

luiscoco / **Spark_DataSources_JDBC_PostgreSQL**

Code Issues Pull requests Actions Projects Wiki Security Insights Set

Spark_DataSources

0 stars 0 forks 1 watching Activity Public repository

main ▾ ...

Branches Tags

luiscoco Update README.md ... 5 minutes ago 40

[View code](#)

Spark DataSources JDBC PostgreSQL

README.md

plugin, Java 11, Spark and winutils in your computer

1.1. Install IntelliJ Community + Scala plugin

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

We're committed to giving back to our wonderful community, which is why IntelliJ IDEA Community Edition is completely free to use

IntelliJ IDEA Community Edition
The IDE for pure Java and Kotlin development

[Download](#) [.exe ▾](#)
Free, built on open source

1.2. Install Java 11 (JDK) and set JAVA_HOME environmental variable



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

Download Type	File Size	Action
Windows x64 Installer	141.39 MB	jdk-11.0.20_windows-x64_bin.exe
Windows x64 Compressed Archive	159.14 MB	jdk-11.0.20_windows-x64_bin.zip

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>

- Download
- Libraries
- Documentation
- Examples
- Community
- Developers

Download Apache Spark™

1. Choose a Spark release:
2. Choose a package type:
3. Download Spark: [spark-3.5.0-bin-hadoop3.tgz](#)
4. Verify this release using the 3.5.0 [signatures](#), [checksums](#) and [project release KEYS](#) by following these [procedures](#).

Note that Spark 3 is pre-built with Scala 2.12 in general and Spark 3.2+ provides additional pre-built distribution with Scala 2.13.

- Community
- Projects
- Downloads
- Learn
- Resources & Tools
- About



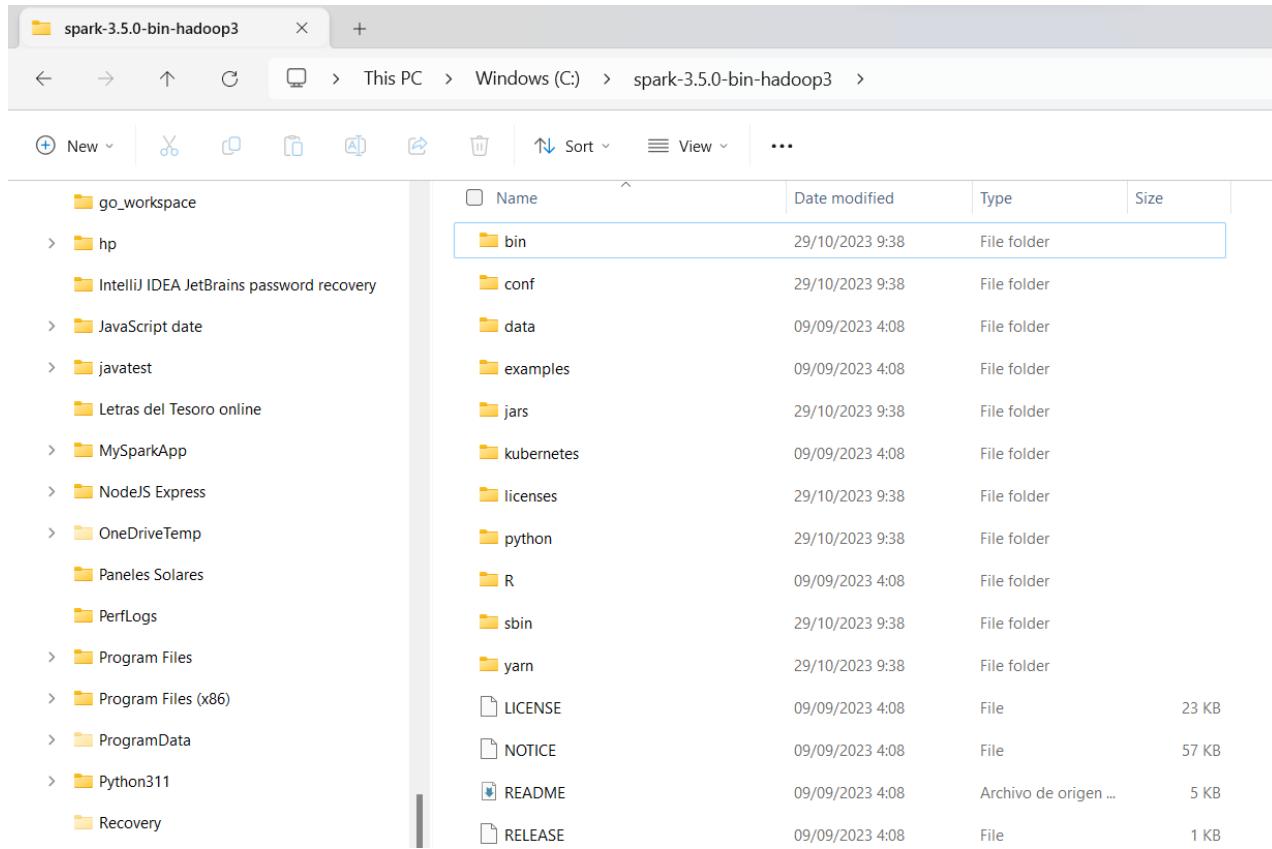
We suggest the following location for your download:

<https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz>

Alternate download locations are suggested below.

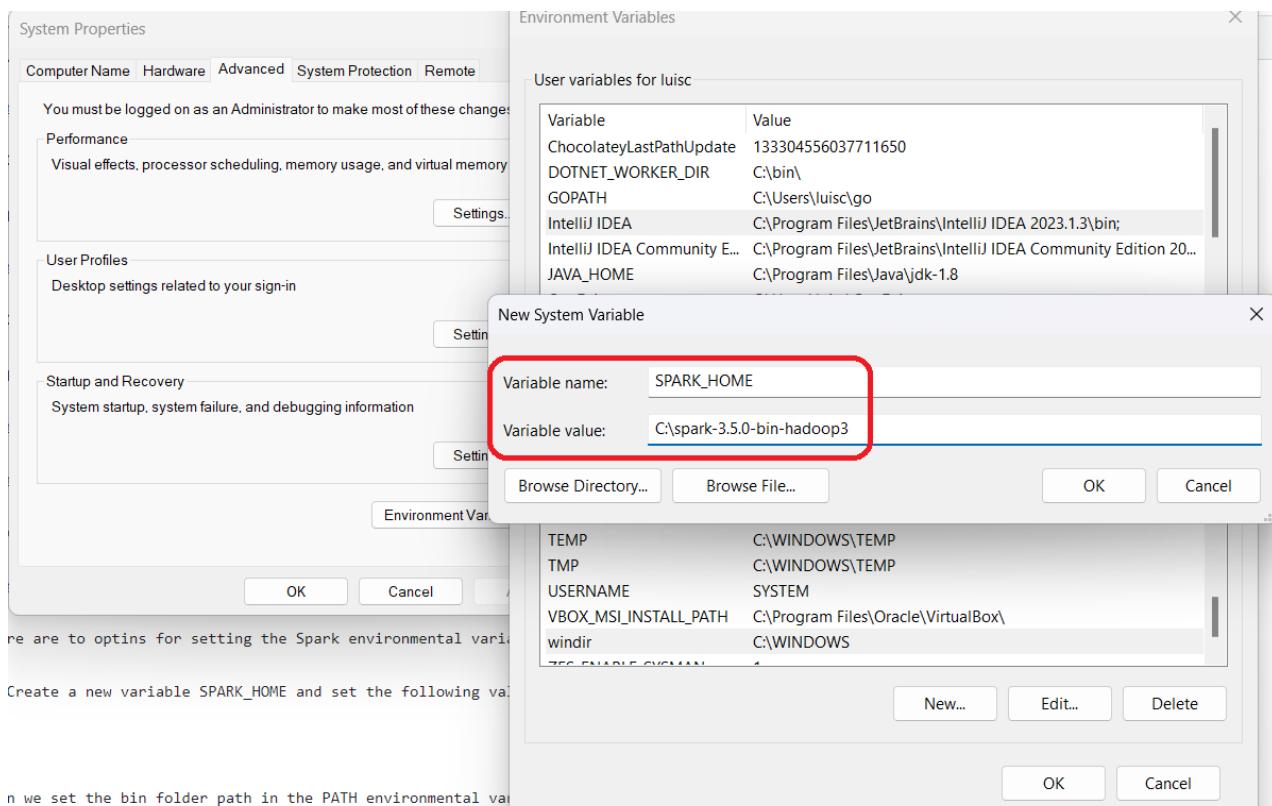
It is essential that you [verify the integrity](#) of the downloaded file using the PGP signature ([.asc](#) file) or a hash ([.md5](#) or [.sha*](#) file).

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

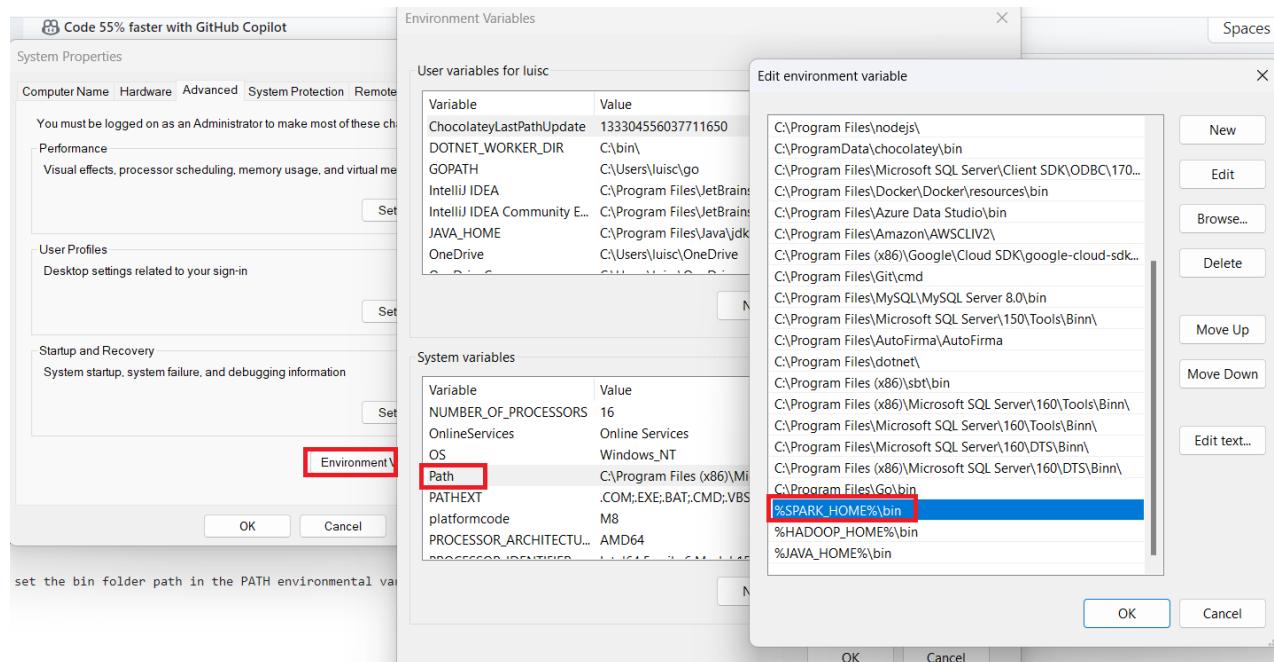


There are two options for setting the Spark environmental variables:

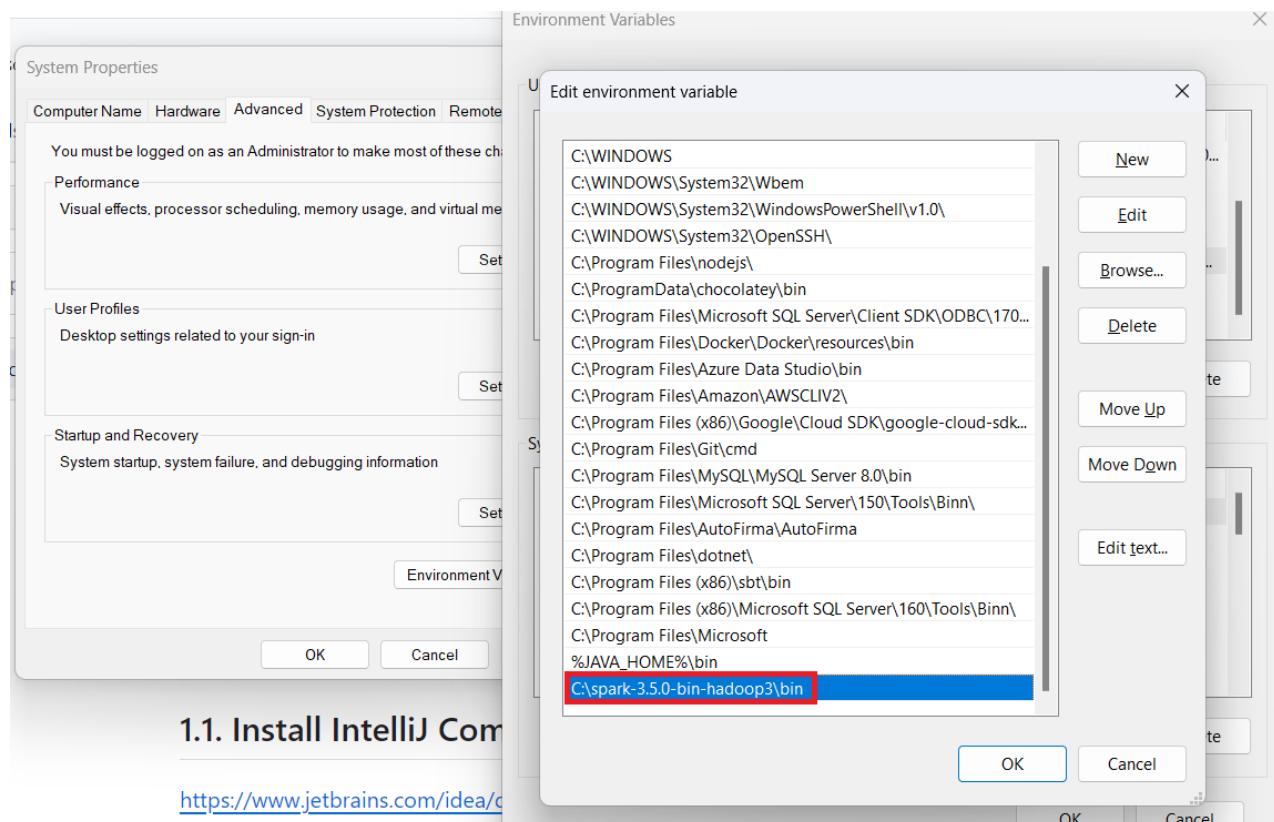
- Create a new variable SPARK_HOME and set the following value: C:\spark-3.5.0-bin-hadoop3



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

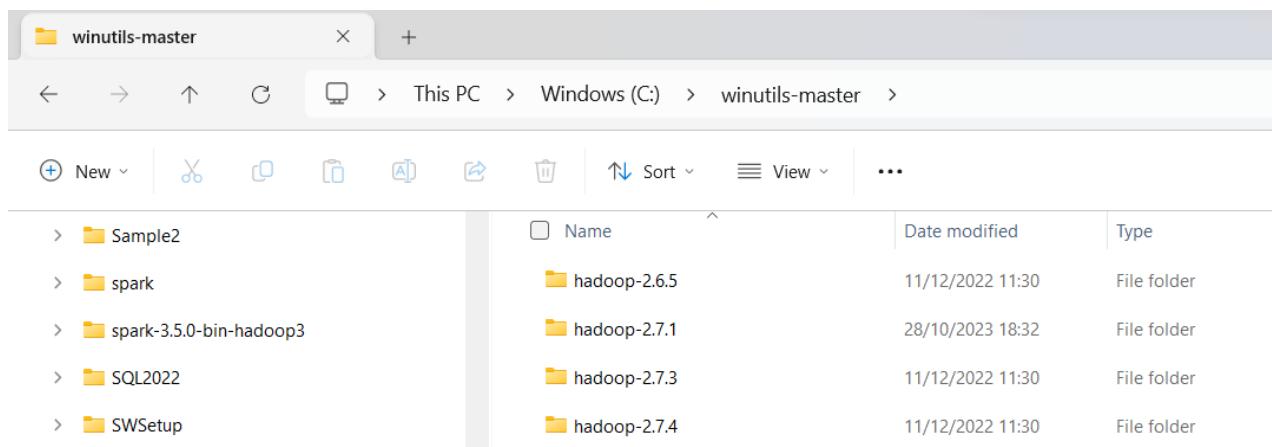
We download or clone this git repository: <https://github.com/kontext-tech/winutils>

This branch is 23 commits ahead, 6 commits behind cdarlint:master.

FahaoTang Updated link to build via Docker tutorial. ff8e529 on Dec 11, 2022 28 commits

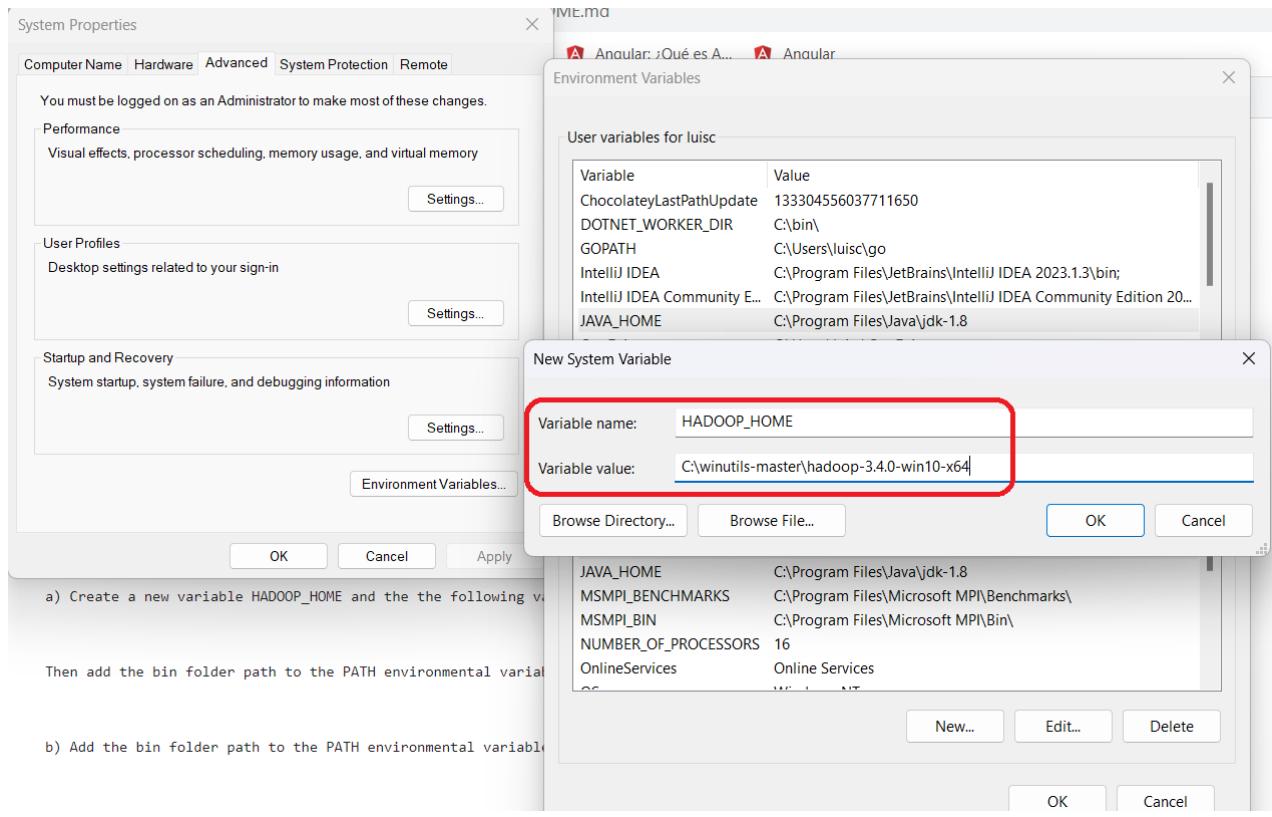
- hadoop-2.6.5/bin fixed exe and lib 265-312 4 years ago
- hadoop-2.7.3/bin fixed exe and lib 265-312 4 years ago
- hadoop-2.7.4/bin fixed exe and lib 265-312 4 years ago
- hadoop-2.7.6/bin fixed exe and lib 265-312 4 years ago

We place the winutils folder in C:

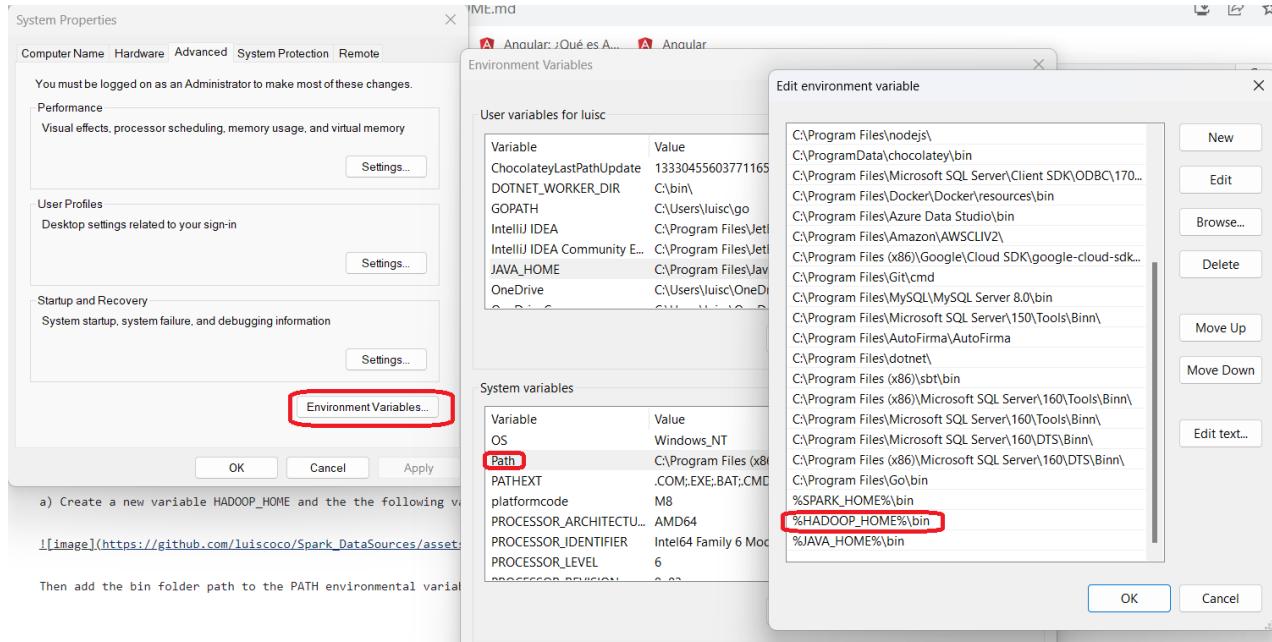


Set HADOOP_HOME environmental variable. There are two options:

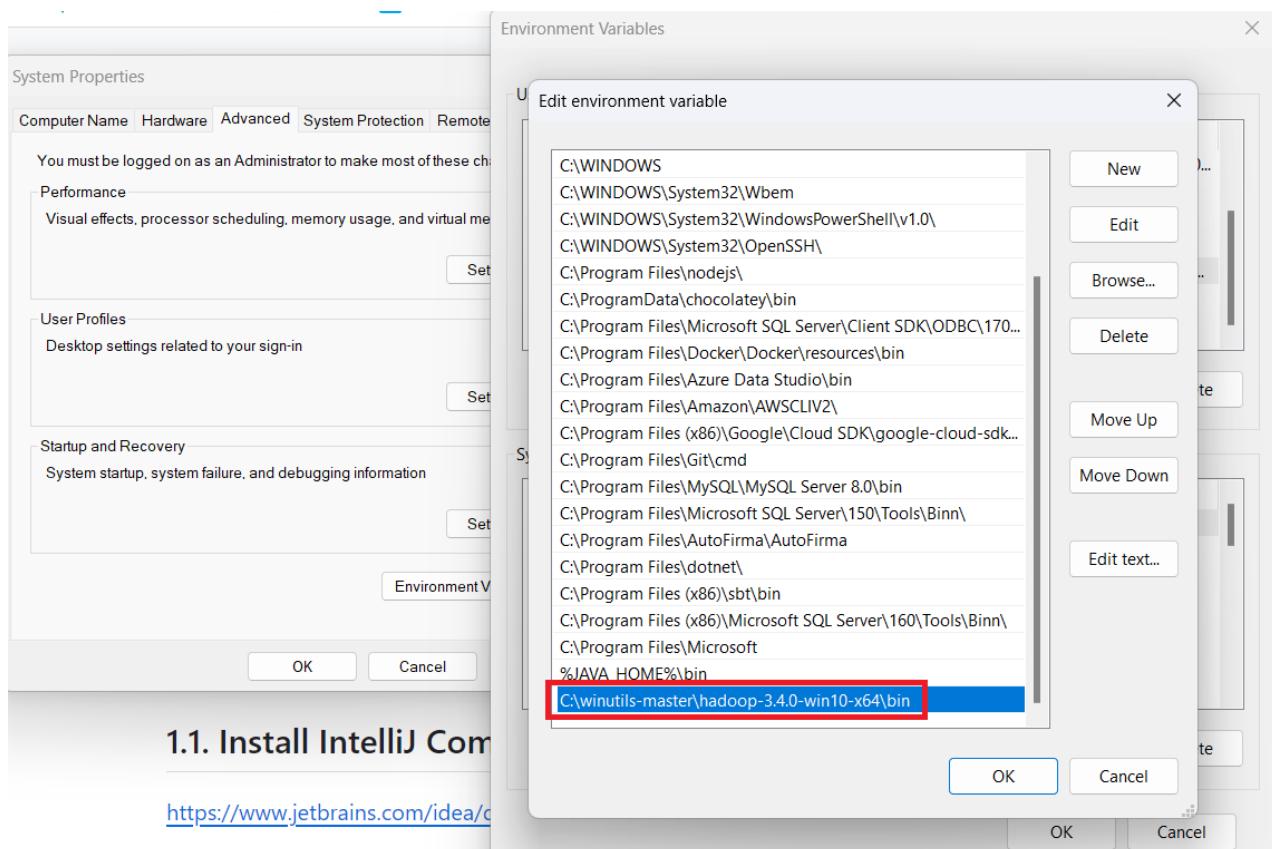
a) Create a new variable HADOOP_HOME and the the following value



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 your 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 download PostgreSQL

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

The screenshot shows the PostgreSQL Downloads page. On the left, there's a 'Quick Links' sidebar with options like 'Downloads', 'Packages', 'Source', 'Software Catalogue', and 'File Browser'. The main content area is titled 'Downloads' with a download icon. Below it is a section titled 'PostgreSQL Downloads'. A note says 'PostgreSQL is available for download as ready-to-use packages or installers for various platforms, as well as a source code archive if you want to build it yourself.' Under 'Packages and Installers', it asks 'Select your operating system family:' and shows five options: Linux (Ubuntu logo), macOS (Apple logo), Windows (Windows logo), BSD (BSD logo), and Solaris (Solaris logo). The Windows button is highlighted with a red circle.

In the following URL you can download pgAdmin

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

The screenshot shows the pgAdmin 4 (Windows) download page. At the top, there's a navigation bar with the pgAdmin logo, Home, Development, Documentation, Download, and Support. The main content area has a blue header 'pgAdmin 4 (Windows)'. To the left is a 'Quick Links' sidebar with icons for 'Download' (down arrow), 'FAQ' (info icon), 'Latest Docs' (book icon), 'Get Help' (question mark icon), and 'Screenshots' (monitor icon). The main content area is titled 'Download' and says 'Maintainer: pgAdmin Development Team'. It notes that pgAdmin is available for 64 bit Windows™ 7 SP1 (desktop) or 2008R2 (server) and above, up to v4.30. It also mentions support for Windows 8 (desktop) or 2012 (server) and above, Windows 10 (desktop) or 2016 (server) and above, and 32 bit Windows support up to v4.29. A list of download links is provided:

- [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 \(released March 9, 2023\)](#)

Below the download links is a 'Info' section.

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/>

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

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

 Products Developers Pricing Blog About Us Partners

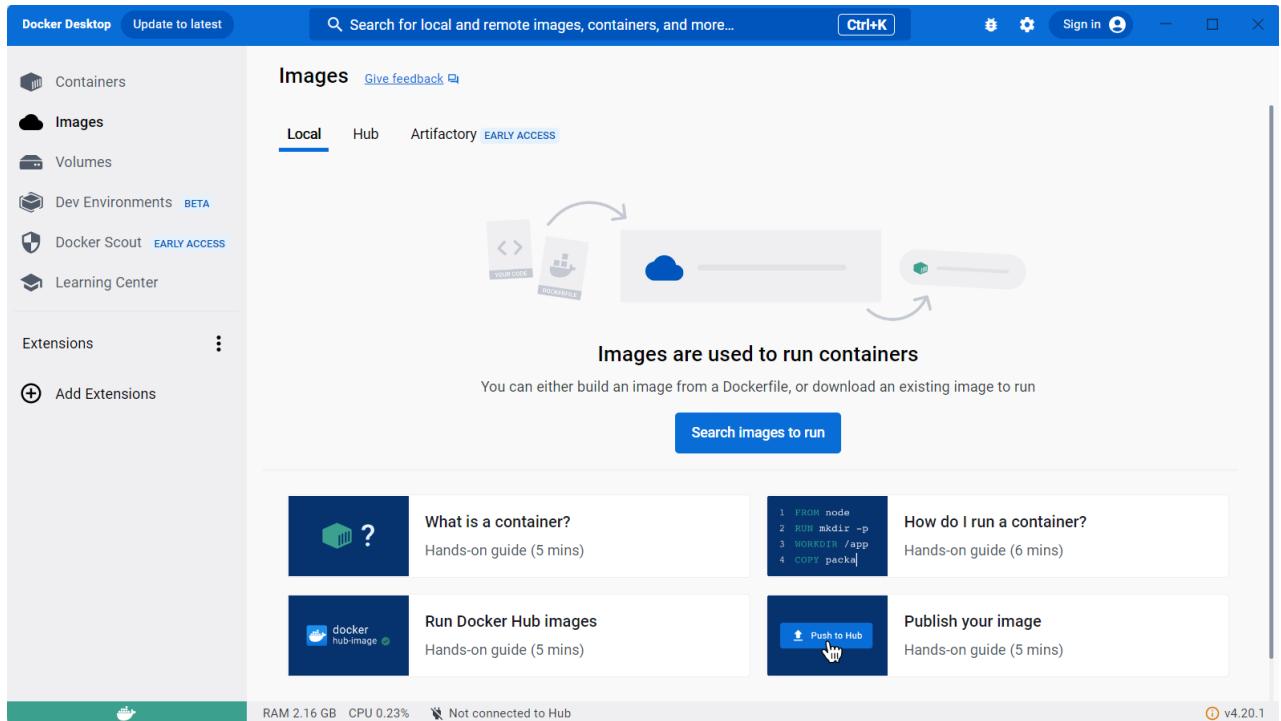
Docker Desktop

The #1 containerization software for developers and teams

Your command center for innovative container development

[Create an account](#) [Download for Windows](#)

Run Docker Desktop



The screenshot shows the Docker Desktop application window. On the left, there's a sidebar with icons for Containers, Images, Volumes, Dev Environments (Beta), Docker Scout (Early Access), and Learning Center. Below that is an 'Extensions' section with an 'Add Extensions' button. The main area is titled 'Images' with a 'Local' tab selected, showing a diagram of a Dockerfile building an image. Below the diagram, text says 'Images are used to run containers' and 'You can either build an image from a Dockerfile, or download an existing image to run'. A 'Search images to run' button is present. At the bottom, status information shows 'RAM 2.16 GB CPU 0.23%' and 'Not connected to Hub', with a version 'v4.20.1' in the bottom right.

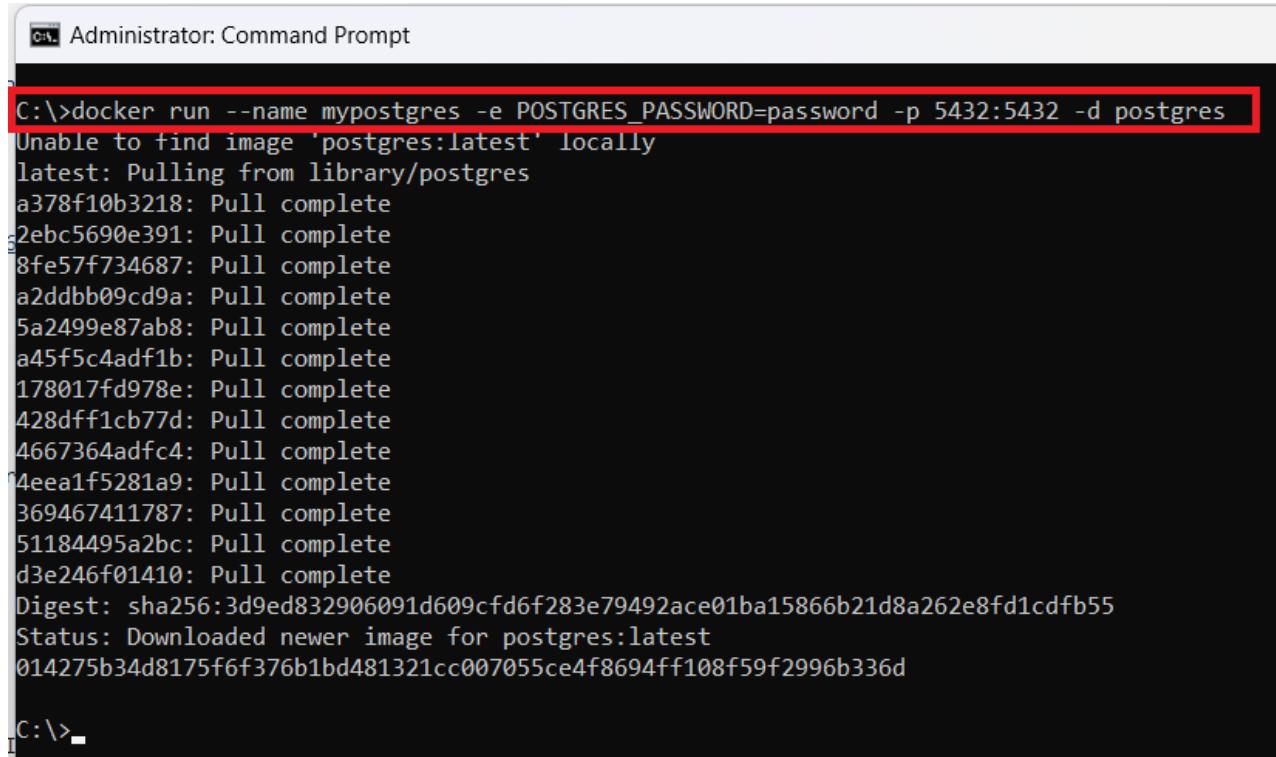
2. Pull and run the PostgreSQL docker container.

For details see: 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
```





```
C:\>docker run --name mypostgres -e POSTGRES_PASSWORD=password -p 5432:5432 -d postgres
Unable to find image 'postgres:latest' locally
latest: Pulling from library/postgres
a378f10b3218: Pull complete
2ebc5690e391: Pull complete
8fe57f734687: Pull complete
a2ddbb09cd9a: Pull complete
5a2499e87ab8: Pull complete
a45f5c4adf1b: Pull complete
178017fd978e: Pull complete
428dff1cb77d: Pull complete
4667364adfc4: Pull complete
4eea1f5281a9: Pull complete
369467411787: Pull complete
51184495a2bc: Pull complete
d3e246f01410: Pull complete
Digest: sha256:3d9ed832906091d609cf6f283e79492ace01ba15866b21d8a262e8fd1cdfb55
Status: Downloaded newer image for postgres:latest
014275b34d8175f6f376b1bd481321cc007055ce4f8694ff108f59f2996b336d

C:\>
```

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
```

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

4. We execute the PostgreSQL container.

```
docker start dockerContainerID
```

```
C:\>docker ps -a
CONTAINER ID        IMAGE       COMMAND             CREATED          STATUS          PORTS
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
014275b34d81
```

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

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

```
C:\>docker exec -it 014275b34d81 bash
root@014275b34d81:/#
```

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
```

```
C:\>docker exec -it 014275b34d81 bash
root@014275b34d81:/# psql -U postgres -W
Password:
psql (16.0 (Debian 16.0-1.pgdg120+1))
Type "help" for help.

postgres=#
```

7. We create a new database called mydb

```
create database mydb;
```

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

8. For listing all the databases

\l



List of databases									
Name	Owner	Encoding	Locale	Provider	Collate	Ctype	ICU Locale	ICU Rules	Access privileges
mydb	postgres	UTF8	libc		en_US.utf8	en_US.utf8			=c/postgres +
postgres	postgres	UTF8	libc		en_US.utf8	en_US.utf8			postgres=CTc/postgres +
template0	postgres	UTF8	libc		en_US.utf8	en_US.utf8			=c/postgres +
template1	postgres	UTF8	libc		en_US.utf8	en_US.utf8			postgres=CTc/postgres

(4 rows)

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

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



```
postgres=# create table t1(id int);
CREATE TABLE
```

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

```
select * from t1;
```



```
postgres=# select * from t1;
 id
-----
(0 rows)
```

11. We insert a row in the table

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



```
postgres=# insert into t1 values(1);
INSERT 0 1
```

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

```
select * from t1;
```



```
postgres=# select * from t1;
 id
-----
 1
(1 row)
```

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

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



```
postgres=# create user myuser with encrypted password 'mypass';
CREATE ROLE
```

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:

```
C:\> Administrator: Command Prompt - 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=# create table t1(id int);
CREATE TABLE
mydb=# insert into t1 values(1);
INSERT 0 1
mydb=#

```

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 5433"
```



```
Microsoft Windows [Version 10.0.22621.2506]
(c) Microsoft Corporation. All rights reserved.

C:\Users\luisc>cd C:\Program Files\PostgreSQL\15\bin

C:\Program Files\PostgreSQL\15\bin>pg_ctl start -D "C:\Program Files\PostgreSQL\15\data" -o "-p 5433"
pg_ctl: otro servidor puede estar en ejecución; tratando de iniciararlo de todas formas.
esperando que el servidor se inicie....2023-10-29 10:36:42.479 CET [25288] LOG:  redirigiendo la salida del registro al
proceso recolector de registro
2023-10-29 10:36:42.479 CET [25288] HINT:  La salida futura del registro aparecerá en el directorio «log».
    listo
servidor iniciado
```

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

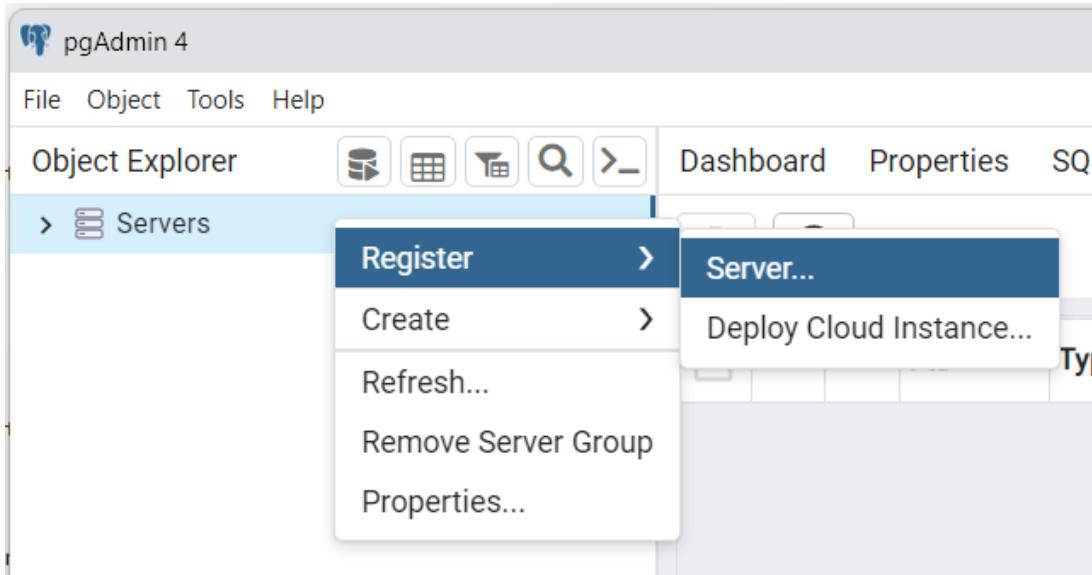
```
C:\Program Files\PostgreSQL\15\bin>pg_ctl status -D "C:\Program Files\PostgreSQL\15\data"
```



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

The screenshot shows the pgAdmin 4 application window. The title bar says "pgAdmin 4". The menu bar includes File, Object, Tools, and Help. The top navigation bar has tabs for Object Explorer, Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and Processes. The Processes tab is currently selected. The left sidebar has a "Servers" section with a plus sign icon. The main pane is titled "Processes" and contains a table with columns: PID, Type, Server, Object, and Start. A message at the bottom of the table says "No rows found".

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".

Register - Server

General **Connection** Parameters SSH Tunnel Advanced

Host name/address	localhost
Port	5432
Maintenance database	mydb
Username	myuser
Kerberos authentication?	<input type="checkbox"/>
Password
Save password?	<input type="checkbox"/>
Role	
Service	

Buttons:

We also have to input the connection name "mypostgres"

The screenshot shows the pgAdmin 4 interface. At the top, there's a menu bar with 'File', 'Object', 'Tools', and 'Help'. Below the menu is the 'Object Explorer' header. Under 'Object Explorer', there's a tree view. A dropdown arrow next to 'Servers (2)' indicates two servers are listed. Underneath, there are two entries: 'PostgreSQL 15' and 'mypostgres'. The 'mypostgres' entry is highlighted with a blue selection bar at the bottom of the tree view.

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

The screenshot shows the pgAdmin 4 interface with the title bar "pgAdmin 4". The menu bar includes "File", "Object", "Tools", and "Help". The toolbar has icons for "New Query", "New Connection", "New Table", "Search", and "Exit". The "Object Explorer" sidebar on the left displays the database structure:

- Servers (2)
 - PostgreSQL 15
 - mypostgres
 - Databases (2)
 - mydb (highlighted with a red box)
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - 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 (highlighted with a red box)

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

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'public' schema, there is one table named 't1'. A context menu is open over this table, specifically at the 'View/Edit Data' option. The 'First 100 Rows' option is highlighted. The main pane displays a query history window with a single entry:

```
27/10/2023 14:40:36
SELECT * FROM public.t1 LIMIT 100
```

Below the query history, an error message is visible:

ERROR: permission denied for table t1
SQL state: 42501

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

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'public' schema, there is one table named 't1'. The main pane displays a query results window. It shows two queries:

```
27/10/2023 14:40:36
SELECT * FROM public.t1 LIMIT 100
1 rows affected
```

Below this, another query is shown:

```
27/10/2023 14:31:20
SELECT id FROM public.t1;
ERROR: permission denied for table t1
SQL state: 42501
```

A messages section indicates:

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

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

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
 - > Rules
 - > Triggers

Count Rows Properties SQL Statistics Dependencies

Create > ic.t1/mydb/myuser@mypostgres

Delete/Drop Refresh... Restore... Backup... Drop Cascade Import/Export Data... Reset Statistics ERD For Table Maintenance... Scripts Truncate View/Edit Data All Rows Search Objects... PSQL Tool Query Tool Properties...

View/Edit Data > First 100 Rows Last 100 Rows Filtered Rows... Actions

	id	integer	lock
1		1	

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
 - > Rules
 - > Triggers

Dashboard Properties SQL Statistics Dependencies

public.t1/mydb/myuser@mypostgres

Query Query History

```
1 SELECT * FROM public.t1
2
3 LIMIT 100
```

Data Output Messages Notifications

	id	integer	lock
1		1	

1.7. Install PostgreSQL JDBC driver in Spark folder [🔗](#)

Download the PostgreSQL JDBC driver from internet

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

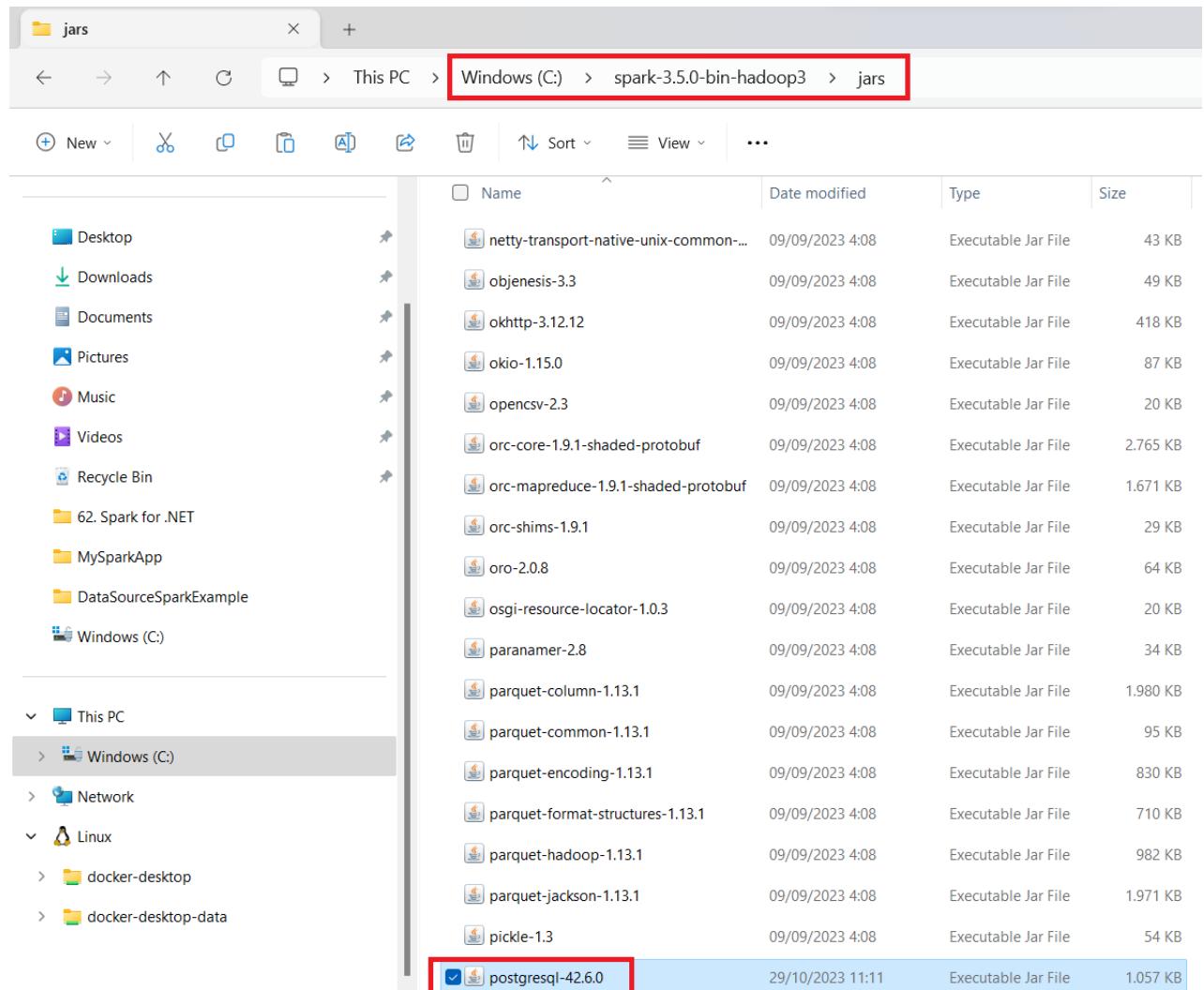
⬇️ Download

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.

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
```



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

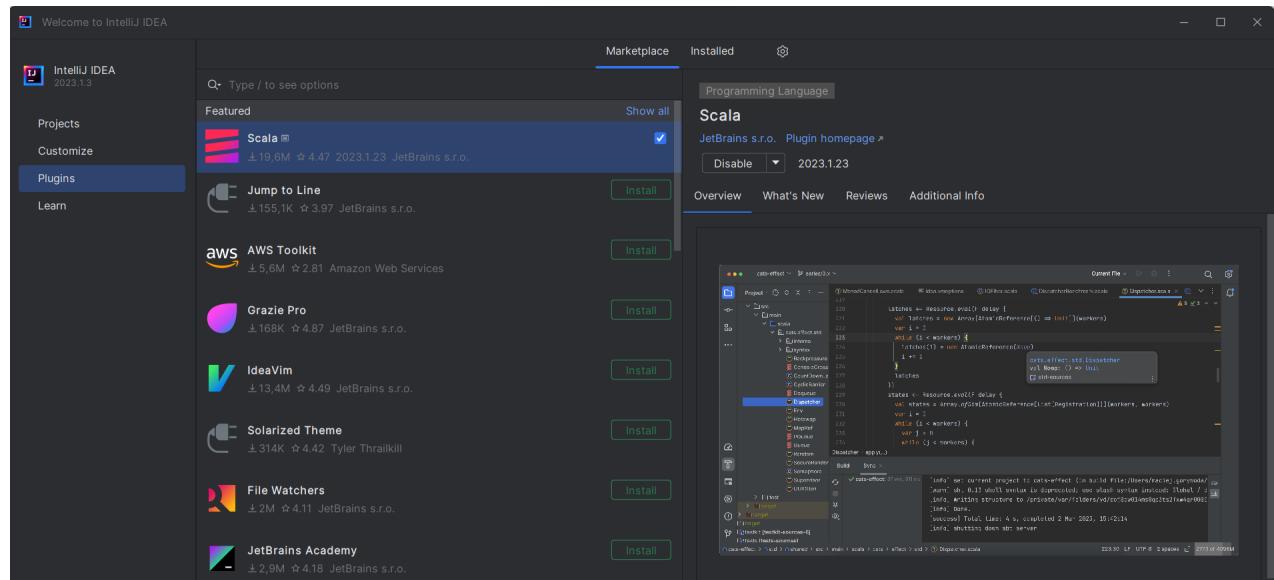
Java version: 11

Spark version: 3.5.0

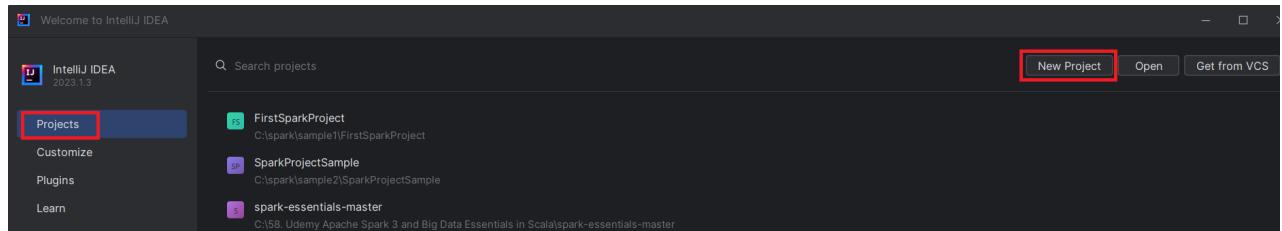
Scala version: 2.12.18

2.2. Run IntelliJ Community IDE.

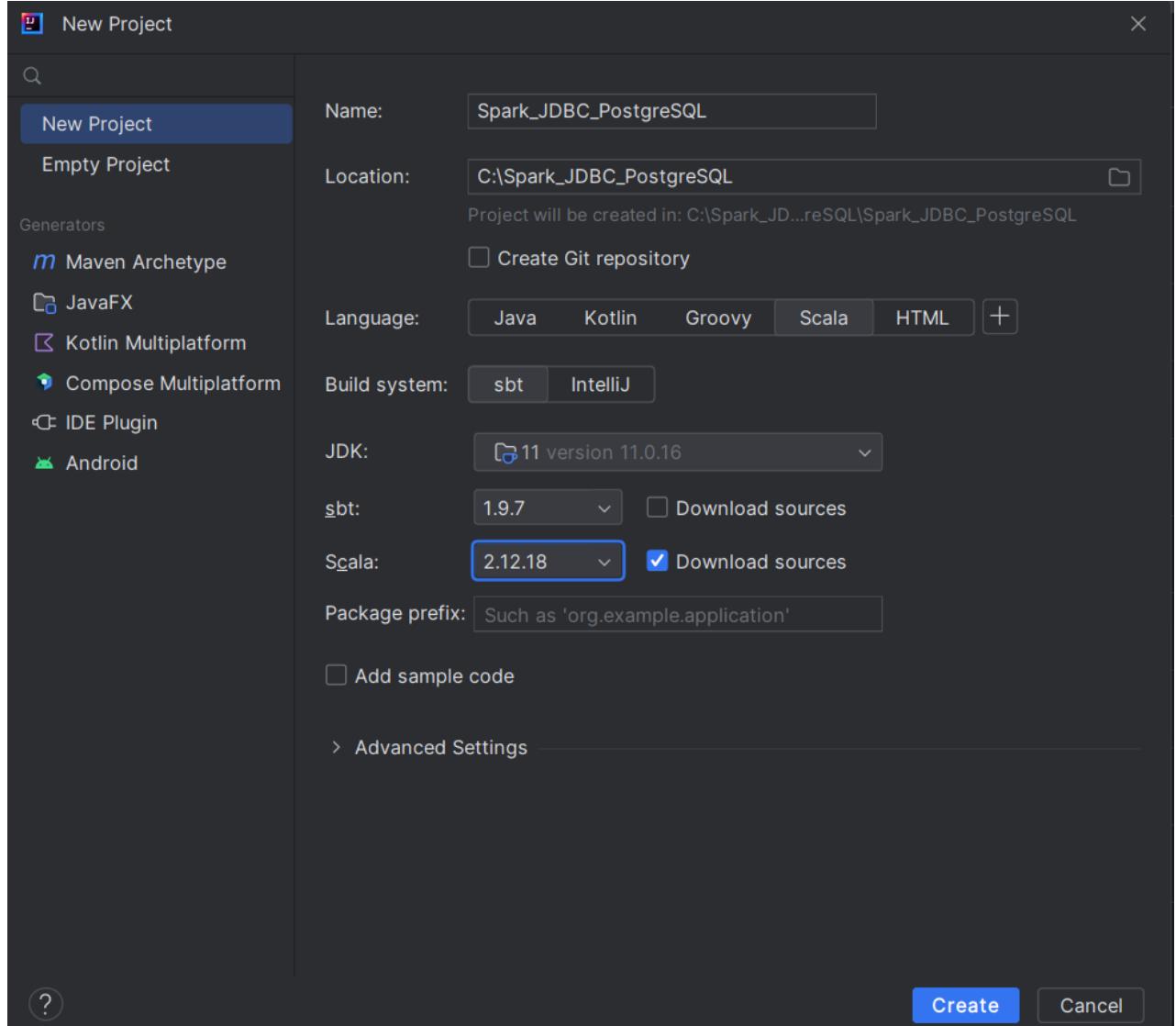
Install the "Scala" plugin



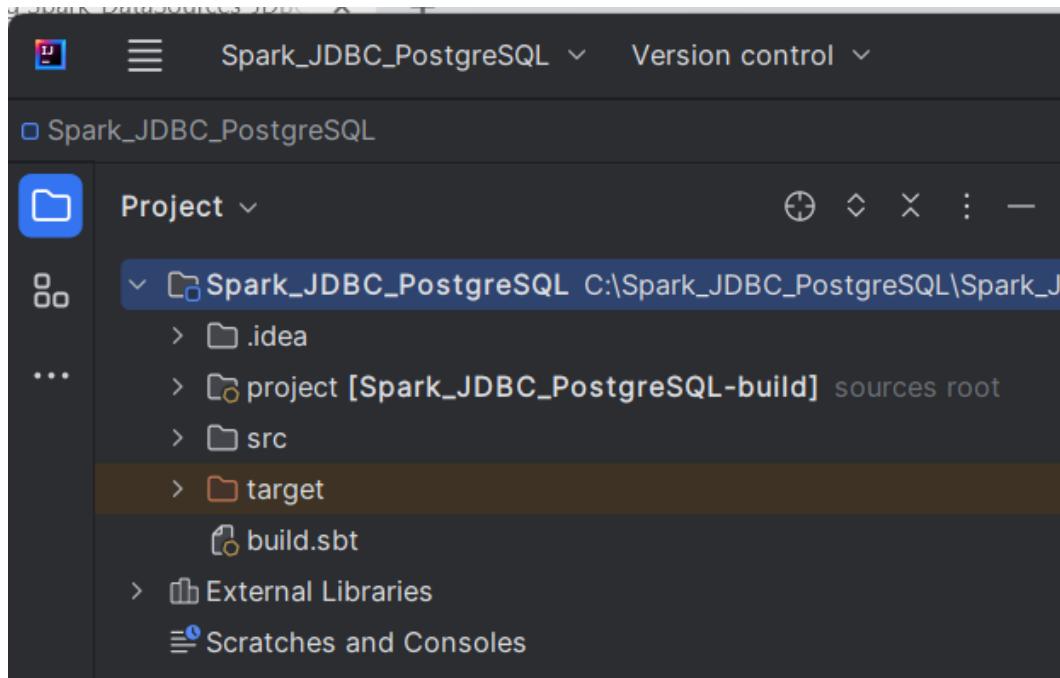
Create a new project



Enter the new project input data



Then we can see the new project in IntelliJ



2.3. We input the bild.sbt file source code ↗

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

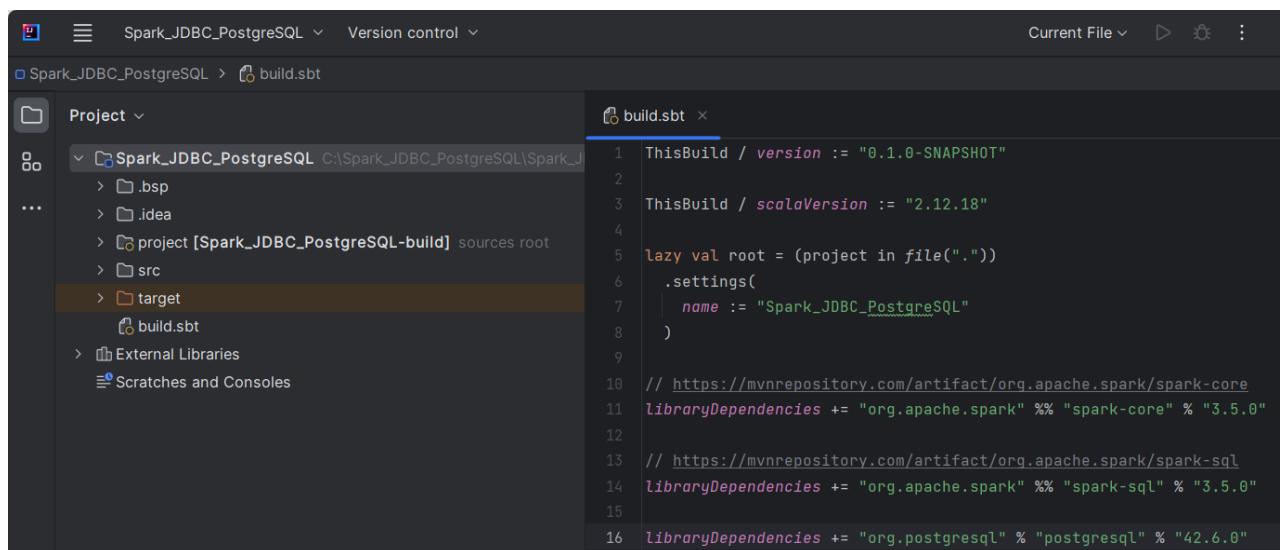
ThisBuild / scalaVersion := "2.12.18"

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

```

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"

```

After reloading the project dependencies we will see this result

```

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"

```

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"

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

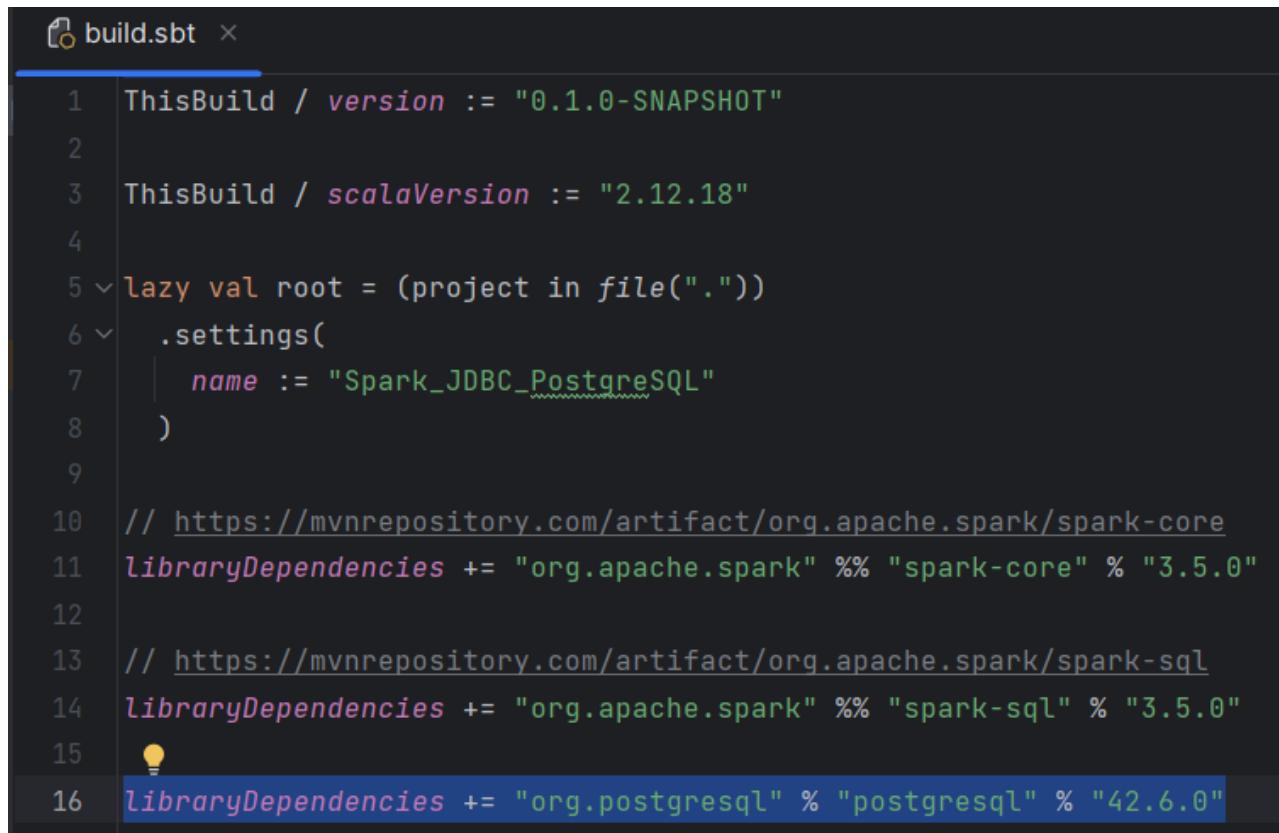
The screenshot shows the MVN Repository homepage with a search bar at the top. Below it, a sidebar lists 'Indexed Artifacts (35.2M)' with a line graph showing the number of projects over time from 2006 to 2018. A section for 'Popular Categories' includes links to Testing Frameworks & Tools, Android Packages, Logging Frameworks, Java Specifications, JSON Libraries, Language Runtime, JVM Languages, and Core Utilities. The main content area shows the 'PostgreSQL JDBC Driver' page for version 42.6.0. It includes tabs for License (BSD 2-clause), Categories (JDBC Drivers), Tags (database, sql, jdbc, postgresql, driver), Ranking (#115 in MvnRepository, #2 in JDBC Drivers), and Used By (4,152 artifacts). Below this is a table of versions with Central as the repository. The version 42.6.0 is highlighted with a red border.

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

This screenshot shows the same PostgreSQL JDBC Driver page as before, but with the 'SBT' tab selected at the bottom of the dependency list. The dependency code is highlighted with a red border:

```
// 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:



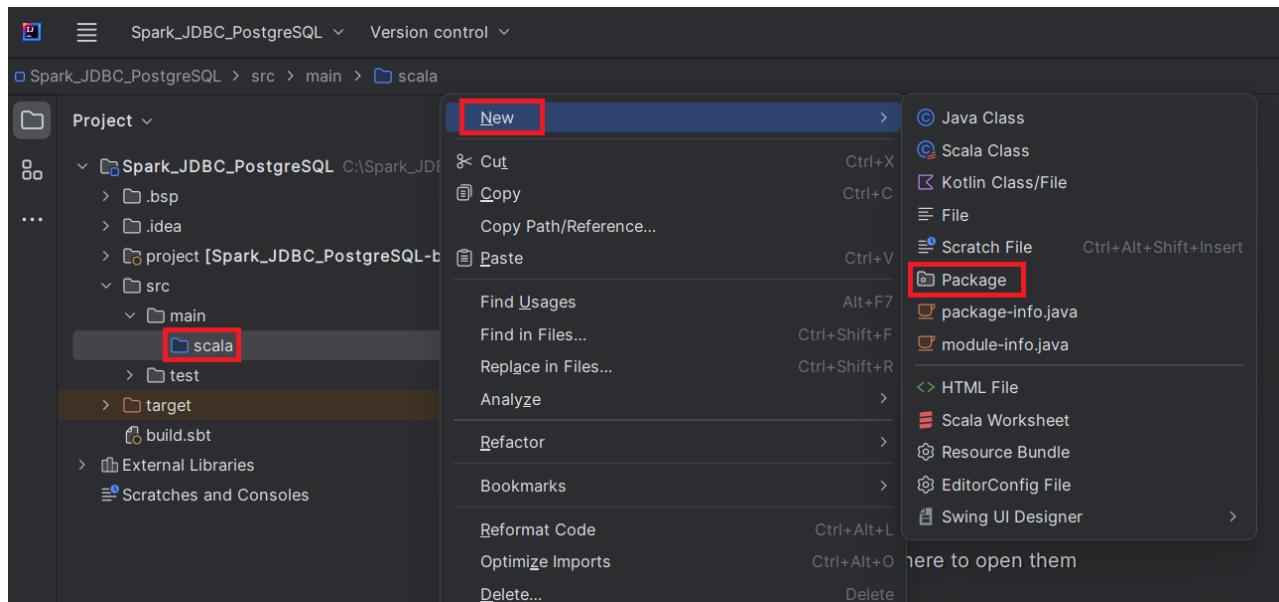
```

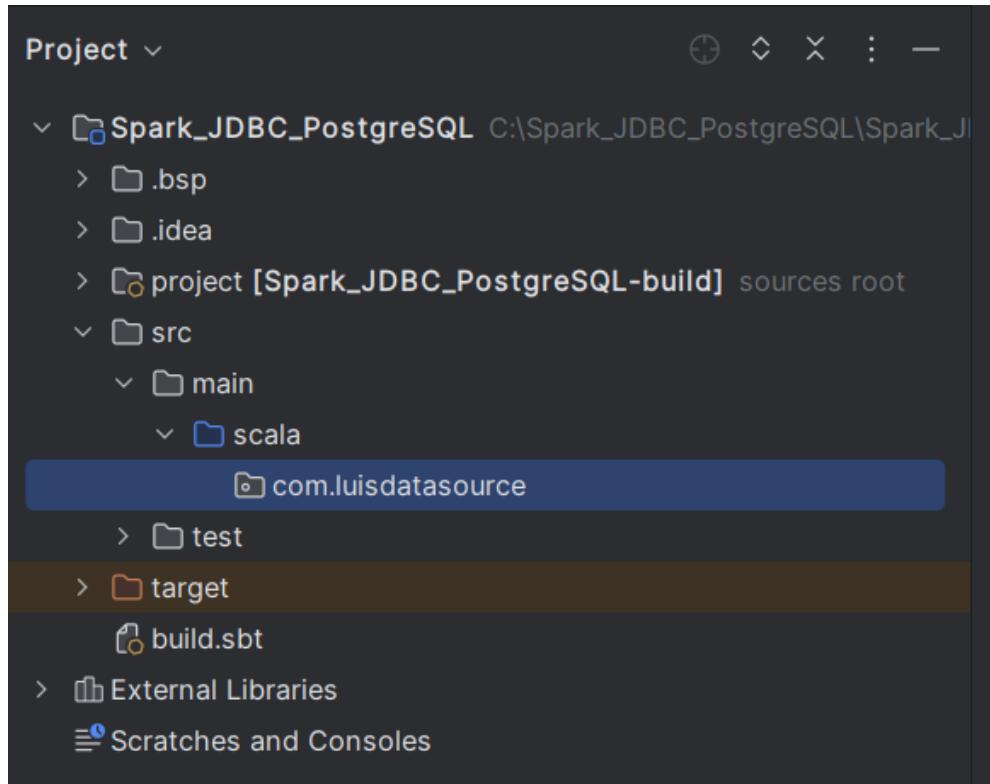
1 ThisBuild / version := "0.1.0-SNAPSHOT"
2
3 ThisBuild / scalaVersion := "2.12.18"
4
5 lazy val root = (project in file("."))
6 .settings(
7   name := "Spark_JDBC_PostgreSQL"
8 )
9
10 // https://mvnrepository.com/artifact/org.apache.spark/spark-core
11 libraryDependencies += "org.apache.spark" %% "spark-core"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

```

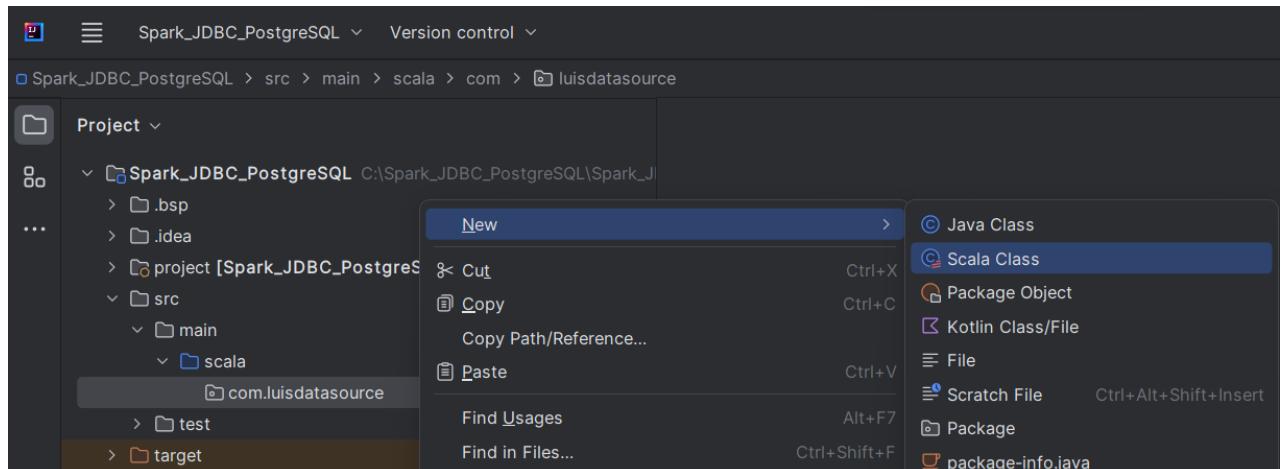
2.4. How to create a new package and new file inside the package ↗

Firs 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

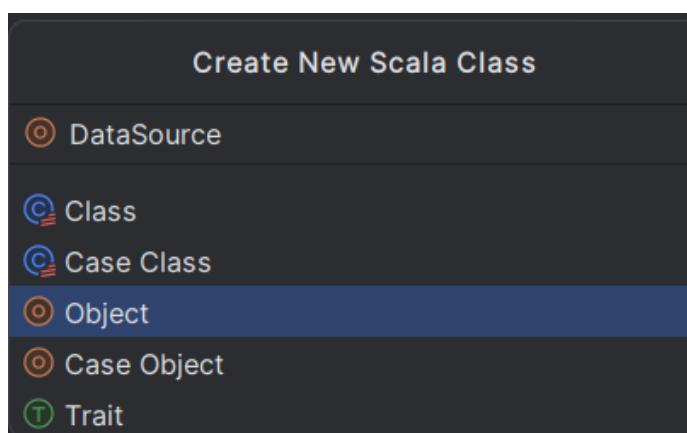




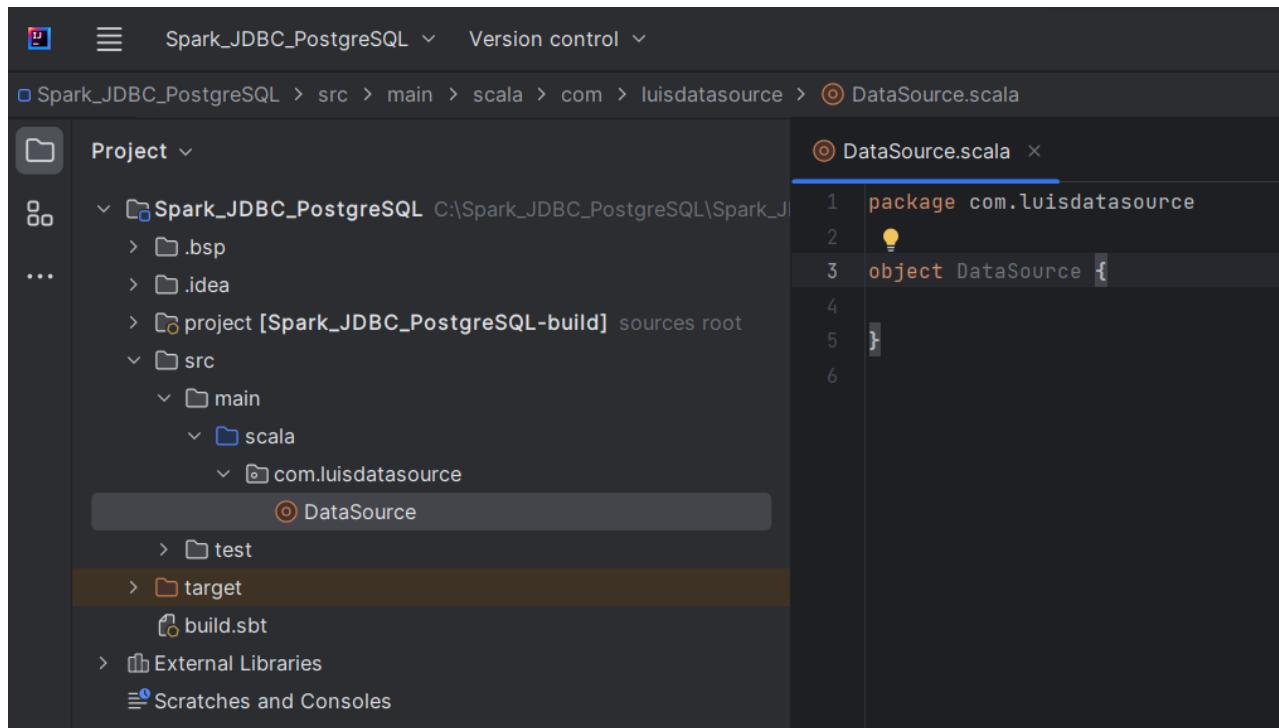
Now we create a new file inside the package. We select the menu options "New->Scala Class" and the "Object"



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. An we can see the execute green button appears in the left margin.

```

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 put
    .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()
```

```
 */
}
```

```

    .format( source = "json")
    .options(Map(
        "mode" -> "failFast",
        "path" -> "src/main/resources/data/cars.json",
        "inferSchema" -> "true"
    ))
    .load()

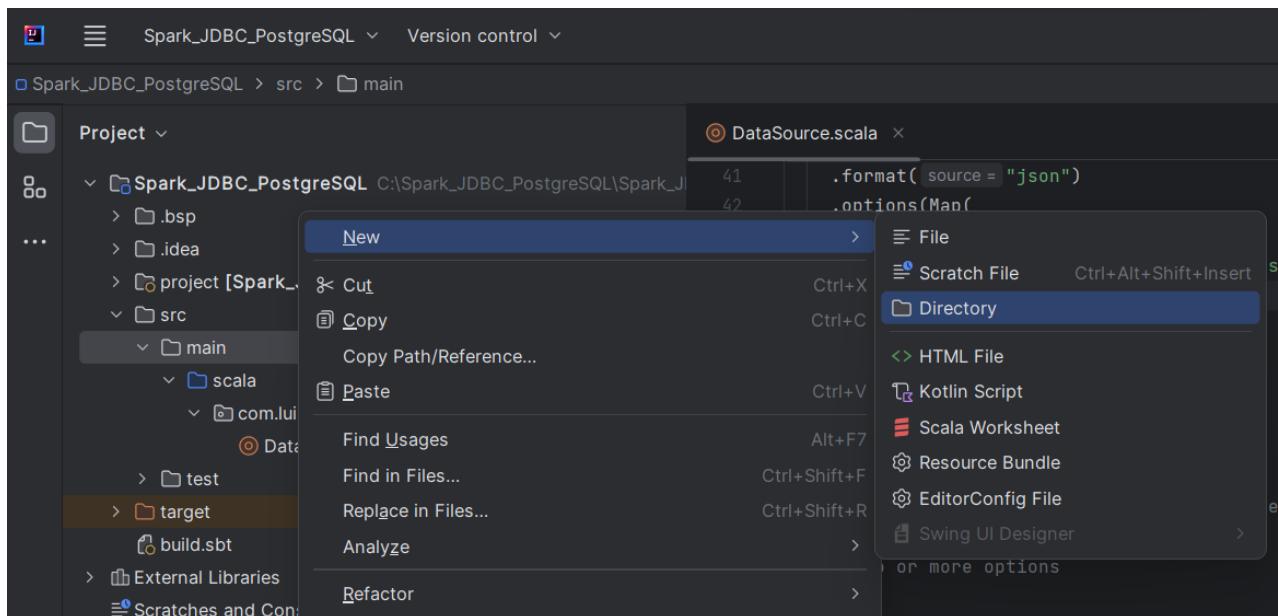
    /*
     Writing DFs
     - format
     - save mode = overwrite, append, ignore, errorIfExists
     - path
     - zero or more options
    */
    carsDF.write
    .format( source = "json")
    .mode(SaveMode.Overwrite)
    .save( path = "src/main/resources/data/cars_dupe.json")

    // JSON flags
    spark.read
    .schema(carsSchema)
    .option("dateFormat", "yyyy-MM-dd") // couple with schema; if Spark fails parsing, it will put null
}

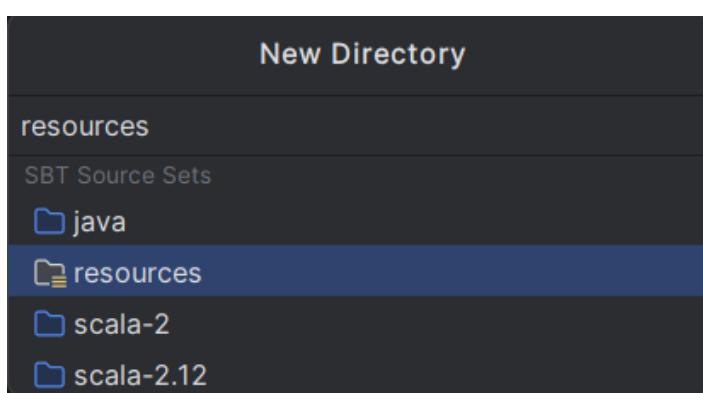
```

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

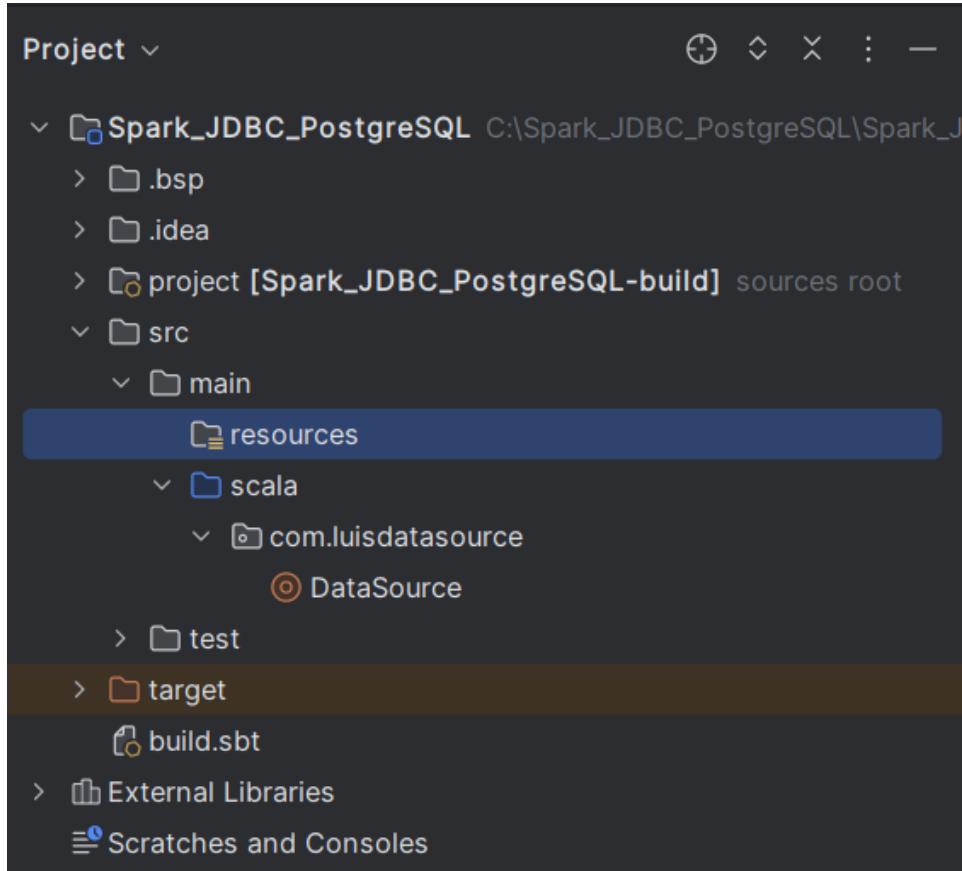
We right click on the "main" folder and we select the menu options "New->Directory"



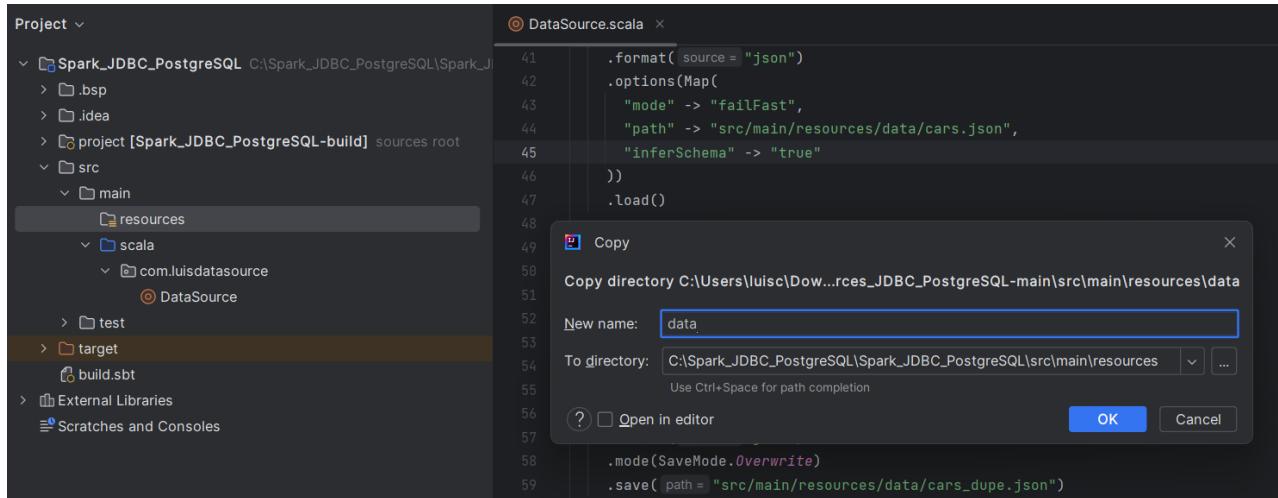
The we select "resources"



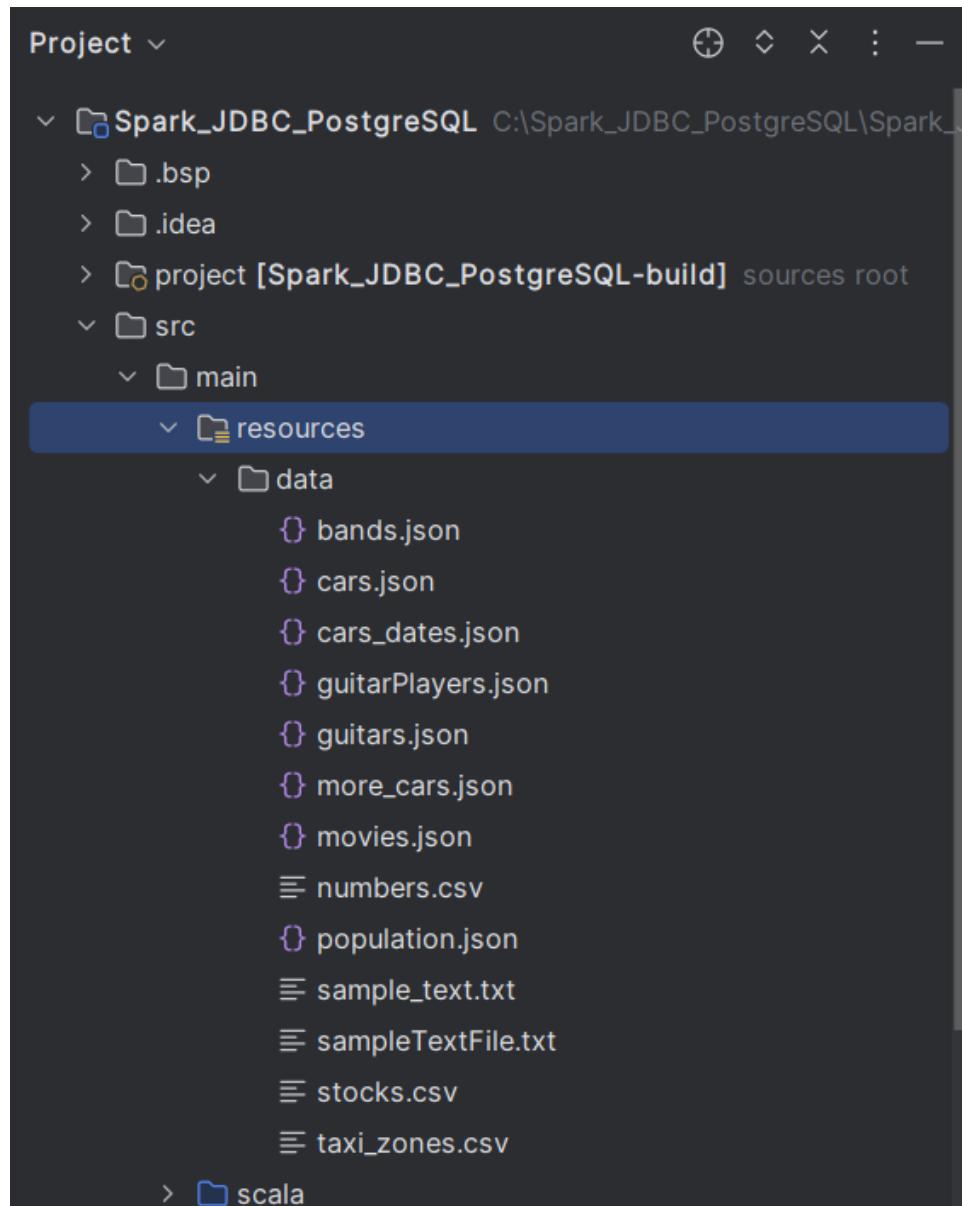
Now we can see the "resources" folder was created



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

The screenshot shows the IntelliJ IDEA code editor with the file 'bands.json' open. The file contains the following JSON data:

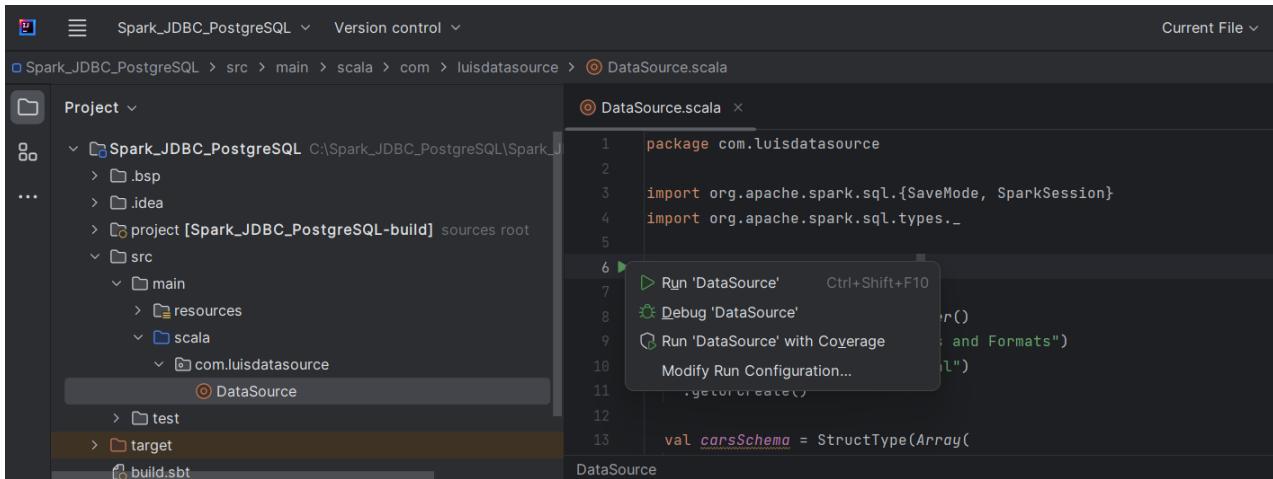
```

1  [{"id":1,"name":"AC/DC","hometown":"Sydney","year":1973}
2  {"id":0,"name":"Led Zeppelin","hometown":"London","year":1968}
3  {"id":3,"name":"Metallica","hometown":"Los Angeles","year":1981}
4  {"id":4,"name":"The Beatles","hometown":"Liverpool","year":1960}
5

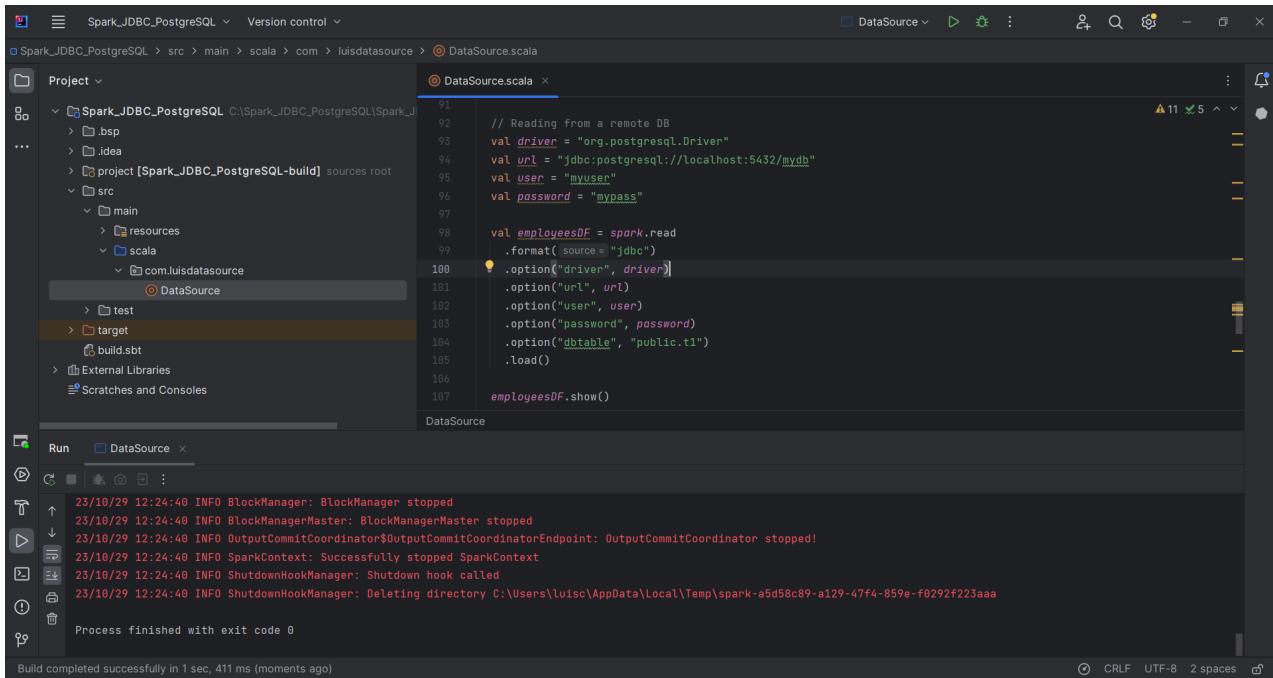
```

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 PostgreSQL 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 5433"
```

2.8. Application output ↗

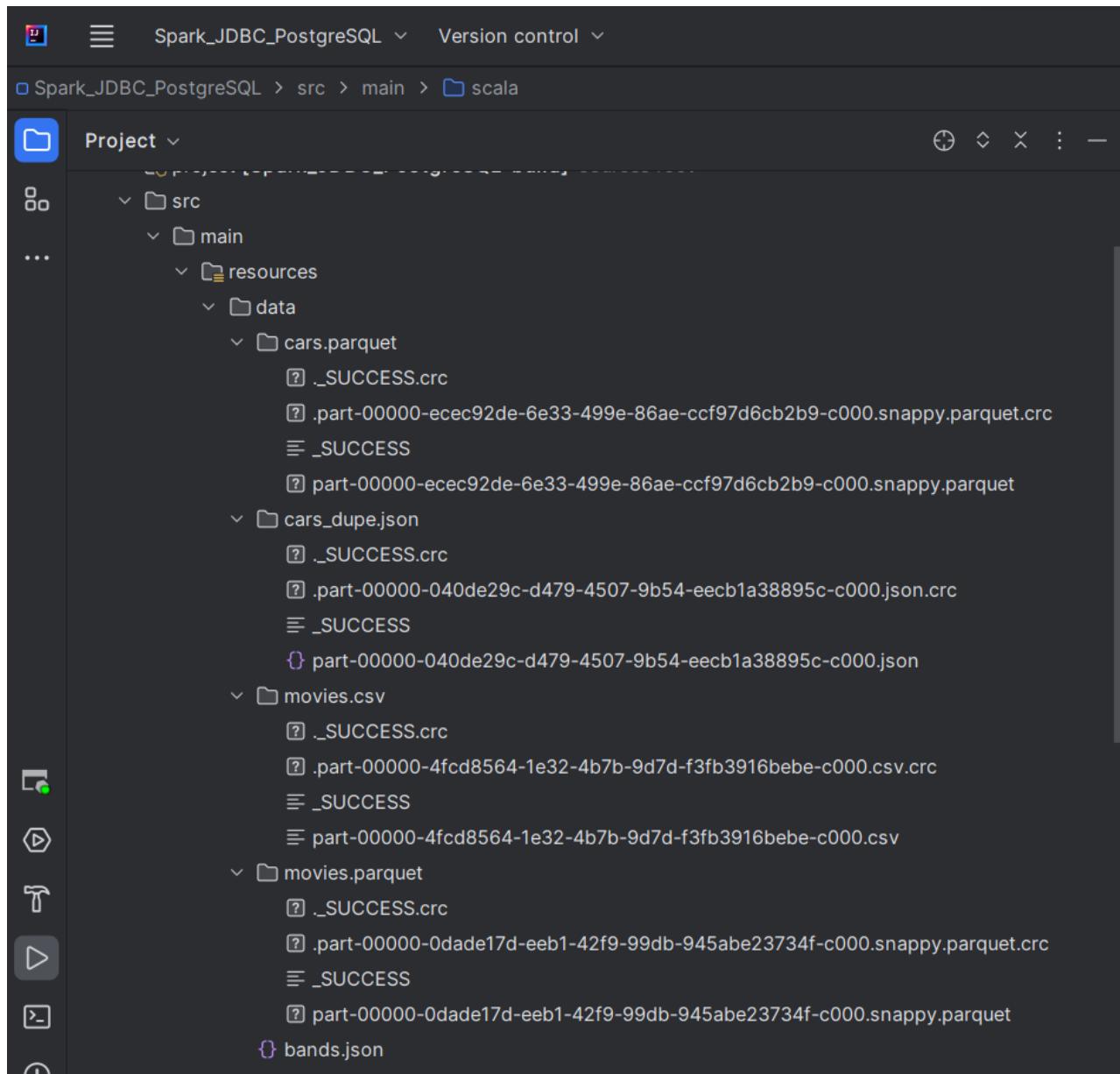
The screenshot shows a terminal window with the following content:

```
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|
+----+
```

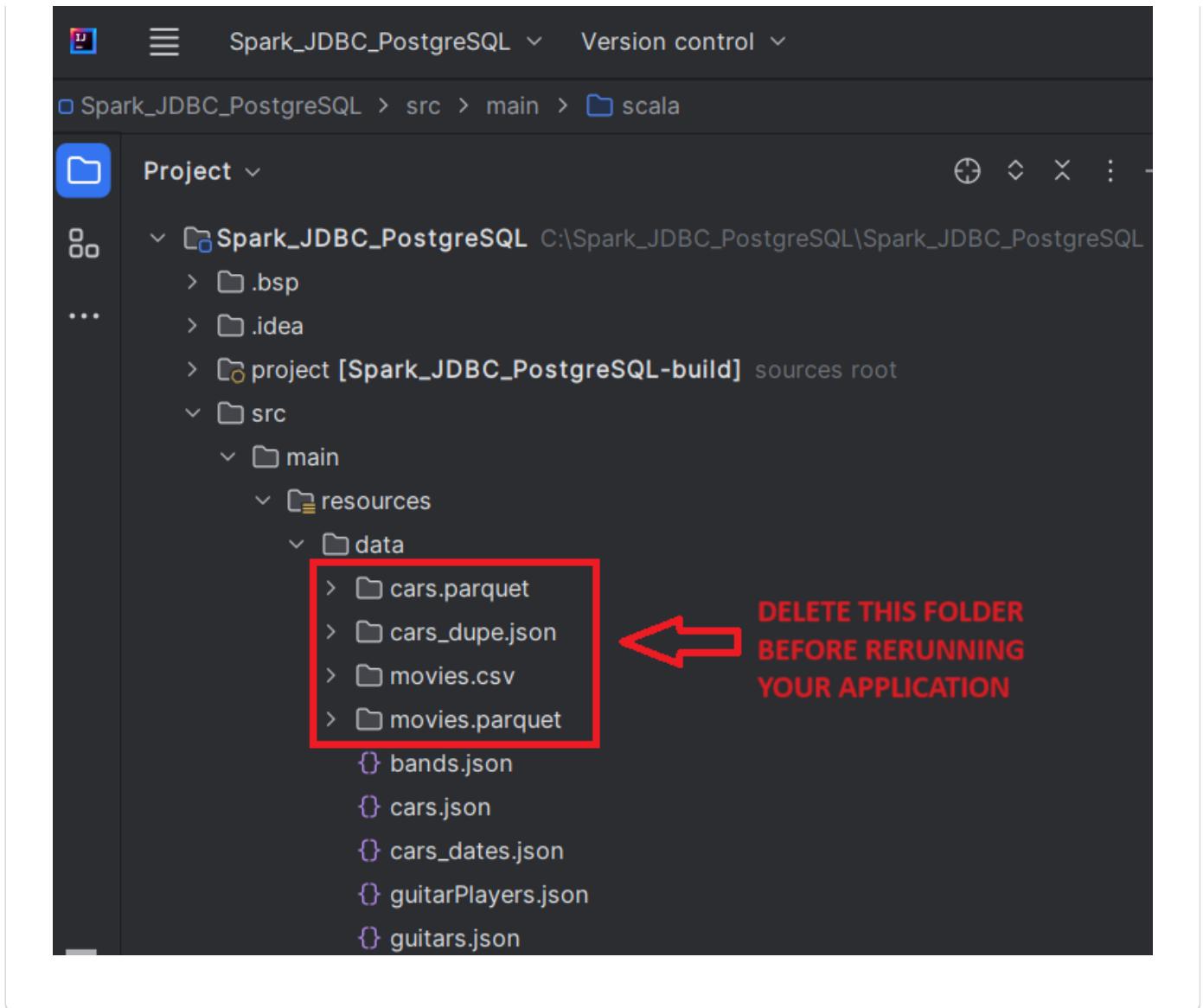
The screenshot shows a terminal window with the following content:

```
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



Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Languages

- Scala 100.0%

Suggested Workflows

Based on your tech stack

 Scala

Build and test a Scala project with SBT.

[Configure](#)

**SLSA Generic generator**[Configure](#)

Generate SLSA3 provenance for your existing release workflows

[More workflows](#)[Dismiss suggestions](#)