**Database Systems**

**Fall-2024 Department of Computer Science**

**The Islamia University Bahawalpur**

**Class 4th MA**

**Submitted by: Abdullah Bin Masood(S23BDOCS1M01061)**

**Submitted to: Muhammad Usman Ghani**

1. **Introduction:**
   1. **Background:**

The code here seems to set up a database schema for keeping track of professors at universities, their affiliations with organizations, and the universities associated with those professors.  
 **Problem:**The data were probably in one table (uni\_professors) initially, with much redundant information and likely inconsistencies.  
  
Such a structure makes data management, querying, and analysis inefficient and prone to errors.  
**Context:**  
The project wants to improve the organization and integrity of the data through normalization and improve the efficiency of the database structure.

**1.2. Goal:  
  
The main goal of this project is to:  
  
Normalize the data:**Remove redundancy in the data by putting information into separate tables (professors, universities, organizations, affiliations).  
Increase consistency of the data and reduce the likelihood of data anomalies.

**Create a well-structured database:**Use the appropriate data types, constraints (primary keys, foreign keys), and indexes to ensure the integrity of the data and improve the performance of the database.  
Allow for efficient data retrieval and analysis:  
  
The schema should allow for many different queries concerning professors, their affiliations, and their universities.

**1.3. Requirements:  
  
User Requirements:  
  
Ability to store and retrieve professor's information:**Store the names of professors (firstname, lastname).  
Store university details (university name, short name, city).  
Relate professors to universities.

**Ability to store and retrieve information about organizations:**Store organization names and their sectors.

**Ability to store and retrieve information about professor affiliations:**

Store professor's affiliations with organizations (function, organization).  
Associate affiliations with both professors and organizations.

**Data Integrity:**

Ensure the uniqueness of university short names and organization names.  
 Prevent null values for essential fields like professor names.

**Data Consistency:** Maintain consistent relationships between professors, universities, and organizations.

1. **Functional Description:**
   1. **Method of use**

Based on the database schema created in the code, this system would be used by the people who need to manage and analyze data related to university professors and their affiliations.

**Possible User Roles:**

**Database Administrators:**

Responsible for maintaining and updating the database.

Would use the system to:

Add, update, and delete professor records.

Add, update, and delete university records.

Add, update, and delete organization records.

Add, update and delete professor affiliation records.

Ensure data integrity and consistency.

Performs database backup and recovery operations.

**Investigators:**

**Could use the system to:**

Identify professors with specific expertise or affiliations.

Investigate collaborative research between professors and organizations.

Study the distribution of professors among various universities and cities.

Generate reports about professor affiliations and research activities.

**University Administrators:**

**Could use the system to:**

Track professor affiliations with outside organizations.

Manage professors' profiles in the university.

Generate reports on professor activities and achievements.

**Human Resources:**

**Could use the system to:**

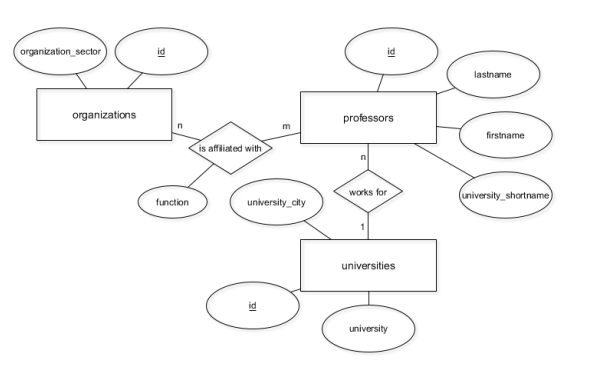
Manage professor personnel records.

Track professor professional development.

Assist in recruitment and hiring processes.

1. **Entity Data Model**

The code defines a relational database schema for keeping track of information about university professors, their affiliations with organizations, and their associated universities. The following diagram shows the EDM:

****

**Description:  
  
The EDM has four main entities:  
  
Professors:**This represents individual professors with attributes including id, firstname, lastname, and university\_shortname. **Universities:**This represents universities with attributes including id, university, and university\_city.  
**Organizations:**This represents organizations with attributes including id and organization\_sector.  
**Affiliations:**This represents the many-to-many relationship between professors and organizations, with attributes including function and university\_city.

**Relationships:  
  
One-to-many relationship between Universities and Professors:**

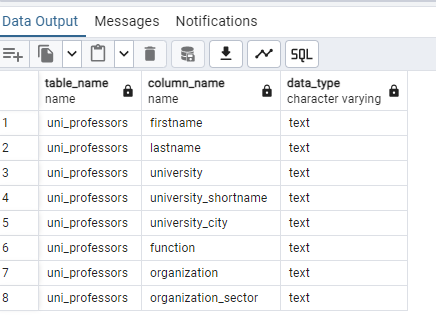
A university can have multiple professors, while a professor is associated with only one university. This is represented by the university\_shortname foreign key in the Professors table.

**Many-to-many relationship between Professors and Organizations:**

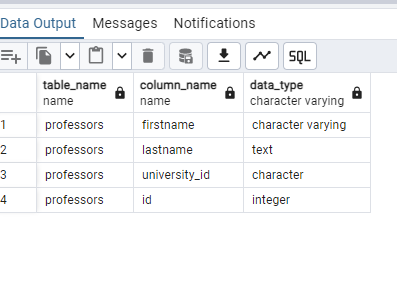
A professor may have multiple affiliations with different organizations, and an organization may have multiple affiliations with different professors. This relationship is implemented through the Affiliations table, which acts as a junction table.  
This EDM provides a clear and concise representation of the data structure, allowing efficient storage, retrieval, and analysis of data.  
 **Note:**The university\_shortname attribute in the Professors table is probably a foreign key referencing the id attribute in the Universities table, implementing the one-to-many relationship.  
The Affiliations table would preferably have foreign keys referring to both id attributes in the Professors and Organizations tables, thus implementing the many-to-many relationship.

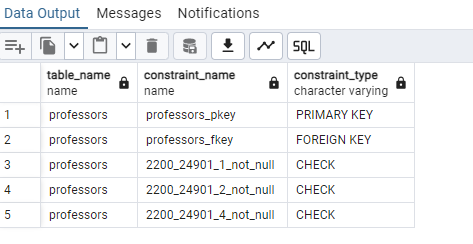
1. **Table Design (Schema) Screenshots**

**🡪Schema of uni\_professors table**

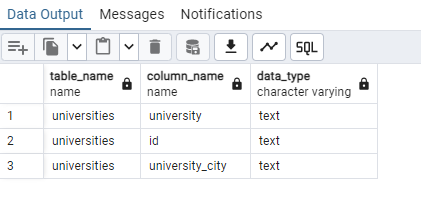


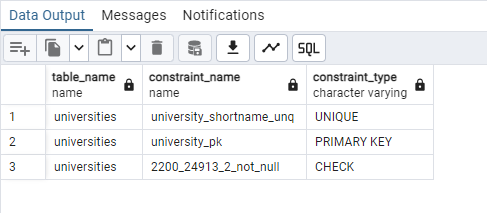
**🡪Schema of professors table**



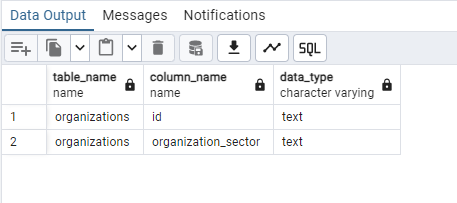


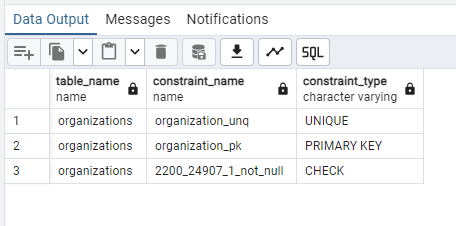
**🡪Schema of universities table**



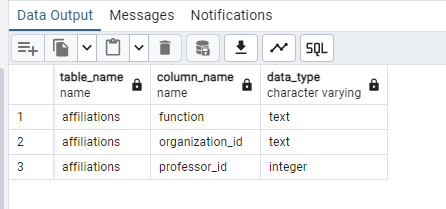


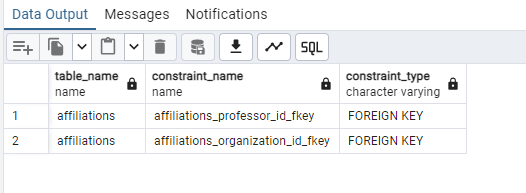
**🡪Schema of organizations table**



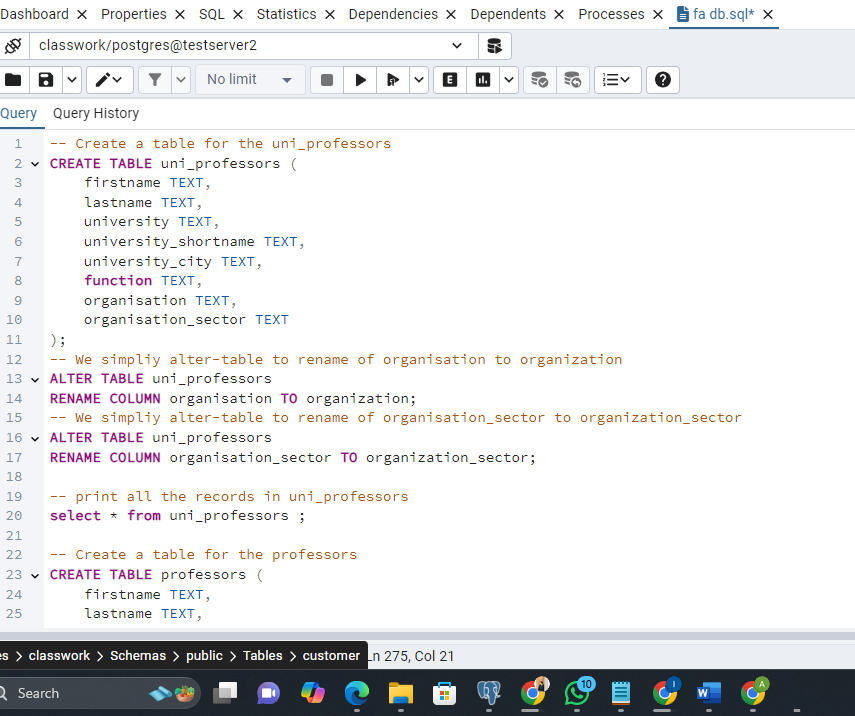


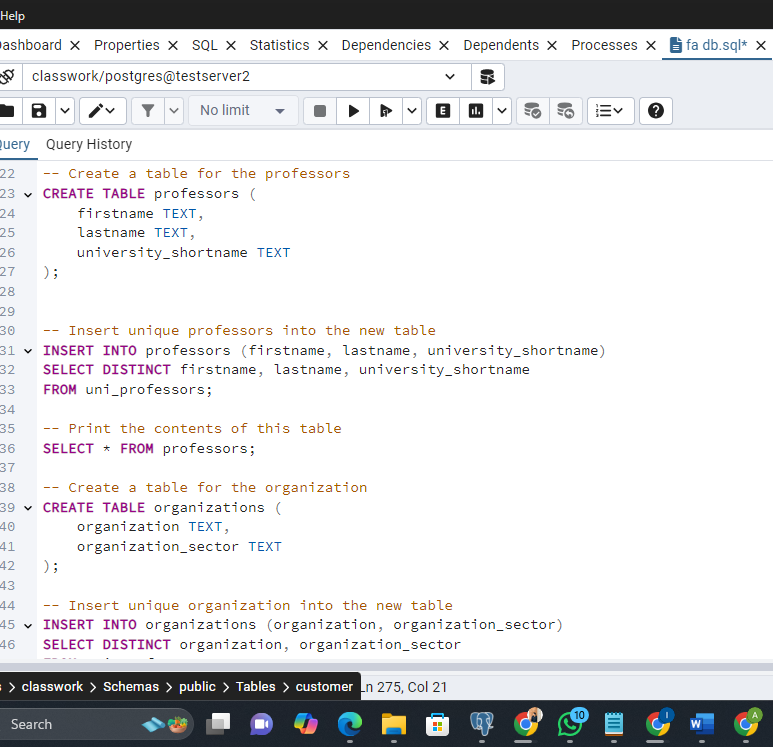
**🡪Schema of affiliations table**

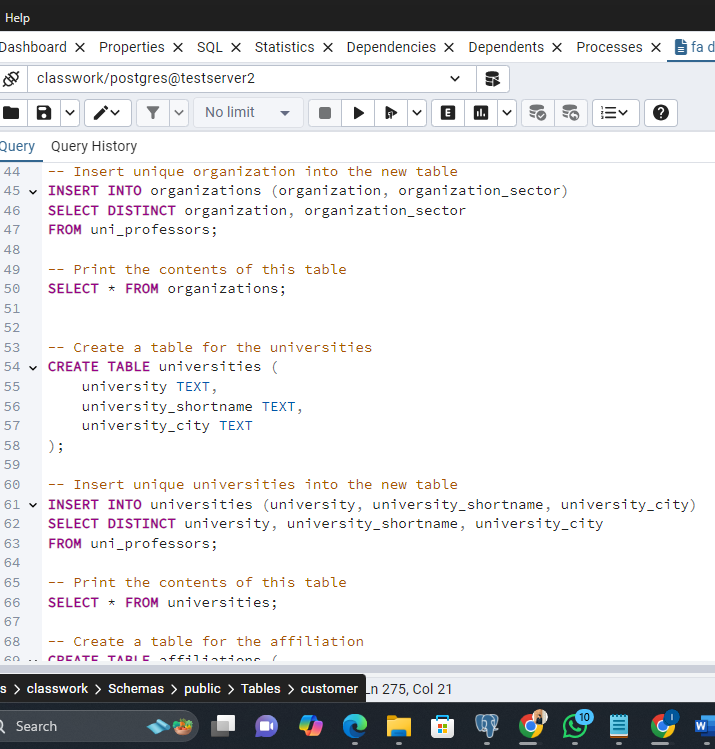


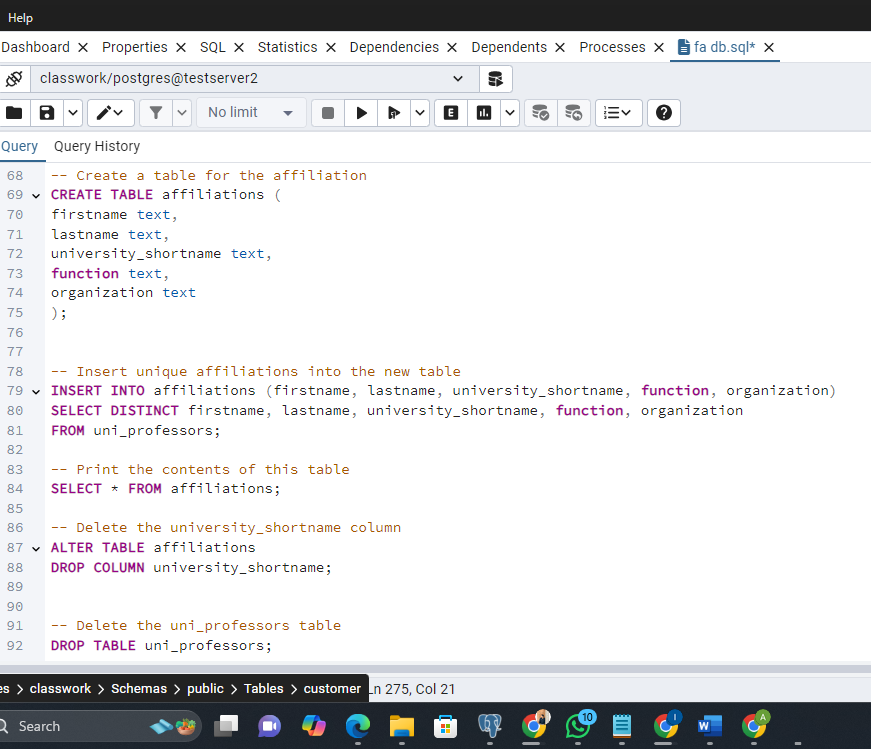


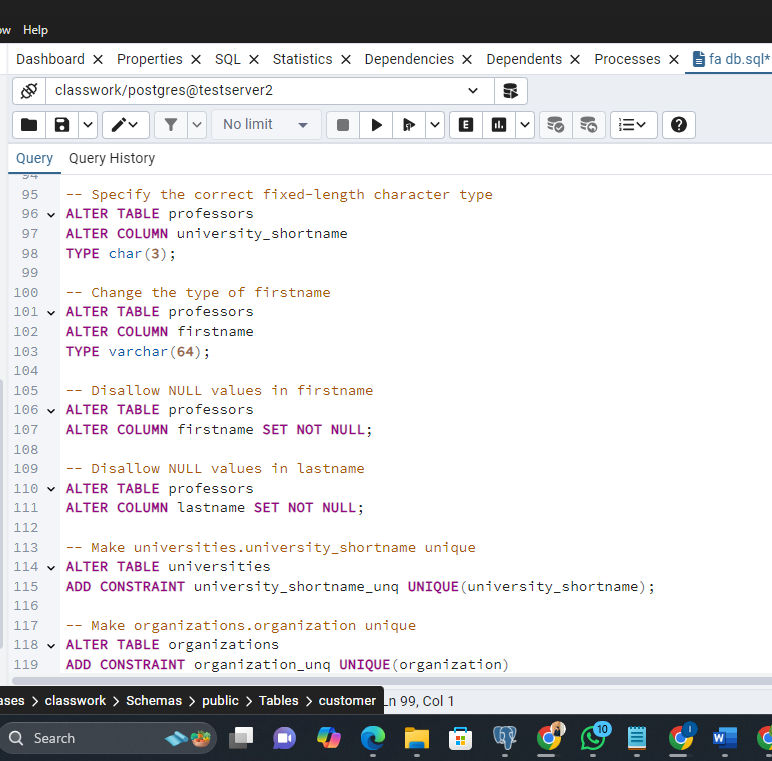
1. **Frontend Screenshots:**

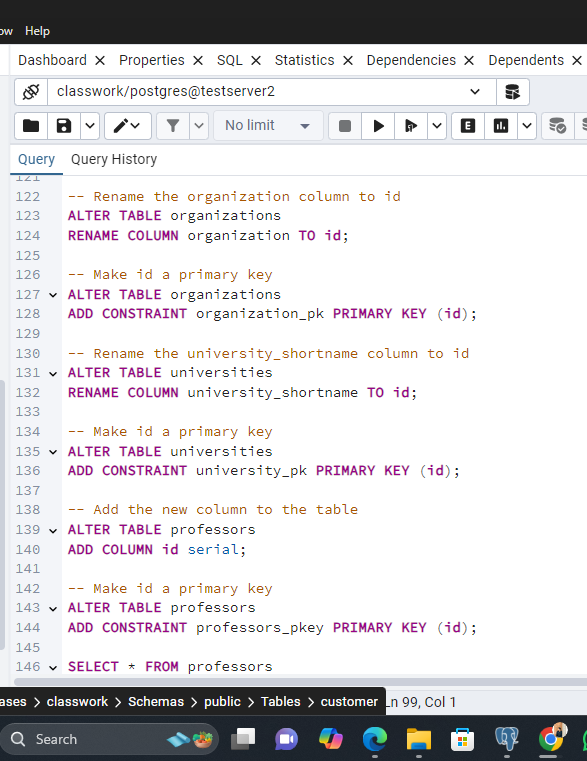


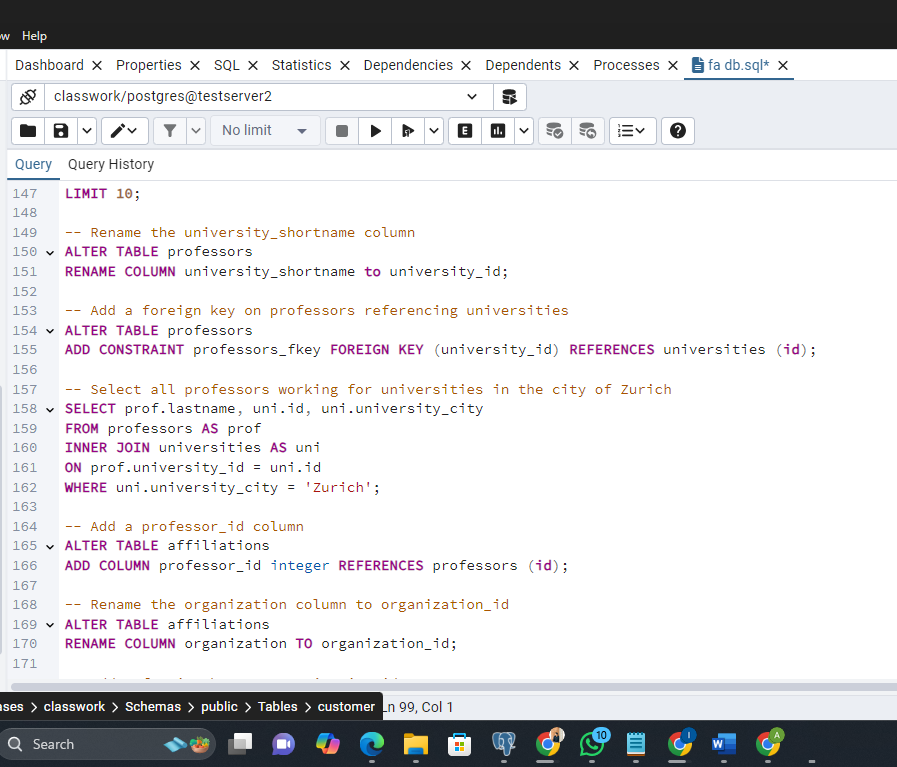


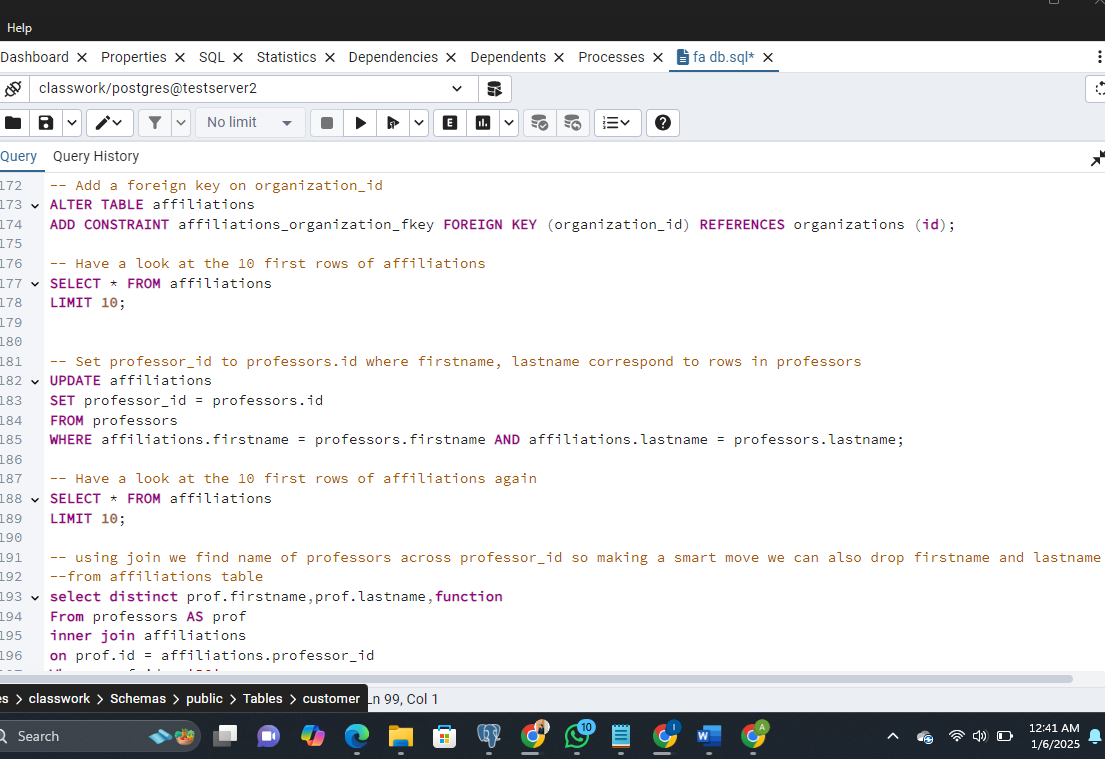


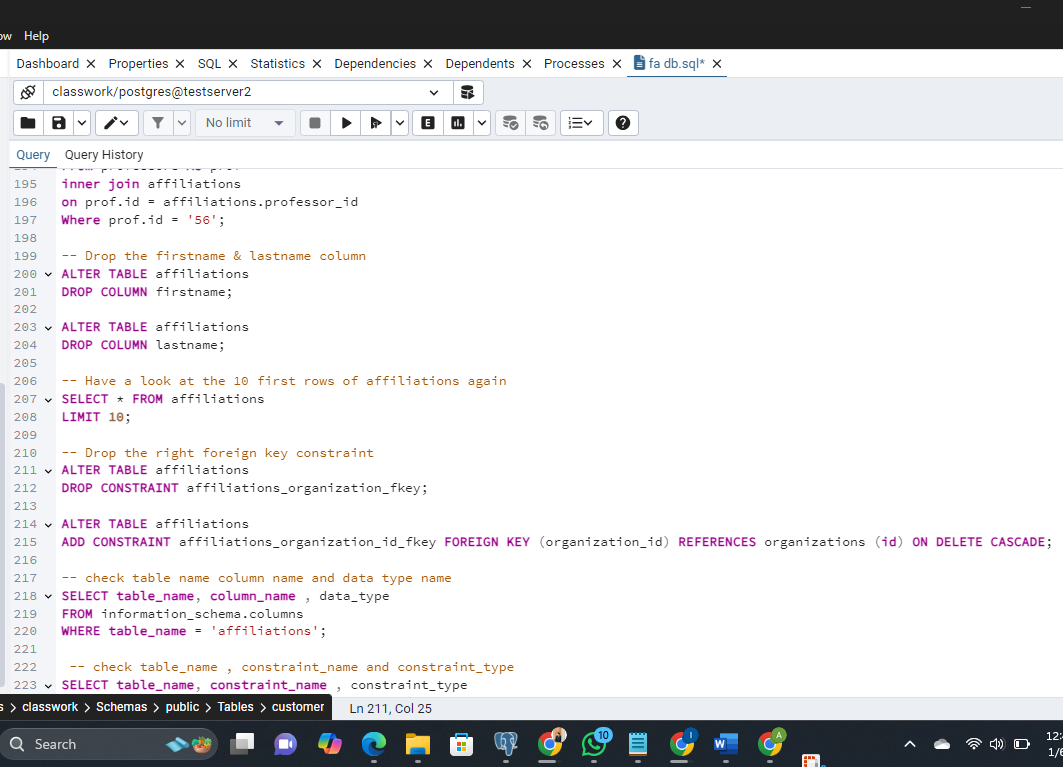


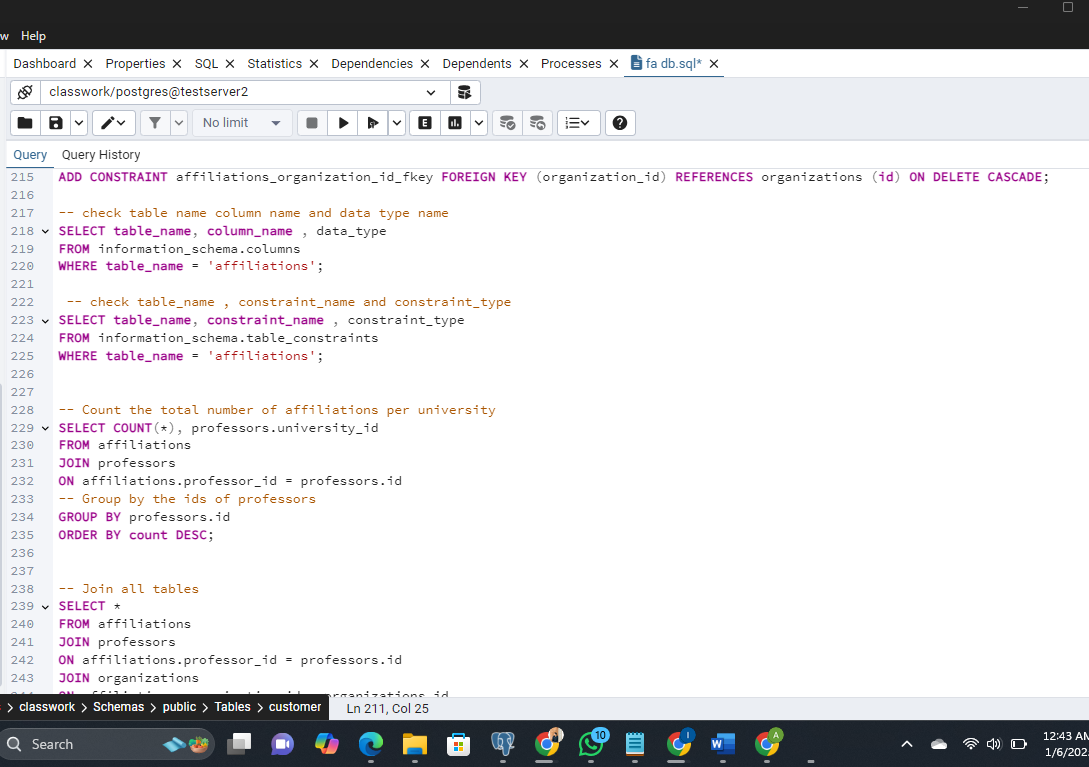


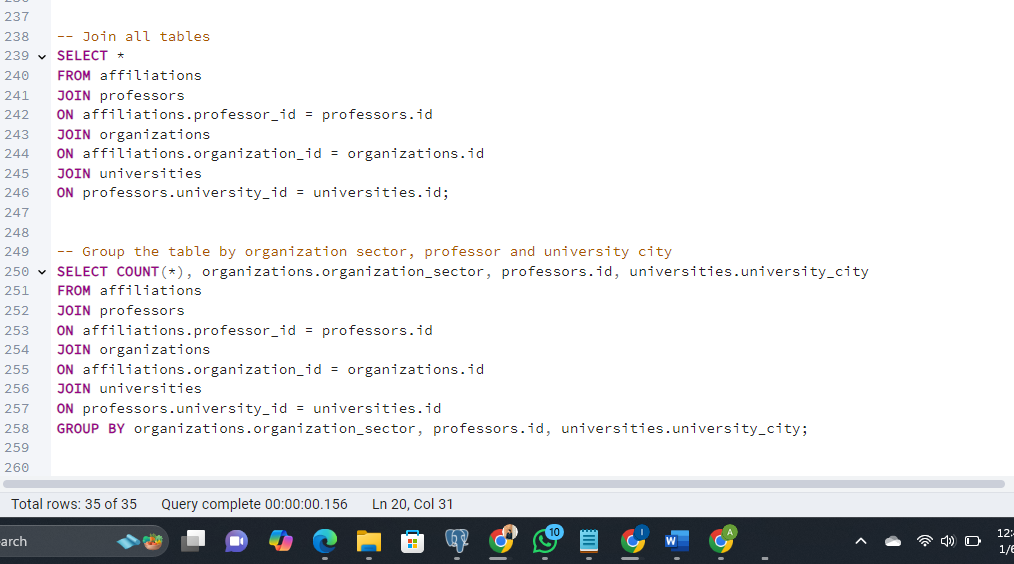


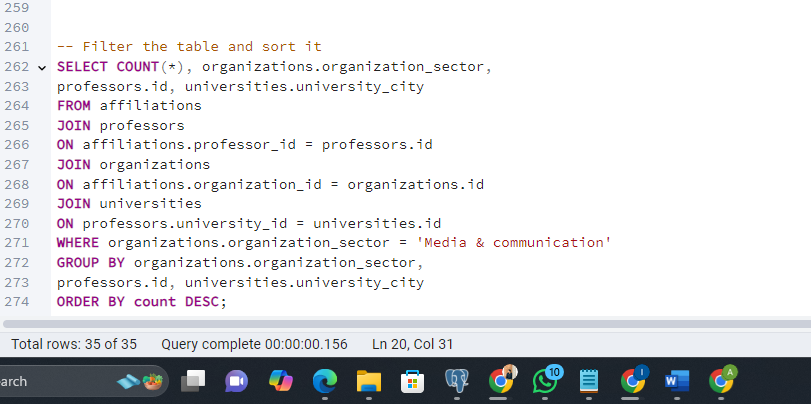












1. **Example Queries and the outputs (create , Insert, update, delete , joins, referential integrity)**

