**Data Generation script:** DatabaseGeneration.java

**System Selected:** PostgreSQL Version 10.7

**Data Generation:**

Implemented a script in Java to automate the data generation process. The script has been given the connection parameters to the database which is running on our local Windows/Mac System. When the script is run, the connection to the database is established and using the established connection we are executing the CREATE TABLE statements and INSERT VALUES statement to populate the database.

The script when executed opens a Scanner instance through which we input the number of tables. Based on the number of tables, the script then takes in the name of each table and number of tuples in each table from user input. The scanner instance is then closed, and the data generation code is executed. In the data generation code, we first create the tables and populate them in a for loop. We are generating the data based on the *Attribute Specification of "Scalable" Wisconsin Benchmark Relations.* Data is generated in a for loop so one iteration of the for loop corresponds to one tuple in that table. The data generated for one tuple is then added to the batch of statements. Once all the tuples for that table have been generated and added to the batch, we execute the batch update statement to inject the data into the created database table.

To establish connection to the PostgreSQL server from the script we have used JDBC API. The JDBC API uses JDBC drivers to connect with the database. (The dependency jar *postgresql-42.2.5* has been included with this file).

**Reasons for choosing PostgreSQL:**

**/\*\*AMEE\*\*/**

We populated 3 tables namely:

ONEKTUP with 1000 tuples

TENKTUP1 with 10000 tuples

TENKTUP2 with 10000 tuples

A screenshot of a computer screen

Description automatically generated

Attached below is ONEKTUP table screenshot.

A screenshot of a computer

Description automatically generated

One of the difficulties we faced while doing the first part of the project was in data generation especially string generation. At first string generation was confusing but with revisions of the code we were able to get clarity. Another difficulty we faced was installing Postgres 10.7.2 on Mac System but after one day’s struggle we were able to successfully install it. One of the lessons we learned while implementing Part 1 of the project was the importance of understanding the data completely before populating the database. Understanding the data gives us a better hold in query execution. Testing the script generating the data for all possible edge cases before actually injecting data into the database is very important. Many of the data anomalies are easier to figure out with the script than after they are inserted into the database.