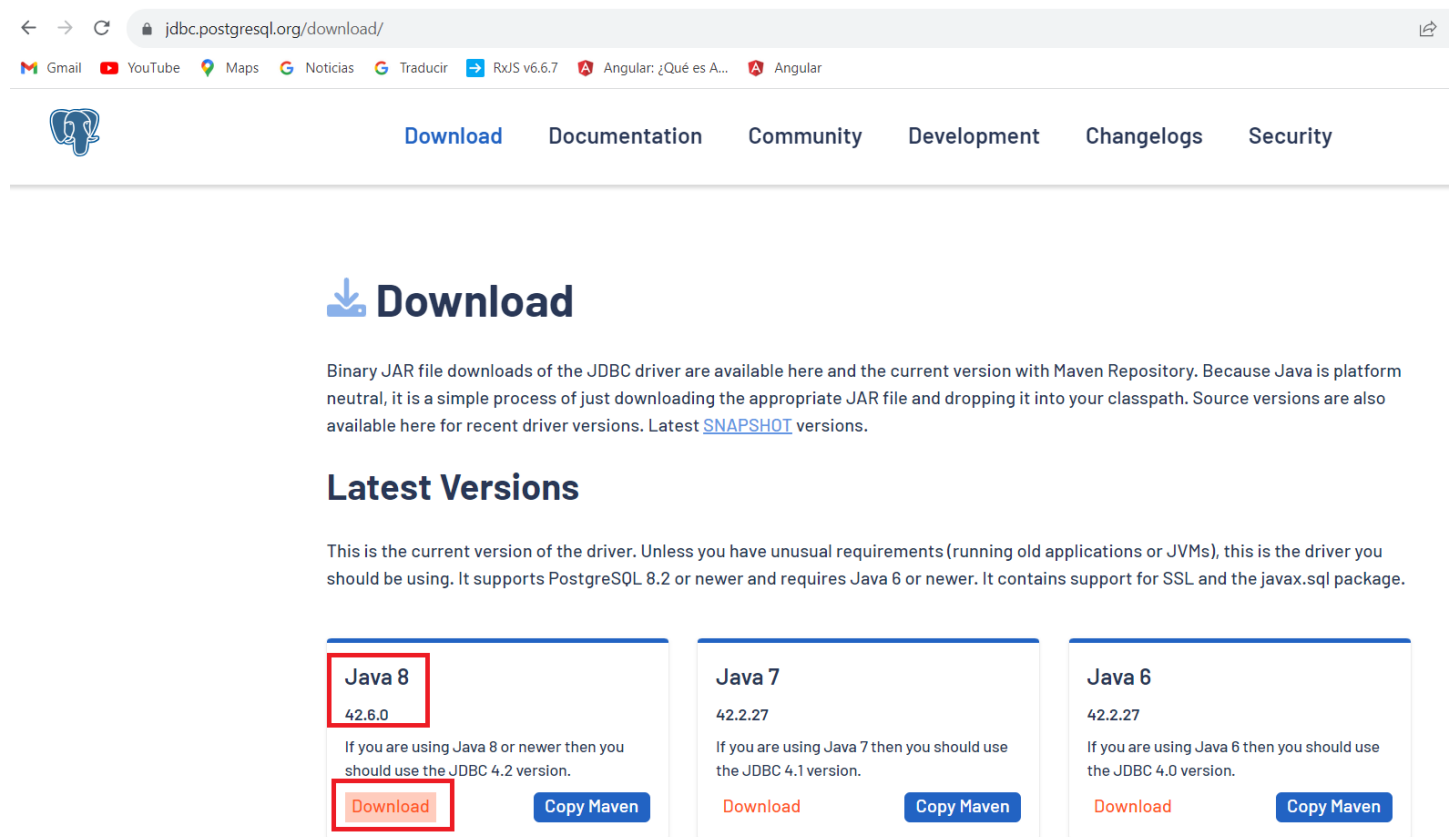
22. Now connect to the PostgreSQL database and table with pgAdmin 4

We

## 1.7. Install PostgreSQL JDBC driver in Spark folder

Download the PostgreSQL JDBC driver from internet

<https://jdbc.postgresql.org/download/>



Binary JAR file downloads of the JDBC driver are available here and the current version with Maven Repository. Because Java is platform neutral, it is a simple process of just downloading the appropriate JAR file and dropping it into your classpath. Source versions are also available here for recent driver versions. Latest [SNAPSHOT](#) versions.

### Latest Versions

This is the current version of the driver. Unless you have unusual requirements (running old applications or JVMs), this is the driver you should be using. It supports PostgreSQL 8.2 or newer and requires Java 6 or newer. It contains support for SSL and the javax.sql package.

Java 8	Java 7	Java 6
42.6.0	42.2.27	42.2.27
If you are using Java 8 or newer then you should use the JDBC 4.2 version.	If you are using Java 7 then you should use the JDBC 4.1 version.	If you are using Java 6 then you should use the JDBC 4.0 version.
<a href="#">Download</a> <a href="#">Copy Maven</a>	<a href="#">Download</a> <a href="#">Copy Maven</a>	<a href="#">Download</a> <a href="#">Copy Maven</a>

Place the jar file "postgresql-42.6.0.jar" inside the path spark jars folder: C:\spark-3.5.0-bin-hadoop3\jars

## 2. Run the application in IntelliJ

### 2.1. Prerequisites

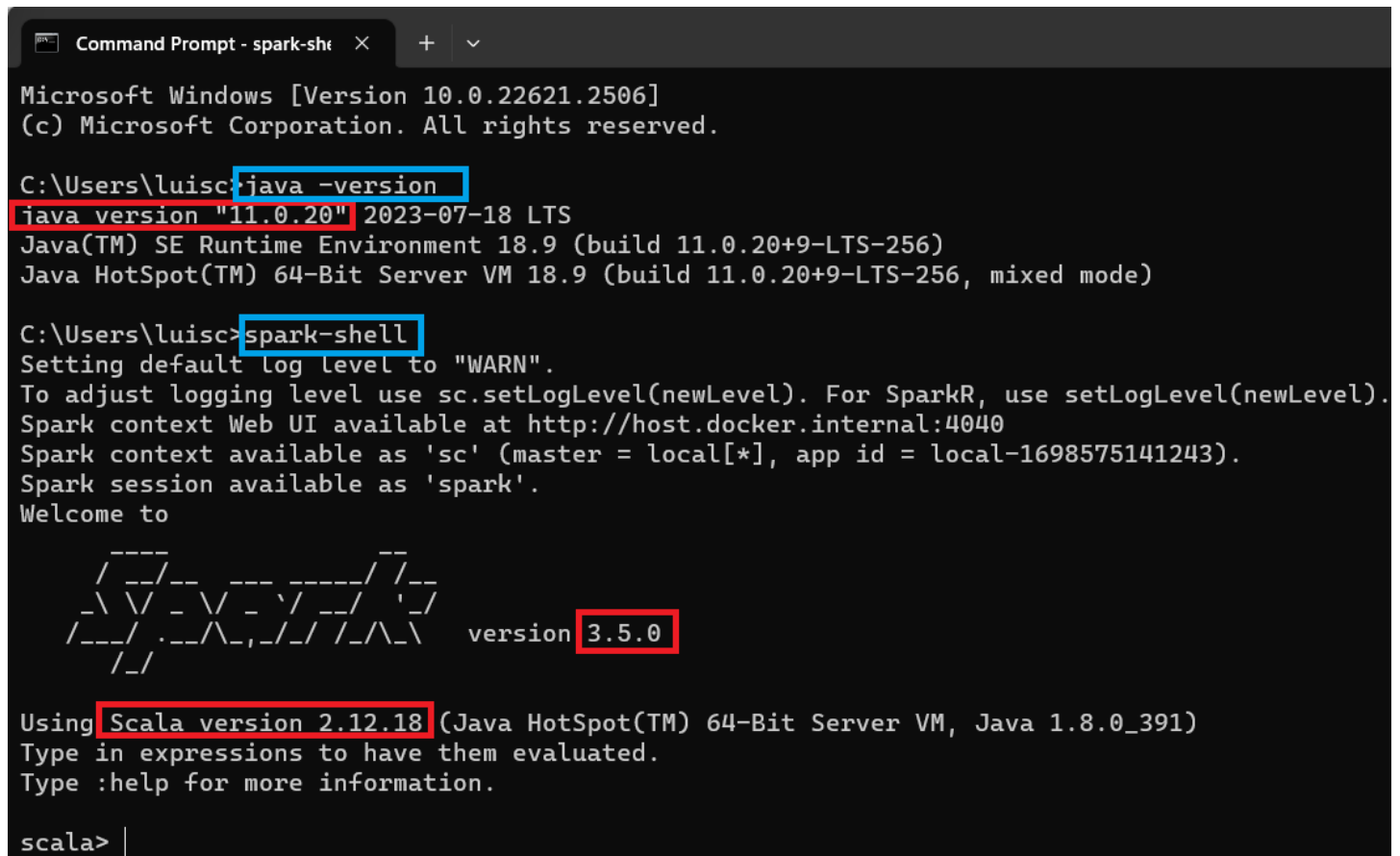
Before running IntelliJ we have to check the Java and Spark installations.

To see the Java version open a command prompt and run the command:

```
java -version
```

Then we also run the command:

```
spark-shell
```



```
Command Prompt - spark-shell X + v
Microsoft Windows [Version 10.0.22621.2506]
(c) Microsoft Corporation. All rights reserved.

C:\Users\luisc>java -version
java version "11.0.20" 2023-07-18 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.20+9-LTS-256)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.20+9-LTS-256, mixed mode)

C:\Users\luisc>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://host.docker.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1698575141243).
Spark session available as 'spark'.
Welcome to

  ____      _
 / ___|    / \
 \___ \  __/ __\
  ___) | /_/\_ \
 / ___|/ __// __ \
 \___|\___|\___|_|\_\

version 3.5.0

Using Scala version 2.12.18 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_391)
Type in expressions to have them evaluated.
Type :help for more information.

scala> |
```

We copy the Java, Spark and Scala versions in order to use this data later when creating out new Spark Scala project in IntelliJ.

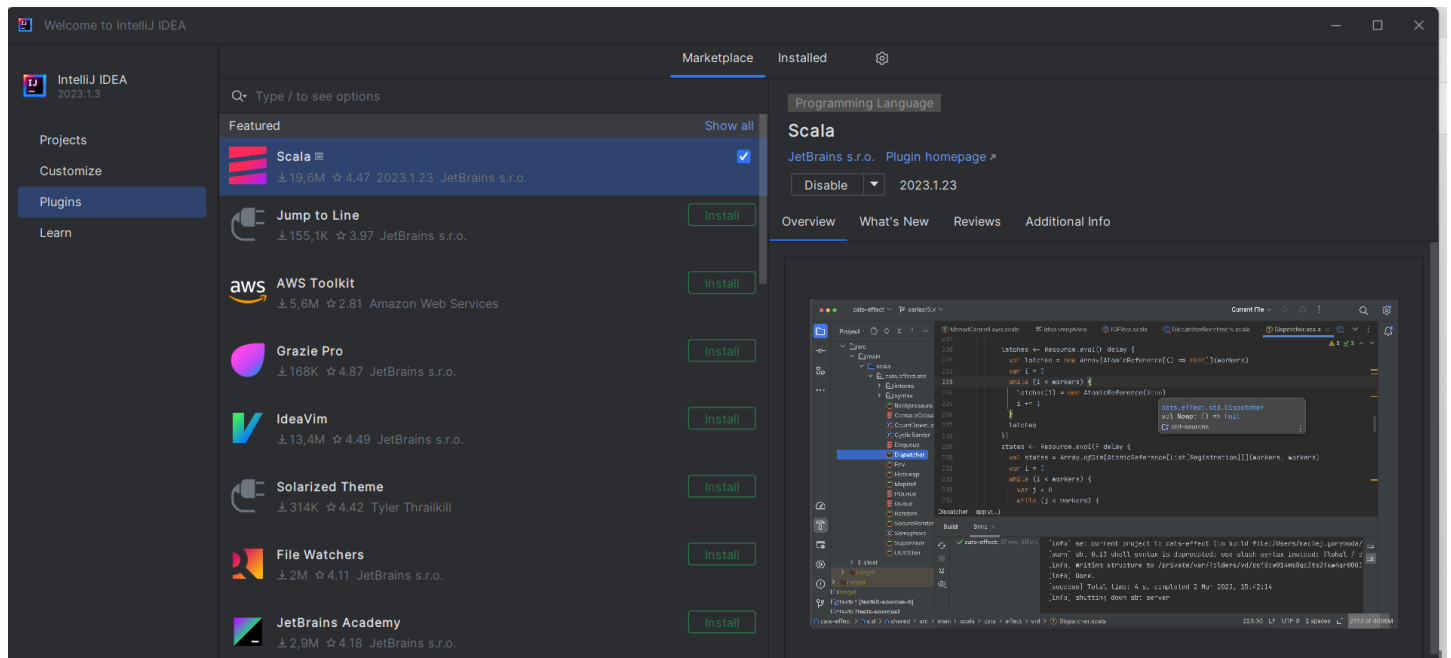
Java version: 11

Spark version: 3.5.0

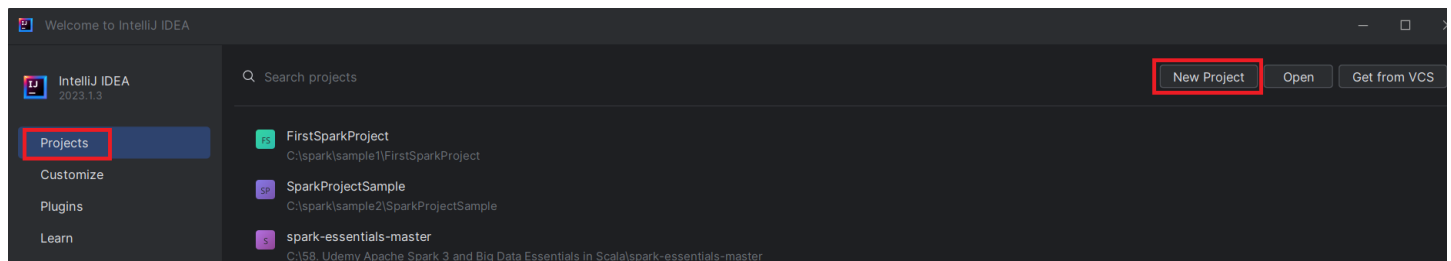
Scala version: 2.12.18

## 2.2. Run IntelliJ Community IDE.

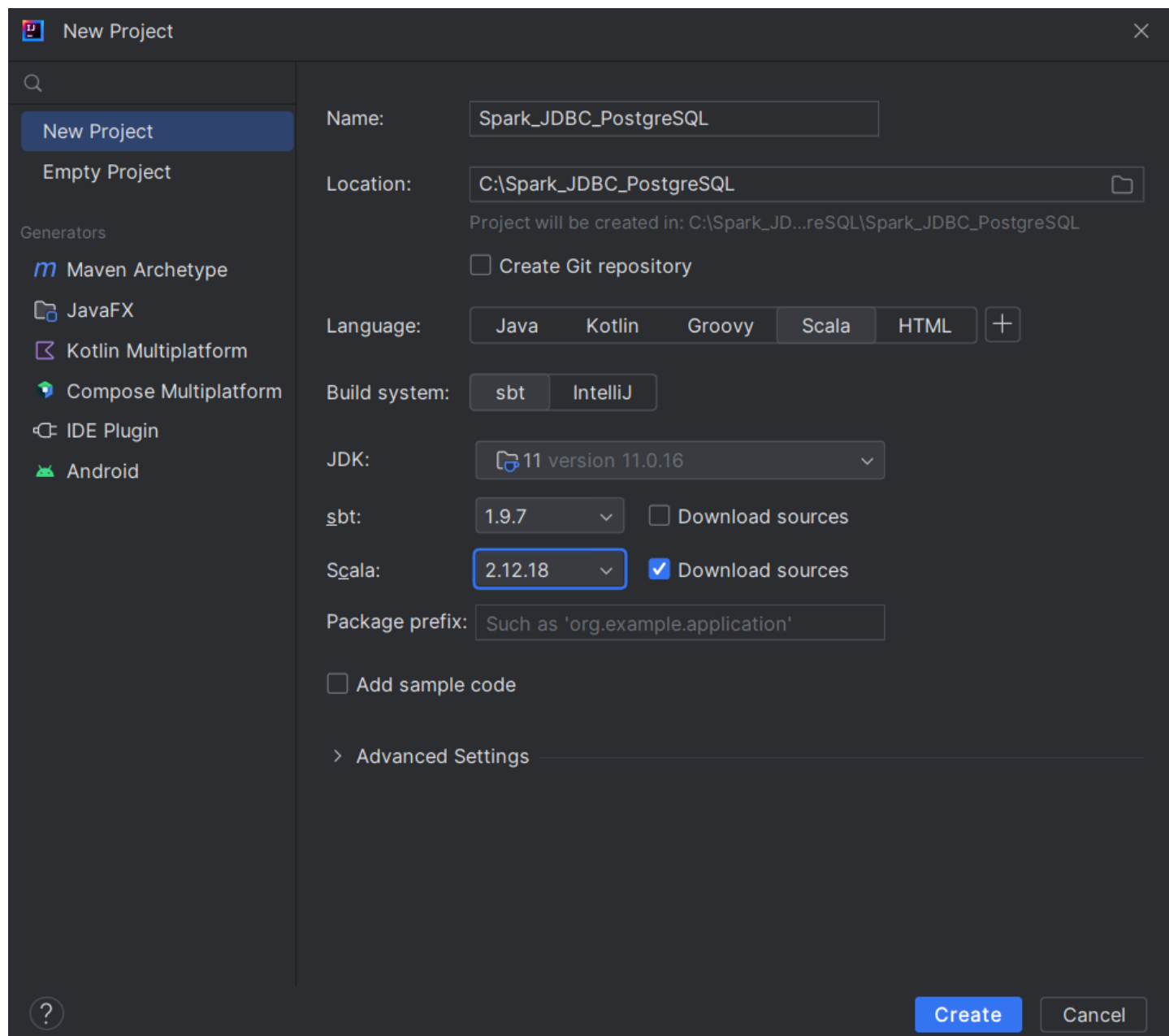
Install the "Scala" plugin



## Create a new project

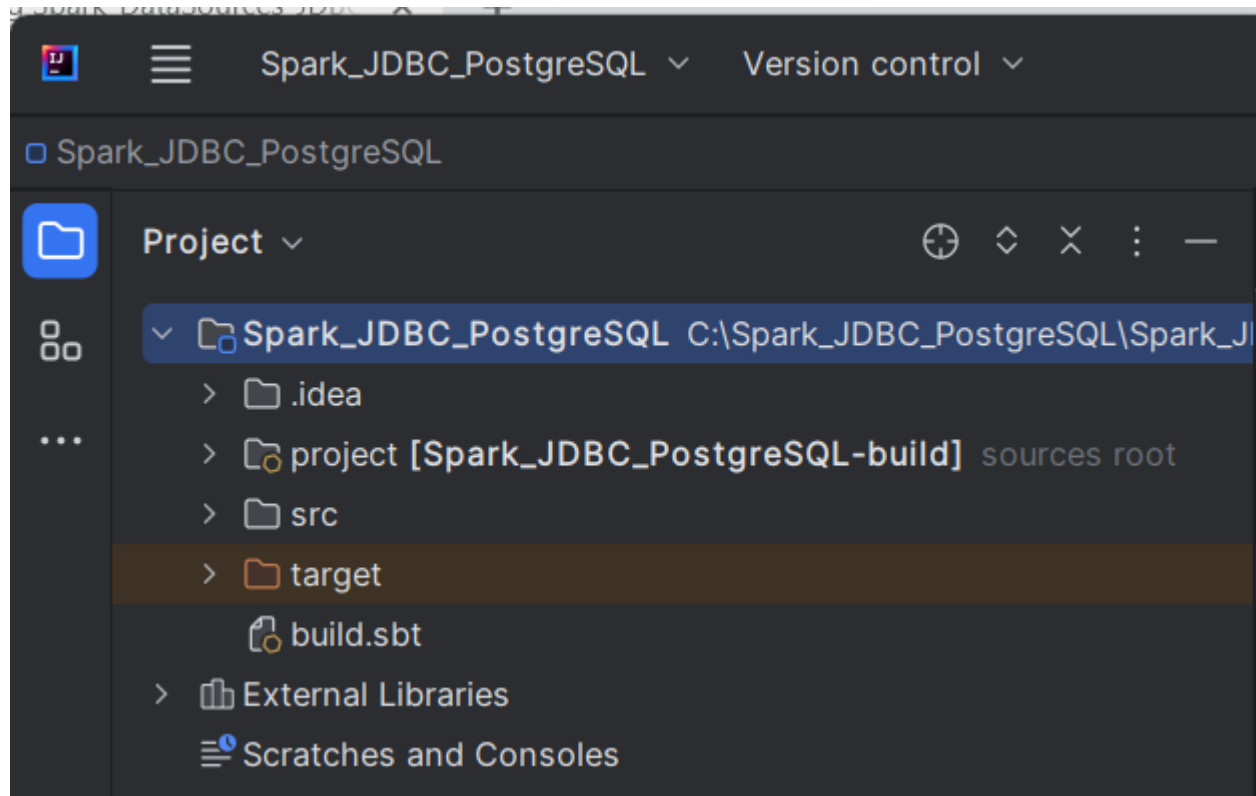


## Enter the new project input data



Then we can see the new project in IntelliJ





## 2.3. We input the build.sbt file source code

```
ThisBuild / version := "0.1.0-SNAPSHOT"

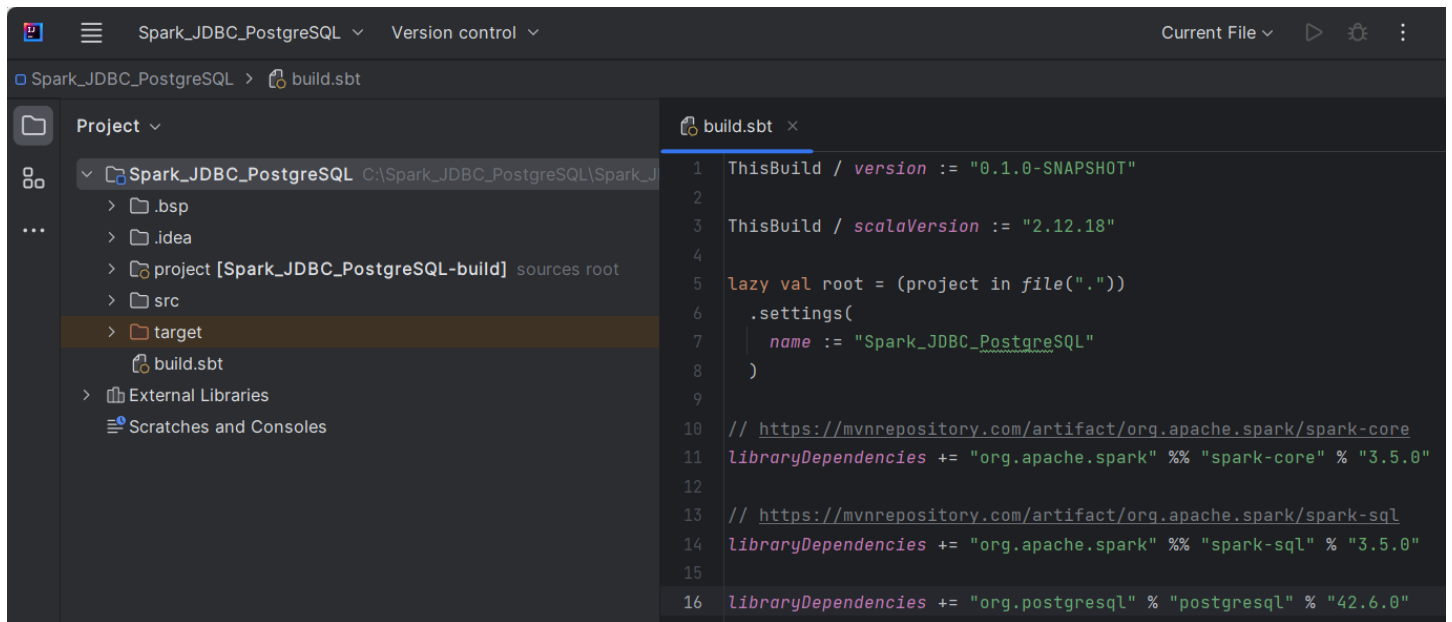
ThisBuild / scalaVersion := "2.12.18"

lazy val root = (project in file("."))
  .settings(
    name := "Spark_JDBC_PostgreSQL"
  )

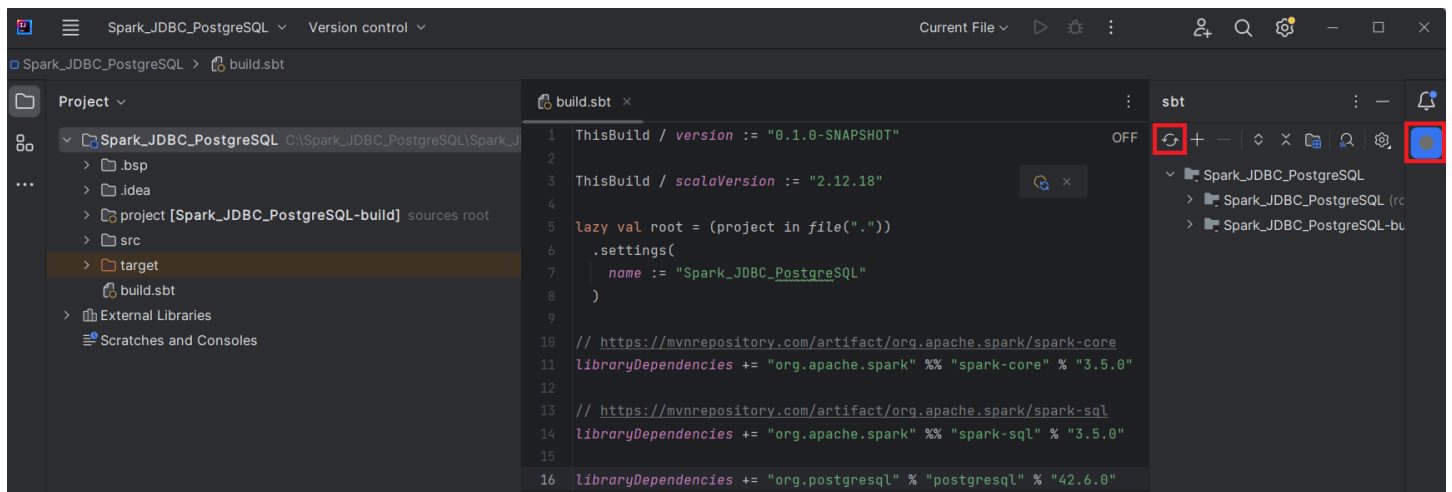
// https://mvnrepository.com/artifact/org.apache.spark/spark-core
libraryDependencies += "org.apache.spark" %% "spark-core" % "3.5.0"

// https://mvnrepository.com/artifact/org.apache.spark/spark-sql
libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.5.0"

libraryDependencies += "org.postgresql" % "postgresql" % "42.6.0"
```



Then we press the sbt button in the right hand side menu and the reload project button



After reloading the project dependencies we will see this result

```

1 ThisBuild / version := "0.1.0-SNAPSHOT"
2
3 ThisBuild / scalaVersion := "2.12.18"
4
5 lazy val root = (project in file("."))
6   .settings(
7     name := "Spark_JDBC_PostgreSQL"
8   )
9
10 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
11 libraryDependencies += "org.apache.spark" %% "spark-core" % "3.5.0"
12
13 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
14 libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.5.0"
15
16 libraryDependencies += "org.postgresql" % "postgresql" % "42.6.0"
17

```

Build Sync

✓ Spark\_JDBC\_PostgreSQL: finished .47 sec, 264 ms

```

[info] The new values will be used by cleanKeepGlobs
[info] Run 'last' for details.
[info] Reapplying settings...
[info] set current project to Spark_JDBC_PostgreSQL (in build file:/C:/Spark_JDBC_PostgreSQL/Spark_JDBC_PostgreSQL/)
[info] Applying State transformations org.jetbrains.sbt.CreateTasks from
C:/Users/luisc/AppData/Roaming/JetBrains/IdeaIC2023.1/plugins/Scala/repo/org/jetbrains/sbt-structure-extractor_2
.12.1.0/2023.1.0/sbt-structure-extractor-2023.1.0.jar
[info] Reapplying settings...
[info] set current project to Spark_JDBC_PostgreSQL (in build file:/C:/Spark_JDBC_PostgreSQL/Spark_JDBC_PostgreSQL/)
[warn] sbt 0.13 shell syntax is deprecated; use slash syntax instead: Global / dumpStructure

```

Microsoft Defender configuration: Project paths were successfully added to the Microsoft Defender exclusion list (6 minutes ago)

**IMPORTANT NOTE:** How to look for libraries dependencies in internet

In **Maven repository** we can find the dependencies in internet. For example if we are looking for the "org.postgresql"

Editing Spark\_DataSources\_JDBC x org.postgresql - Search x MVN Maven Repository: org.postgresql x +

bing.com/search?EID=MBSE&FORM=HI4CDF&PC=HI4C&q=org.postgresql

Gmail YouTube Maps Noticias Traducir RxJS v6.7 Angular: ¿Qué es A... Angular

Maven Repository  
https://mvnrepository.com/artifact/org.postgresql/postgresql

**Maven Repository: org.postgresql » postgresql**

Web **PostgreSQL JDBC Driver PostgreSQL**. License. BSD 2-clause. Categories. JDBC Drivers.  
Tags. database sql jdbc postgresql driver. Ranking. #115 in MvnRepository ( See Top ...

We press in the version link, in this case we press in the 42.6.0 link

mvnrepository.com/artifact/org.postgresql/postgresql

Mail YouTube Maps Noticias Traducir RxJS v6.6.7 Angular: ¿Qué es A... Angular

# MVN REPOSITORY

Search for groups, artifacts, categories

Home » org.postgresql » postgresql

## PostgreSQL JDBC Driver

PostgreSQL JDBC Driver Postgresql

License	BSD 2-clause
Categories	JDBC Drivers
Tags	database sql jdbc postgresql driver
Ranking	#115 in MvnRepository (See Top Artifacts) #2 in JDBC Drivers
Used By	4,152 artifacts

Central (177) Atlassian (1) Redhat GA (11) Redhat EA (3) HuaweiCloudSDK (1) Brekka (1) OrbisGIS (1) ICM (3)

Version	Vulnerabilities	Repository	Usages	Date
42.6.x				
42.6.0		Central	471	Mar 18, 2023
42.5.4		Central	202	Feb 16, 2023

And then we select the SBT tab and copy the dependency reference in our build.sbt file

Home » org.postgresql » postgresql » 42.6.0

## PostgreSQL JDBC Driver » 42.6.0

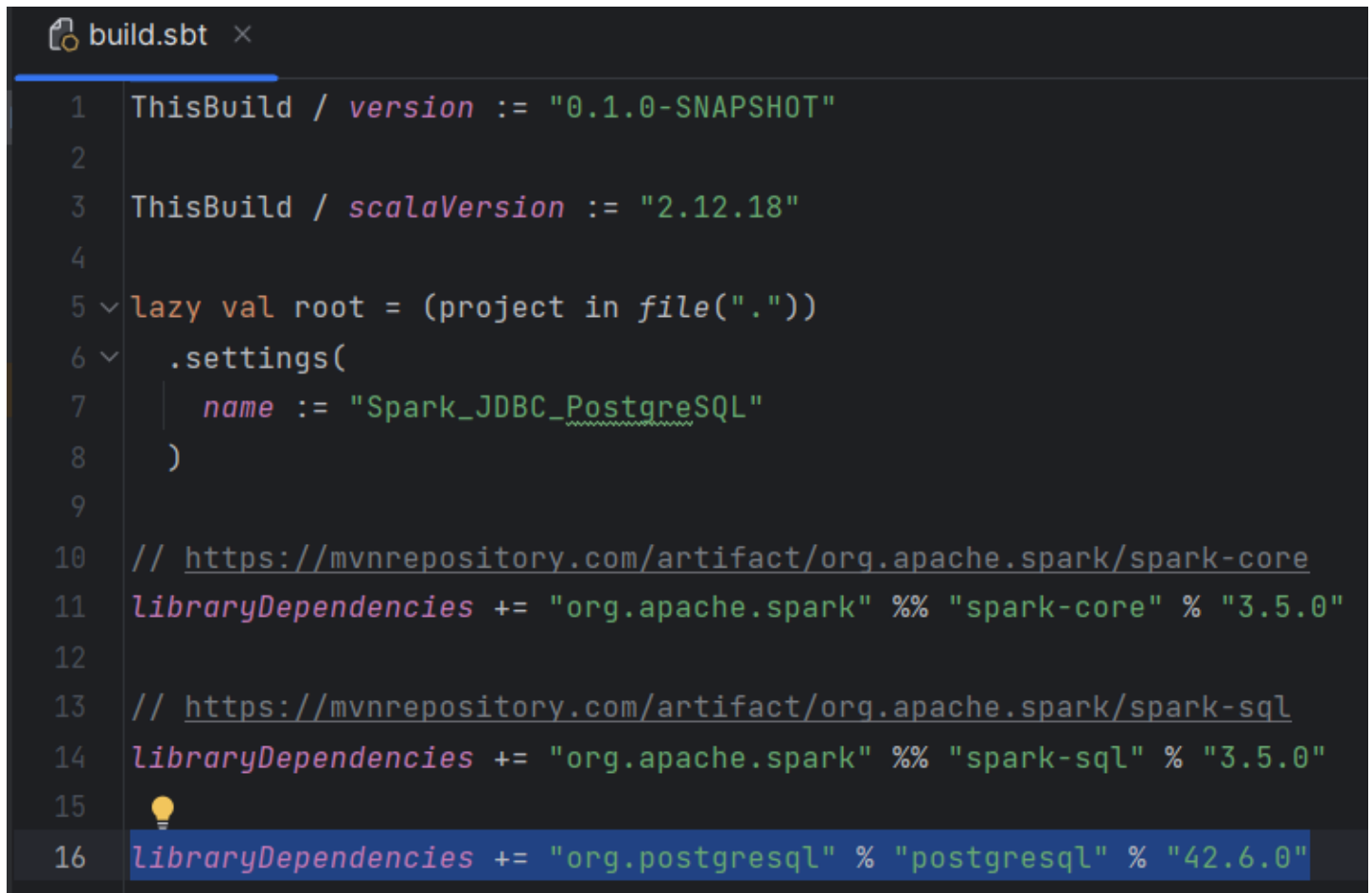
PostgreSQL JDBC Driver Postgresql

License	BSD 2-clause
Categories	JDBC Drivers
Tags	database sql jdbc postgresql driver
Organization	PostgreSQL Global Development Group
HomePage	https://jdbc.postgresql.org
Date	Mar 18, 2023
Files	<a href="#">pom (2 KB)</a> <a href="#">jar (1.0 MB)</a> <a href="#">View All</a>
Repositories	Central
Ranking	#115 in MvnRepository (See Top Artifacts) #2 in JDBC Drivers
Used By	4,152 artifacts

Maven Gradle Gradle (Short) Gradle (Kotlin) **SBT** Ivy Grape Leiningen Buildr

```
// https://mvnrepository.com/artifact/org.postgresql/postgresql
libraryDependencies += "org.postgresql" % "postgresql" % "42.6.0"
```

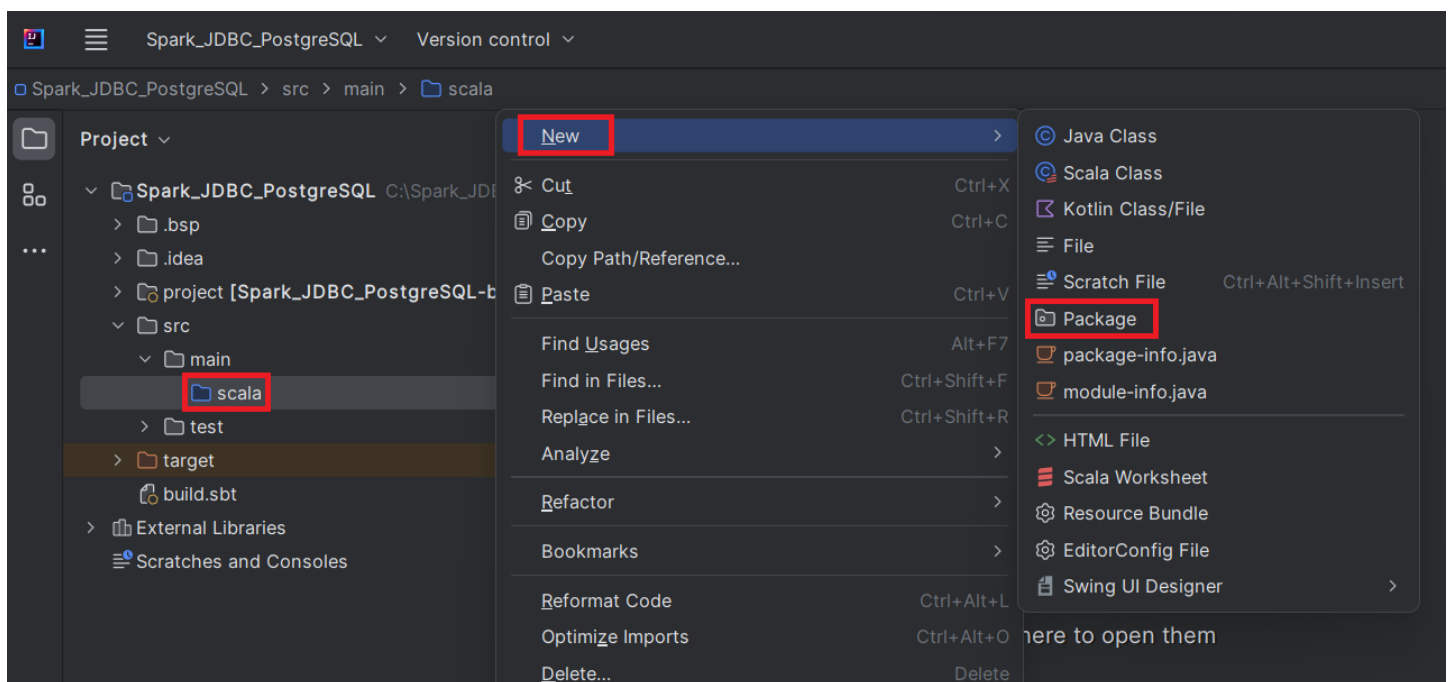
Then we go to the build.sbt file and we copy the library dependency code:

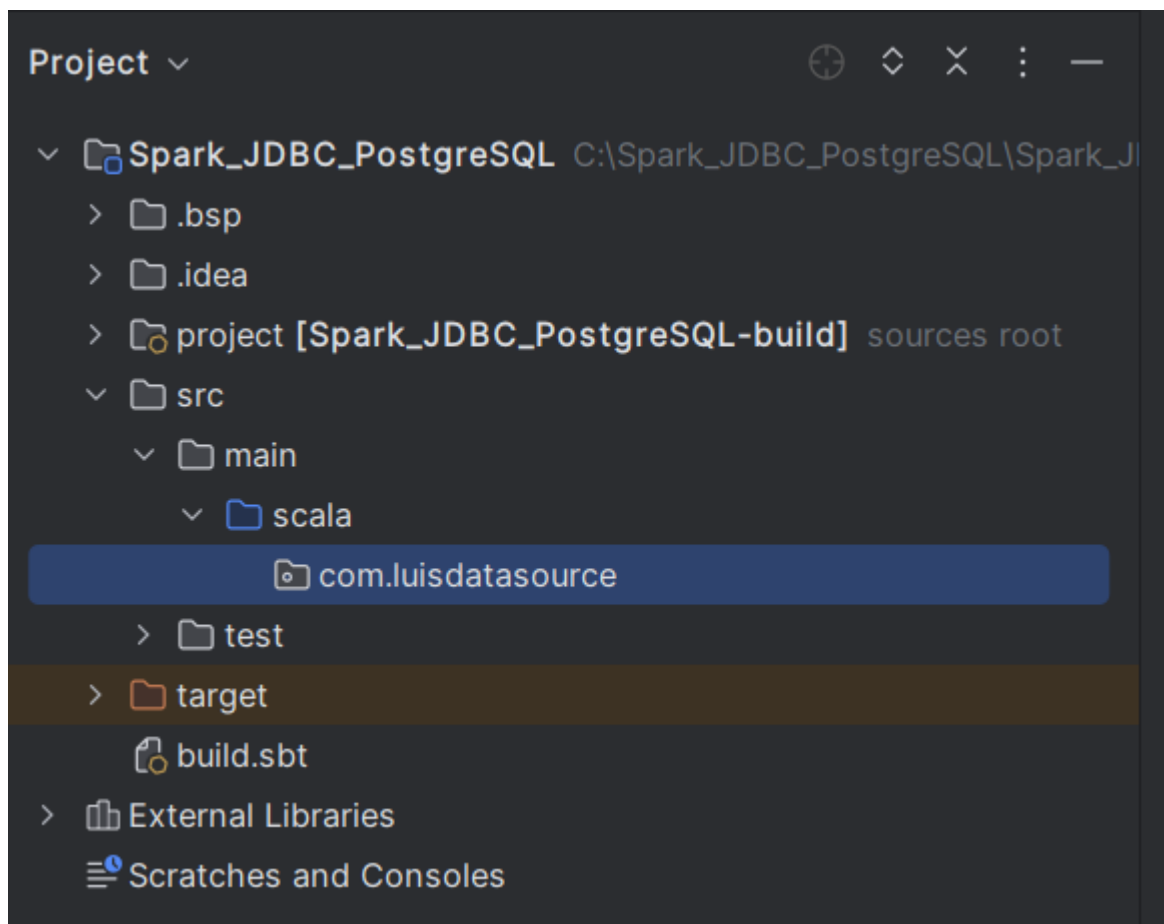
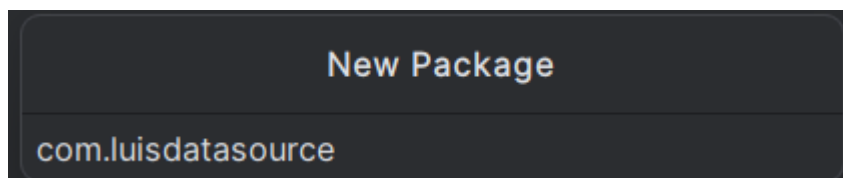


```
1 ThisBuild / version := "0.1.0-SNAPSHOT"
2
3 ThisBuild / scalaVersion := "2.12.18"
4
5 lazy val root = (project in file("."))
6   .settings(
7     name := "Spark_JDBC_PostgreSQL"
8   )
9
10 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
11 libraryDependencies += "org.apache.spark" %% "spark-core" % "3.5.0"
12
13 // https://mvnrepository.com/artifact/org.apache.spark/spark-sql
14 libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.5.0"
15
16 libraryDependencies += "org.postgresql" % "postgresql" % "42.6.0"
```

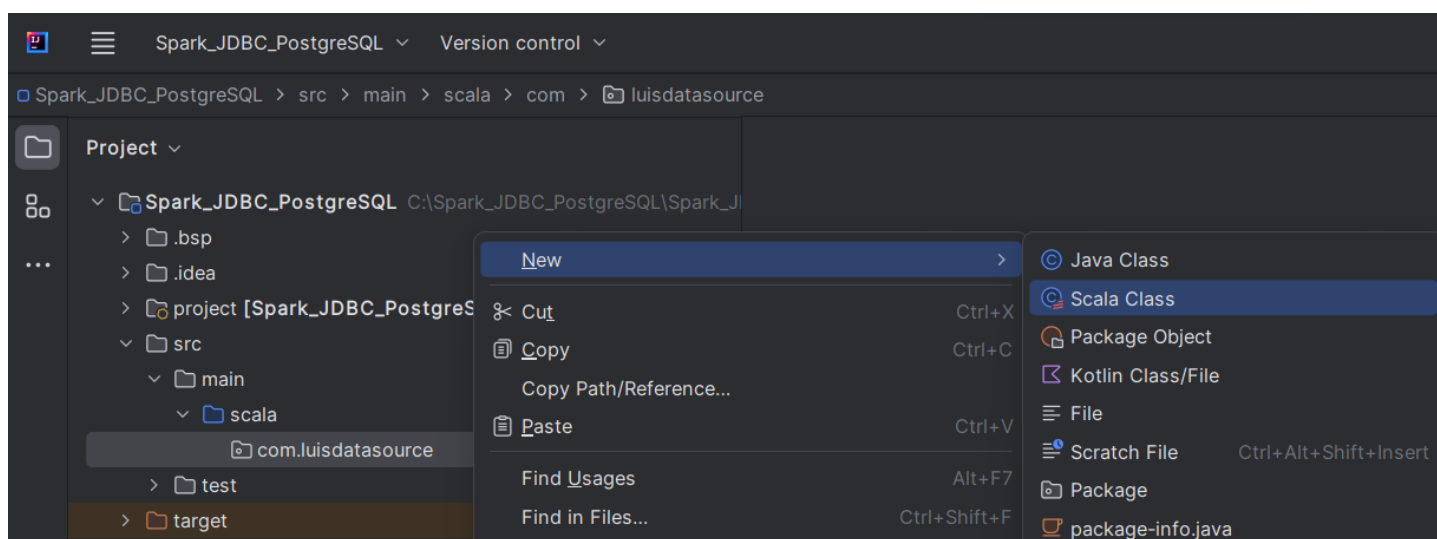
## 2.4. How to create a new package and new file inside the package

First we create the package. For that we right click on the scala folder and we select the menu options New->Package and we input the new package name

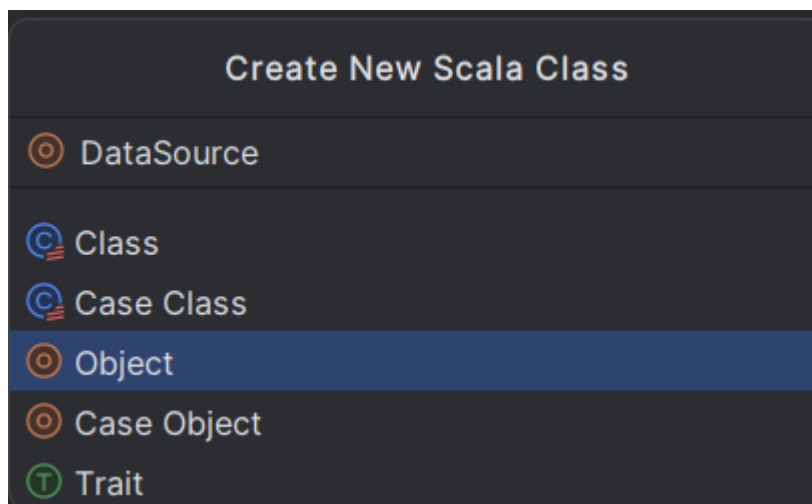




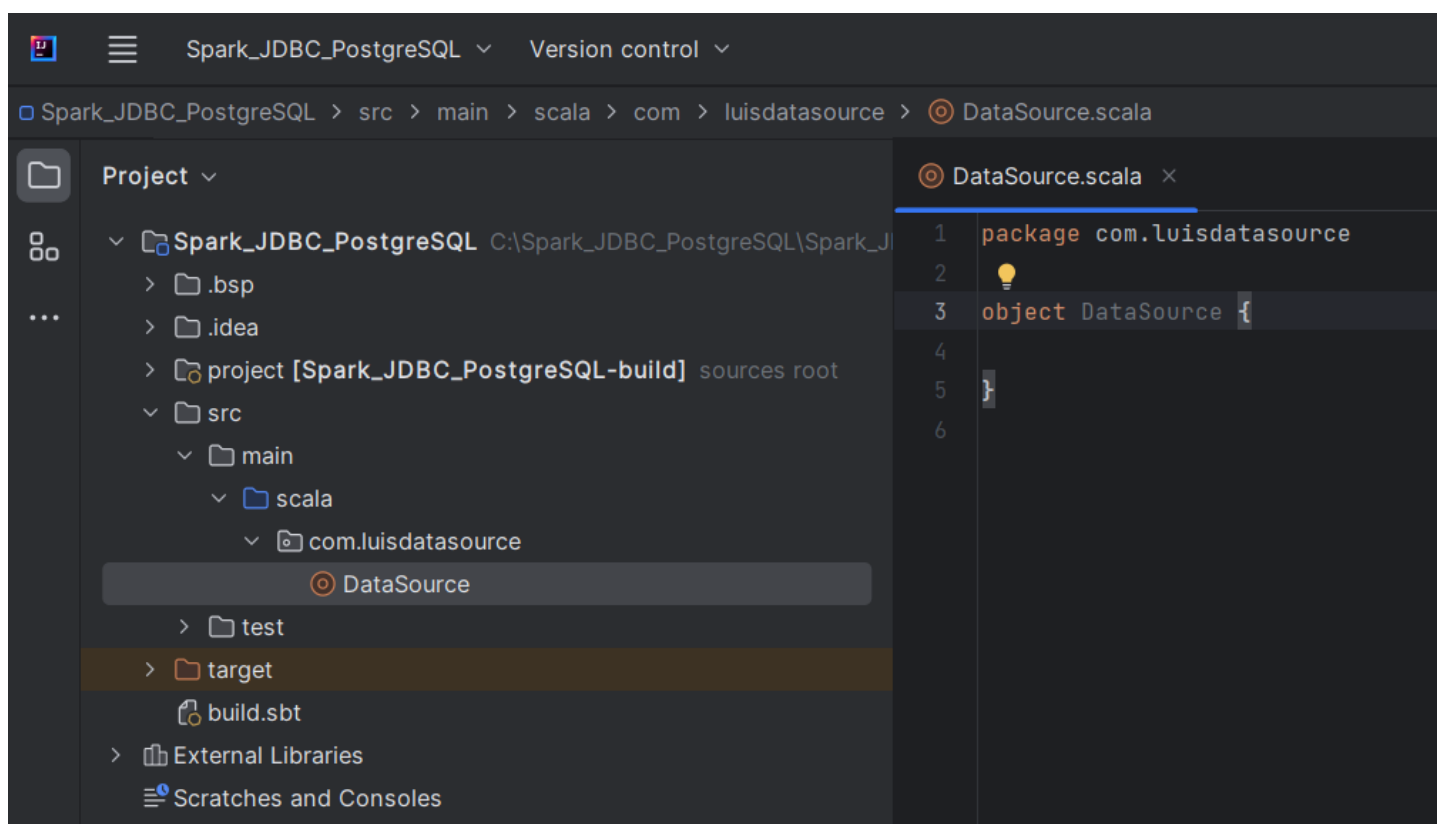
Now we create a new file inside the package. We select the menu options "New->Scala Class" and the "Object"



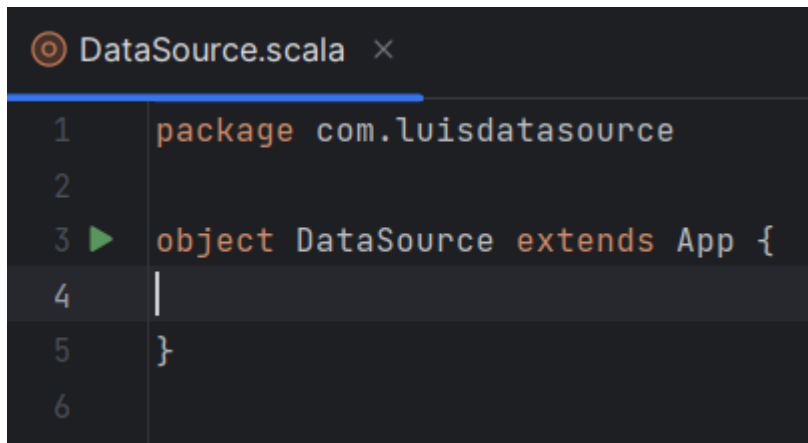
We input the new "Object" name and we set as "DataSource", an arbitrary name we invented



Now you can see the new package and inside the new file "DataSource.scala" with the new object "DataSource"



We "extends App" for the "DataSource" object. And we can see the execute green button appears in the left margin.



```
DataSource.scala x
1 package com.luisdatasource
2
3 object DataSource extends App {
4
5 }
6
```

Now we can input the rest of the application source code.

## 2.5. See the scala source code: scala\com.sparkExample1\DataSources.scala file

---

```
package com.luisdatasource

import org.apache.spark.sql.{SaveMode, SparkSession}
import org.apache.spark.sql.types._

object DataSource extends App {

  val spark = SparkSession.builder()
    .appName("Data Sources and Formats")
    .config("spark.master", "local")
    .getOrCreate()

  val carsSchema = StructType(Array(
    StructField("Name", StringType),
    StructField("Miles_per_Gallon", DoubleType),
    StructField("Cylinders", LongType),
    StructField("Displacement", DoubleType),
    StructField("Horsepower", LongType),
    StructField("Weight_in_lbs", LongType),
    StructField("Acceleration", DoubleType),
    StructField("Year", DateType),
    StructField("Origin", StringType)
  ))

  /*
   Reading a DF:
   - format
   - schema or inferSchema = true
   - path
   - zero or more options
   */
  val carsDF = spark.read
    .format("json")
}
```



```
.schema(carsSchema) // enforce a schema
.option("mode", "failFast") // dropMalformed, permissive (default)
.option("path", "src/main/resources/data/cars.json")
.load()

// alternative reading with options map
val carsDFWithOptionMap = spark.read
  .format("json")
  .options(Map(
    "mode" -> "failFast",
    "path" -> "src/main/resources/data/cars.json",
    "inferSchema" -> "true"
  ))
  .load()

/*
Writing DFs
- format
- save mode = overwrite, append, ignore, errorIfExists
- path
- zero or more options
*/
carsDF.write
  .format("json")
  .mode(SaveMode.Overwrite)
  .save("src/main/resources/data/cars_dupe.json")

// JSON flags
spark.read
  .schema(carsSchema)
  .option("dateFormat", "yyyy-MM-dd") // couple with schema; if Spark fails parsing, it will
  .option("allowSingleQuotes", "true")
  .option("compression", "uncompressed") // bzip2, gzip, lz4, snappy, deflate
  .json("src/main/resources/data/cars.json")

// CSV flags
val stocksSchema = StructType(Array(
  StructField("symbol", StringType),
  StructField("date", DateType),
  StructField("price", DoubleType)
))

spark.read
  .schema(stocksSchema)
  .option("dateFormat", "MMM d yyyy")
  .option("header", "true")
  .option("sep", ",")
  .option("nullValue", "")
  .csv("src/main/resources/data/stocks.csv")

// Parquet
carsDF.write
  .mode(SaveMode.Overwrite)
```

```
.save("src/main/resources/data/cars.parquet")

// Text files
spark.read.text("src/main/resources/data/sampleTextFile.txt").show()

// Reading from a remote DB
val driver = "org.postgresql.Driver"
val url = "jdbc:postgresql://localhost:5432/mydb"
val user = "myuser"
val password = "mypass"

val employeesDF = spark.read
  .format("jdbc")
  .option("driver", driver)
  .option("url", url)
  .option("user", user)
  .option("password", password)
  .option("dbtable", "public.t1")
  .load()

employeesDF.show()

/**
 * Exercise: read the movies DF, then write it as
 * - tab-separated values file
 * - snappy Parquet
 * - table "public.movies" in the Postgres DB
 */

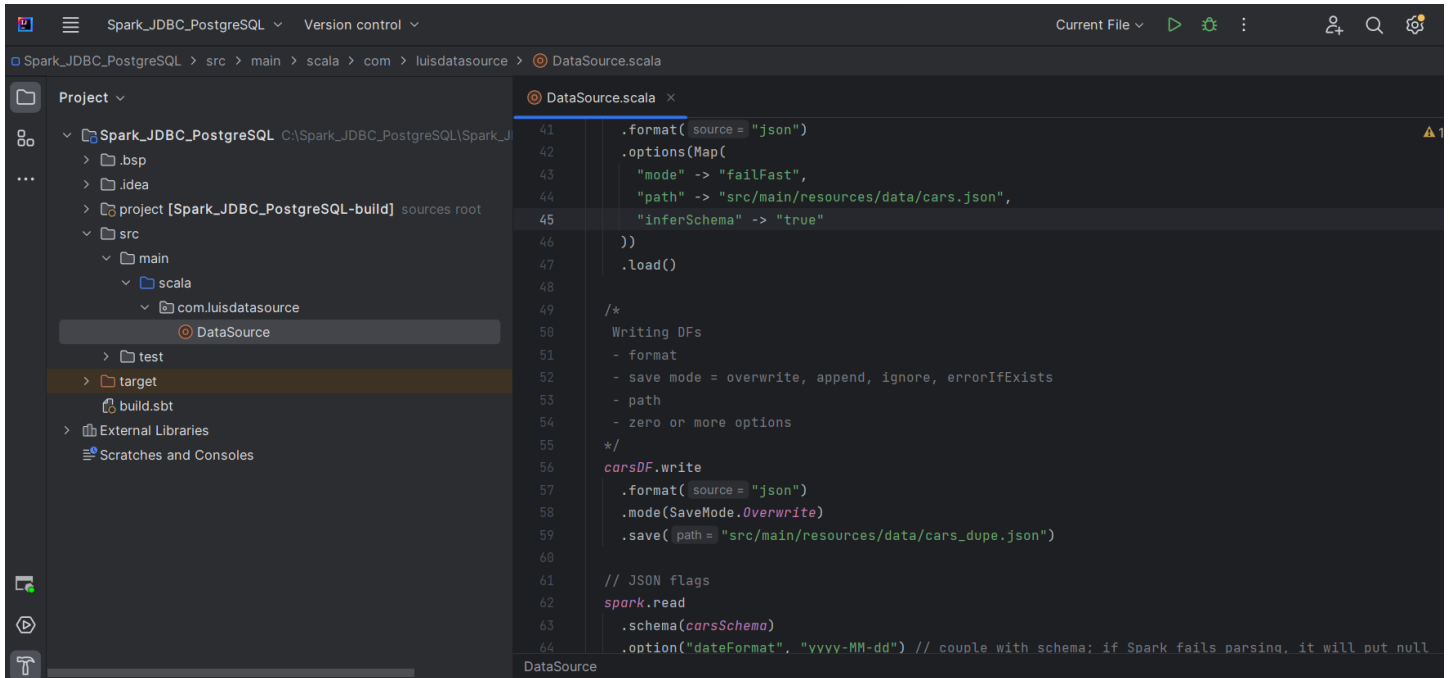
val moviesDF = spark.read.json("src/main/resources/data/movies.json")

// TSV
moviesDF.write
  .format("csv")
  .option("header", "true")
  .option("sep", "\t")
  .save("src/main/resources/data/movies.csv")

// Parquet
moviesDF.write.save("src/main/resources/data/movies.parquet")

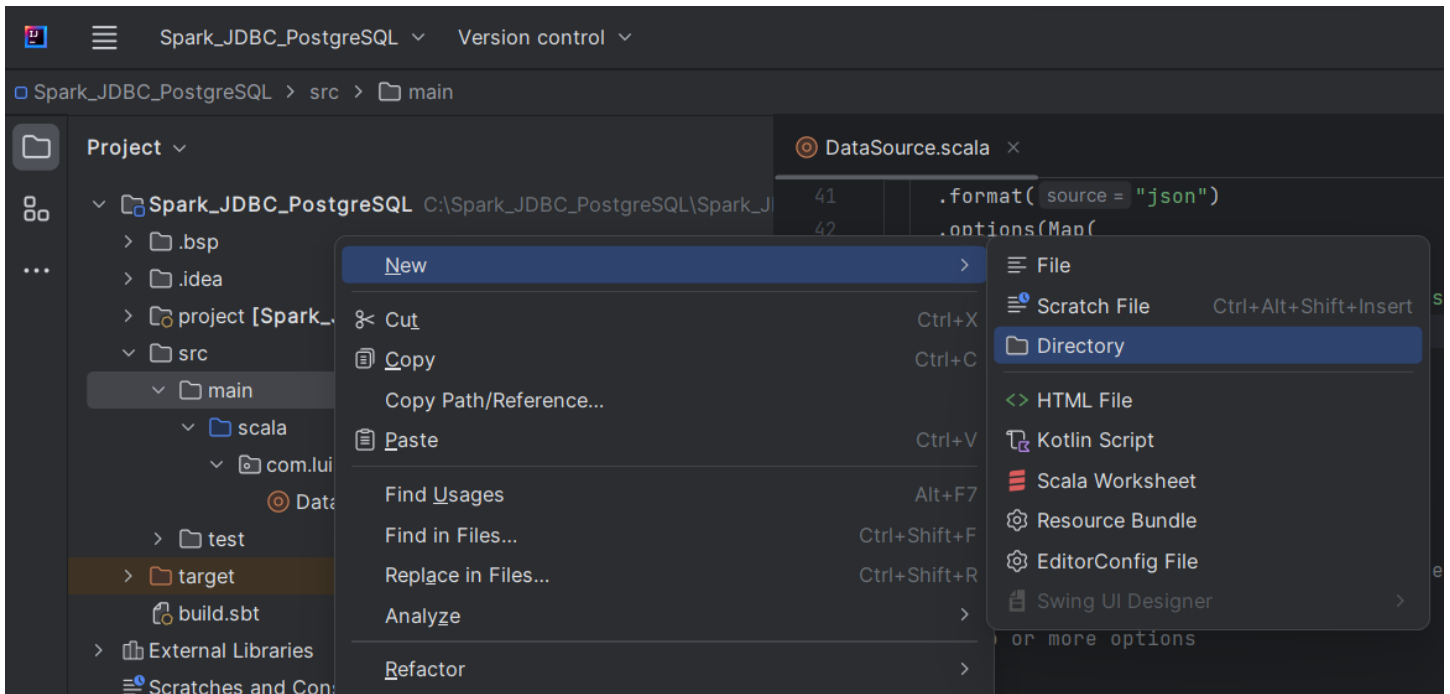
// save to DF
/**
moviesDF.write
  .format("jdbc")
  .option("driver", driver)
  .option("url", url)
  .option("user", user)
  .option("password", password)
  .option("dbtable", "public.t1")
  .save()
```

```
*/
}
```

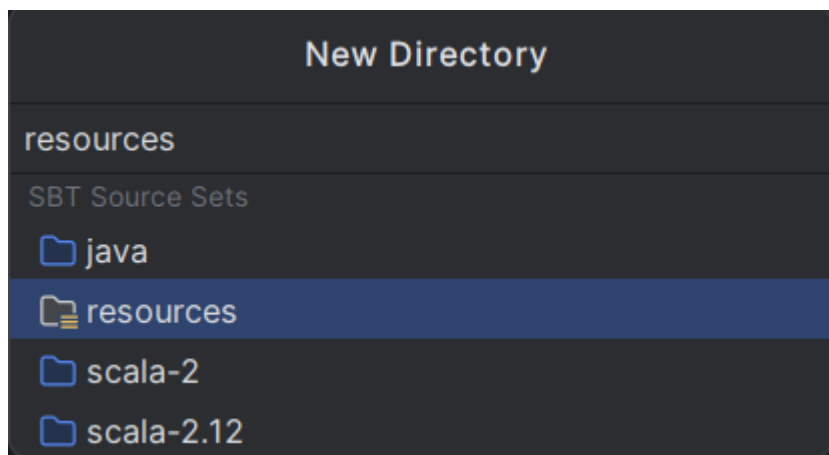


## 2.6. We create the resource folder to place the data

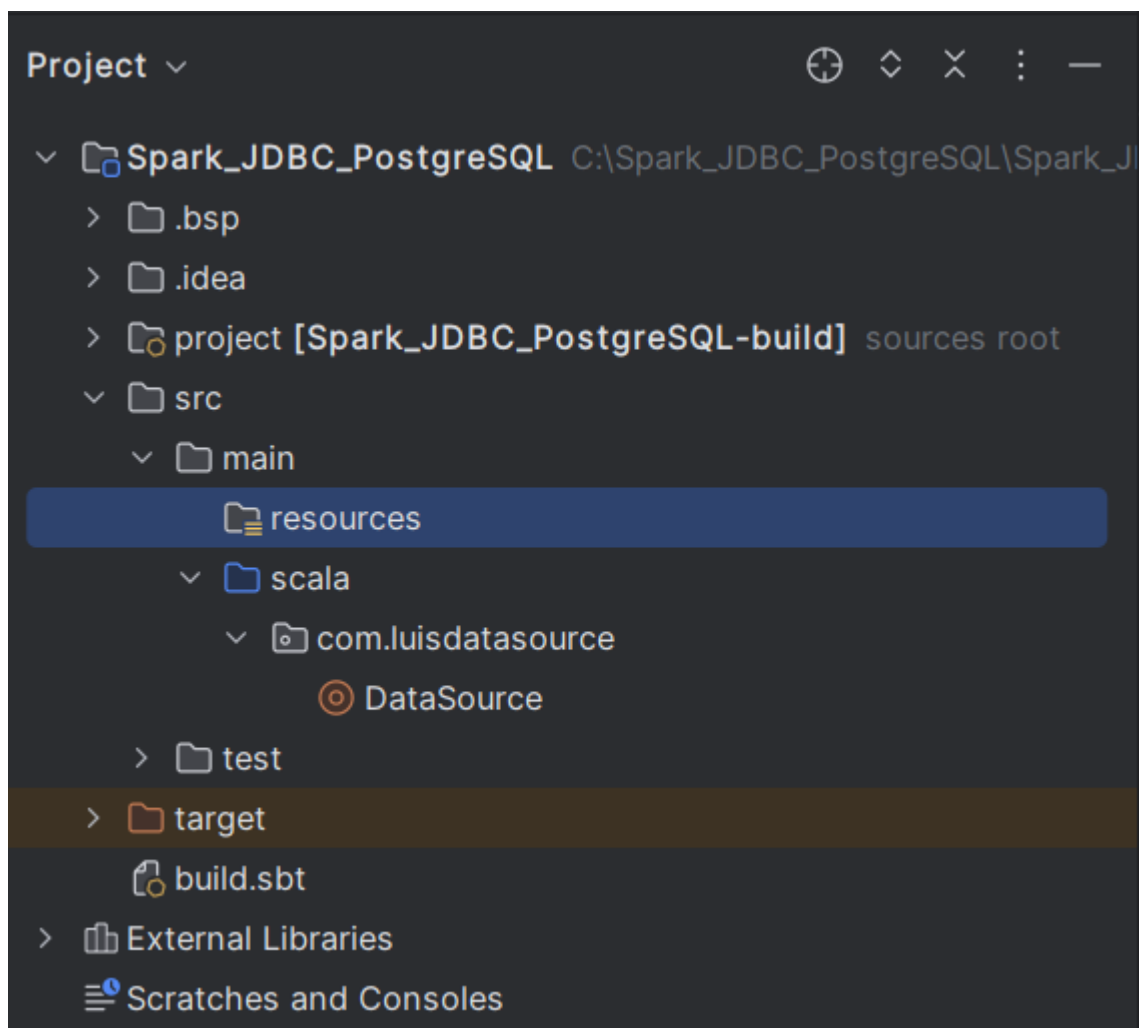
We right click on the "main" folder and we select the menu options "New->Directory"



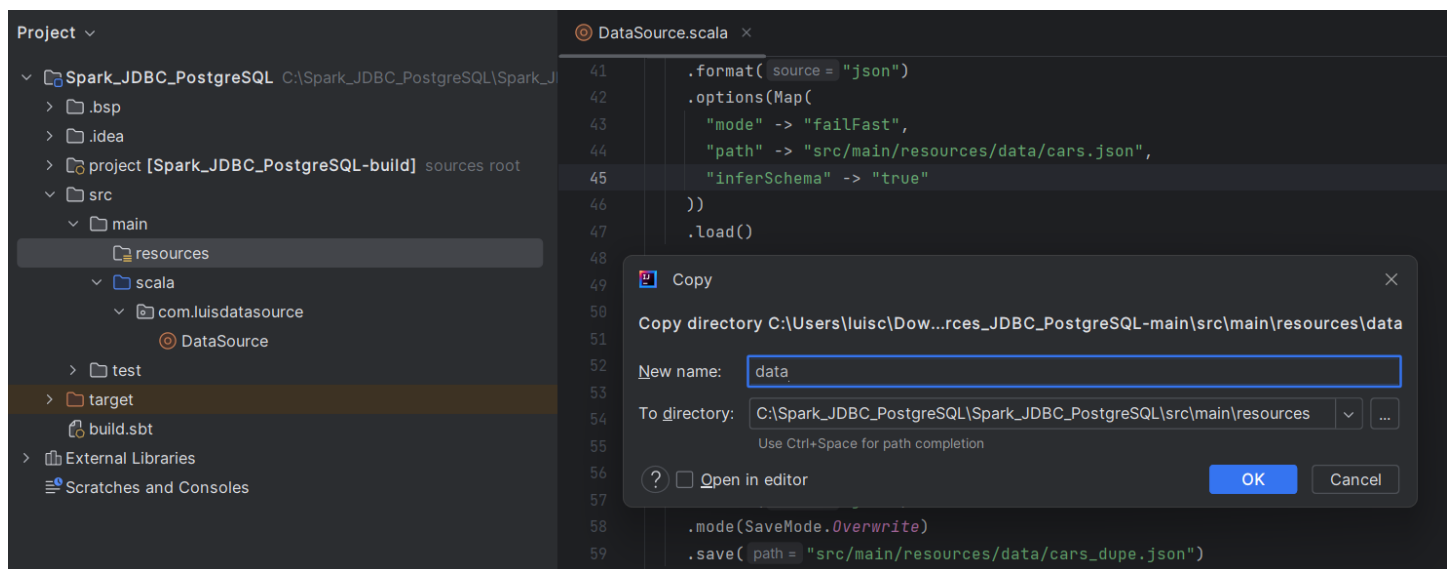
The we select "resources"



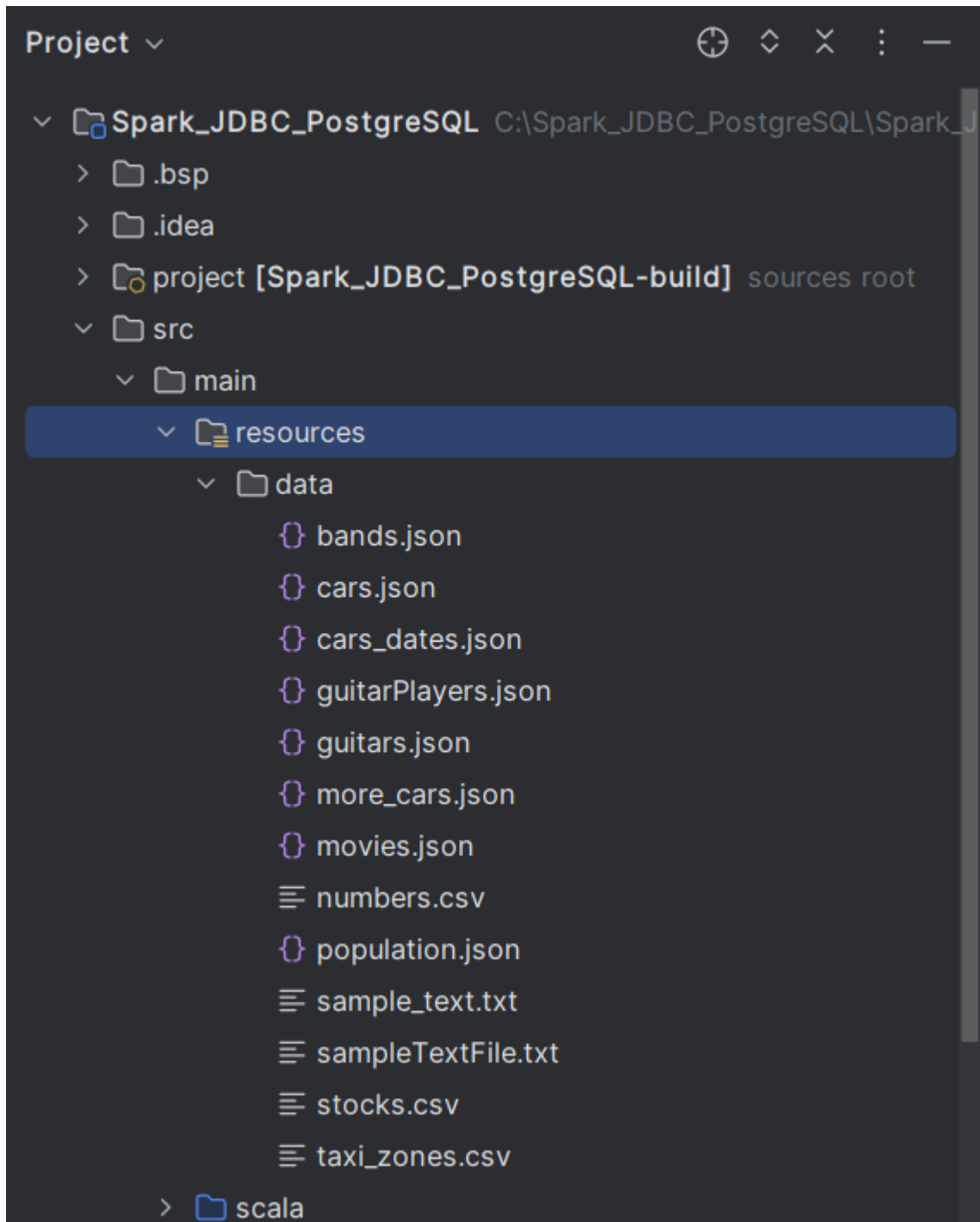
Now we can see the "resources" folder was created



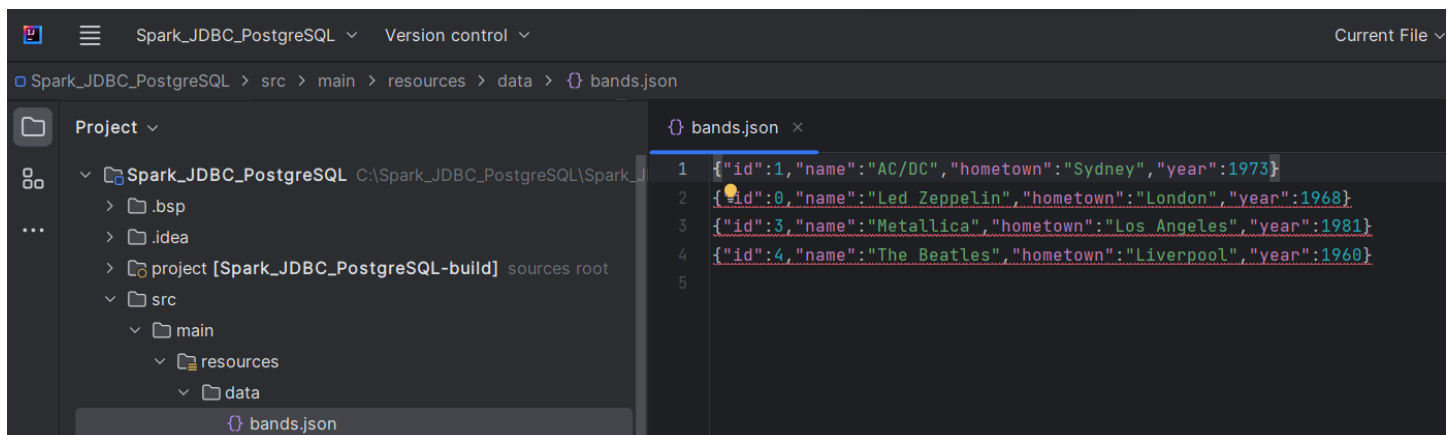
Now we right click on the "resources" folder and paste the "data" folder with all the data files inside.



You can see the "data" folder with all the data files inside

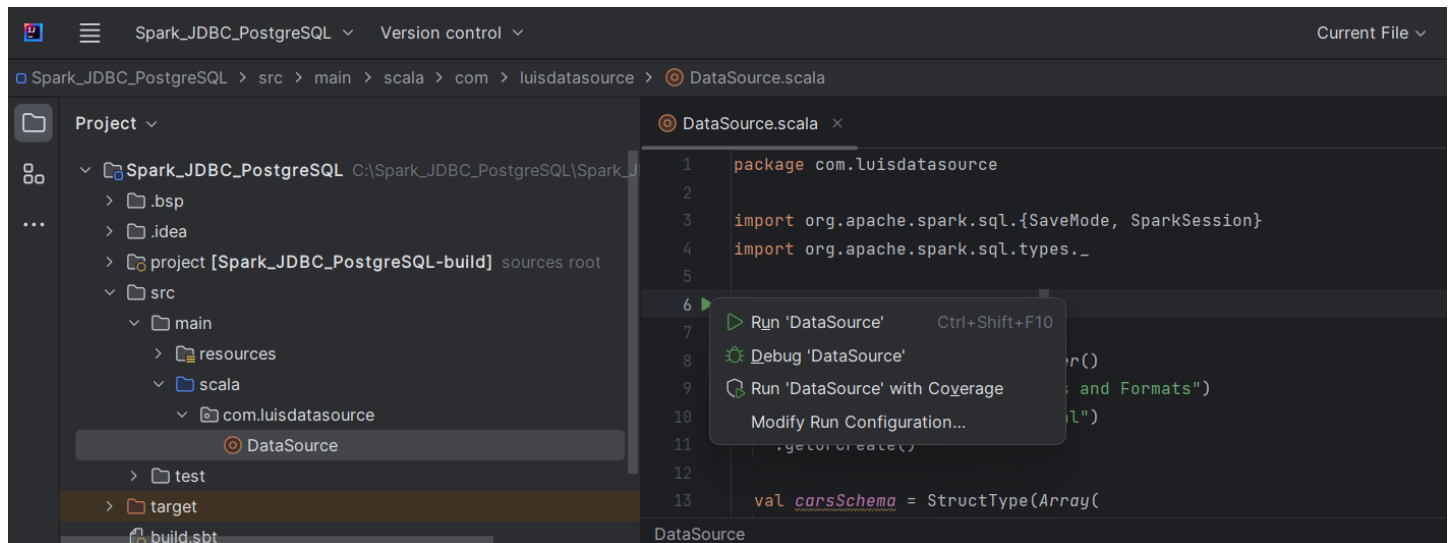


See as an example the bands.json file

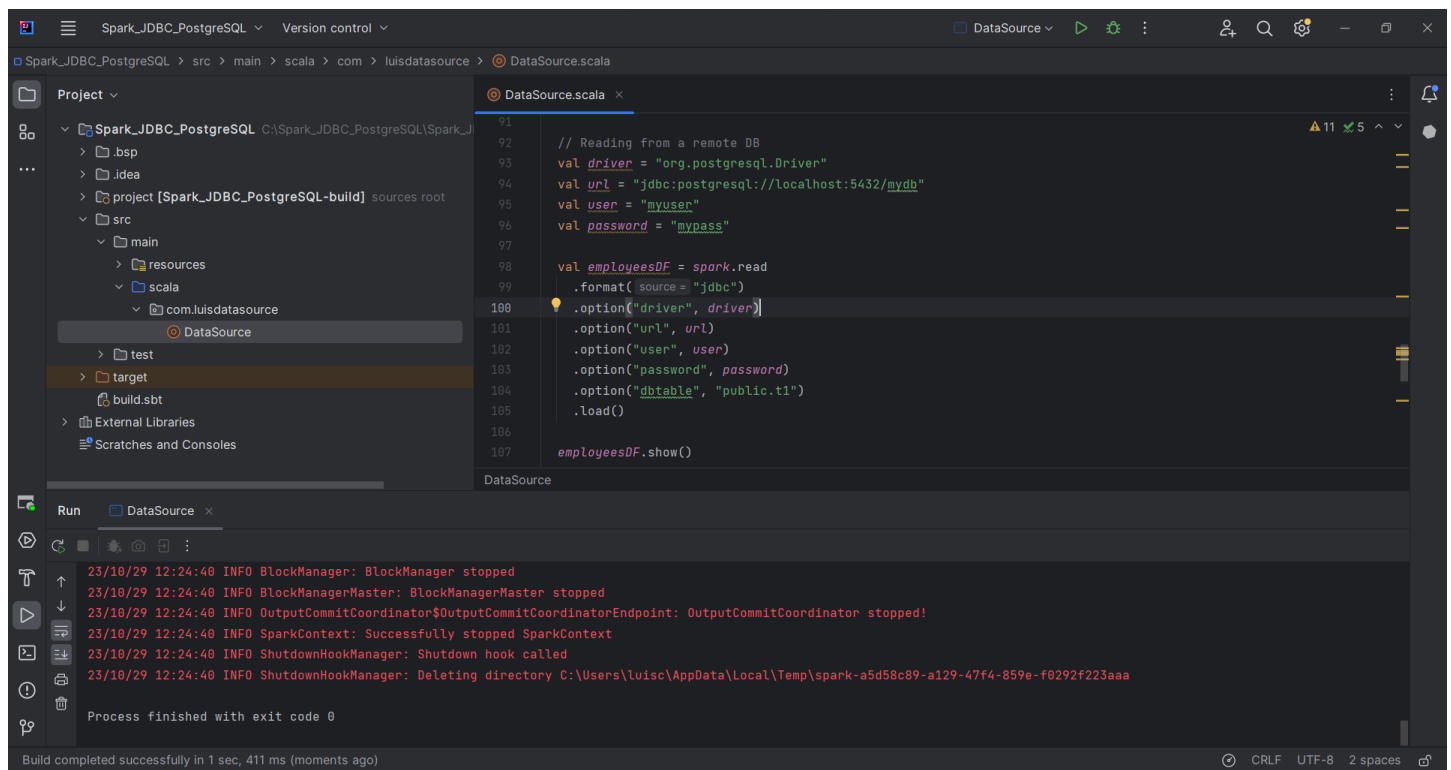


## 2.7. How to run the application

Press in the execution green button and select the option "Run 'DataSource'"



You can see the "Build completed successfully" and the "Process finished with exit code 0"



IMPORTANT NOTE: take care if you restart your computer, if this is the case then:

- Run Docker Desktop
- Run the postgresSQL container.
- Enter in the container and follow all the steps defined above to create a new user and grant permission
- Be sure to start the server in your local machine also.

```
C:\Program Files\PostgreSQL\15\bin>pg_ctl start -D "C:\Program Files\PostgreSQL\15\data" -o "-
```

## 2.8. Application output

```

23/10/29 12:24:38 INFO DAGScheduler: Job 3 finished: show at DataSource.scala:90, took 0,108340 s
23/10/29 12:24:38 INFO CodeGenerator: Code generated in 13.1209 ms
+-----+
|value|
+-----+
| this|
|  is|
|Scala|
| and|
|  I|
| love|
|Spark|
+-----+

```

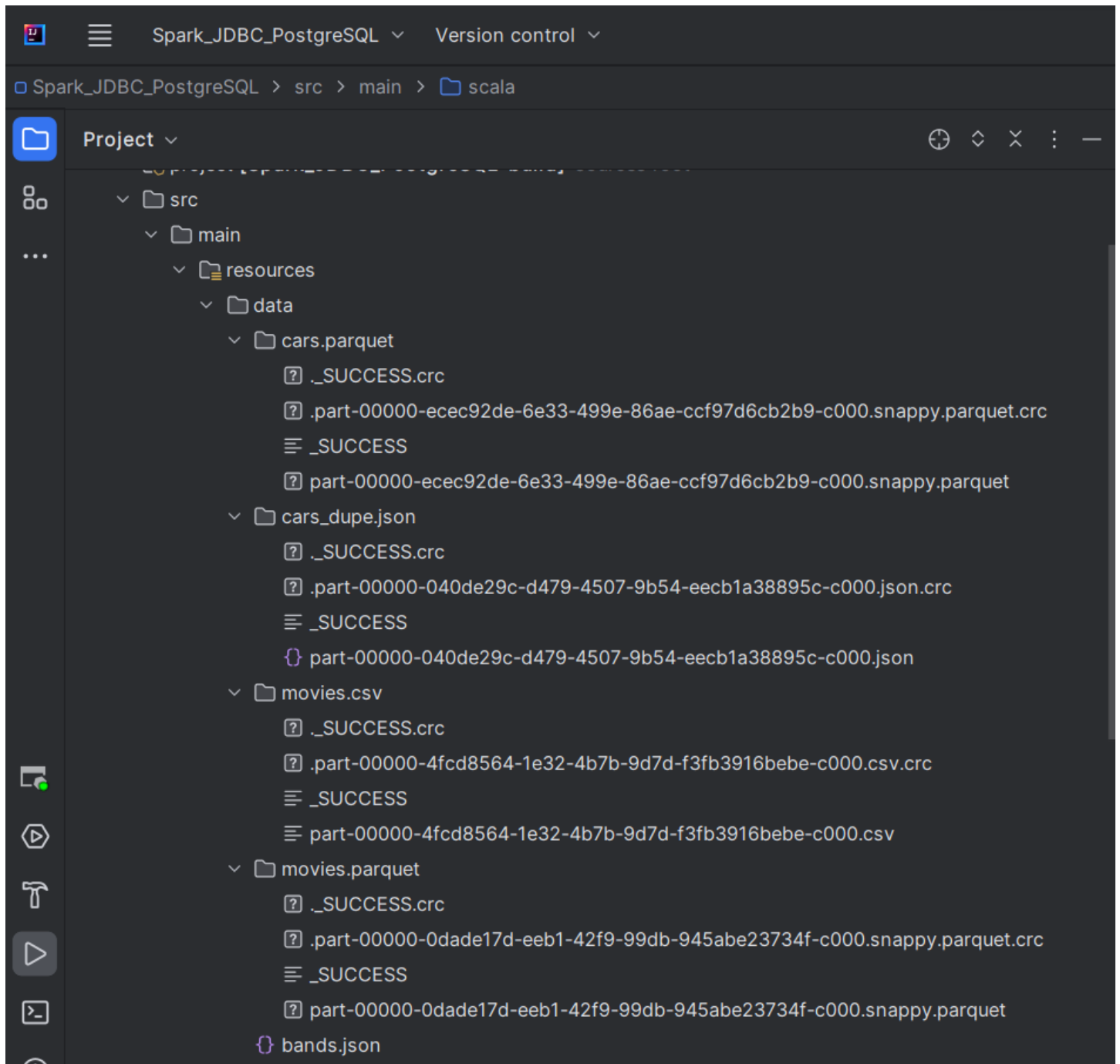
```

23/10/29 12:24:39 INFO TaskSchedulerImpl: Killing all running tasks in stage 4: Stage finished
23/10/29 12:24:39 INFO DAGScheduler: Job 4 finished: show at DataSource.scala:107, took 0,170807 s
+----+
| id|
+----+
|  1|
+----+

```

Also this new data files were created





**IMPORTANT NOTE:** if you would like to rerun the application you should delete the following folders before running the application again

