Data Management And Database Design

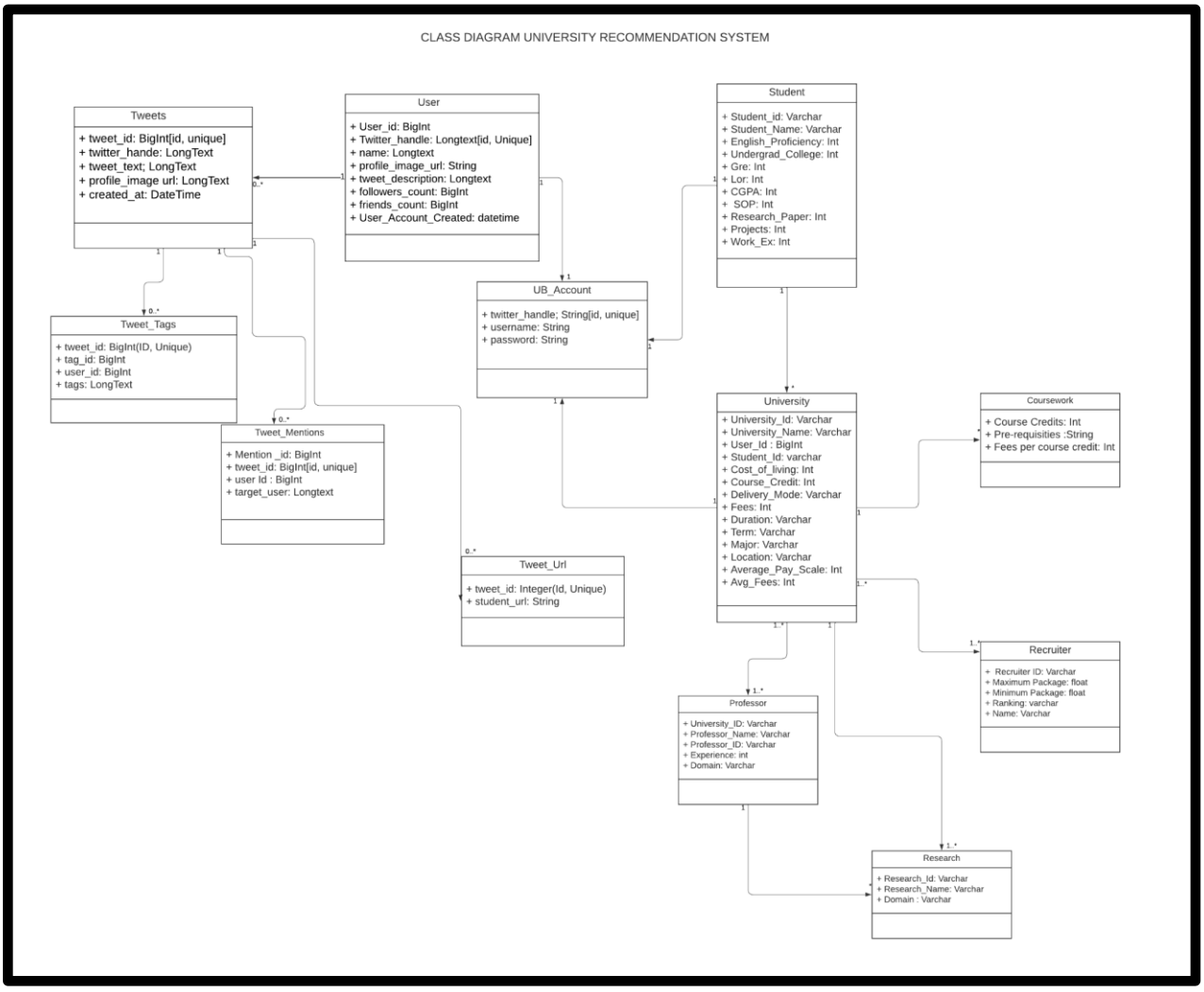
Assignment - 2

GitHub Repository: https://github.com/Muskansri1/University_Recommendation_System

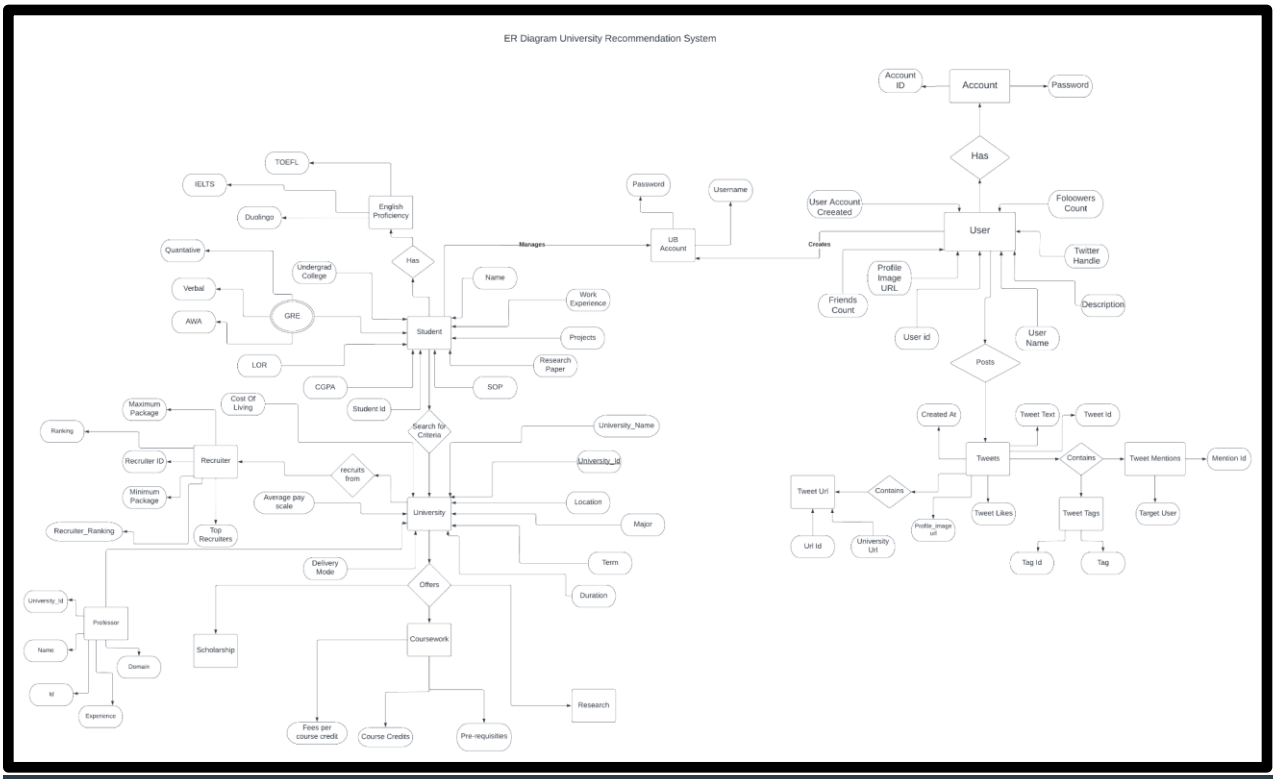
Group Members: Sameer Sanjay Nimse (002752914)

Muskan Srivastava (002794929)

Physical Model Of University Recommendation System



ER Diagram of University Recommendation System



SQL STATEMENTS FOR THE CONCEPTUAL MODEL

User Table:

```
CREATE TABLE user (
user_id bigint,
twitter_handle longtext,
user_name longtext,
profile_img_url longtext,
tweet_description longtext,
friends_count bigint,
followers_count bigint,
user_account_created DATETIME,
PRIMARY KEY (user_id)
);
```

Student Table:

```
CREATE TABLE Student (
student_id Varchar(255),
student_name VARCHAR(255),
TOEFL int,
IELTS int,
GRE int,
LOR int,
CGPA int,
SOP int,
Research_paper int,
Projects int,
```

```
Work_Ex int,
user_id BIGINT,
PRIMARY KEY (student_id),
FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Research Table:

```
CREATE TABLE Research(
Research_ID Varchar(255),
Research_Name VARCHAR(255),
Domain VARCHAR(255),
University_ID Varchar(255),
Professor_ID Varchar(255),
PRIMARY KEY (Research_ID),
FOREIGN KEY (University_ID) REFERENCES university(University_ID)
);
```

Tweets Table:

```
CREATE TABLE tweets (
tweet_id bigint,
user_id bigint,
tweet_text longtext,
created_at DATETIME,
tweet_likes bigint,
PRIMARY KEY (tweet_id),
FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

University Table:

```
CREATE TABLE university(
University_ID VARCHAR(255),
University_Name VARCHAR(255),
user_id BIGINT,
student_id Varchar(255),
avg_pay_scale INT,
Delivery_Mode VARCHAR(255),
Duration VARCHAR(255),
Term VARCHAR(255),
Major VARCHAR(255),
Location VARCHAR(255),
PRIMARY KEY (University_ID),
FOREIGN KEY (student_id) REFERENCES student(student_id)
);
```

Tweet Tags Table:

```
CREATE TABLE tweet_tags (
tag_id BIGINT,
user_id BIGINT,
tags longtext,
tweet_id BIGINT,
PRIMARY KEY (tag_id),
FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id),
FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Tweet Mentions Table:

```
CREATE TABLE tweet_mentions (
mention_id BIGINT,
tweet_id BIGINT,
user_id BIGINT,
target_user LONGTEXT,
PRIMARY KEY (mention_id),
FOREIGN KEY (tweet_id) REFERENCES Tweets(tweet_id),
FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```

Recruiter Table:

```
CREATE TABLE Recruiter(
Recruiter_ID Varchar(255),
Recruiter_Name Varchar(255),
Recruiter_Ranking INT,
min_package float,
max__package float,
University_ID Varchar(255),
PRIMARY KEY (Recruiter_ID),
FOREIGN KEY (University_ID) REFERENCES university(University_ID)
);
```

Professor Table:

```
CREATE TABLE Professor (
Professor_ID VARCHAR(255),
Professor_Name VARCHAR(255),
Domain VARCHAR(255),
University_ID VARCHAR(255),
Experience Int,
PRIMARY KEY (Professor_ID),
FOREIGN KEY (University_ID) REFERENCES university(University_ID)
);
```

Course Table:

```
CREATE TABLE Course (
Course_ID VARCHAR(255),
Course_Name VARCHAR(255),
Pre_Req VARCHAR(255),
Credits FLOAT,
Fees_per_credit FLOAT,
University_ID VARCHAR(255),
PRIMARY KEY (Course_ID),
FOREIGN KEY (University_ID) REFERENCES university(University_ID)
);
```

1. **What user posted this tweet?**

SQL Query

SELECT user.user_name, tweets.tweet_text

FROM User, Tweets

WHERE user.user_id = Tweets.user_id AND

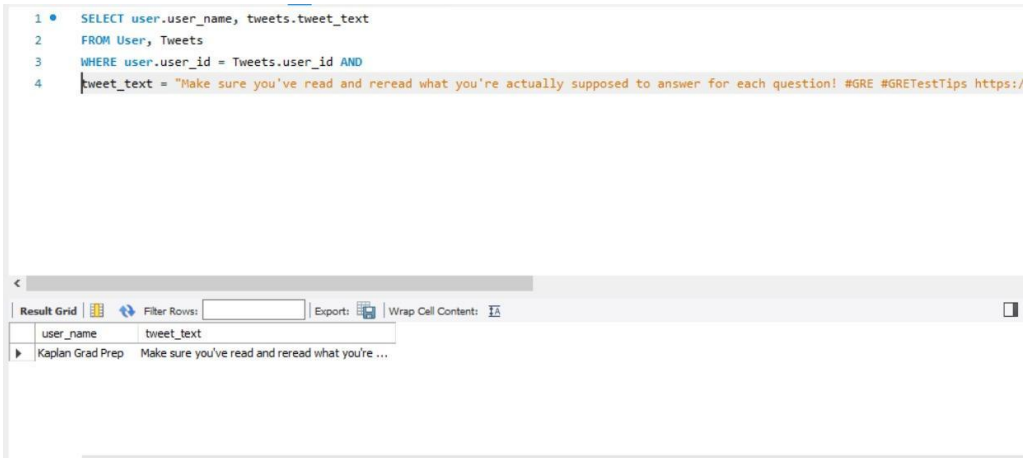
tweet_text = "Make sure you've read and reread what you're actually supposed to answer for each question! #GRE #GRETestTips https://t.co/tHUH4SlRrK";

Relational Algebra:

π user . user_name, tweets . tweet_text

σ user . user_id = tweets . user_id AND tweet_text = "Make sure you've read and reread what you're actually supposed to answer for each question! #GRE #GRETestTips https://t.co/tHUH4SlRrK" (user × tweets)

OUTPUT:



2. When did the user post this tweet?

SQL Query

SELECT Tweets.created_at, tweets.tweet_text, user.user_name FROM Tweets, User

WHERE user.user_id = Tweets.user_id AND

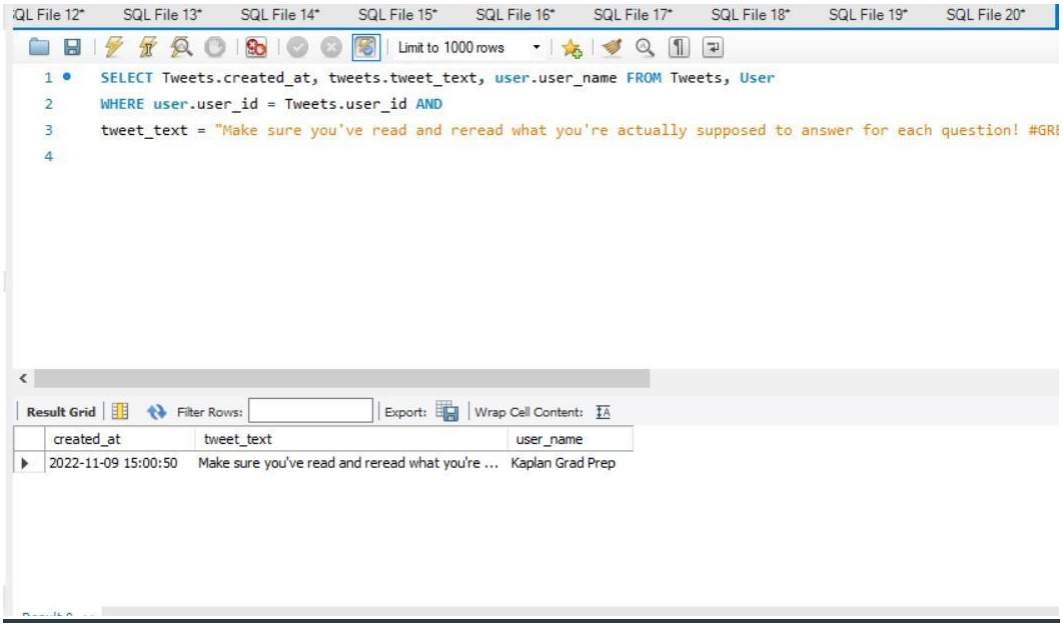
tweet_text = "Make sure you've read and reread what you're actually supposed to answer for each question! #GRE #GRETestTips https://t.co/tHUH4SlRrK";

Relational Algebra:

π tweets . created_at, tweets . tweet_text, user . user_name

σ user . user_id = tweets . user_id AND tweet_text = "Make sure you've read and reread what you're actually supposed to answer for each question! #GRE #GRETestTips https://t.co/tHUH4SlRrK" (tweets × user)

OUTPUT



3 What tweets have this user posted in the past 24 hours?

SQL Query

SELECT user.user_name, tweets.tweet_text

FROM User, Tweets

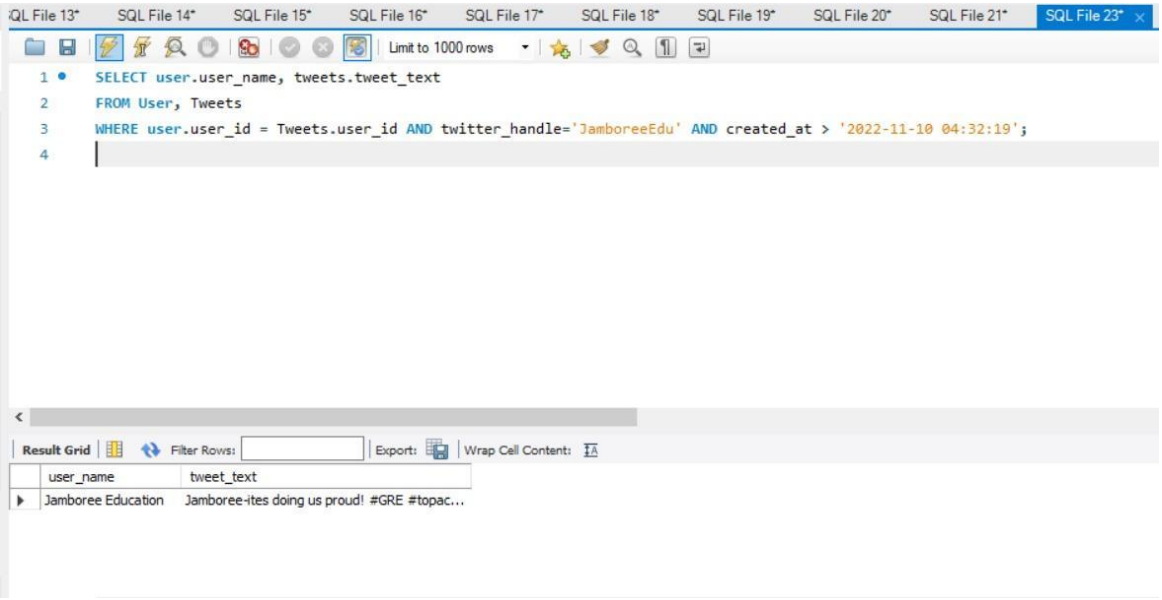
WHERE user.user_id = Tweets.user_id AND twitter_handle='JamboreeEdu' AND created_at > '2022-11-10 04:32:19';

Relational Algebra

π user . user_name, tweets . tweet_text

σ user . user_id = tweets . user_id AND twitter_handle = "JamboreeEdu" AND created_at > "2022-11-10 04:32:19" (user × tweets)

OUTPUT



4. How many tweets have this user posted in the past 24 hours?

SQL Query

SELECT user.user_name, count(tweets.tweet_text) AS No_of_Tweets

FROM User, Tweets

WHERE user.user_id = Tweets.user_id AND twitter_handle='JamboreeEdu' AND created_at > '2022-11-10 04:32:19';

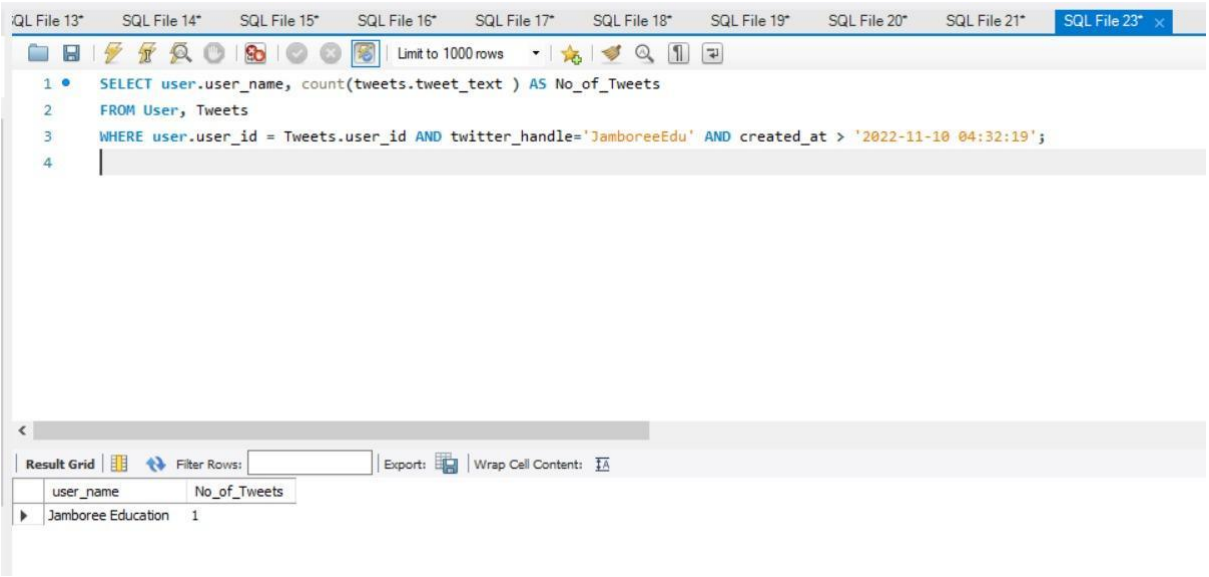
Relational Algebra

π user . user_name, COUNT (tweet_text) → no_of_tweets

γ COUNT (tweet_text)

σ user . user_id = tweets . user_id AND twitter_handle = "JamboreeEdu" AND created_at > "2022-11-10 04:32:19" (user × tweets)

OUTPUT



5. When did this user join Twitter?

SQL Query

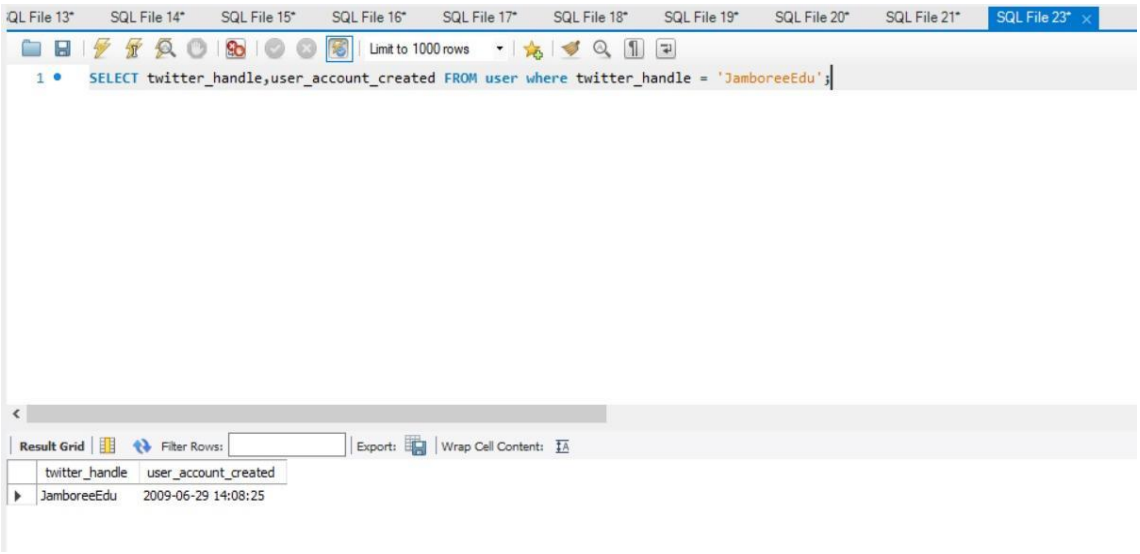
SELECT twitter_handle,user_account_created FROM user where twitter_handle = 'JamboreeEdu';

Relational Algebra

π twitter_handle, user_account_created

σ twitter_handle = "JamboreeEdu" user

OUTPUT



6. What keywords/ hashtags are popular?

SQL Query

Select tags,COUNT(tags) AS Frequency

from tweet_tags group by tags order by Frequency DESC

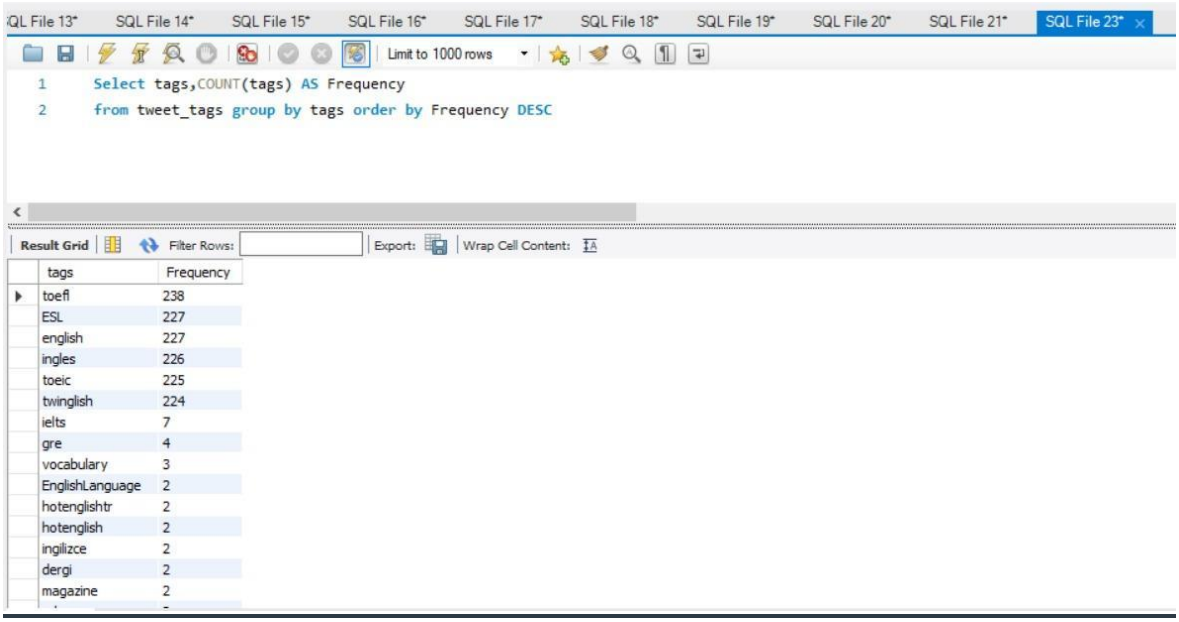
Relational Algebra

T frequency ↓

ΤΤ tags, COUNT (tags) → frequency

γ tags, COUNT (tags) tweet_tags

OUTPUT



tags	Frequency
toefl	238
ESL	227
english	227
ingles	226
toeic	225
twinglish	224
ielts	7
gre	4
vocabulary	3
EnglishLanguage	2
hotenglishtr	2
hotenglish	2
ingilizce	2
dergi	2
magazine	2

7.What tweets are popular?

SQL Query

SELECT tweet_text, tweet_likes from tweets where tweet_likes>500 order by tweet_likes DESC;

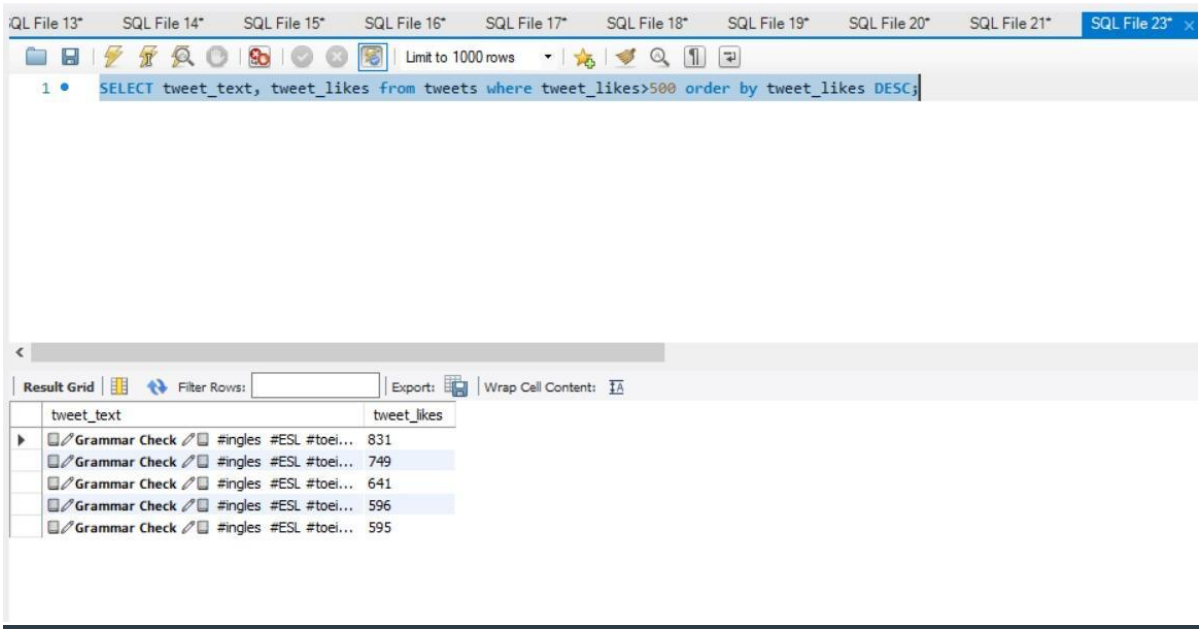
RELATIONAL ALGEBRA

T tweet_likes ↓

TT tweet_text, tweet_likes

σ tweet_likes > 500 tweets

OUTPUT


QL File 13*
SQL File 14*
SQL File 15*
SQL File 16*
SQL File 17*
SQL File 18*
SQL File 19*
SQL File 20*
SQL File 21*
SQL File 23*
Limit to 1000 rows
1 SELECT tweet_text, tweet_likes from tweets where tweet_likes>500 order by tweet_likes DESC;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
tweet_text
tweet_likes
Grammar Check #ingles #ESL #toei... 831
Grammar Check #ingles #ESL #toei... 749
Grammar Check #ingles #ESL #toei... 641
Grammar Check #ingles #ESL #toei... 596
Grammar Check #ingles #ESL #toei... 595

USE-CASES PARTICULAR TO THE DOMAIN

1. **Use Case: View the list of top 5 recruiters with their average package**

   Description: The student selects a university to find its top 5 recruiters

   Actor: Student

   Precondition: When a student wants to look up top 5 recruiters in a particular university, the student selects a particular university

   Steps:

   Actor action: Student request for top 5 recruiters

   System Responses: If student information is correct then the system displays a list of top 5 recruiters of a university

   Post Condition: The system displays the name of the universities

   Alternate Path: The student request is not correct and the system throws an error

   Error: The information is incorrect

   SQL Query:

   SELECT recruiter_name, recruiter_ranking, ((Min_Package+Max__package)/2) as AVG_PACKAGE

   FROM RECRUITER

   WHERE recruiter_ranking < 5 ORDER BY recruiter_ranking;

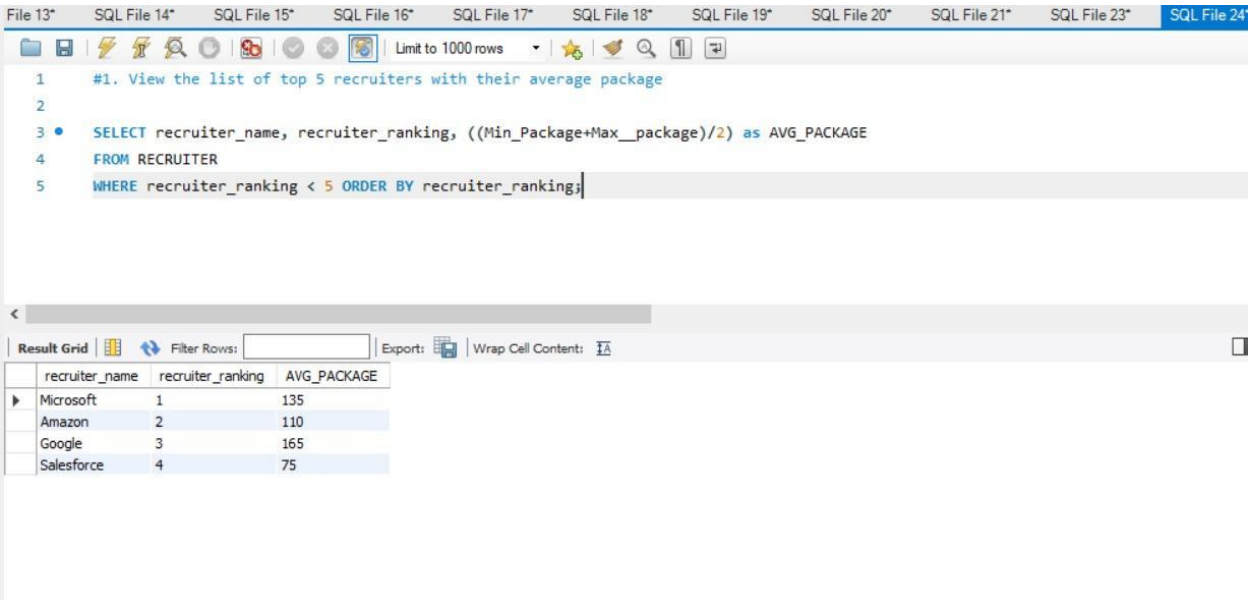
   Relational Algebra:

   τ recruiter_ranking

   π recruiter_name, recruiter_ranking, (min_package + max__package)/ 2 → avg_package

   σ recruiter_ranking < 5 recruiter

OUTPUT



2. **Use Case: View a university based on the ROI**

Description: Student selects a university to view its ROI

Actors: Student

Precondition: Student must keep in mind cost of living, average pay scale and fees

Steps:

Actor action – Student tweets about a university to view its ROI/ student views UB application to view the ROI

System Responses – An response is made for the request that matches the university that fits the condition average payscale > fees+cost of living

Post Condition: ROI for a university is generated

Alternate Path: The university is not currently available on the website

Error: University Not Available

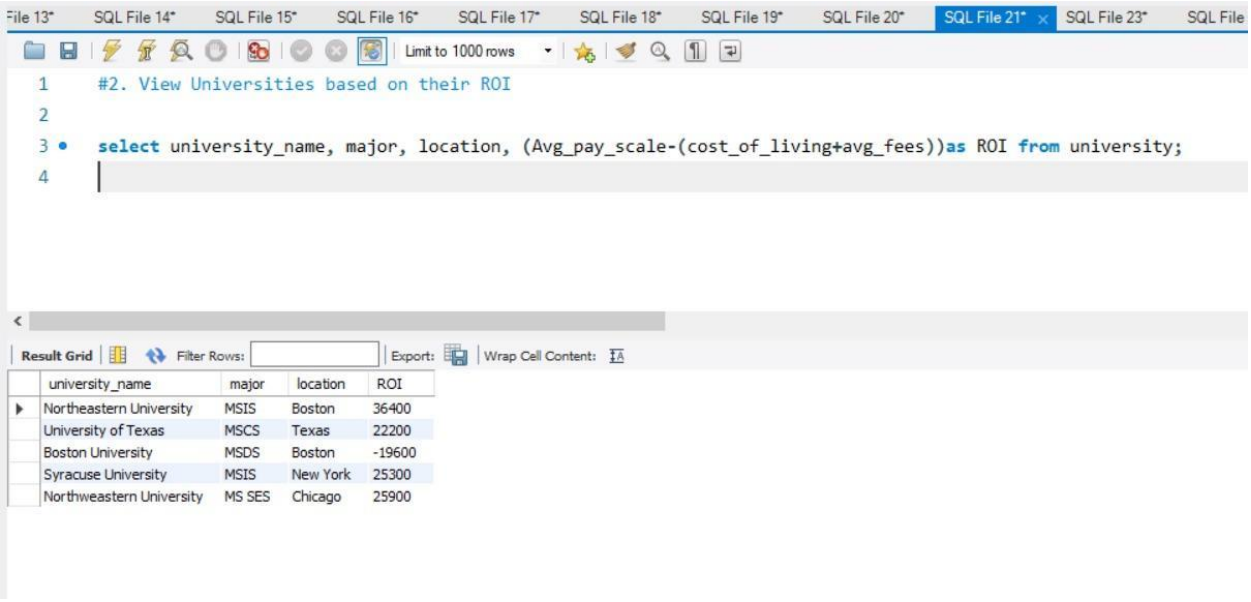
SQL Query:

select university_name, major, location, (Avg_pay_scale-(cost_of_living+avg_fees))as ROI from university;

Relational Algebra:

TT university_name, major, location, avg_pay_scale - (cost_of_living + avg_fees) → roi university

OUTPUT



3. **Use Case: View All the professors of universities with their experience**

Description: The student views the professor based on their experience

Actors: Student

Precondition: The student must have entered a professsor

Steps:

Actor action – The student views the professor on the basis of experience

System Responses – The system generates the results of the professors on the basis of years of experience

Post Condition: System displays Universities most experienced professor

SQL Query:

select professor.Professor_Name, professor.Domain, professor.Experience, university.University_Name

from professor, university

where professor.University_ID = university.University_ID order by professor.Experience desc;

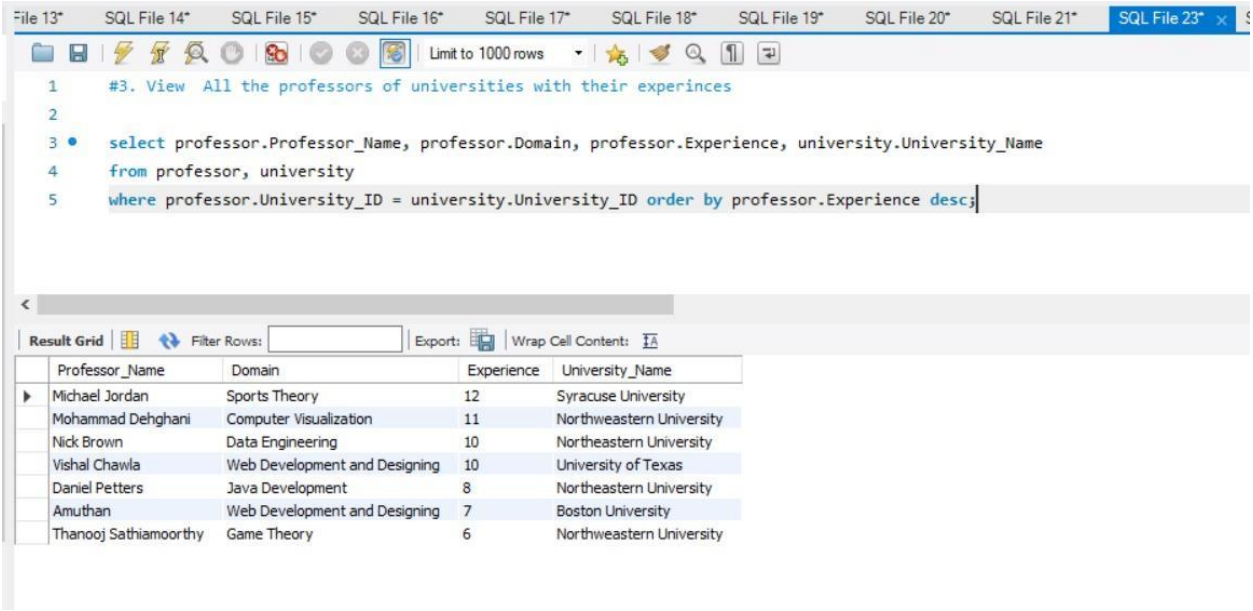
Relational Algebra:

T professor . experience ↓

Tπ professor . professor_name, professor . domain, professor . experience, university . university_name

σ professor . university_id = university . university_id (professor × university)

OUTPUT



4. **Use Case: View the List of eligible Universities for students based on student GRE and University GRE**

Description: Use views of the universities above a particular gre cutoff

Actor: Student

Precondition: The student must enter the gre score

Steps:

Actor action: Student views the universities above a particular cutoff

System Responses: the list of universities above a cutoff is displayed

Post Condition: system displays the list of universities for the condition

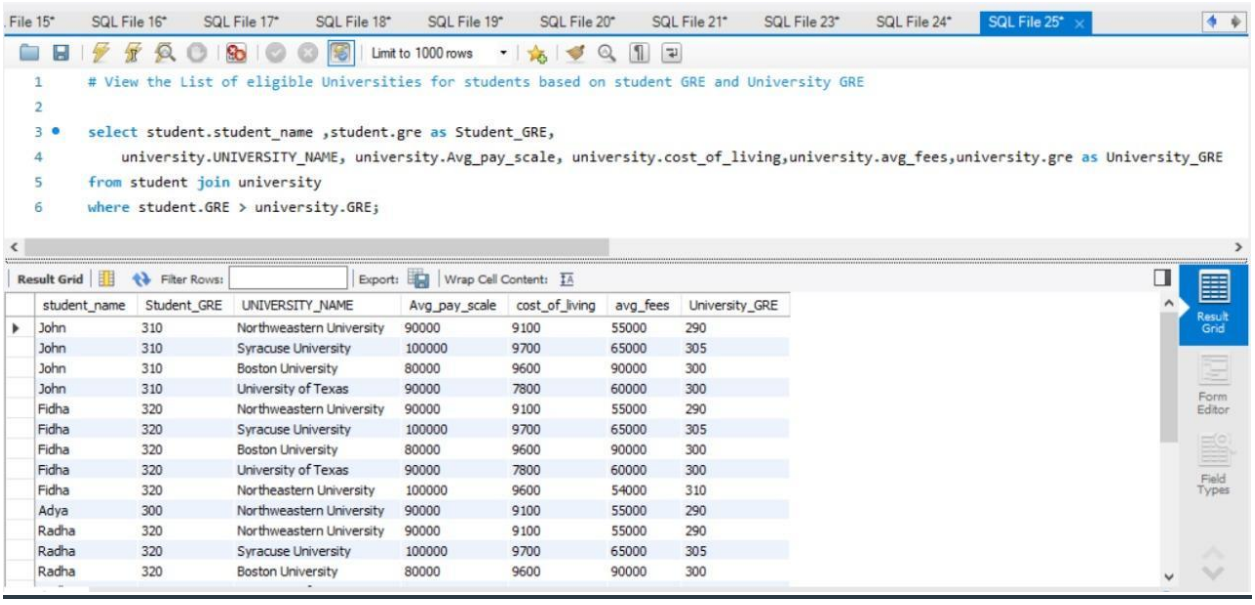
SQL Query:

```
select student.student_name ,student.gre as Student_GRE,
        university.UNIVERSITY_NAME, university.Avg_pay_scale,
university.cost_of_living,university.avg_fees,university.gre as University_GRE
from student join university
where student.GRE > university.GRE;
```

Relational Algebra:

π student . student_name, student . gre → student_gre, university . university_name, university . avg_pay_scale, university . cost_of_living, university . avg_fees, university . gre → university_gre (student ⋈ student . gre > university . gre university)

OUTPUT



student_name	Student_GRE	UNIVERSITY_NAME	Avg_pay_scale	cost_of_living	avg_fees	University_GRE
John	310	Northweastern University	90000	9100	55000	290
John	310	Syracuse University	100000	9700	65000	305
John	310	Boston University	80000	9600	90000	300
John	310	University of Texas	90000	7800	60000	300
Fidha	320	Northweastern University	90000	9100	55000	290
Fidha	320	Syracuse University	100000	9700	65000	305
Fidha	320	Boston University	80000	9600	90000	300
Fidha	320	University of Texas	90000	7800	60000	300
Fidha	320	Northeastern University	100000	9600	54000	310
Adya	300	Northweastern University	90000	9100	55000	290
Radha	320	Northweastern University	90000	9100	55000	290
Radha	320	Syracuse University	100000	9700	65000	305
Radha	320	Boston University	80000	9600	90000	300

5. **Use Case: View the Universities in a particular location**

   Description: Student views the universities based on a particular location

   Actor: Student

Precondition: Student must have entered a particular location

Steps:

Actor action: Student views the university based on the location

System Responses: Displays all the universities which are in the range of the location

Alternate Path: The system doesn't display universities in that location

Error: There are no universities in this location

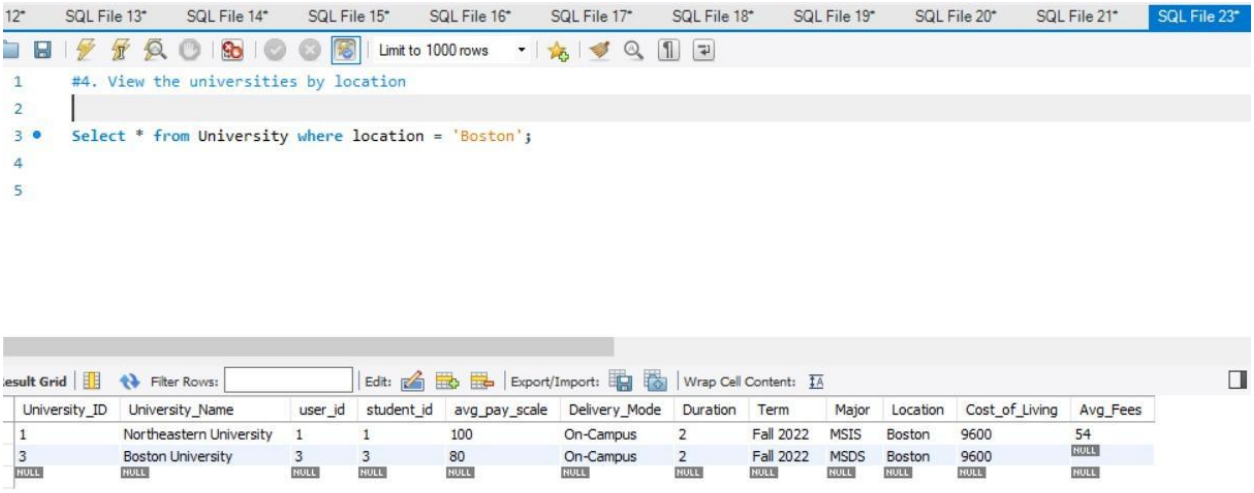
SQL Query:

Select * from University where location = 'Boston';

Relational Algebra:

$\sigma_{\text{location = "Boston"}}$ university

OUTPUT



---

6. Use Case: View information about the university for the Major available as MSIS for Fall 2022

Description: Student views the universities by major it provides

Actor: Student

Precondition: Student must select at least one university to view its major

Steps:

Actor action: Student views the major of universities

System Responses: Displays all the majors in the university

Post Condition: system displays all the majors in a particular university

Alternate Path: There are no majors in that university

Error: No major of history of universities is available

SQL Query:

Select University_Name, avg_pay_scale AS Expected_Job_Salary_in_thousands, Delivery_Mode,Duration,Term,Major,Location

From university

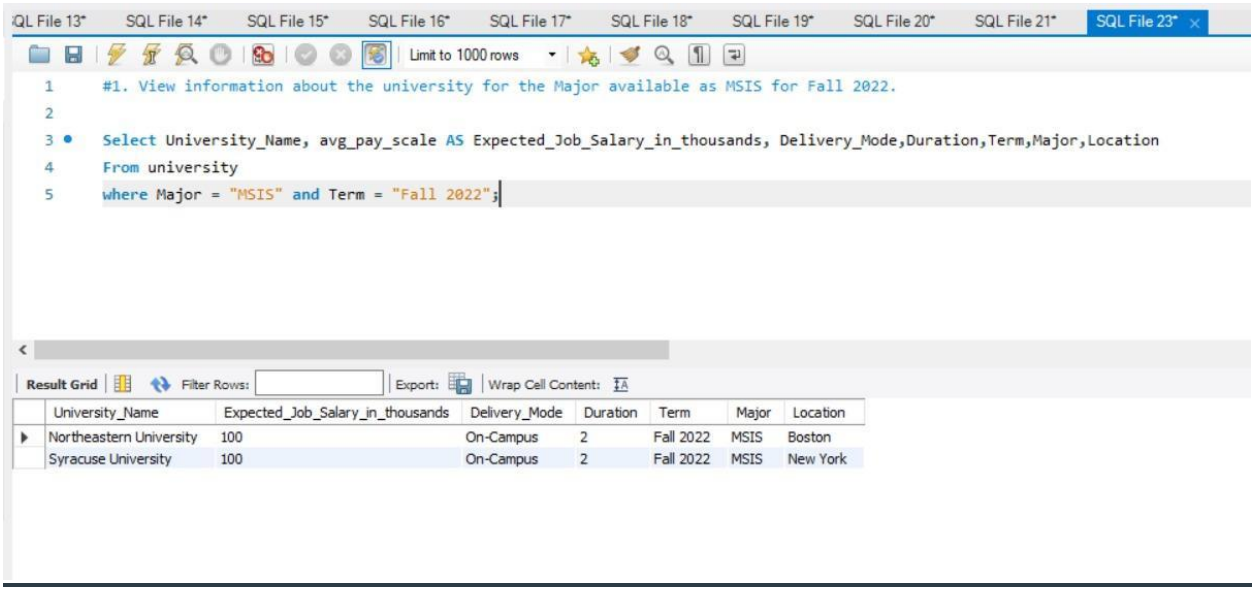
where Major = "MSIS" and Term = "Fall 2022";

Relational Algebra:

π university_name, avg_pay_scale → expected_job_salary_in_thousands, delivery_mode, duration, term, major, location

σ major = "MSIS" AND term = "Fall 2022" university

OUTPUT



7. Use Case: View the ongoing research of a particular university

Description: The research opportunities available in a university

Actor: Student

Precondition: Research opportunities must be available in a university

Steps:

Actor action: Student views the universities as a research opportunity

System Responses: the list of research of a particular university is generated

Post Condition: system displays the list of research opportunities available in a university

SQL Query:

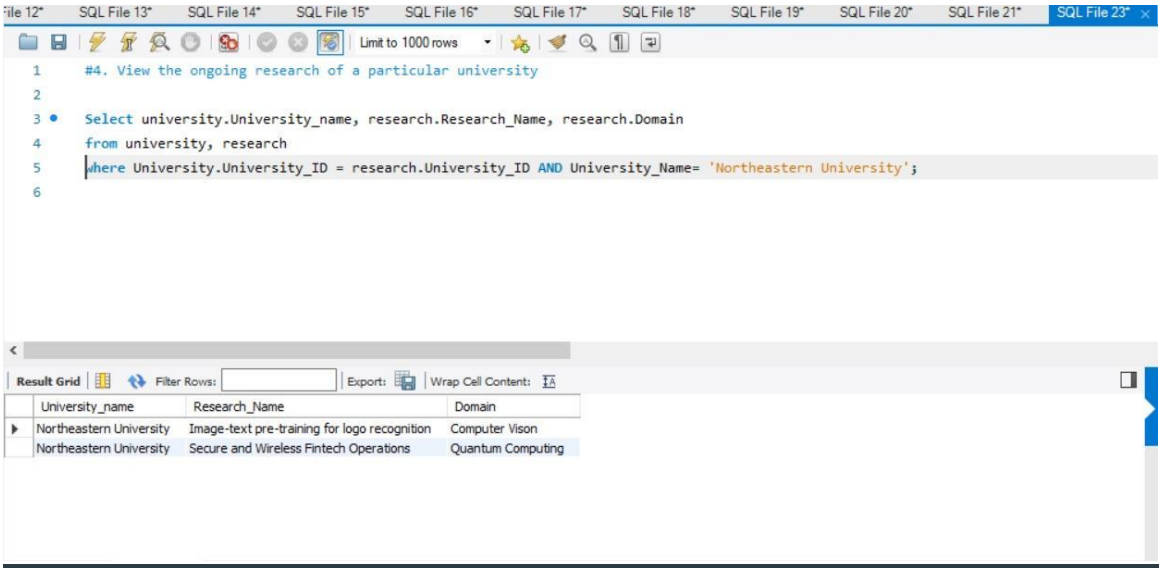
```
Select university.University_name, research.Research_Name, research.Domain

from university, research

where University.University_ID = research.University_ID AND University_Name= 'Northeastern University';
```

Relational Algebra:

π university . university_name, research . research_name, research . domain

σ university . university_id = research . university_id AND university_name = "Northeastern University" (university $\times$ research)

OUTPUT



8. Use Case: View the total Cost for courses offered by Northeastern University

Description: Use the cost per credit to calculate the total cost a student will incur for master's in a university

Actor: Student

Precondition: The student must enter the University name

Steps:

Actor action: Student views the cost of courses offered by the university

System Responses: the total cost is displayed by the university

Post Condition: System displays the overall cost of courses offered by the university

SQL Query:

```
select
        (select count(Course_Name)from course where University_ID = 1) AS No_of_Subjects_Available,
                (Select count(Course_Name) from course where University_ID = 1)*(Credits*Fees_per_credit) AS TOTAL_COST
from course
```

where University_ID = 1

group by University_ID;

Relational Algebra:

Let's break the queries in sub-parts

Let **q1** be the 1st nested Select statement

Let **q2** be the 2nd nested Select statement

q1 - π COUNT (course_name)

γ COUNT (course_name)

σ university_id = 1 course

q2 - π COUNT (course_name)

γ COUNT (course_name)

σ university_id = 1 course

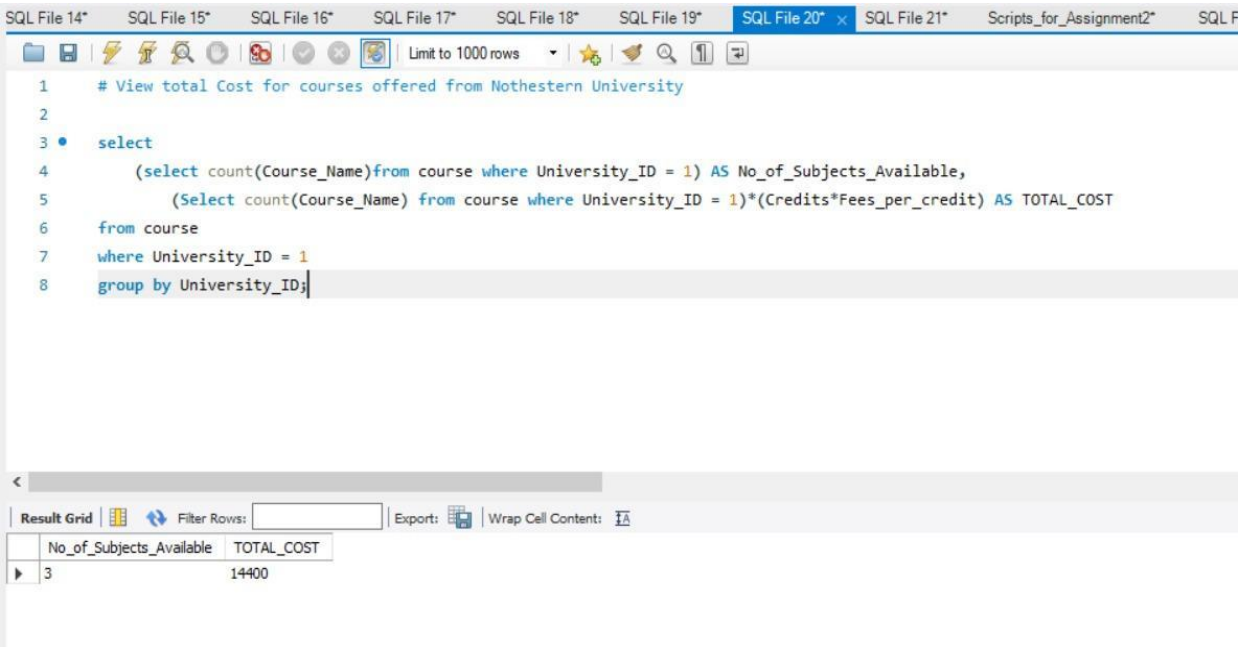
Therefore, the entire relational algebra results in -

π q1 → no_of_subjects_available, q2 * (credits * fees_per_credit) → total_cost

γ university_id,

σ university_id = 1 course

OUTPUT



9. Use Case: <u>View the universities in a particular fees range (say 50k to 60k)</u>

Description: Display the universities in between a particular fee range

Actor: Student

Precondition: University fees structure must be present

Steps:

Actor action: Student views the universities in between a particular fee structure

System Responses: the system generated a list of universities in between a particular fee range

Post Condition: system displays the list of universities for the condition

<u>SQL Query:</u>

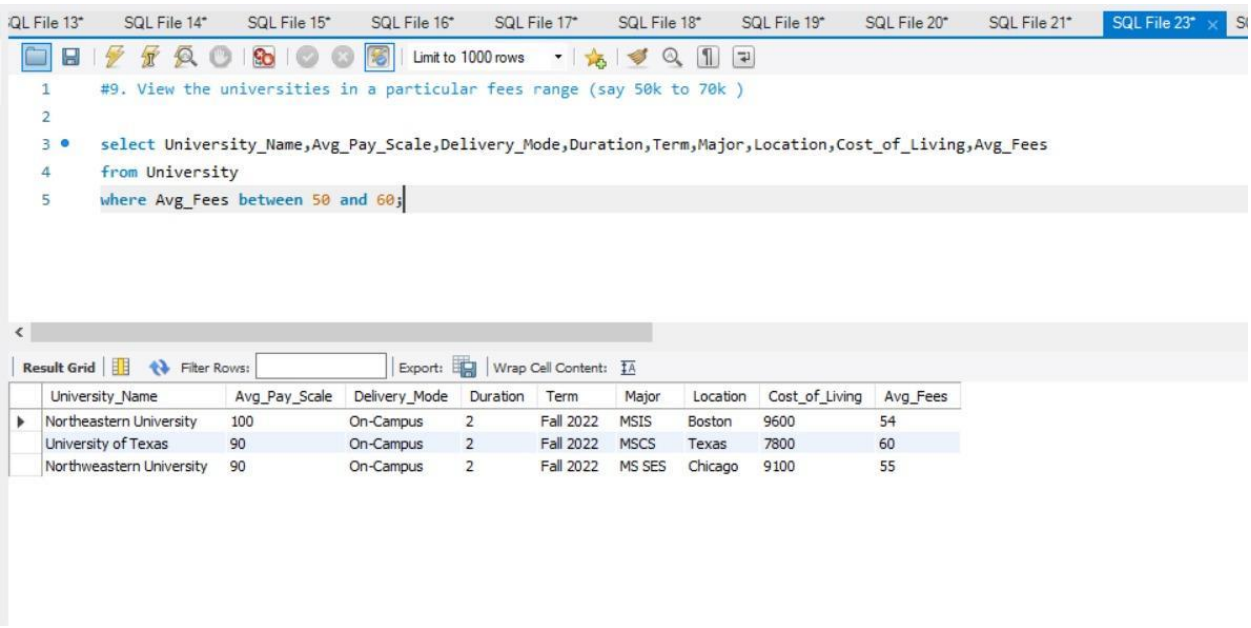
```
select
University_Name,Avg_Pay_Scale,Delivery_Mode,Duration,Term,Major,Location,Cost_of_Living,Avg_Fees

from University

where Avg_Fees between 50 and 60;
```

Relational Algebra:

π university_name, avg_pay_scale, delivery_mode, duration, term, major, location, cost_of_living, avg_fees

σ 50 <= avg_fees AND avg_fees <= 60 university

OUTPUT



10. Use Case: View the universities by their course offering

Description: Student views the universities by the course that the university offers

Actor: Student

Precondition: Student must have made at least one university to view their course

Steps:

Actor action: Student views the coursework of universities

System Responses: Displays all the courses offered by the university

Alternate Path: There are no relevant courses in that university

Error: No history of courses available.

SQL Query:

```
Select university.University_name, course.Course_Name
from university, course
where University.University_ID = course.University_ID;
```

Relational Algebra:

π university . university_name, course . course_name

σ university . university_id = course . university_id (university × course)

OUTPUT: