

Manual for Postgresql Installation and Load part of the ETL

1. Open the terminal and type

```
ardent@ardent:~$ sudo apt install postgresql
[sudo] password for ardent:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm17t64
  libtypes-serialiser-perl postgresql-16 postgresql-client-16
  postgresql-client-common postgresql-common
```

Enter your password when/if prompted

2. After the installation has been completed. Let us change to the postgres user which is the default user for postgres database. Then as a postgres user run psql command.

```
ardent@ardent:~$ sudo -i -u postgres
postgres@ardent:~$ psql
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

postgres=#
```

3. Let us setup a password for the default user which we will need for future data transfer and query.

```
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

postgres=# alter user postgres with password 'postgres';
ALTER ROLE
postgres=#
```

Follow the above alter user syntax. The first postgres is the name of the user, the second postgres in single quotation is the password. Make sure you keep a password of your choice. While this is local system and the strength does not matter much, it is good behavior to keep strong password always.

4. Exit out of psql then logout of user postgres. User \q and exit respectively

```
postgres=# \q
postgres@ardent:~$ exit
logout
ardent@ardent:~$
```

5. Try accessing psql as user ardent or as user postgres from outside user postgres area

```
ardent@ardent:~$ psql
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432"
failed: FATAL:  role "ardent" does not exist
ardent@ardent:~$ psql -U postgres
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432"
failed: FATAL:  Peer authentication failed for user "postgres"
ardent@ardent:~$
```

The first command fails because you are trying to login as user ardent which does not exist. The second command fails because only peer based authentication has been set and no password based authentication is available. So let us change that next

6. Update pg_hba.conf file of postgres

```
ardent@ardent:~$ sudo vim /etc/postgresql/16/main/pg_hba.conf
```

Please use the text editor of your choice. Make sure you are accessing it as sudo as other users are not allowed to make changes. Alternatively you could login to the postgres user as before using `sudo -i -u postgres` and then open the file normally

```
# Database administrative login by Unix domain socket
local all postgres peer

# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all peer
# IPv4 local connections:
host all all 127.0.0.1/32 scram-sha-256
```

If we scroll down we will find the following texts. Let us change peer to md5

```

# Database administrative login by Unix domain socket
local    all             postgres                                md5

# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local    all             all              md5
# IPv4 local connections:
host     all             all              127.0.0.1/32           scram-sha-256

```

Save and close the file

7. Restart postgresql services to update the changes

```

ardent@ardent:~$ sudo vim /etc/postgresql/16/main/pg_hba.conf
ardent@ardent:~$ sudo systemctl restart postgresql

```

8. Try to run psql as postgres user

```

ardent@ardent:~$ psql -U postgres
Password for user postgres:
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

postgres=#

```

You will be prompted for a password and when provided you can login. Congratulations! Postgres with password based authentication has been setup in your system.

9. Setup gui tool pgadmin to view postgres db data

Use below commands in succession. This should work for most part, if it does not use the instruction provided here: <https://www.pgadmin.org/download/pgadmin-4-apt/>

```
curl -fsS https://www.pgadmin.org/static/packages_pgadmin_org.pub | sudo gpg --dearmor -o /usr/share/keyrings/packages-pgadmin-org.gpg
```

```
sudo sh -c 'echo "deb [signed-by=/usr/share/keyrings/packages-pgadmin-org.gpg] https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/$(lsb_release -cs) pgadmin4 main" > /etc/apt/sources.list.d/pgadmin4.list && apt update'
```

```
sudo apt install pgadmin4
```

```

ardent@ardent:~$
ardent@ardent:~$ curl -fsS https://www.pgadmin.org/static/packages_pgadmin_org.p
ub | sudo gpg --dearmor -o /usr/share/keyrings/packages-pgadmin-org.gpg

sudo sh -c 'echo "deb [signed-by=/usr/share/keyrings/packages-pgadmin-org.gpg] h
ttps://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/$(lsb_release -cs) pgadmin4 m
ain" > /etc/apt/sources.list.d/pgadmin4.list && apt update'

sudo apt install pgadmin4
[sudo] password for ardent:
Get:1 https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/noble pgadmin4 InRelea
se [4,217 B]
Hit:2 https://packages.microsoft.com/repos/code stable InRelease
Get:3 https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/noble pgadmin4/main al
l Packages [5,385 B]

```

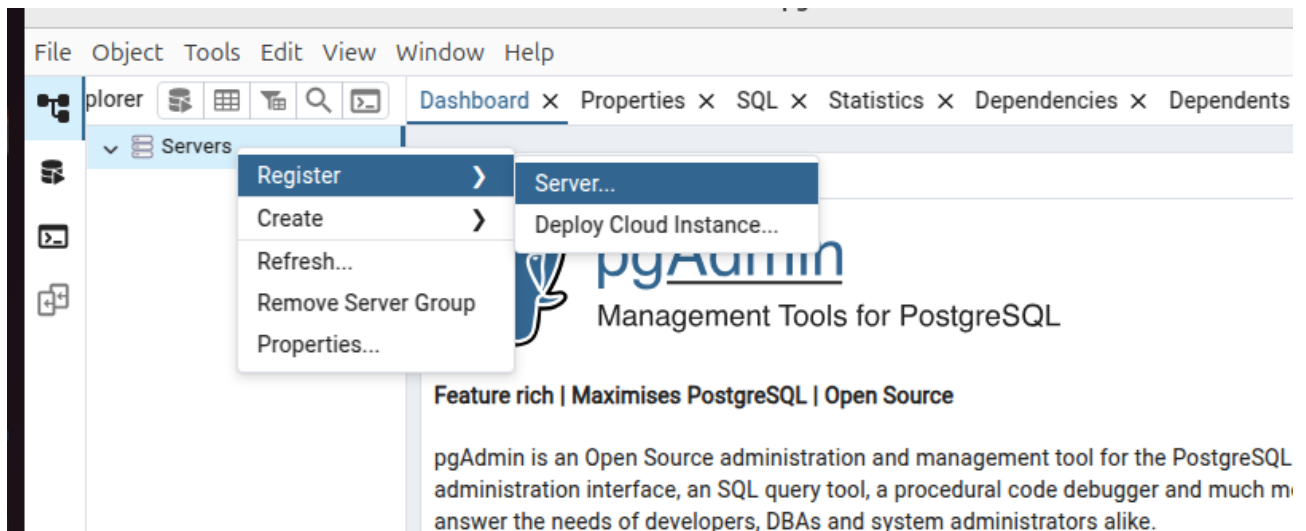
Enter password and Press Y when prompted

Wait for the installation to completed

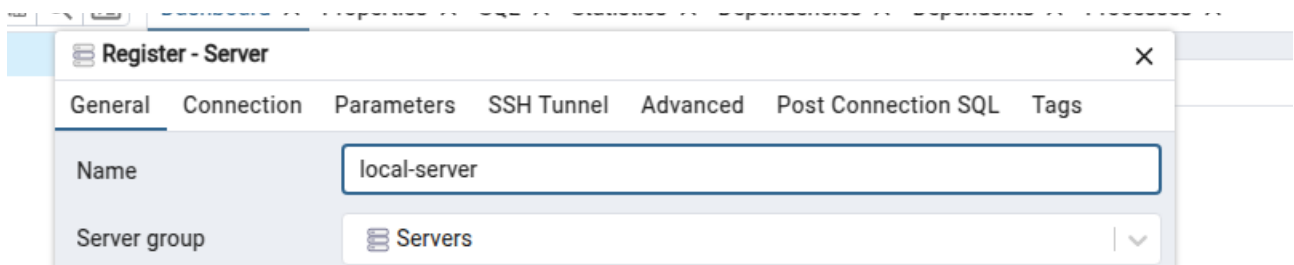
10. Go to search menu by pressing Windows key and search for pgadmin and open it



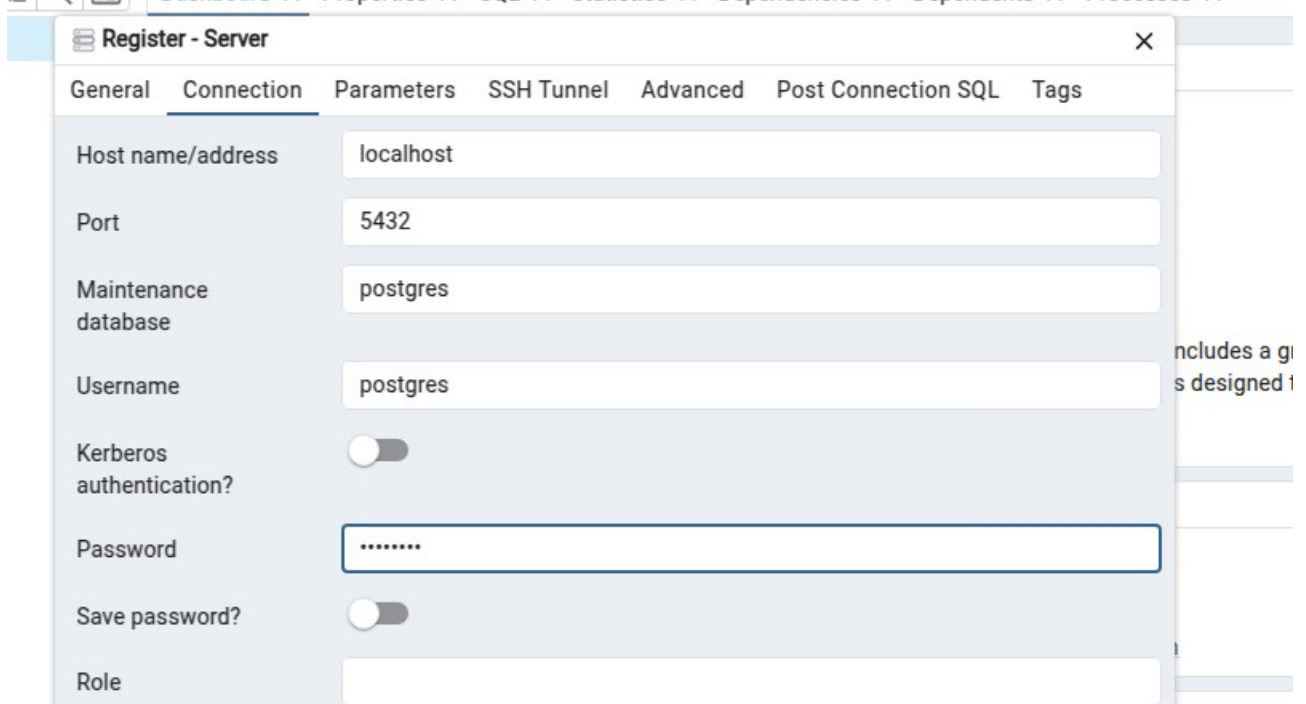
11. Go to Servers Right Click → Register → Server



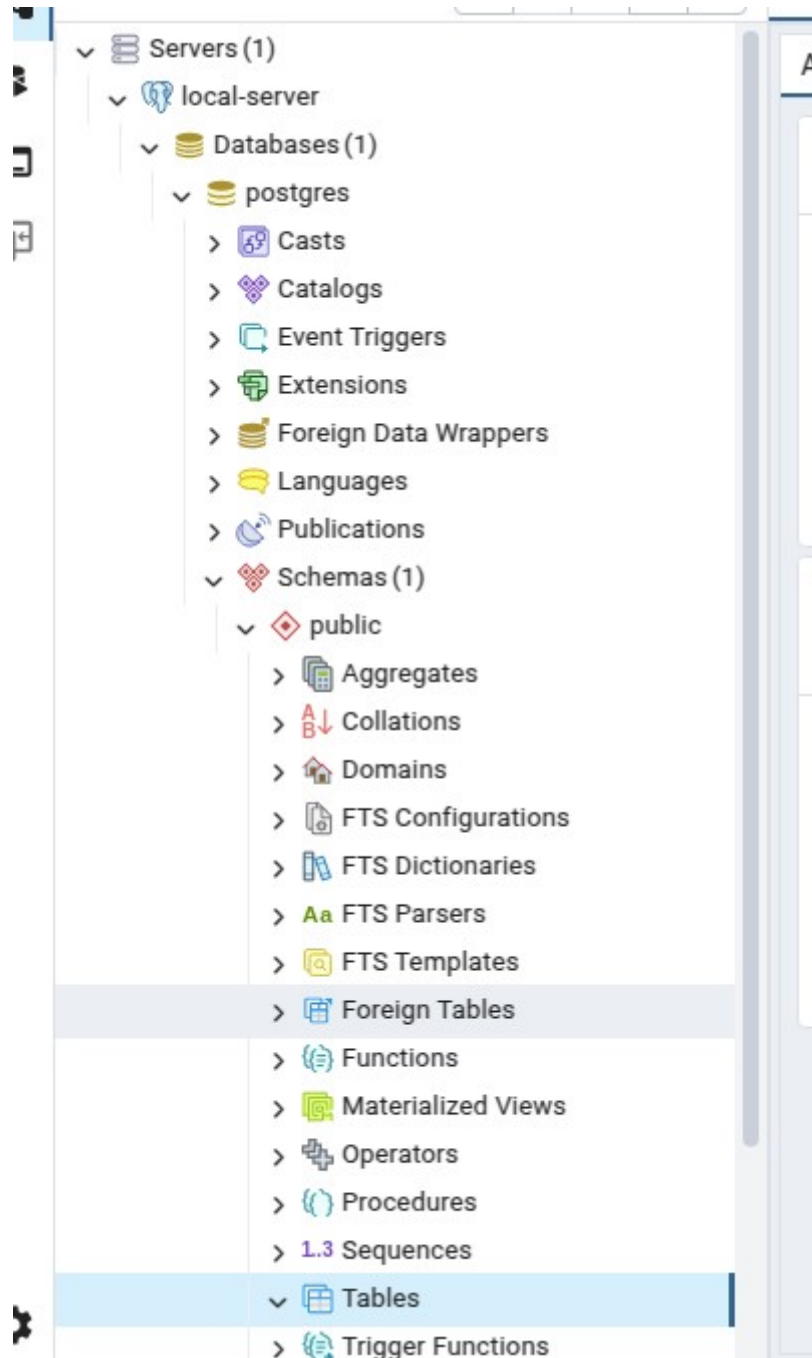
12. Fill out the required information



In the general tab give name to the server



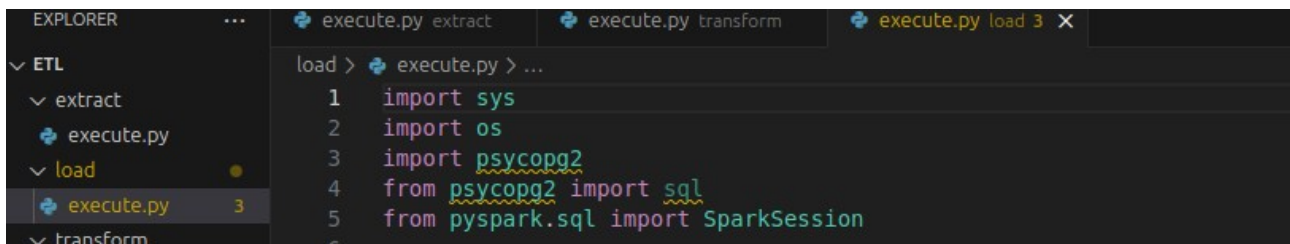
In the connection tab provide the username, password and set the host name to localhost then press save. If all the credentials were correct you should now be able to browse your local database



If you get above view you have completed the GUI setup as well. If you check the tables tab now there will be nothing.

Now let us implement the load part of our ETL pipeline. We will use pyscopg2 to create tables in the database and use pyspark jdbc driver to transfer the data to postgres. We will need to download the jar file later.

1. Open vs code in the etl directory and inside the load directory create new file execute.py and at the top of the file import the required libraries

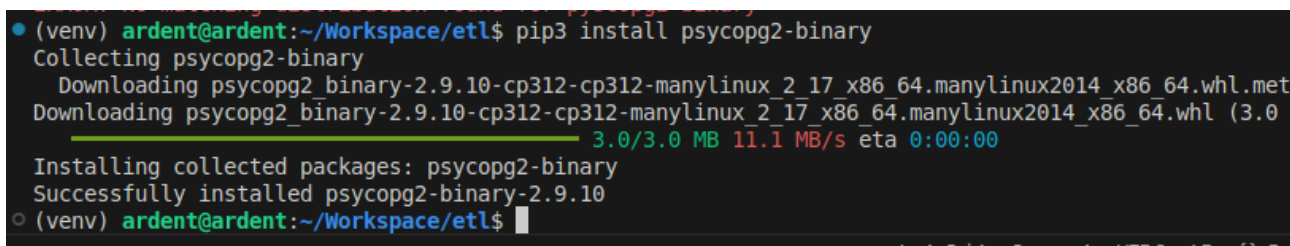


The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows the 'ETL' directory structure: 'extract', 'load', and 'transform'. Inside 'load', there is a file named 'execute.py'. The main editor window shows the content of 'execute.py' with the following code:

```
1 import sys
2 import os
3 import pyscopg2
4 from pyscopg2 import sql
5 from pyspark.sql import SparkSession
```

Since the pyscopg2 has not been installed yet. It will be underlined in yellow.

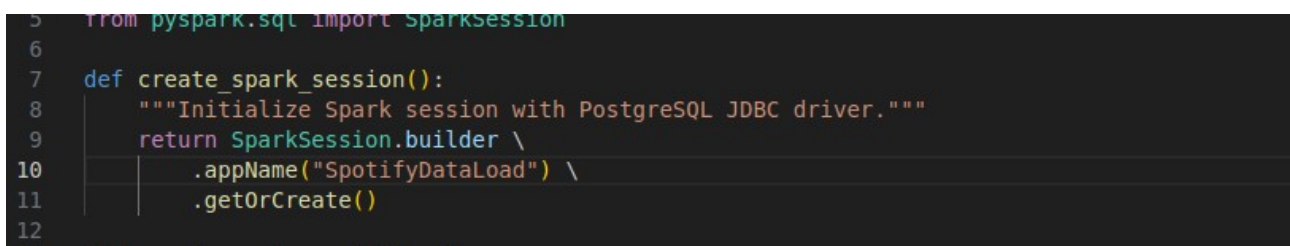
2. Make sure you have the virtual environment activated and install pyscopg2-binary package



The screenshot shows a terminal window with the following commands and output:

```
(venv) ardent@ardent:~/Workspace/etl$ pip3 install pyscopg2-binary
Collecting pyscopg2-binary
  Downloading pyscopg2_binary-2.9.10-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.met
  Downloading pyscopg2_binary-2.9.10-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.0
    3.0/3.0 MB 11.1 MB/s eta 0:00:00
Installing collected packages: pyscopg2-binary
Successfully installed pyscopg2-binary-2.9.10
(venv) ardent@ardent:~/Workspace/etl$
```

3. Similar to that for transform create a function that gets/creates and returns SparkSession



The screenshot shows the VS Code Editor with the following code in 'execute.py' (lines 5-12):

```
5 from pyspark.sql import SparkSession
6
7 def create_spark_session():
8     """Initialize Spark session with PostgreSQL JDBC driver."""
9     return SparkSession.builder \
10         .appName("SpotifyDataLoad") \
11         .getOrCreate()
12
```


4. Create the function to connect to database and create table if it does not exists

```
def create_postgres_tables():  
    """Create PostgreSQL tables if they don't exist using psycopg2."""  
    conn = None  
    try:  
        conn = psycopg2.connect(  
            dbname="postgres",  
            user="postgres",  
            password="postgres",  
            host="localhost",  
            port="5432"  
        )  
        cursor = conn.cursor()
```

Initially psycopg2 is used to connect to database. Now let us create a list of queries to run

```
create_table_queries = [  
    """  
    CREATE TABLE IF NOT EXISTS master_table (  
        track_id VARCHAR(50),  
        track_name TEXT,  
        track_popularity INTEGER,  
        artist_id VARCHAR(50),  
        artist_name TEXT,  
        followers FLOAT,  
        genres TEXT,  
        artist_popularity INTEGER,  
        danceability FLOAT,  
        energy FLOAT,  
        tempo FLOAT,  
        related_ids TEXT[]  
    );  
    """,  
    """  
    CREATE TABLE IF NOT EXISTS recommendations_explored (  
        id VARCHAR(50),  
        related_id VARCHAR(50)  
    );  
    """,  
    """  
    CREATE TABLE IF NOT EXISTS artist_track (  
        """
```



```

"""
CREATE TABLE IF NOT EXISTS artist_track (
    id VARCHAR(50),
    artist_id VARCHAR(50)
);
"""
,
"""
CREATE TABLE IF NOT EXISTS track_metadata (
    id VARCHAR(50) PRIMARY KEY,
    name TEXT,
    popularity INTEGER,
    duration_ms INTEGER,
    danceability FLOAT,
    energy FLOAT,
    tempo FLOAT
);
"""
,
"""
CREATE TABLE IF NOT EXISTS artist_metadata (
    id VARCHAR(50) PRIMARY KEY,
    name TEXT,
    followers FLOAT,
    popularity INTEGER
);
"""

```

]

Data type shown here might not be optimal so make sure you design your own create queries as per need.

```

        popularity INTEGER
    );
    """
]

for query in create_table_queries:
    cursor.execute(query)
conn.commit()
print("PostgreSQL tables created successfully")

except Exception as e:
    print(f"Error creating tables: {e}")
finally:
    if cursor:
        cursor.close()
    if conn:
        conn.close()

```

Finally execute the queries and commit them to database. Exception message is printed if it occurs and finally opened resources are closed.

5. Create function to read the transformed files and transfer them to postgres database. This function will take two input parameter: SparkSession and directory path of transformed files

```

88 def load_to_postgres(spark, input_dir):
89     """Load Parquet files to PostgreSQL."""
90     jdbc_url = "jdbc:postgresql://localhost:5432/postgres"
91     connection_properties = {
92         "user": "postgres",
93         "password": "postgres",
94         "driver": "org.postgresql.Driver"
95     }
96
97     tables = [
98         ("stage2/master_table", "master_table"),
99         ("stage3/recommendations_exploded", "recommendations_exploded"),
100        ("stage3/artist_track", "artist_track"),
101        ("stage3/track_metadata", "track_metadata"),
102        ("stage3/artist_metadata", "artist_metadata")
103    ]
104
105    for parquet_path, table_name in tables:

```

First we create the jdbc url. which contains the name of the server (localhost), the port to use (5432), and the name of database (postgres). Then we create a dict containing information required

for connection for pyspark. A list of tuple is created which contains mapping of nested folder with their table names.

```
        ("stage3/artist_metadata", "artist_metadata")
    ]

    for parquet_path, table_name in tables:
        try:
            df = spark.read.parquet(os.path.join(input_dir, parquet_path))
            mode = "append" if 'master' in parquet_path else "overwrite"
            df.write \
                .mode(mode) \
                .jdbc(url=jdbc_url, table=table_name, properties=connection_properties)
            print(f"Loaded {table_name} to PostgreSQL")
        except Exception as e:
            print(f"Error loading {table_name}: {e}")

if __name__ == "__main__":
```

Then we iterate over that list. And load each file into dataframe and then write to postgres using the jdbc_url, table_name, and connection properties we defined earlier. Also notice the logic for master table contains append meaning we want to keep on adding data to it while for other table we want to overwrite

6. Write the main logic to be run at startup

```
6  if __name__ == "__main__":
7      if len(sys.argv) != 2:
8          print("Usage: python load/execute.py <input_dir>")
9          sys.exit(1)
10
11     input_dir = sys.argv[1]
12
13     if not os.path.exists(input_dir):
14         print(f"Error: Input directory {input_dir} does not exist")
15         sys.exit(1)
16
17     spark = create_spark_session()
18     create_postgres_tables()
19     load_to_postgres(spark, input_dir)
20
21     print("Load stage completed")
22
```

Like before it checks if required parameter the directory path of transformed files is provided or not. If the path exists then spark session is created, tables are created in database and finally the data is transferred using pyspark.

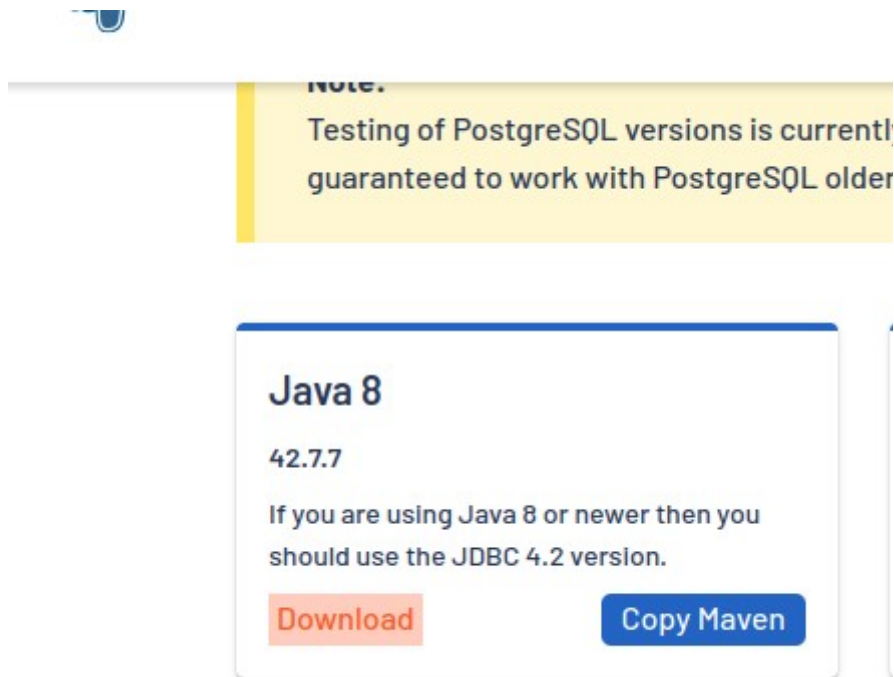
7. Run the Load script. It won't work as we have not added the jdbc driver for postgresql

```
Successfully installed psycopg2-binary-2.9.10
(venv) ardent@ardent:~/Workspace/etl$ python3 load/execute.py /home/ardent/Data/transform
25/06/29 16:08:44 WARN Utils: Your hostname, ardent resolves to a loopback address: 127.0.1.1; using 192.168.122.50 instead (on interface enp1s0)
25/06/29 16:08:44 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/06/29 16:08:44 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
PostgreSQL tables created successfully
Error loading master table: An error occurred while calling o30.jdbc.
: java.lang.ClassNotFoundException: org.postgresql.Driver
    at java.base/java.net.URLClassLoader.findClass(URLClassLoader.java:476)
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:594)
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:527)
    at org.apache.spark.sql.execution.datasources.jdbc.DriverRegistry$.register(DriverRegistry.scala:46)
    at org.apache.spark.sql.execution.datasources.jdbc.JDBCOptions.$anonfun$driverClass$1(JDBCOptions.scala:101)
    at org.apache.spark.sql.execution.datasources.jdbc.JDBCOptions.$anonfun$driverClass$1$adapted(JDBCOptions.scala:101)
101)
```

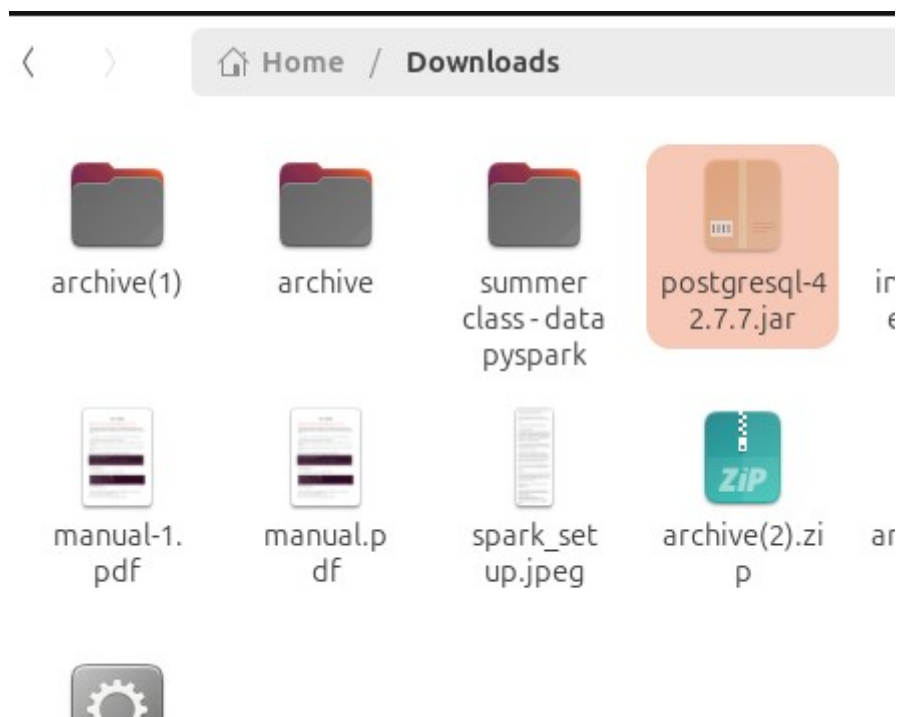
Go to google and search for the driver. Select the Download link



Download the JAVA 8 or newer supported jdbc

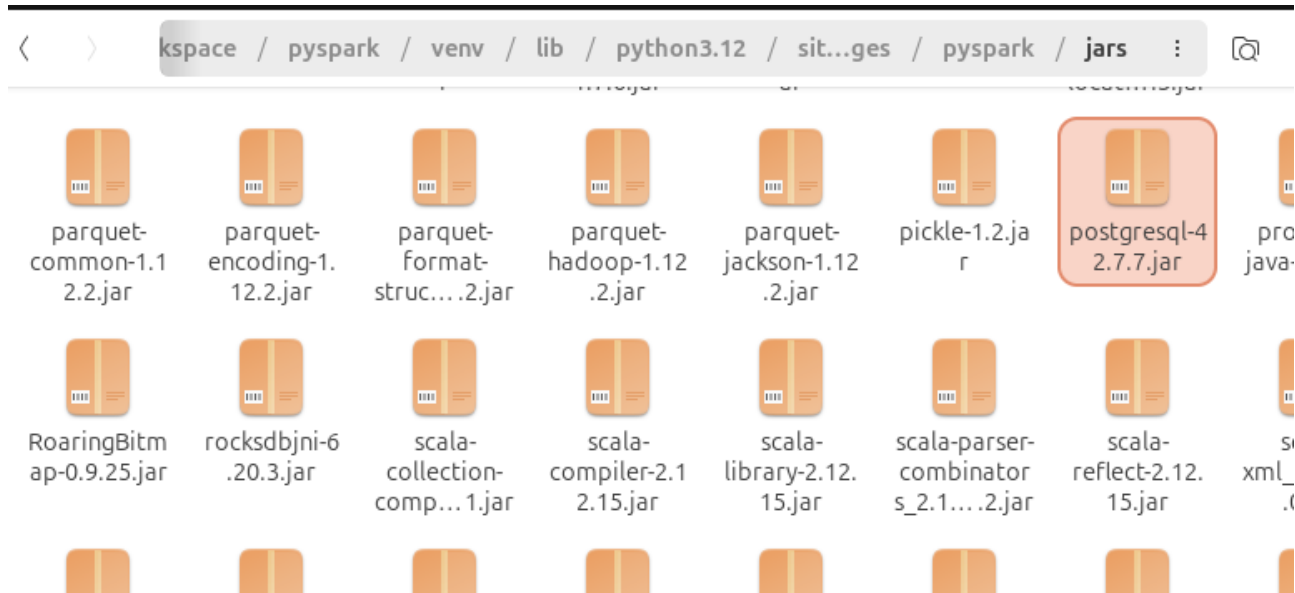


Go to Downloads and cut the jar file and move it to the pyspark jar installation file



Generally the pyspark jar is located at the location: venv(virtual environment folder)/lib/python3.12/site-packages/pyspark/jars

Paste the jar file you copied from Downloads and paste it here



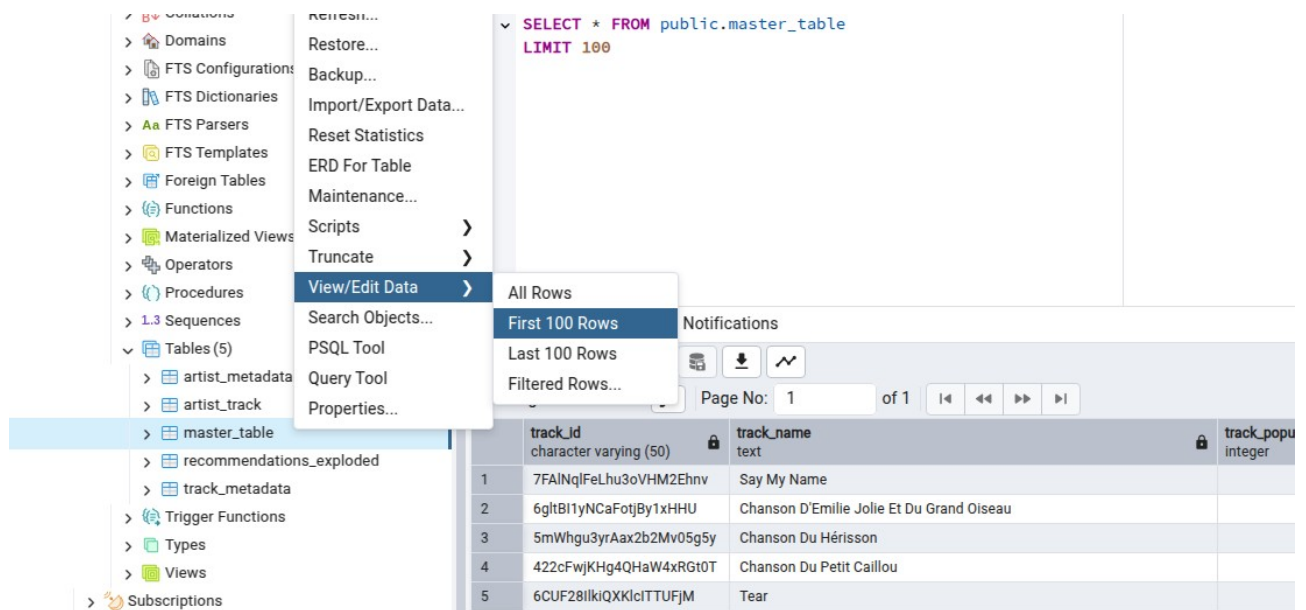
8. Rerun the script again

```
at java.base/java.lang.Thread.run(Thread.java:829)

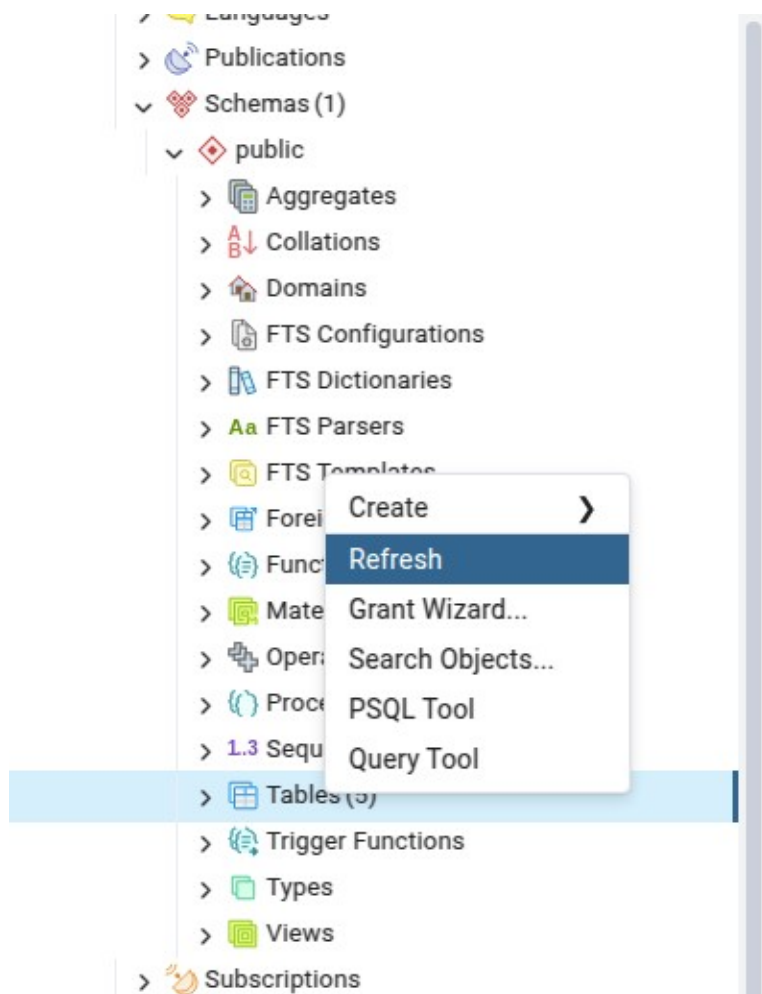
Load stage completed
(venv) ardent@ardent:~/Workspace/etl$ python3 load/execute.py /home/ardent/Data/transform
25/06/29 16:14:10 WARN Utils: Your hostname, ardent resolves to a loopback address: 127.0.1.1; using 192.168.122.50 instead (on interface enp1s0)
25/06/29 16:14:10 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/06/29 16:14:11 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
PostgreSQL tables created successfully
Loaded master_table to PostgreSQL
Loaded recommendations_explored to PostgreSQL
Loaded artist_track to PostgreSQL
Loaded track_metadata to PostgreSQL
Loaded artist_metadata to PostgreSQL
Load stage completed
(venv) ardent@ardent:~/Workspace/etl$
```

The tables should be created and populated properly

Let us check the pgadmin gui to verify that tables have been rightly created and populated.



Right click on the table and view data to verify all the tables have been correctly populated



If the tables have not populated you might have to right click and select refresh

Awesome! The ETL pipeline is complete. You have extracted the data from remote source, processed it and populated it to a database engine. Now data analyst or data scientist can query the database to get the required outputs. Let us do the task we did in week 2 for you favorite artists but by querying the database

Understand the below query before you use it for your own favorite artist. Also the query might be taking few seconds, explore indexing so that query response is fast. Generally BTREE index on the filter or join column helps a lot.

Query










Query History

```
1 select (select name from artist_metadata where id = r.related_id)
2 from recommendations_exploded r
3 inner join artist_metadata a
4 on r.id = a.id
5 where a.name = 'Nepathya'
```

Data Output

Messages

Notifications



SQL

Showing rows: 1 to 20

Page No:

	name text	
1	The Uglyz	
2	Arun Thapa	
3	Deepak Bajracharya	
4	Deep Shrestha	
5	Adrian Pradhan	
6	Cobweb	
7	1974 Ad	
8	Narayan Gopal	
9	Prashant Tamang	
10	Yogeshwor Amatya	
11	Mongolian Heart	
12	Anil Singh	