

# Data Management And Database Design

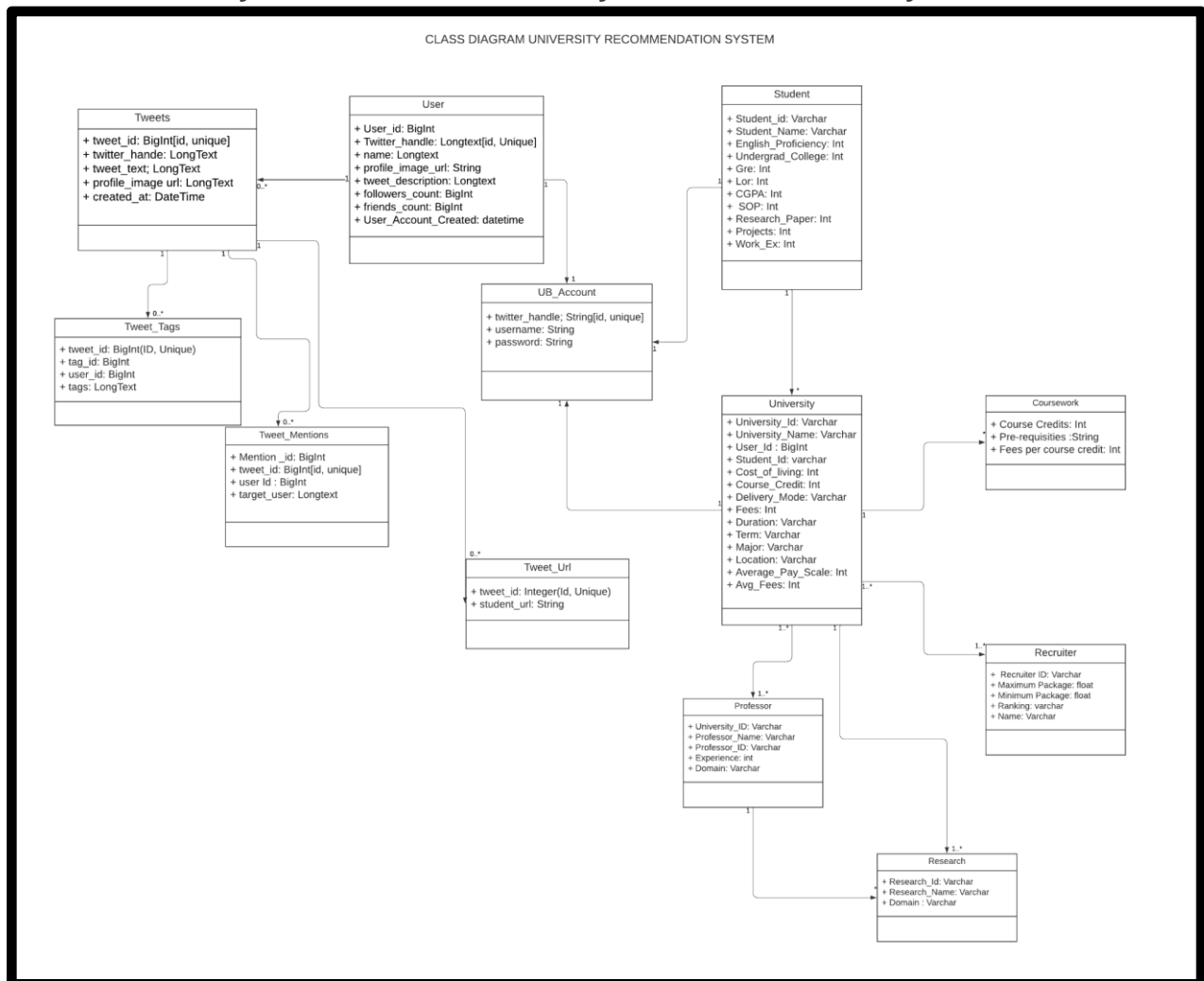
## Assignment - 2

**GitHub Repository:** [https://github.com/Muskansri1/University\\_Recommendation\\_System](https://github.com/Muskansri1/University_Recommendation_System)

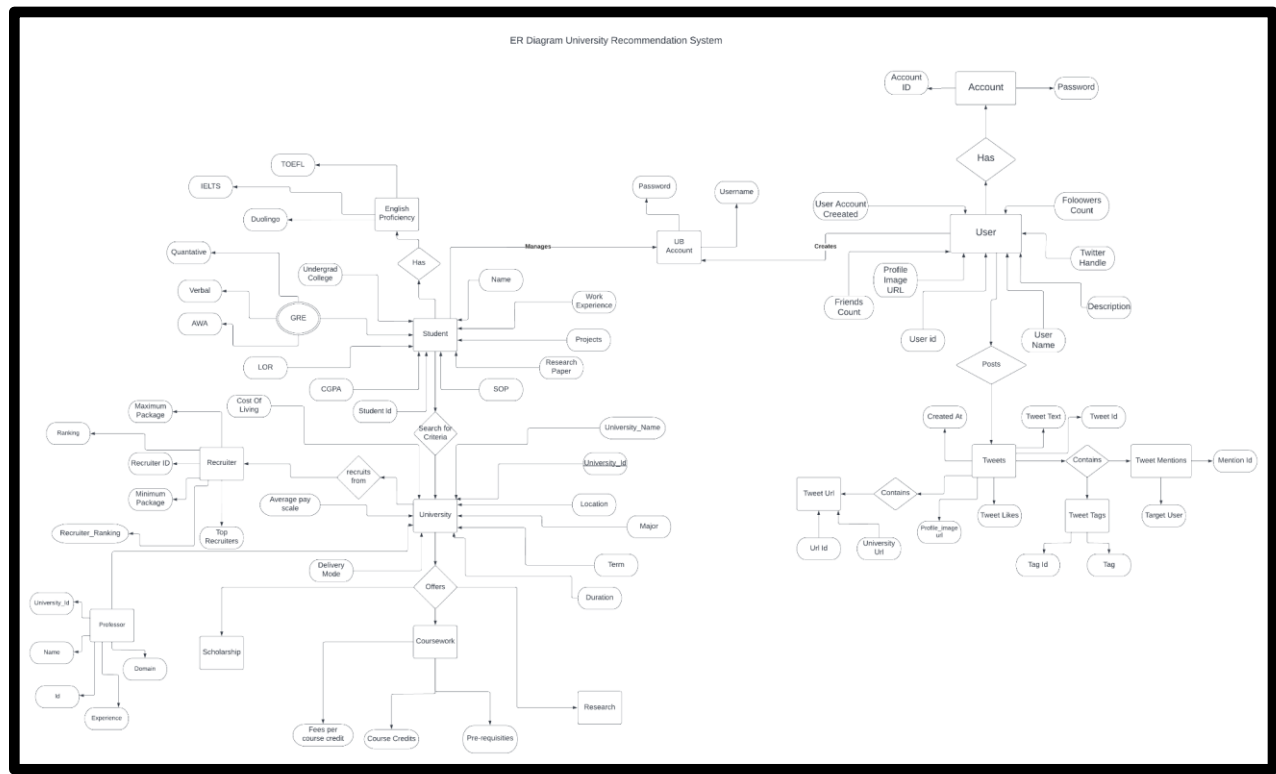
**Group Members:** Sameer Sanjay Nimse (002752914)

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
```

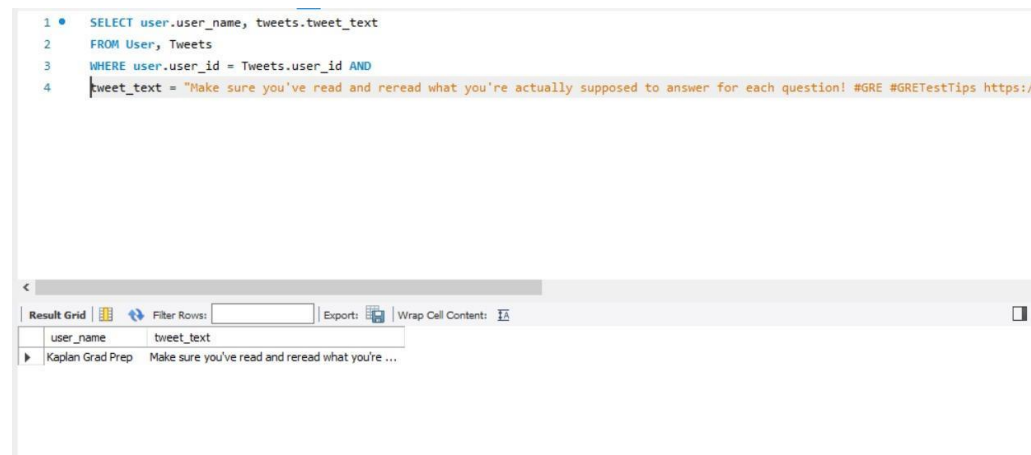
```
tweet_text = "Make sure you've read and reread what you're actually supposed to  
answer for each question! #GRE #GRETestTips https://t.co/tHUH4SIRrK";
```

### Relational Algebra:

$\Pi$  user . user\_name, tweets . tweet\_text

$\sigma$  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/tHUH4SIRrK>" (user  $\times$  tweets)

### OUTPUT:



```
1 • SELECT user.user_name, tweets.tweet_text
2 FROM User, Tweets
3 WHERE user.user_id = Tweets.user_id AND
4 tweet_text = "Make sure you've read and reread what you're actually supposed to answer for each question! #GRE #GRETestTips https://t.co/tHUH4SIRrK";
```

user_name	tweet_text
Kaplan Grad Prep	Make sure you've read and reread what you're ...

## 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/tHUH4SIRrK";
```

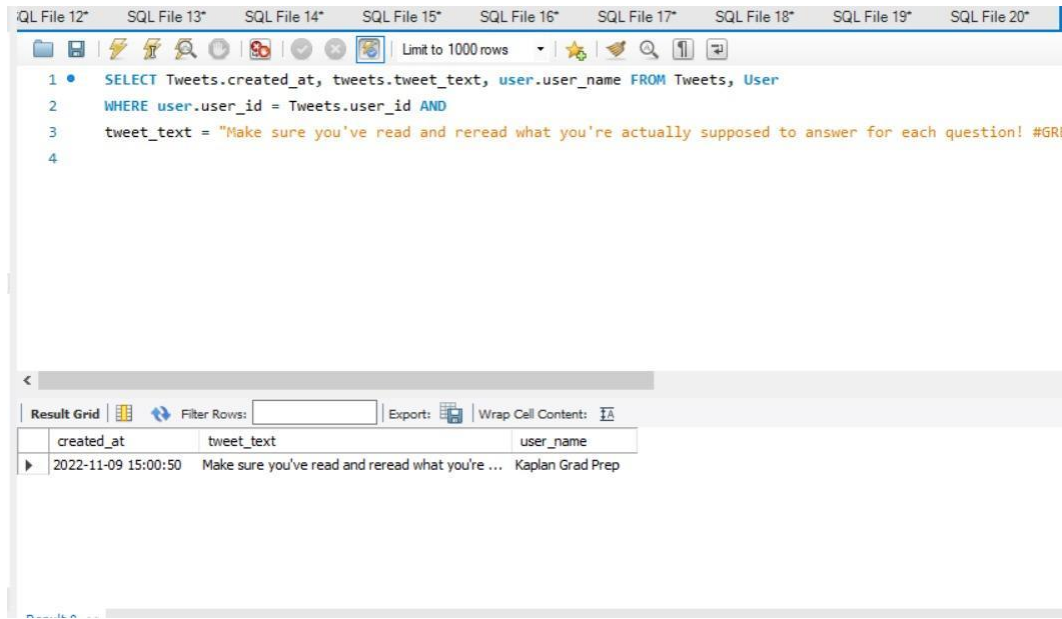


## Relational Algebra:

$\Pi$  tweets . created\_at, tweets . tweet\_text, user . user\_name

$\sigma$  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/THUH4SIRrK>" (tweets  $\times$  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

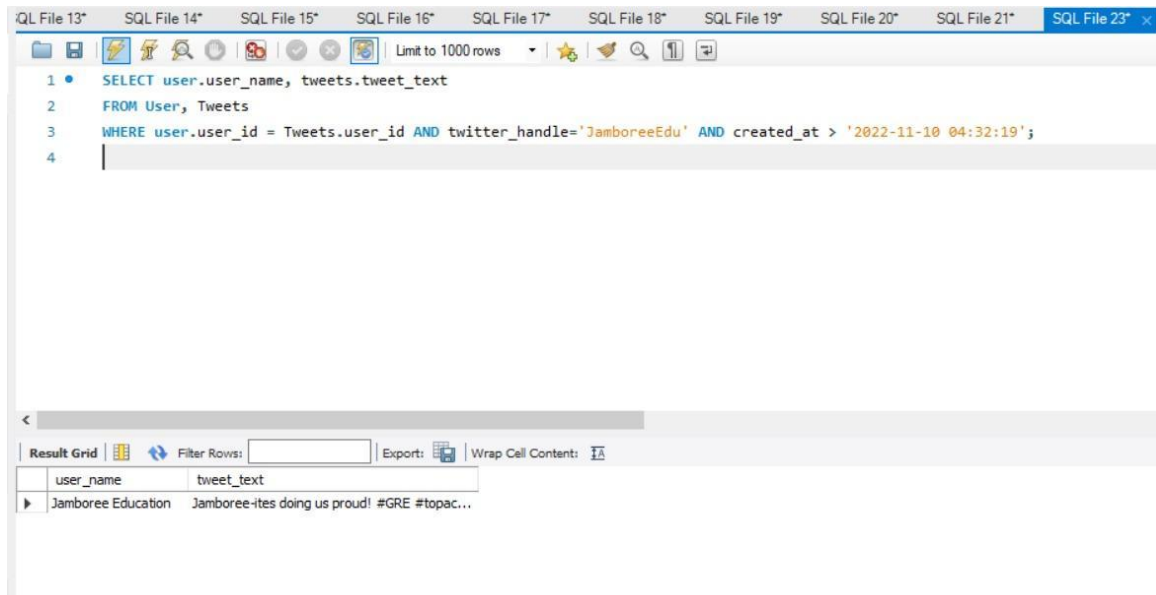
WHERE user.user\_id = Tweets.user\_id AND twitter\_handle='JamboreeEdu' AND  
created\_at > '2022-11-10 04:32:19';

## Relational Algebra

$\Pi$  user . user\_name, tweets . tweet\_text

$\sigma$  user . user\_id = tweets . user\_id AND twitter\_handle = "JamboreeEdu" AND created\_at > "2022-11-10 04:32:19" (user  $\times$  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';

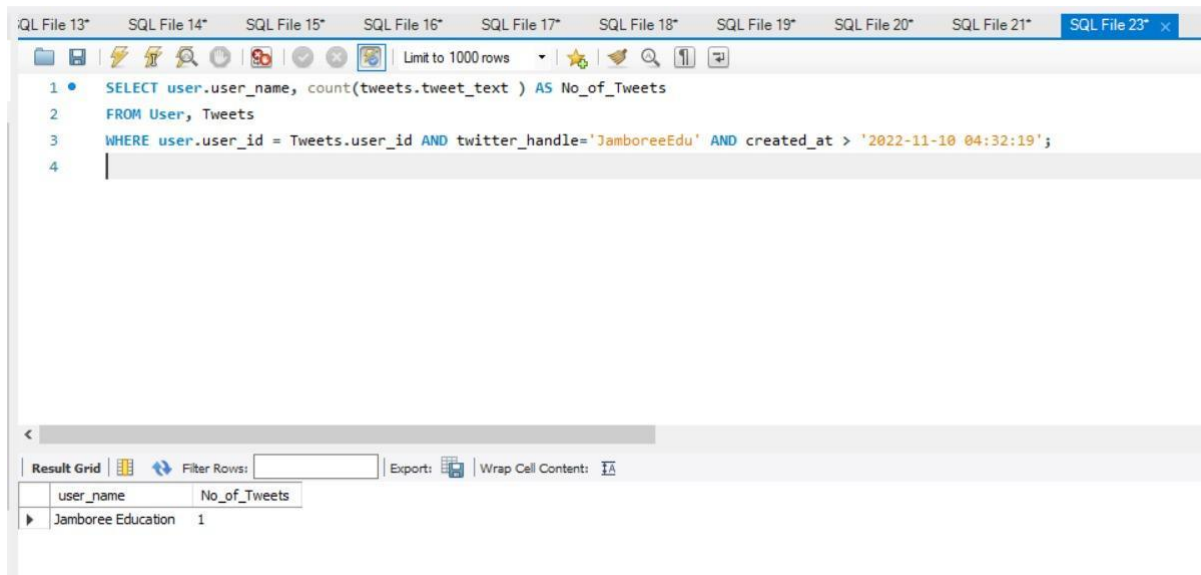
### Relational Algebra

$\Pi$  user . user\_name, COUNT (tweet\_text)  $\rightarrow$  no\_of\_tweets

$\gamma$  COUNT (tweet\_text)

$\sigma$  user . user\_id = tweets . user\_id AND twitter\_handle = "JamboreeEdu" AND created\_at > "2022-11-10 04:32:19" (user  $\times$  tweets)

## OUTPUT



The screenshot shows a SQL IDE with multiple tabs. The active tab is 'SQL File 23'. The query editor contains the following SQL query:

```
1 • SELECT user.user_name, count(tweets.tweet_text ) AS No_of_Tweets
2 FROM User, Tweets
3 WHERE user.user_id = Tweets.user_id AND twitter_handle='JamboreeEdu' AND created_at > '2022-11-10 04:32:19';
4 |
```

Below the query editor, the 'Result Grid' is visible. It shows a single row of results:

user_name	No_of_Tweets
Jamboree Education	1

## 5. When did this user join Twitter?

### SQL Query

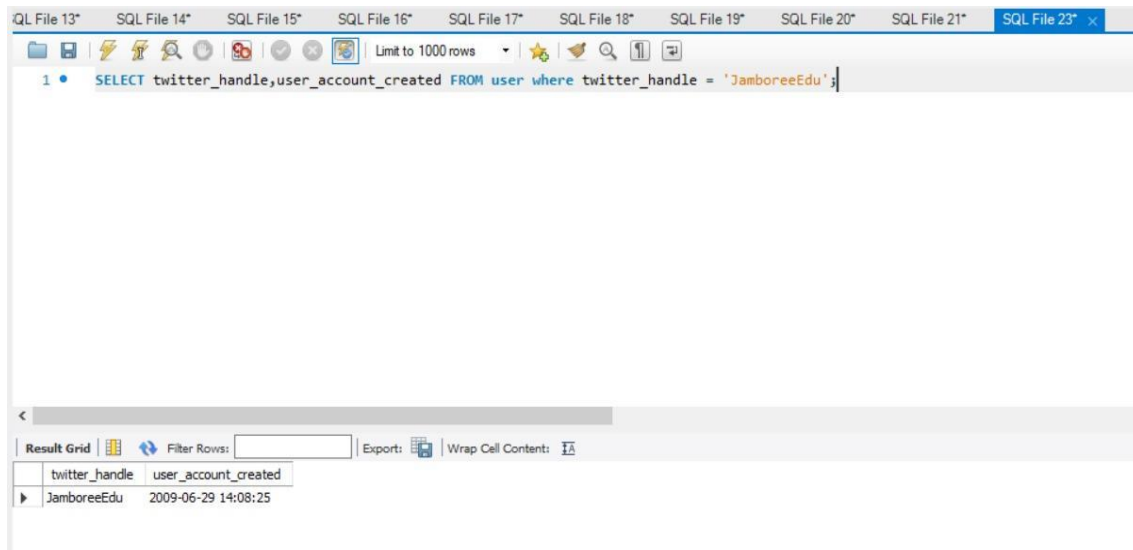
SELECT twitter\_handle,user\_account\_created FROM user where twitter\_handle = 'JamboreeEdu';

### Relational Algebra

$\Pi$  twitter\_handle, user\_account\_created

$\sigma$  twitter\_handle = "JamboreeEdu" USER

## OUTPUT



The screenshot shows a SQL IDE interface with multiple tabs labeled 'SQL File 13\*' through 'SQL File 23\*'. The active tab is 'SQL File 23\*'. The query editor contains the following SQL query:

```
1 • SELECT twitter_handle,user_account_created FROM user where twitter_handle = 'JamboreeEdu';
```

Below the query editor, the 'Result Grid' is displayed. It shows a table with two columns: 'twitter\_handle' and 'user\_account\_created'. The table contains one row of data:

twitter_handle	user_account_created
JamboreeEdu	2009-06-29 14:08:25

## 6. What keywords/ hashtags are popular?

### SQL Query

Select tags,COUNT(tags) AS Frequency

from tweet\_tags group by tags order by Frequency DESC

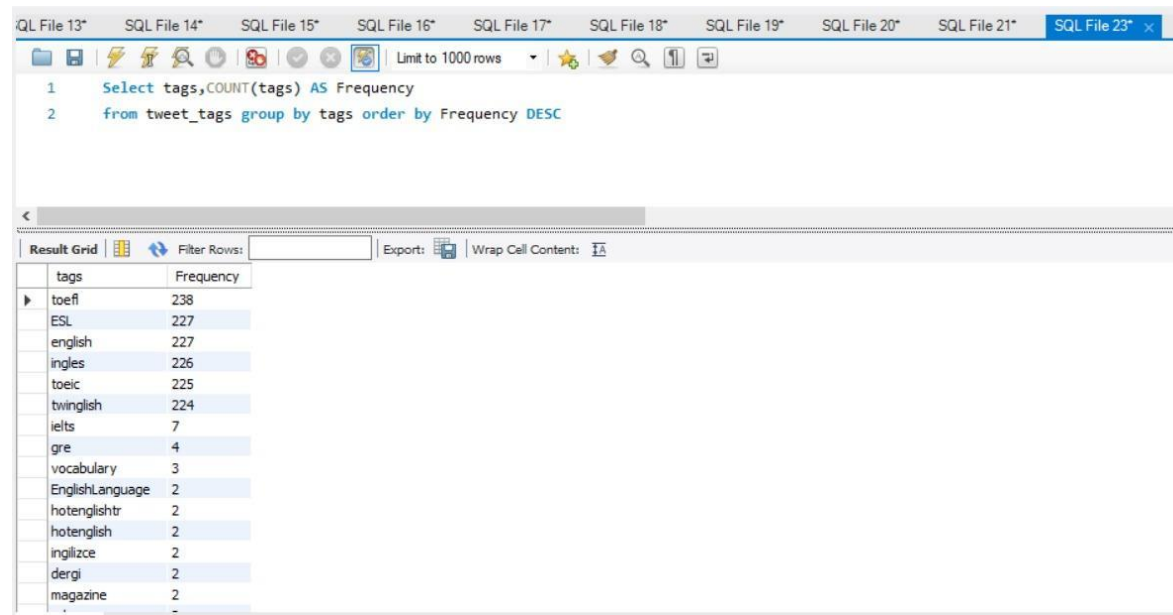
### Relational Algebra

$\pi$  frequency  $\downarrow$

$\pi$  tags, COUNT (tags)  $\rightarrow$  frequency

$\gamma$  tags, COUNT (tags) tweet\_tags

## OUTPUT



The screenshot shows a SQL IDE with multiple tabs. The active tab is 'SQL File 23\*'. The query editor contains the following SQL query:

```
1 Select tags,COUNT(tags) AS Frequency
2 from tweet_tags group by tags order by Frequency DESC
```

Below the query editor, the 'Result Grid' tab is selected, displaying the results of the query. The results are as follows:

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

$\sigma$  tweet\_likes > 500 tweets











## OUTPUT

SQL 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: [IA](#)

	tweet_text	tweet_likes
▶	  #ingles #ESL #toei... 831	831
	  #ingles #ESL #toei... 749	749
	  #ingles #ESL #toei... 641	641
	  #ingles #ESL #toei... 596	596
	  #ingles #ESL #toei... 595	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;
```

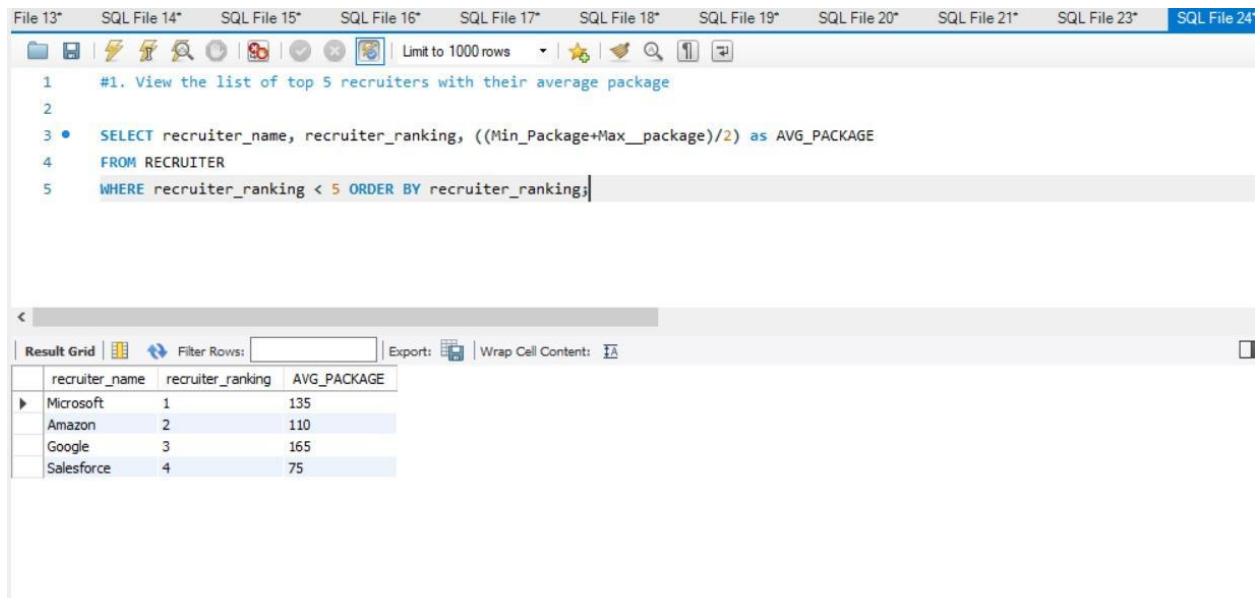
**Relational Algebra:**

$\pi$  recruiter\_ranking

$\pi$  recruiter\_name, recruiter\_ranking, (min\_package + max\_\_package)/ 2  $\rightarrow$  avg\_package

$\sigma$  recruiter\_ranking < 5 recruiter

## OUTPUT



The screenshot shows a SQL IDE interface with multiple tabs at the top, including '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\*', and 'SQL File 24\*'. The active tab is 'SQL File 24\*'. The query editor contains the following SQL code:

```
1 #1. View the list of top 5 recruiters with their average package
2
3 • SELECT recruiter_name, recruiter_ranking, ((Min_Package+Max_package)/2) as AVG_PACKAGE
4 FROM RECRUITER
5 WHERE recruiter_ranking < 5 ORDER BY recruiter_ranking;
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query. The grid has three columns: 'recruiter\_name', 'recruiter\_ranking', and 'AVG\_PACKAGE'. The results are as follows:

recruiter_name	recruiter_ranking	AVG_PACKAGE
Microsoft	1	135
Amazon	2	110
Google	3	165
Salesforce	4	75

## 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:**

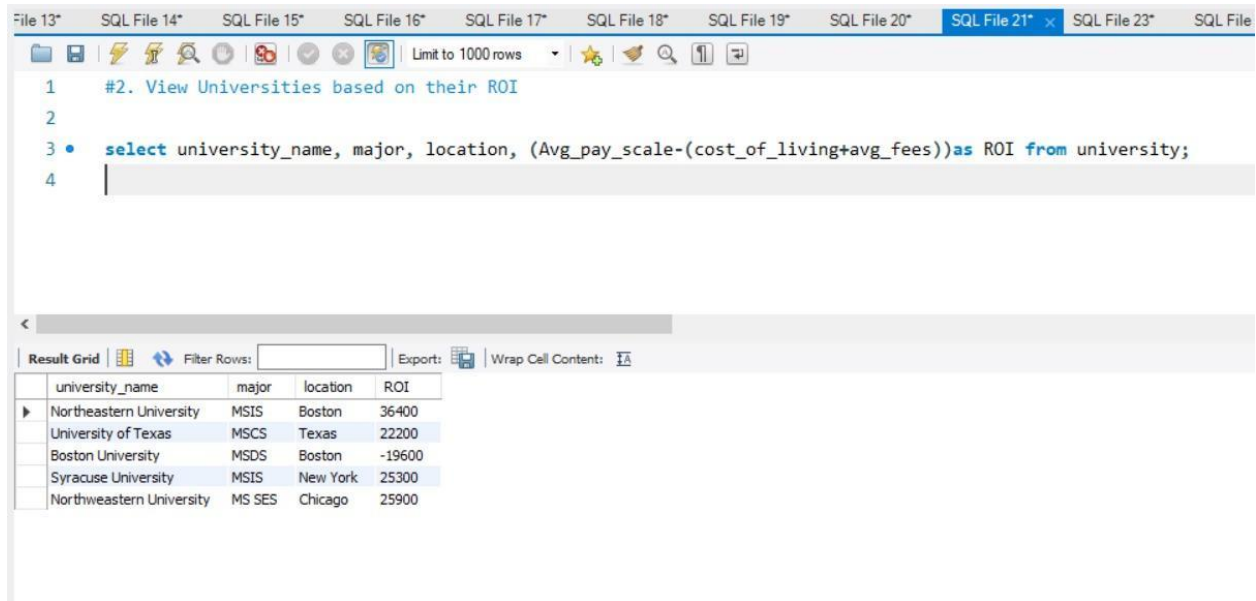
```
select university_name, major, location, (Avg_pay_scale-(cost_of_living+avg_fees))as  
ROI from university;
```

**Relational Algebra:**



TT university\_name, major, location, avg\_pay\_scale - (cost\_of\_living + avg\_fees) → roi university

## OUTPUT



The screenshot shows a SQL IDE interface with multiple tabs at the top. The active tab is 'SQL File 21\*'. The query editor contains the following SQL query:

```
#2. View Universities based on their ROI  
  
select university_name, major, location, (Avg_pay_scale-(cost_of_living+avg_fees))as ROI from university;
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query. The table has four columns: university\_name, major, location, and ROI. The results are as follows:

university_name	major	location	ROI
Northeastern University	MSIS	Boston	36400
University of Texas	MSCS	Texas	22200
Boston University	MSDS	Boston	-19600
Syracuse University	MSIS	New York	25300
Northwestern University	MS SES	Chicago	25900

### 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 professor

**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;

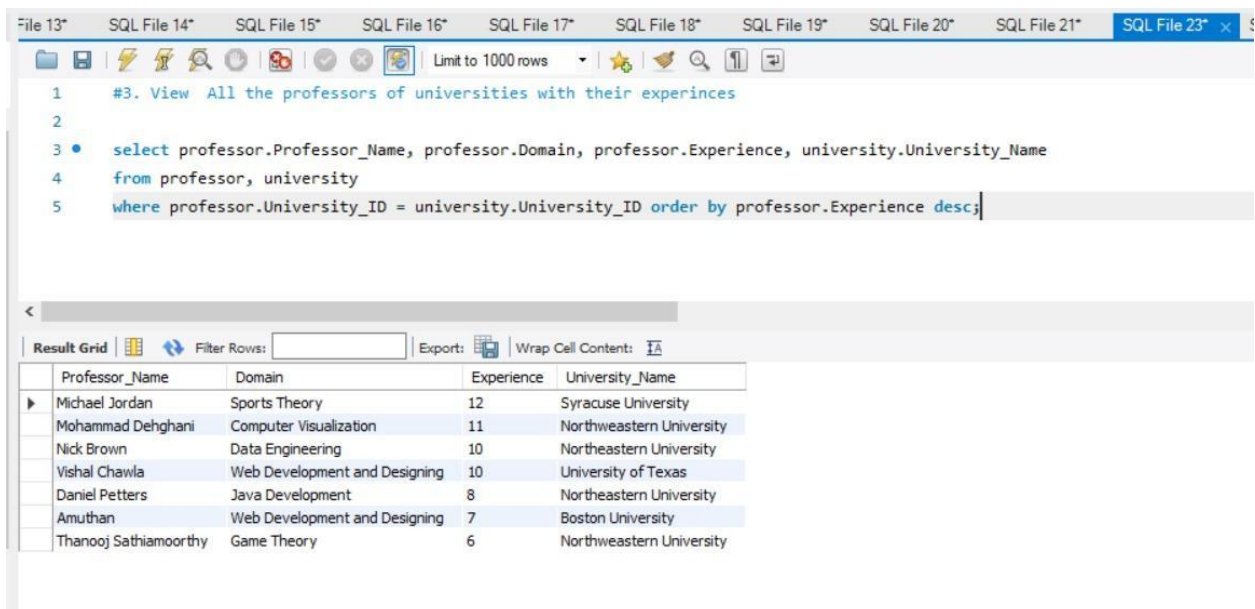
### Relational Algebra:

$\pi$  professor . experience ↓

$\pi$  professor . professor\_name, professor . domain, professor . experience, university . university\_name

$\sigma$  professor . university\_id = university . university\_id (professor  $\times$  university)

### OUTPUT



The screenshot shows a SQL IDE with multiple tabs. The active tab is 'SQL File 23'. The query editor contains the following SQL code:

```
1 #3. View All the professors of universities with their experiences
2
3 • select professor.Professor_Name, professor.Domain, professor.Experience, university.University_Name
4 from professor, university
5 where professor.University_ID = university.University_ID order by professor.Experience desc;
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query. The table has 5 columns: Professor\_Name, Domain, Experience, and University\_Name. The results are ordered by Experience in descending order.

Professor_Name	Domain	Experience	University_Name
Michael Jordan	Sports Theory	12	Syracuse University
Mohammad Dehghani	Computer Visualization	11	Northwestern University
Nick Brown	Data Engineering	10	Northeastern University
Vishal Chawla	Web Development and Designing	10	University of Texas
Daniel Petters	Java Development	8	Northeastern University
Amuthan	Web Development and Designing	7	Boston University
Thanooj Sathiamoorthy	Game Theory	6	Northwestern University

#### 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:**

$\pi$  student . student\_name, student . gre  $\rightarrow$  student\_gre, university . university\_name,  
university . avg\_pay\_scale, university . cost\_of\_living, university . avg\_fees, university .  
gre  $\rightarrow$  university\_gre (student  $\bowtie$  student . gre > university . gre university)

**OUTPUT**

The screenshot shows a database application interface. At the top, there are tabs for multiple SQL files, with 'SQL File 25\*' currently selected. Below the tabs is a toolbar with various icons and a 'Limit to 1000 rows' dropdown. The main area displays an SQL query with line numbers 1 through 6. Below the query editor is a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Contents' checkbox. The grid itself contains 8 columns: student\_name, Student\_GRE, UNIVERSITY\_NAME, Avg\_pay\_scale, cost\_of\_living, avg\_fees, and University\_GRE. There are 16 rows of data, each representing a student and their associated university information. On the right side of the grid, there are buttons for 'Result Grid', 'Form Editor', and 'Field Types'.

	student_name	Student_GRE	UNIVERSITY_NAME	Avg_pay_scale	cost_of_living	avg_fees	University_GRE
▶	John	310	Northwestern 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	Northwestern 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	Northwestern University	100000	9600	54000	310
	Adya	300	Northwestern University	90000	9100	55000	290
	Radha	320	Northwestern 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

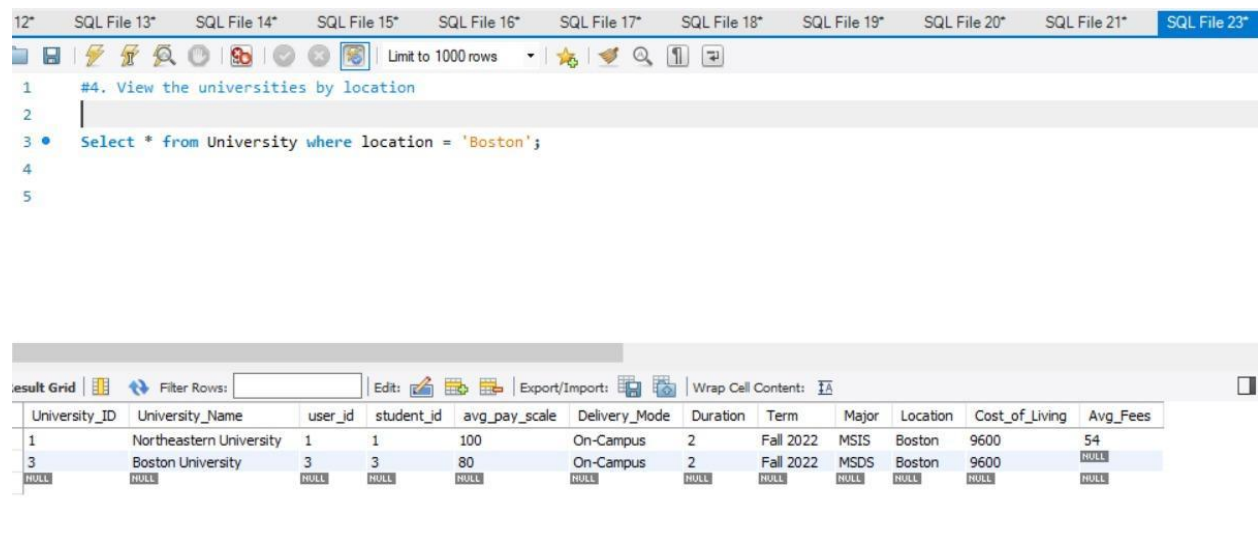
### SQL Query:

Select \* from University where location = 'Boston';

### Relational Algebra:

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

### OUTPUT



The screenshot shows a SQL IDE interface with a toolbar at the top containing icons for file operations, search, and execution. Below the toolbar, a list of SQL files is displayed, with 'SQL File 23\*' selected. The main editor area shows a SQL query: `Select * from University where location = 'Boston';`. Below the query, a 'result Grid' is visible, displaying the results of the query. The grid has columns for University\_ID, University\_Name, user\_id, student\_id, avg\_pay\_scale, Delivery\_Mode, Duration, Term, Major, Location, Cost\_of\_Living, and Avg\_Fees. The results show two rows: one for Northeastern University and one for Boston University, both located in Boston.

University_ID	University_Name	user_id	student_id	avg_pay_scale	Delivery_Mode	Duration	Term	Major	Location	Cost_of_Living	Avg_Fees
1	Northeastern University	1	1	100	On-Campus	2	Fall 2022	MSIS	Boston	9600	54
3	Boston University	3	3	80	On-Campus	2	Fall 2022	MSDS	Boston	9600	NULL

## 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

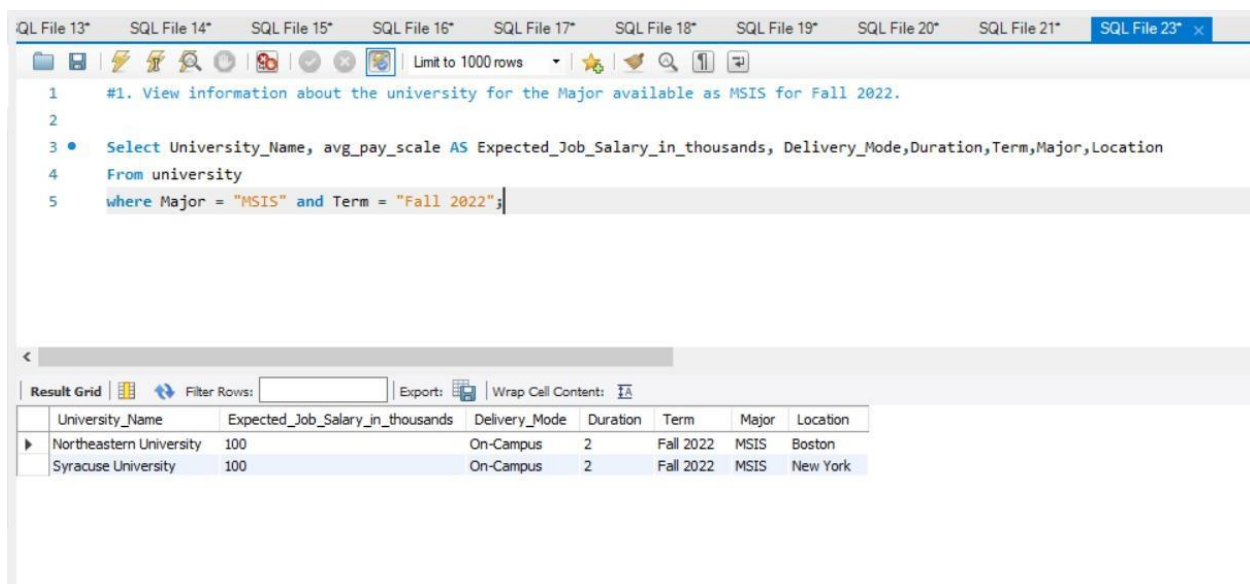
where Major = "MSIS" and Term = "Fall 2022";

## Relational Algebra:

$\pi$  university\_name, avg\_pay\_scale  $\rightarrow$  expected\_job\_salary\_in\_thousands, delivery\_mode, duration, term, major, location

$\sigma$  major = "MSIS" AND term = "Fall 2022" university

## OUTPUT



The screenshot shows a SQL IDE with multiple tabs. The active tab, 'SQL File 23', contains the following SQL query:

```
1 #1. View information about the university for the Major available as MSIS for Fall 2022.
2
3 • Select University_Name, avg_pay_scale AS Expected_Job_Salary_in_thousands, Delivery_Mode,Duration,Term,Major,Location
4 From university
5 where Major = "MSIS" and Term = "Fall 2022";
```

Below the query editor, the 'Result Grid' displays the output of the query. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are as follows:

University_Name	Expected_Job_Salary_in_thousands	Delivery_Mode	Duration	Term	Major	Location
Northeastern University	100	On-Campus	2	Fall 2022	MSIS	Boston
Syracuse University	100	On-Campus	2	Fall 2022	MSIS	New York

## 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:

$\Pi$  university . university\_name, research . research\_name, research . domain

$\sigma$  university . university\_id = research . university\_id AND university\_name = "Northeastern University" (university  $\times$  research)

## OUTPUT

The screenshot shows a SQL IDE with multiple tabs. The active tab is 'SQL File 23'. The query editor contains the following SQL code:

```
1 #4. View the ongoing research of a particular university
2
3 • Select university.University_name, research.Research_Name, research.Domain
4 from university, research
5 where University.University_ID = research.University_ID AND University_Name= 'Northeastern University';
6
```

Below the query editor is the 'Result Grid' showing the results of the query. The grid has three columns: 'University\_name', 'Research\_Name', and 'Domain'. There are two rows of data:

University_name	Research_Name	Domain
Northeastern University	Image-text pre-training for logo recognition	Computer Vision
Northeastern University	Secure and Wireless Fintech Operations	Quantum Computing

### 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** -  $\pi$  COUNT (course\_name)

$\gamma$  COUNT (course\_name)

$\sigma$  university\_id = 1 course

**q2** -  $\pi$  COUNT (course\_name)

$\gamma$  COUNT (course\_name)

$\sigma$  university\_id = 1 course

**Therefore, the entire relational algebra results in -**

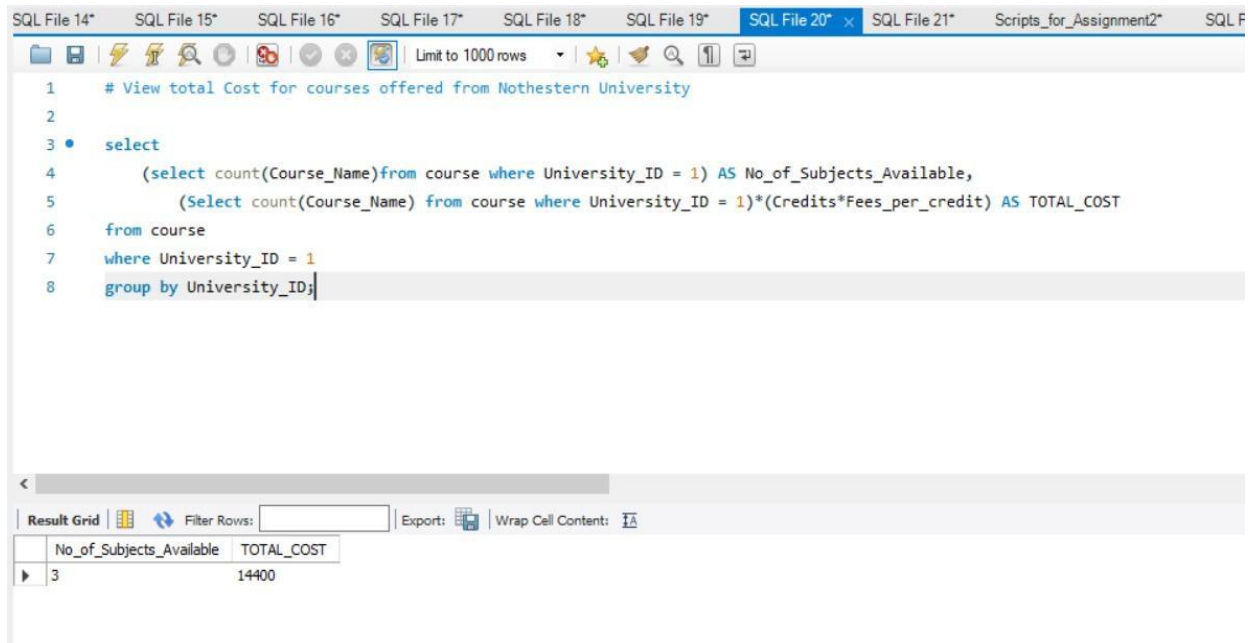
$\pi$  q1  $\rightarrow$  no\_of\_subjects\_available, q2 \* (credits \* fees\_per\_credit)  $\rightarrow$  total\_cost

$\gamma$  university\_id,

$\sigma$  university\_id = 1 course



## OUTPUT



The screenshot shows a SQL IDE with multiple tabs. The active tab is 'SQL File 20\*', which contains the following SQL query:

```
1  # View total Cost for courses offered from Notheastern University
2
3  • select
4      (select count(Course_Name) from course where University_ID = 1) AS No_of_Subjects_Available,
5      (Select count(Course_Name) from course where University_ID = 1)*(Credits*Fees_per_credit) AS TOTAL_COST
6  from course
7  where University_ID = 1
8  group by University_ID;
```

Below the query editor, the 'Result Grid' is displayed, showing the following data:

No_of_Subjects_Available	TOTAL_COST
3	14400

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

**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

**SQL Query:**

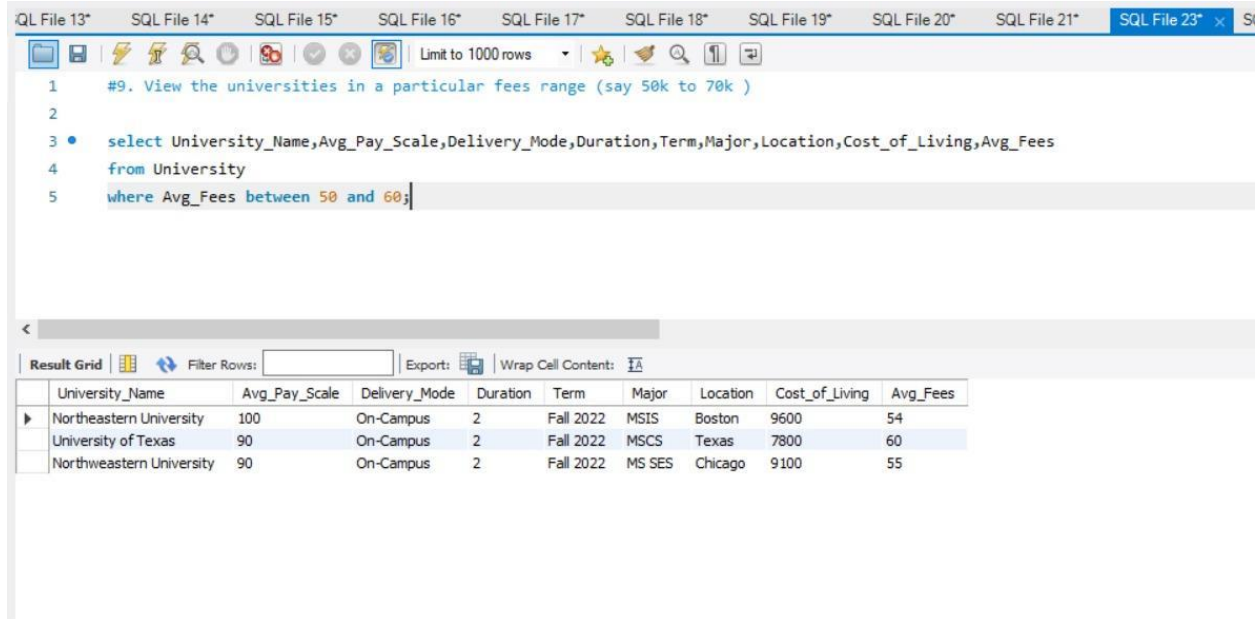
```
select
University_Name,Avg_Pay_Scale,Delivery_Mode,Duration,Term,Major,Location,Cost_o
f_Living,Avg_Fees
from University
where Avg_Fees between 50 and 60;
```

## Relational Algebra:

TT university\_name, avg\_pay\_scale, delivery\_mode, duration, term, major, location, cost\_of\_living, avg\_fees

$\sigma_{50 \leq \text{avg\_fees} \text{ AND } \text{avg\_fees} \leq 60}$  university

## OUTPUT



The screenshot shows a SQL IDE with multiple tabs. The active tab is 'SQL File 23'. The query editor contains the following SQL code:

```
1 #9. View the universities in a particular fees range (say 50k to 70k )
2
3 • select University_Name,Avg_Pay_Scale,Delivery_Mode,Duration,Term,Major,Location,Cost_of_Living,Avg_Fees
4 from University
5 where Avg_Fees between 50 and 60;
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query. The grid has columns for University\_Name, Avg\_Pay\_Scale, Delivery\_Mode, Duration, Term, Major, Location, Cost\_of\_Living, and Avg\_Fees. The results are as follows:

University_Name	Avg_Pay_Scale	Delivery_Mode	Duration	Term	Major	Location	Cost_of_Living	Avg_Fees
Northeastern University	100	On-Campus	2	Fall 2022	MSIS	Boston	9600	54
University of Texas	90	On-Campus	2	Fall 2022	MSCS	Texas	7800	60
Northwestern University	90	On-Campus	2	Fall 2022	MS SES	Chicago	9100	55

### 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

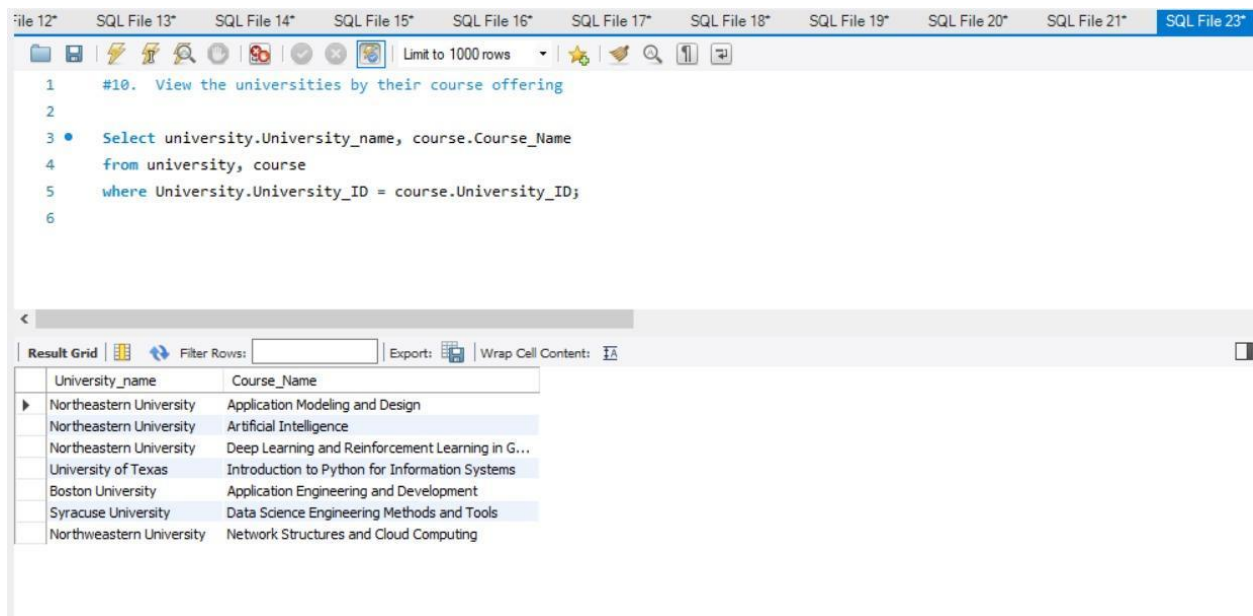
where University.University\_ID = course.University\_ID;

**Relational Algebra:**

$\Pi$  university . university\_name, course . course\_name

$\sigma$  university . university\_id = course . university\_id (university  $\times$  course)

**OUTPUT:**



The screenshot shows a SQL IDE interface with multiple tabs at the top, including 'file 12\*', 'SQL 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\*', and 'SQL File 23\*'. The active tab is 'SQL File 23\*'. The main editor area contains a SQL query:

```
1 #10. View the universities by their course offering
2
3 • Select university.University_name, course.Course_Name
4 from university, course
5 where University.University_ID = course.University_ID;
6
```

Below the editor, there is a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The result grid displays the following data:

University_name	Course_Name
Northeastern University	Application Modeling and Design
Northeastern University	Artificial Intelligence
Northeastern University	Deep Learning and Reinforcement Learning in G...
University of Texas	Introduction to Python for Information Systems
Boston University	Application Engineering and Development
Syracuse University	Data Science Engineering Methods and Tools
Northwestern University	Network Structures and Cloud Computing