***Name : M.HUSNAIN ASGHAR***

***Class : 2nd BsCs (M8)***

***Roll-No : F24BDOCS1M01464***

***Subject : DATABASE SYSTEM***

***Submitted To : SIR USMAN GHANI***



# PROJECT

1. **Introduction:**

***1.1 Background:***

***Problem :***

* Academic data about professors, their universities, and their connections to different organizations is often stored in one large, flat table. This kind of setup causes problems like repeated data, errors, and difficulty when searching or updating information. As the amount of data grows, managing it becomes slower and more confusing.

***Solution:***

* To fix this, we built a relational database using the university\_professors dataset. The data is organized into four connected tables: **professors**, **universities**, **organizations**, and **affiliations**. This design removes duplication, keeps the data accurate, and makes it easier and faster to search or update information. It's also a better solution for handling large datasets in the future.

***1.2 Goal:***

* The main objective of this project is to build a relational database system that accurately represents the relationships among university professors, their universities, and affiliated organizations. By the end of the project, the database will allow for easy insertion, retrieval, and updating of academic and organizational data, while maintaining data integrity and avoiding redundancy through normalization and relational modeling.

***1.3 Requirements:***

* The system must store professor details like first name, last name, and university.
* It must keep a list of all universities and organizations.
* Professors can be linked to many organizations (many-to-many relationship).
* See which organizations professors are affiliated with
* Users should be able to add, update, or delete records without losing data.
* The system should support join queries (e.g., get professors with their university and organizations).
* Use primary and foreign keys to connect tables and keep data correct.

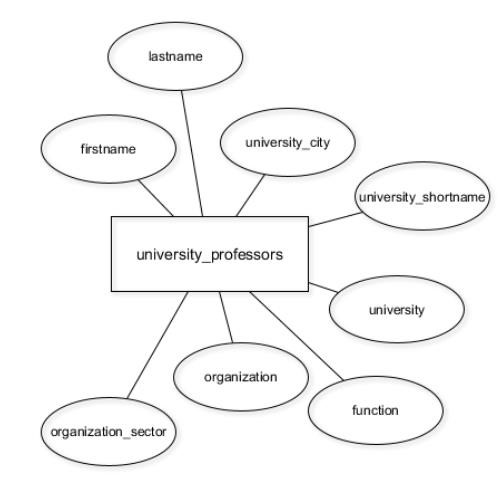
1. **Functional Description:**

***2.1. Method of use:***

* This system will be used by university staff, teachers, and researchers who need to keep track of information about professors. They can use it to add, update, or view details about professors, the universities they work at, and the organizations they are connected to. This helps keep everything in one place and makes it easier to find and manage the data.

1. **Entity Data Model:**

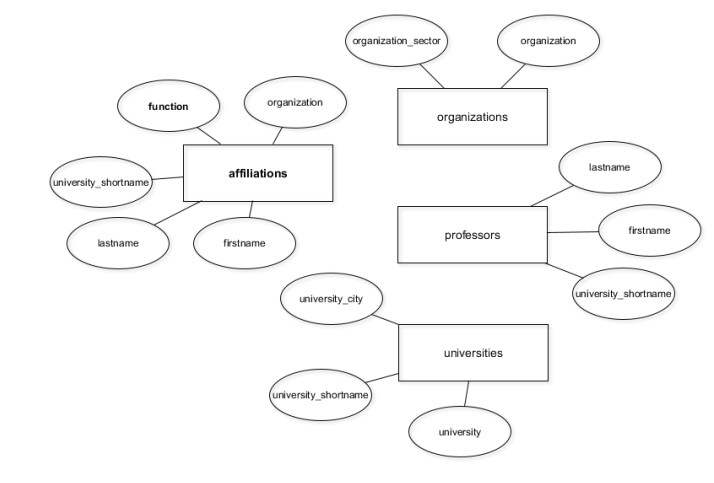
**OLD:**



***Problem – Single Table Design:***

* The original design used one table to store all data about professors, universities, and organizations. This caused repeated data, errors, and made searching or updating information harder

## *NEW:*



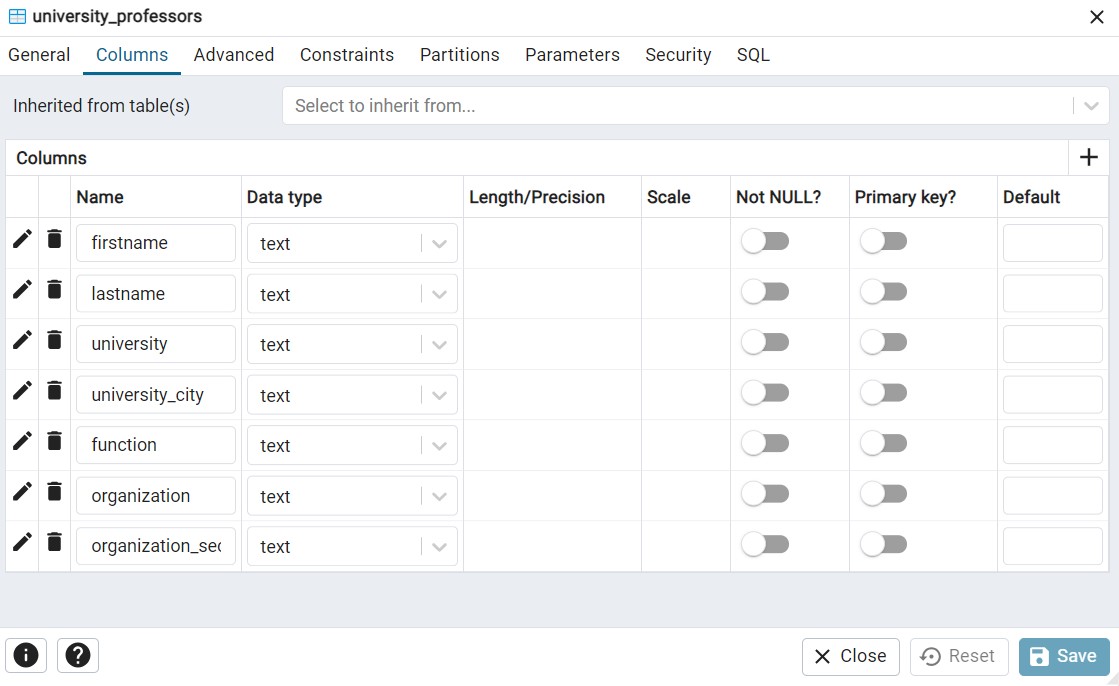
**Solution – Separated Tables Design:**

* The data was split into four linked tables: **professors**, **universities**, **organizations**, and **affiliations**. This setup removes duplication, keeps data accurate, and makes the system easier to use and scale.

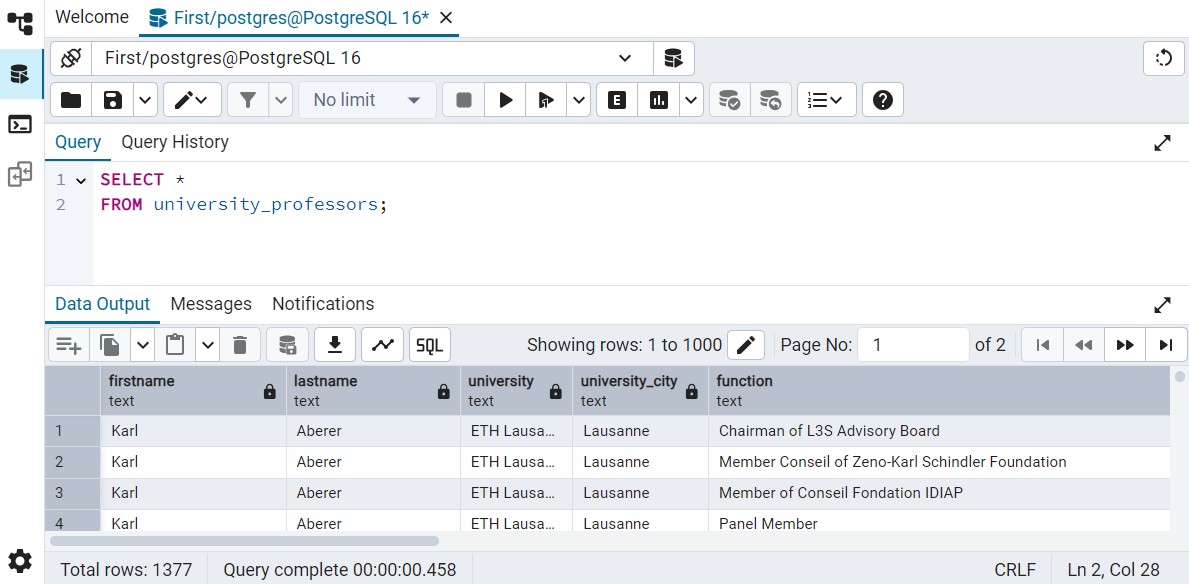
***4. Table Design (Schema) Screenshots:***

* First, we created a table called **university\_professors** in pgAdmin and defined its columns. Then, we **import**ed data into this table from a **CSV** file. After importing, we normalized the data by creating four separate tables: **professors**, **universities**, **organizations**, and **affiliations**.

### *Table: university\_professors*



**Now lets take a look at the current table we have:**



* The data is organized into four connected tables: **professors**, **universities**, **organizations**, and **affiliations**.

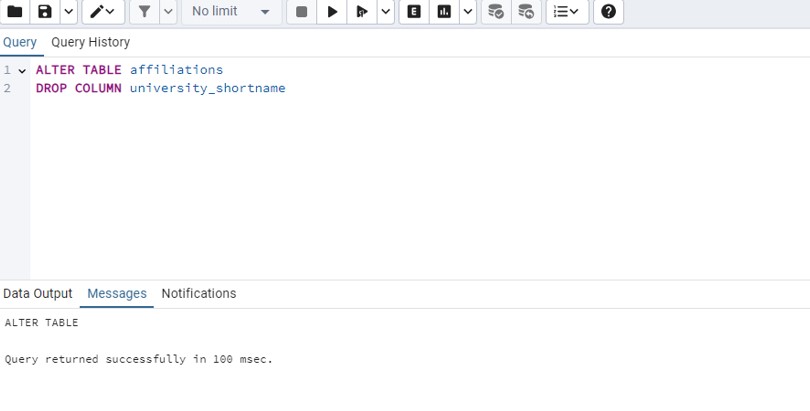


* And insert data into four connected tables: **professors**, **universities**, **organizations**, and **affiliations;**

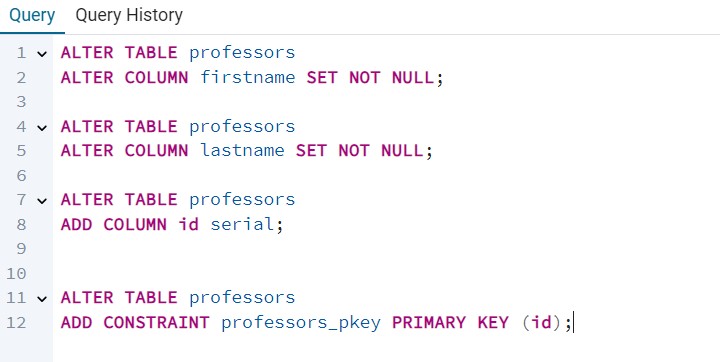


### *5. Frontend Screenshots:*

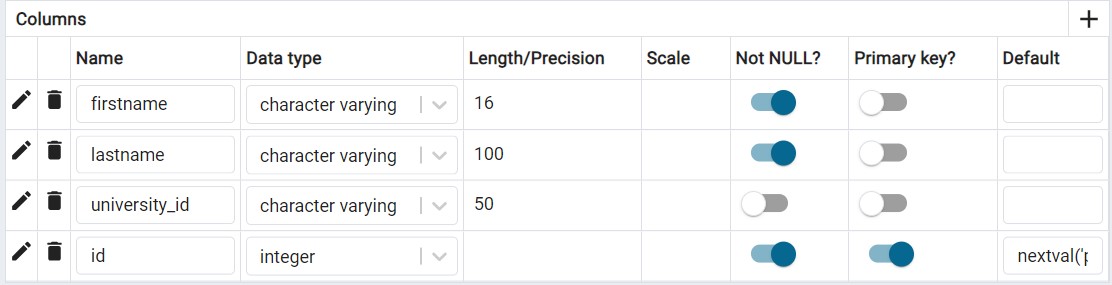
* Here a professor is uniquely identified by **firstname** and **lastname** columns , so there is no needof **univerty\_shortname** column in the **affiliations table** .



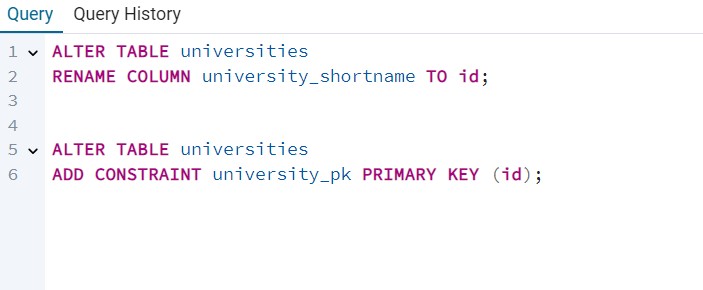
* We updated the professors table by setting the **firstname** and **lastname** columns to **NOT NULL**. Then we added an id column with serial to auto-generate unique numbers for each professor and set it as the **primary key** to uniquely identify each record..



#### Here is professors table:



* We renamed the column to **id** to make the table more consistent and easier to join with other tables. Adding a **PRIMARY KEY** ensures that each **university** can be **uniquely** identified, which improves **data integrity** and allows for better **relationships** between tables.



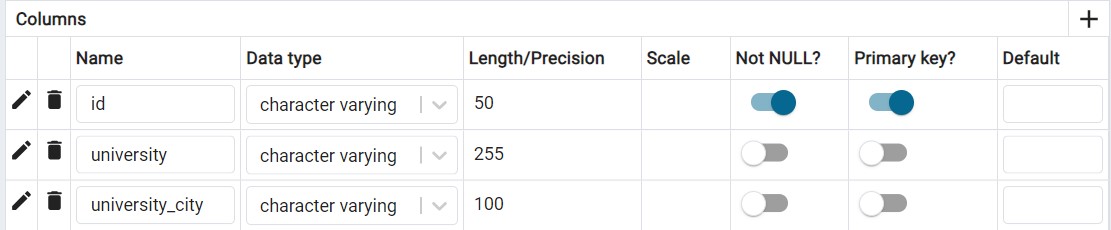
* We added a **PRIMARY KEY** constraint on the id column in the **universities** table to uniquely identify each row. Then, we deleted duplicate rows by keeping only one row per id using the ctid system column to remove duplicates and ensure data consistency.



* This SQL command adds a **foreign key constraint** to the professors table. It links the university\_id column in professors to the id column in the universities table. This ensures that every university\_id in professors must exist in universities, helping maintain **referential integrity** between the two tables.



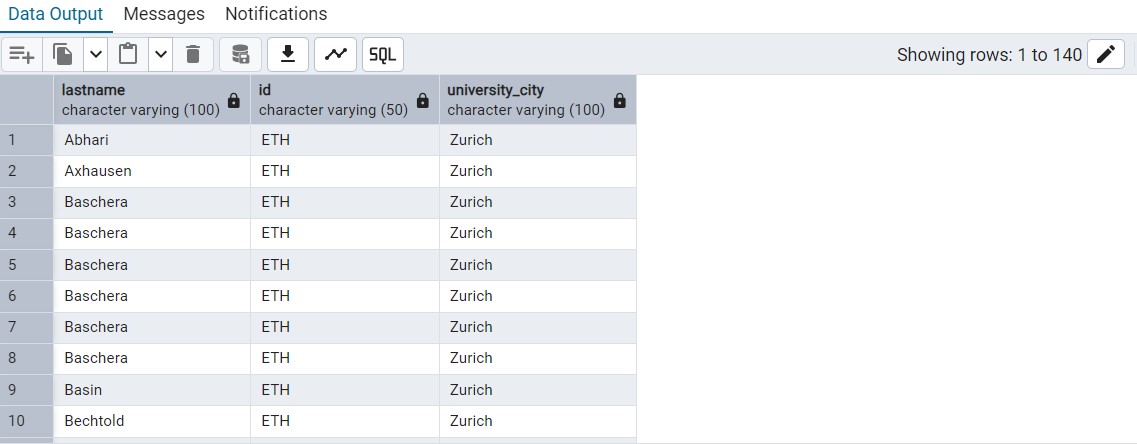
#### Here is universities table:



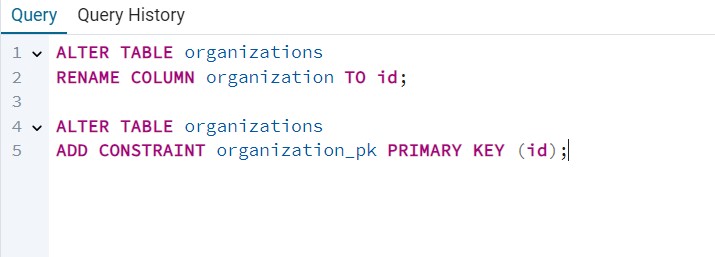
**Here we add some query:** This query finds the last names of professors who work at universities located in the city of Zurich. It joins the professors and universities tables using the university’s unique ID to match professors with their universities. The WHERE clause filters the results to only show professors from universities in Zurich. This helps get specific information about professors based on the location of their university.



## OUTPUT :

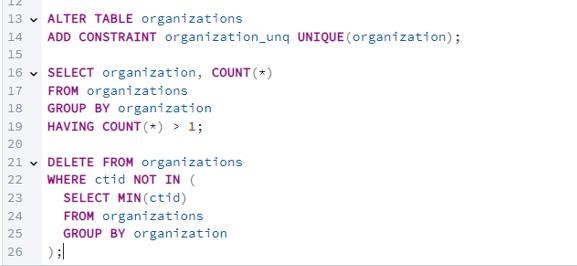


* We renamed the column to id to follow a standard naming convention and added a **PRIMARY KEY** so each organization has a unique identifier for better data management and **relationships** with **other tables**.

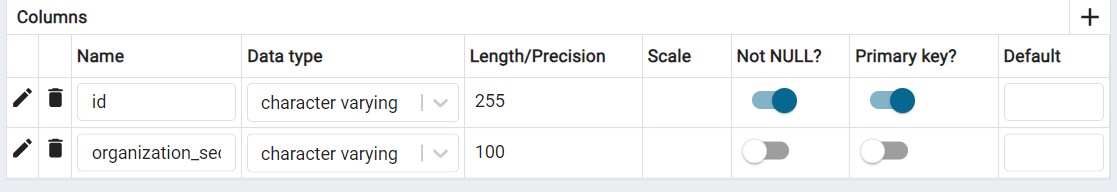


.

* This SQL script is used to clean up duplicate records in the organizations table and ensure data integrity going forward. It first identifies and deletes duplicate rows by keeping only one entry for each unique organization using the ctid (a unique row identifier in PostgreSQL). After the duplicates are removed, it adds a **UNIQUE constraint** on the organization column to prevent any future duplicate entries from being inserted into the table. This process helps maintain consistent and accurate data in the database.



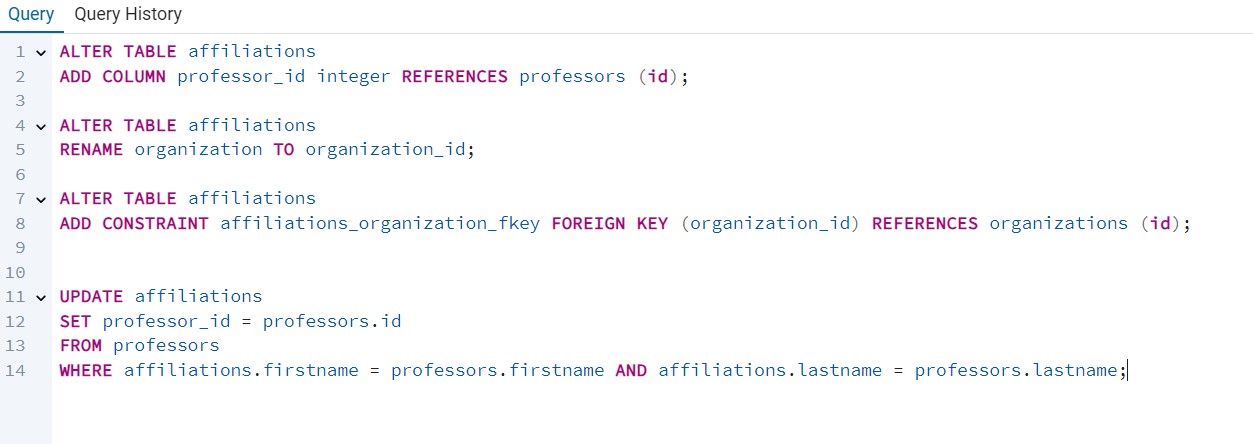
### Here is organizations table:



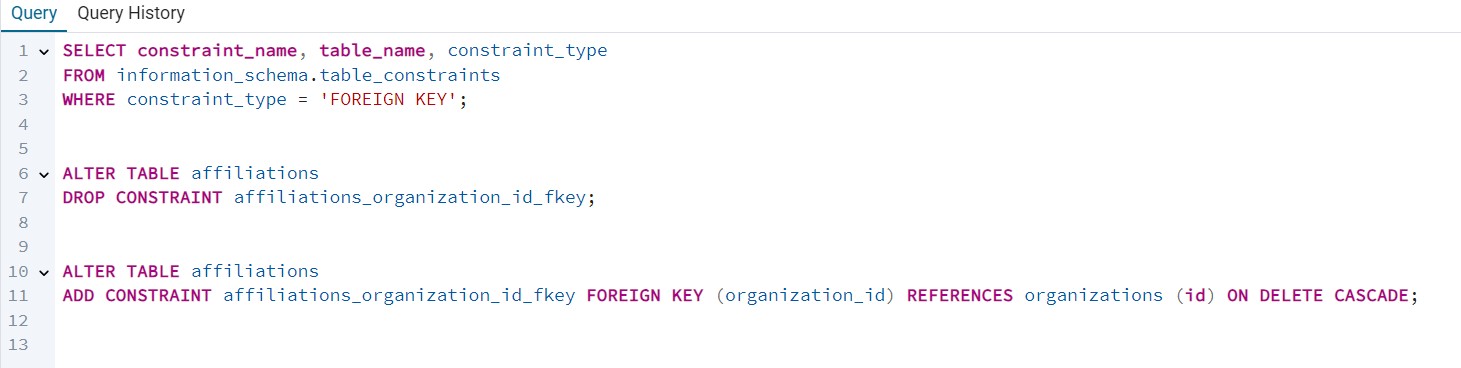
We **DROP** columns from **affiliation** table



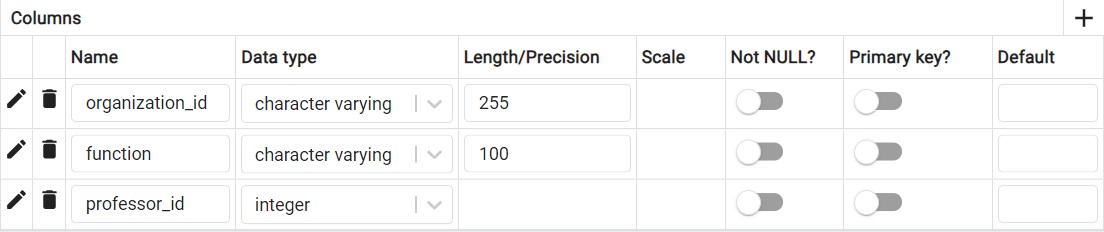
* You added a new column **professor**\_**id** to the **affiliations** table to link each affiliation to a specific professor using their unique ID. You renamed the **organization** column to **organization\_id** to clearly show it stores an ID referencing the **organizations** table. Then, you added foreign key constraints to both **professor\_id** and **organization\_id** to make sure these IDs match valid records in their respective tables. Finally, you updated the **professor\_id** column by matching professors’ first and last names so that existing affiliation records correctly point to the right professor. These changes improve data accuracy and make it easier to manage relationships between professors, organizations, and their affiliations.



* First, you check **all foreign** **key** constraints in the database to find the exact name of the **constraint** on the **affiliations** table. Then, you **drop** the current foreign key constraint on **organization**\_**id** so you can modify it. After that, you add the **foreign** **key** constraint again but this time with ON **DELETE** **CASCADE**, which means if a row in the **organizations** table is **deleted**, all related rows in affiliations will also be automatically deleted to keep the data consistent.



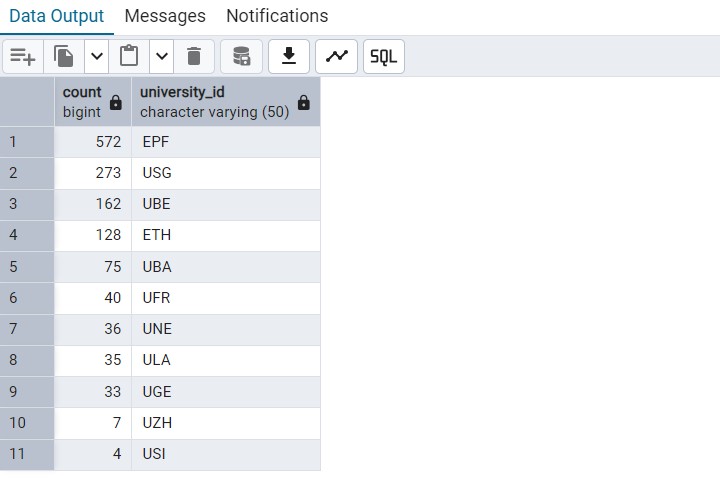
### Here is affiliation table :



#### Here we add some query:



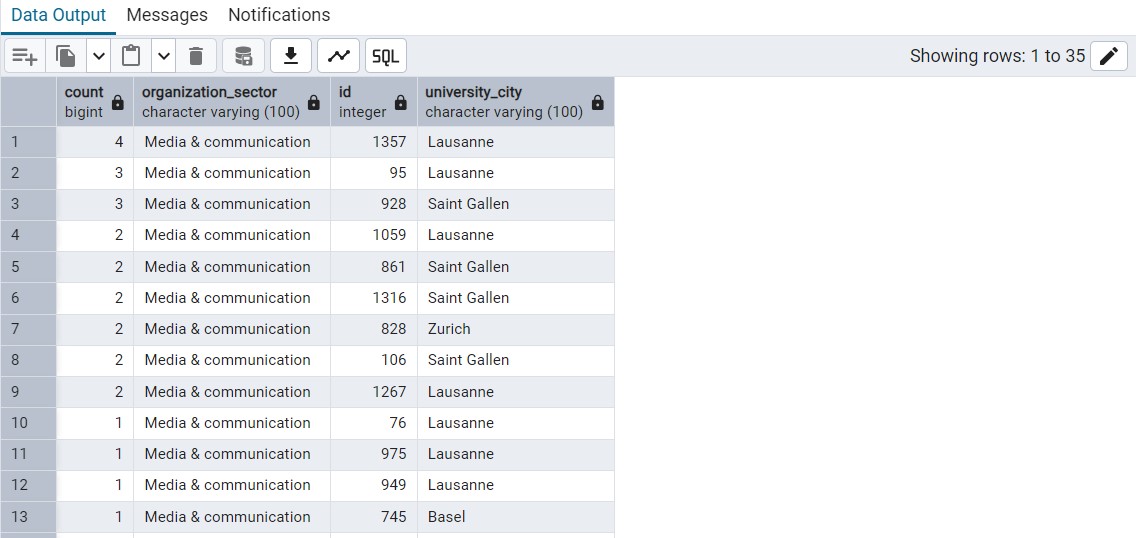
### OUTPUT :



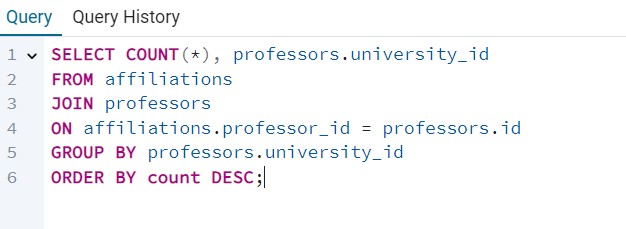
#### Here we add some query:



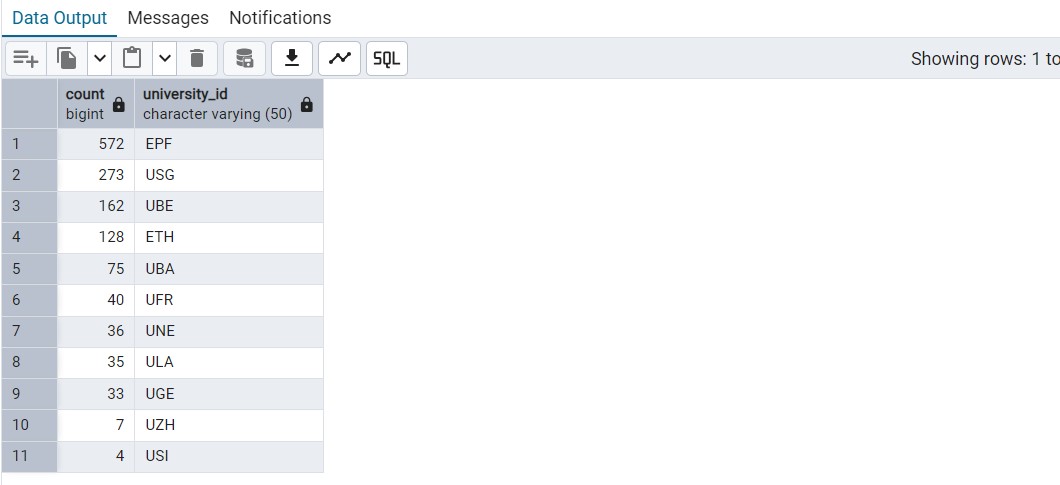
### OUTPUT:



#### Here we add some query :



### OUTPUT:



**Referential Integrity:**

* The process of making a relationship between tables by using FOREIGN KEY which is the PRIMARY KEY of any table is known as referential integrity

**Why ?**

* it will allow us to maintain data between related tables i-e we cant ADD or DELETE rows in the table containing FOREIGN KEY unless we don’t DELETE or UPDATE the PRIMARY KEY of the table

For this purpose , we use the format:

* ALTER TABLE b\_table
* ADD CONSRAINT constraint\_name FOREIGN KEY (b\_table\_column) REFERENCES a\_table (a\_table\_column) ON DELETE CASCADE

So we can only use this format during the formation of FOREIGN KEY so that we are required to DROP previous FOREIGN CONSTRAINTS and add new ones.

### At last the ER diagram of our database :

