**Data Models & Query Languages: Milestone 2**

**Project Name:** Campus Recruitment Database   
**Team Name:** Seekers  
Member1: Katipelly Shreya Reddy   
UBID: skatipel  
Member2: Desireddy Sai Sankeerthana   
UBID: saisanke  
Member3: Abhigna Sree Tumati   
UBID: abhignas

**Problem statement:**

With each passing year, thousands of students will be graduating, and are exposed to various job offers in various companies based on their educational qualifications. To keep track of each person’s data and infer information on how the records of the students in the campus are varying and to take any measures to improve the statistics and to understand how each factor is affecting the placements opportunities each year, The college officials must be able to understand how things are working for students in campus. And maintaining such huge data in a file system will not be of much use since we it is hard to infer information from a file system. So, in here, database would be of good use and helps them to get a good picture of everything related to campus placements and how their educational background is helping them to grab better opportunities.

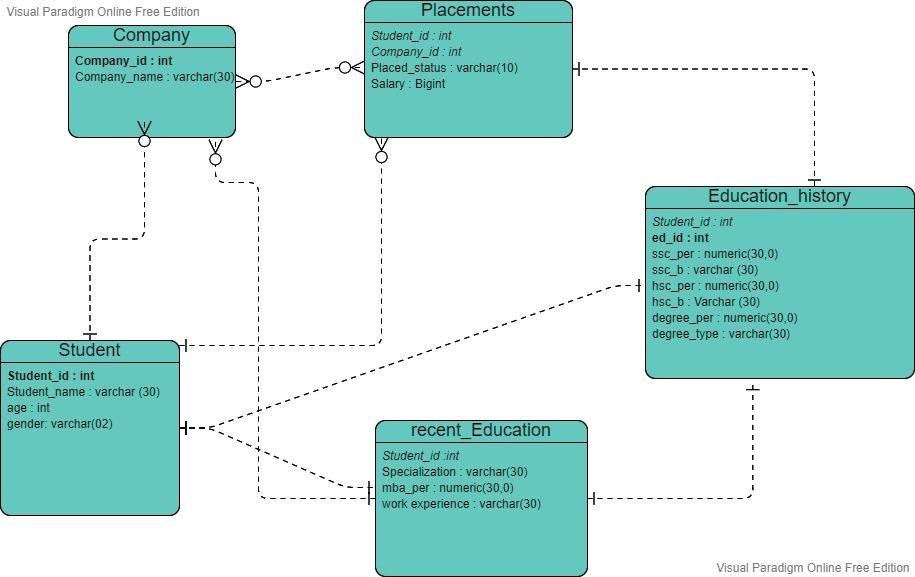
We will be using a database instead of an excel sheet to store the data of our campus recruitment information because in a database we can define relations between various schemas and retrieve data easily for understanding. We can restrict the type of data entered in the records by ‘Data Integrity’ so that we can avoid mistakes while handling large data. In a database we can also use the concept of

‘Referential Integrity’ which ensures there is a value in another table which can be referenced. A database can handle large amount of data which a excel sheet cannot. A database can be accessed by multiple users at a given time whereas a excel can only be used by a single person at any point of time. If we want to secure our data and provide privileges to users to perform various operations on our data, then databases are very efficient where it provides security to our data by applying certain restrictions on who can use our data which an excel sheet fails to provide.

**Target User:**

The target audience and the administrator for the database would be the College officials. Every College wants to be recognized as one of the prestigious one's in terms of the education it offers to the students. To be recognized as one, the college must be providing the students with a great academic program. And the campus placement percentage would be considered as metric since a good program should be able to equip the students with skills to get ready for the industry. So having a database to store all such data and inferring information could help them in getting an idea of what is going well and what needs improvisation. This way, they can make necessary changes to their program to keep the student’s industry ready and helps the college officials to achieve their end goal to provide their best for the students.

**E-R DIAGRAM:**



**Details on Schema:**

Final Schema of the implemented database is the same as the initial one.

**Action on the existing foreign keys in case of deletion of a primary key value:**

Student Id : Delete cascade

Upon the deletion of a student details in the student table, no further data regarding the student is necessary anymore. So the action taken here is delete cascade

Company Id : No action

The deletion of a company tuple from the company table in the present doesn’t have to effect the rest of the student data since we are having records of students past data. So in this case we are taking no action

**Dependencies:**

All our tables are already in BCNF and following are the FD’s for each table.

**Company**

Company (company\_id, company\_name)

1. company\_id -> company\_name

**Placements**

Placements (student\_id, company\_id, placed\_status, salary)

1. student\_id, company\_id -> placed\_status
2. student\_id, company\_id -> salary

**Student**

Student (student\_id, student\_name, age, gender)

1. student\_id -> student\_name
2. student\_id -> age
3. student\_id -> gender

**Recent\_Education**

Recent\_Education (student\_id, specialization, mba\_per, work\_experience)

1. student\_id -> specialization
2. student\_id -> mba\_per
3. student\_id -> work\_experience

**Education\_History**

Education\_History (student\_id, ed\_id, ssc\_per, ssc\_b, hsc\_per, hsc\_b, degree\_per, degree\_type)

1. ed\_id -> student\_id
2. ed\_id -> ed\_id
3. ed\_id -> ssc\_per
4. ed\_id -> ssc\_b
5. ed\_id -> hsc\_per
6. ed\_id -> hsc\_b
7. ed\_id -> degree\_per
8. ed\_id -> degree\_type

**Query Execution:**

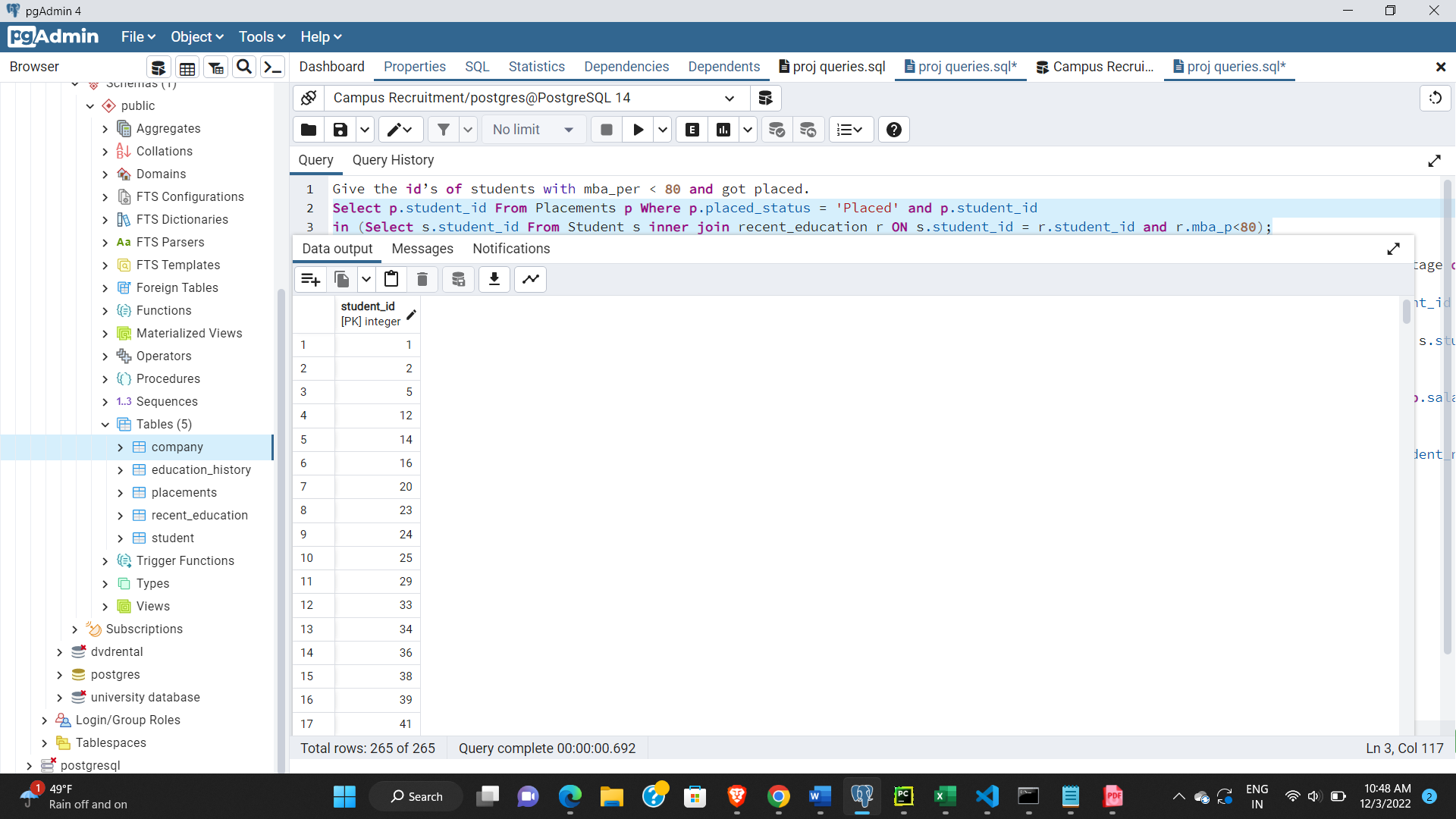
We’ve executed multiple queries on the implemented Campus Recruitment Database including various concepts like subqueries, joins, order by, group by, aggregate functions etc.

1. Selection
2. Deletion
3. Insertion
4. Updating

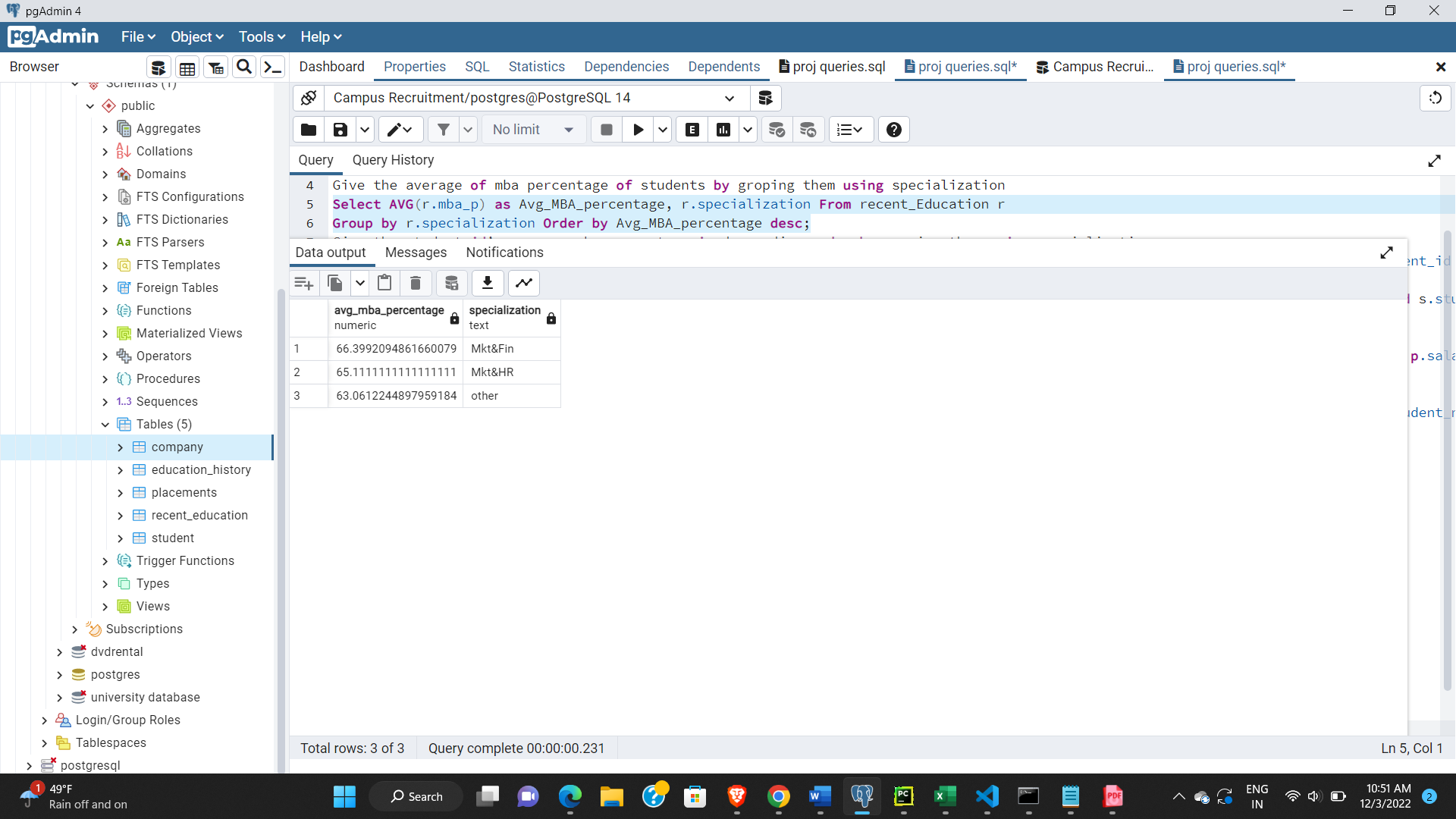
The following are the screenshots of few of them:

**Selection queries:**

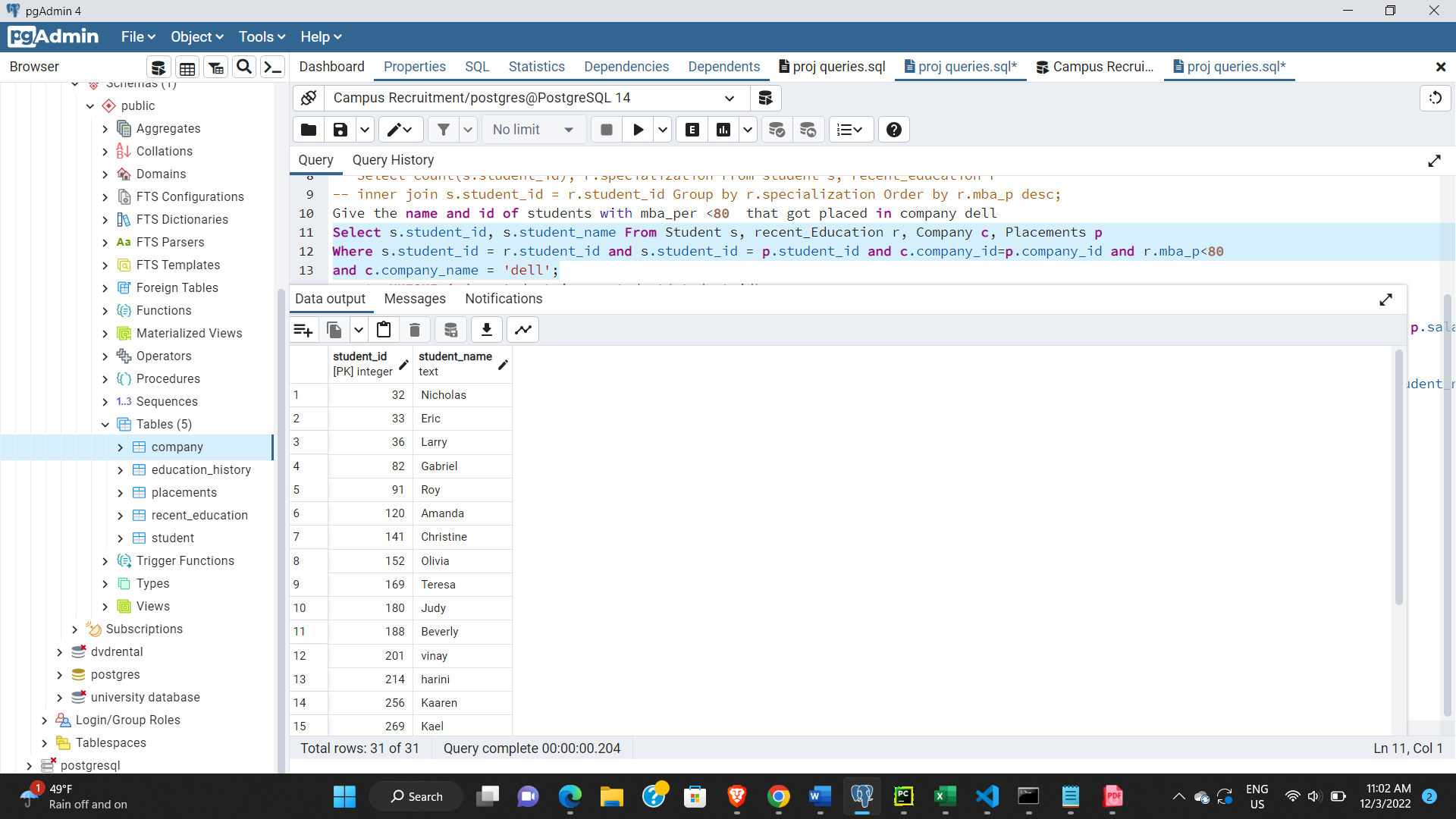
1. Query to find id’s of students who got MBA percentage less than 80 and got placed.



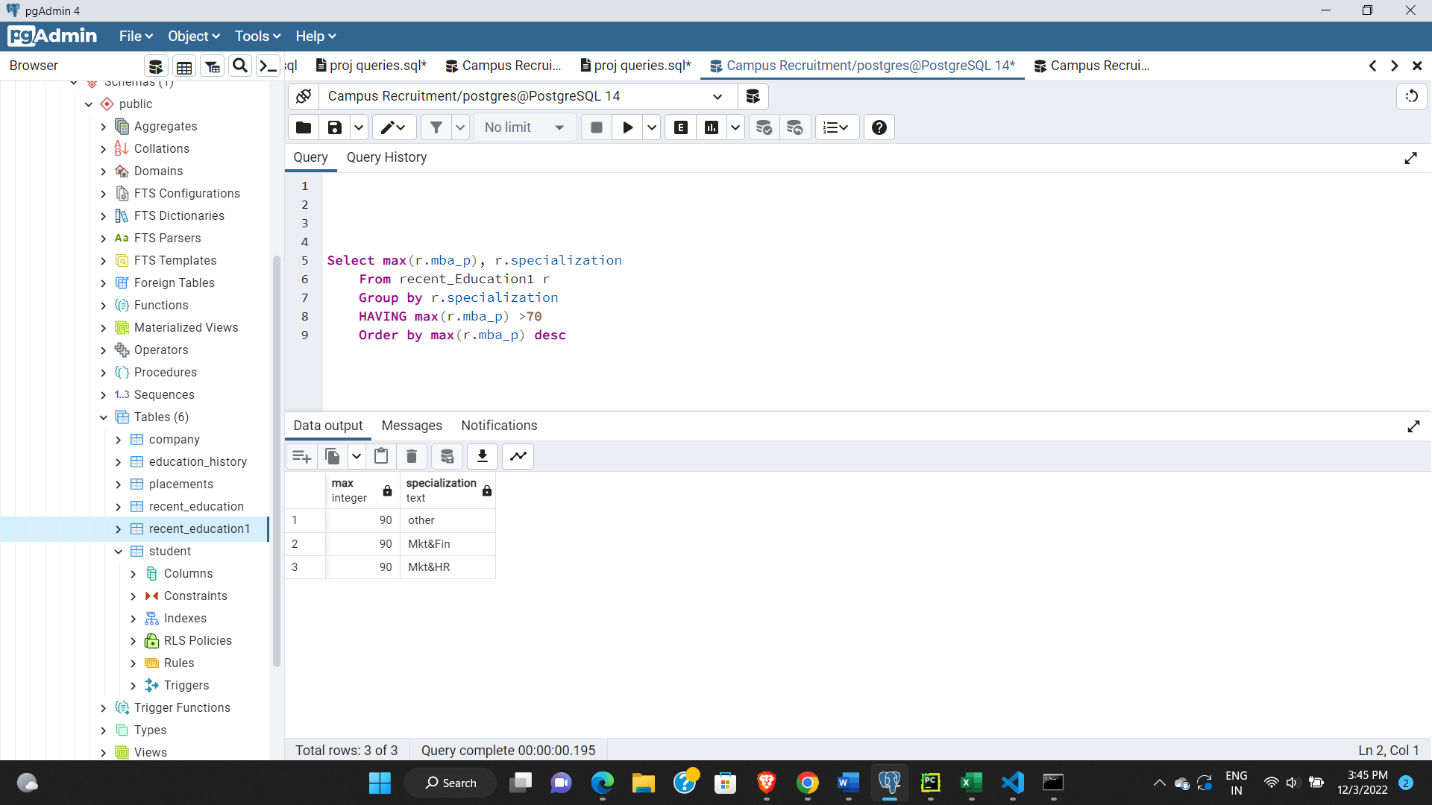
1. Query to find Average MBA percentage of students group by specialization.



1. Query to find names and id’s of student who got less than 80 % in MBA and got placed in company Dell.



1. Query to give the maximum percentage of students in each specialization having highest percentage greater than 70.



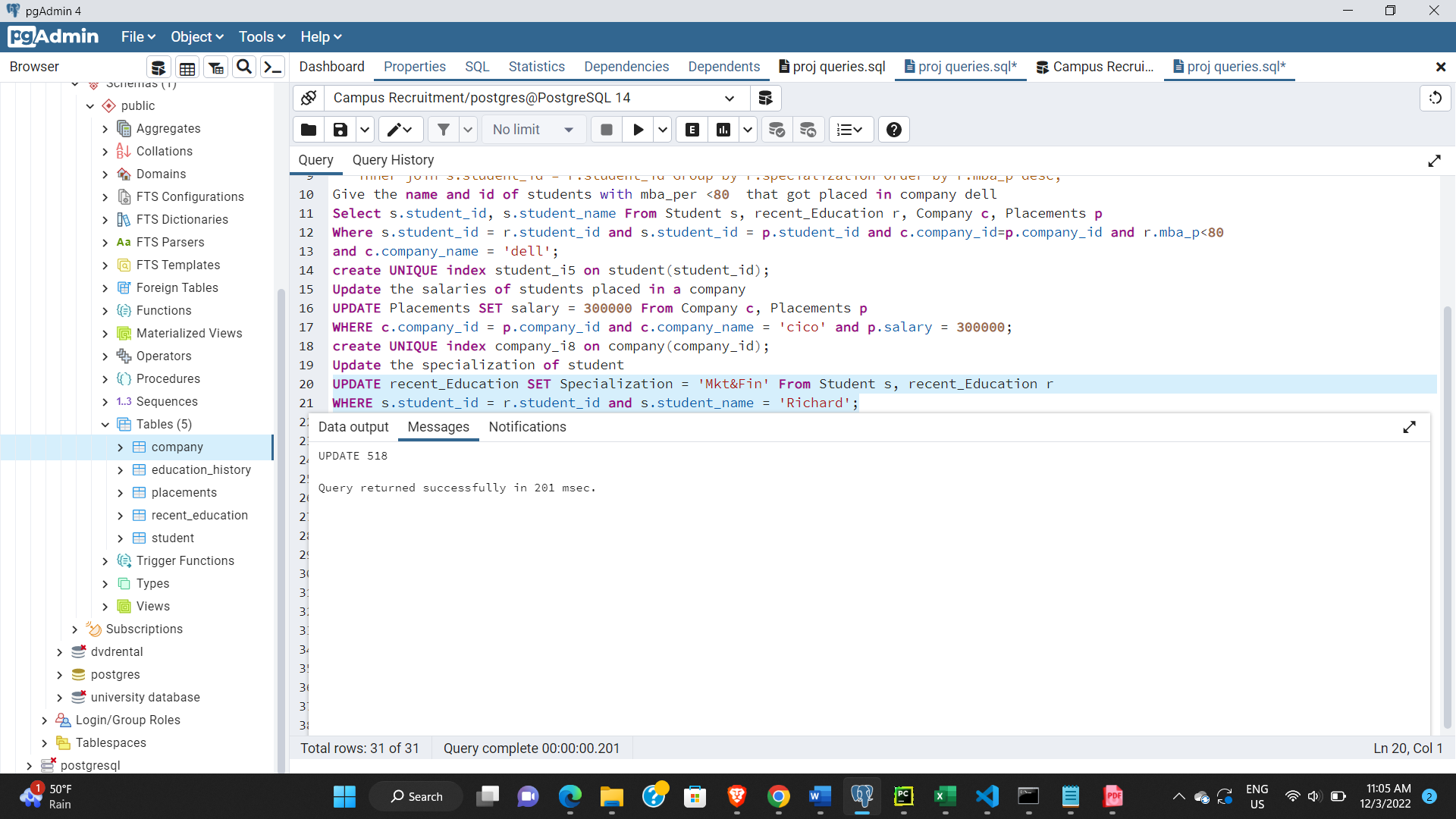
**Update queries:**

1. Query to update salaries to 400k for the students who got placed in company Cisco with salary 300k.

Graphical user interface, text, application

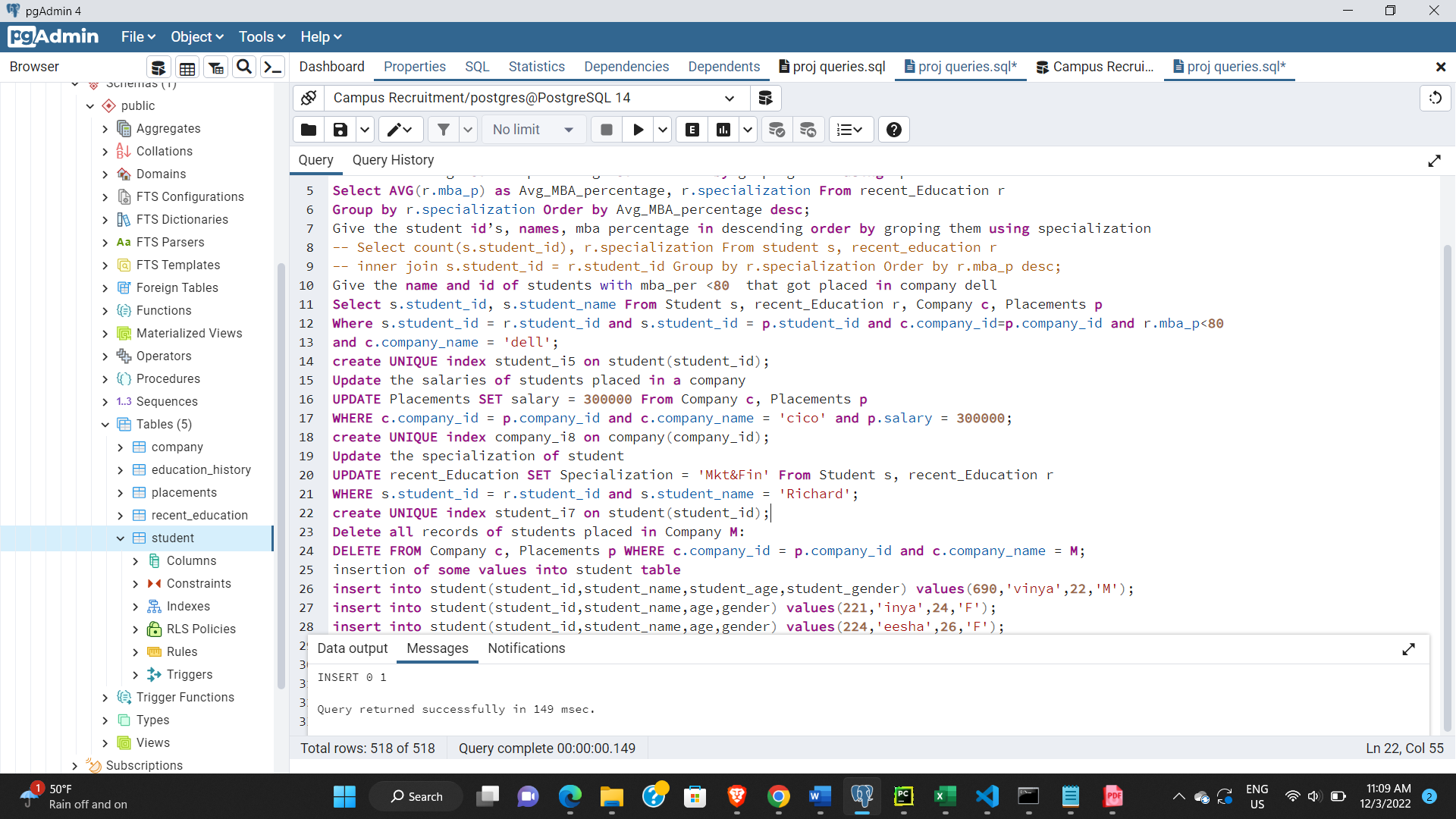
Description automatically generated

1. Query to update specialization of a student named Richard to Mkt&Fin.



**Insertion queries:**

1. Query for insertion of some random records into the student table.



**Deletion queries:**

1. Query to delete records of student from placements who age is less than 25

Graphical user interface, text, application

Description automatically generated

1. Query to delete record of students who have been place in IBM.

Graphical user interface, text, application, email

Description automatically generated

**Indexing:**

Upon the execution of various queries on the relations in the implemented database, to further improvise the performance we have used indexing. After indexing we have noticed a reduction the query execution time. The following are some examples implying the reduction of query execution time upon indexing.

*Example 1: The execution time of the below query has reduced from 117ms to 62ms. Below are the screenshots of the same before and after indexing.*

Graphical user interface, application, Word

Description automatically generated

Before Indexing(i)

Graphical user interface, application, Word

Description automatically generated

After Indexing(ii)

*Example 2: The execution time of the below query has reduced from 102ms to 42ms. Below are the screenshots of the same before and after indexing.*

Graphical user interface, text, application

Description automatically generated

Before Indexing(i)

Graphical user interface, text, application

Description automatically generated

After Indexing(ii)

*Example 3: The execution time of the below query has reduced from 661ms to 48ms. Below are the screenshots of the same before and after indexing.*

A screenshot of a computer

Description automatically generated

Before Indexing(i)

A screenshot of a computer

Description automatically generated

After Indexing(ii)

**Team Contributions:**

**Sankeerthana:**

* + Problem statement selection
  + Design overview of the database
  + ER diagram implementation

**Shreya:**

* + Database Creation
  + Cleaning and modification on dataset
  + Record Insertion into relations

**Abhigna:**

* + Writing queries
  + Testing queries
  + Indexing

**Report Generation** : Sankeerthana,Shreya,Abhigna