CS 432: Databases
Assignment 2: DEVELOPING THE DBMS

Group Name: La Retro's

Topic: Placement Management

Group Members-

G1: Aaryan Darad - 21110001
Abhay Kumar Upparwal - 21110004
Aditya Deshmukh - 21110014
Anugu Arun Reddy - 21110029

G2: Adit Kaushik - 21110010
Gaurav Joshi - 21110065
Ahaan Giriya - 21110015
Rutwik More - 21110133

Important Links

1) ER Diagram: Draw.io file: https://app.diagrams.net/?client=1#G1yCpFEy0jyzqNLsKFg6Lim8za2PPASyPd#%7B%22pageId%22%3A%22n4RG28VcgyLvL5a97sLb%22%7D

2) Code for the Data Generation process: Google Colab notebook: https://colab.research.google.com/drive/1thD8FtA0pBjFwbtnWxz85L4WhKXOa4s-

3) Generated Dataset for the Database: Google Drive: https://drive.google.com/drive/folders/1lr40jKZI_w58t7_DY7AVztuXKtXEc2A2

4) SQL Files, and data dump: https://drive.google.com/drive/folders/1_MqzNJfXfNdrbH-PaVhR7fRd7r1Yt9vL

3.1 Responsibility of G1:

1. *Populate the tables that you created in the previous assignment with random data with the following constraints. All tables must follow the ACID properties and the previous constraints mentioned in Assignment 1.*

Populating the tables with data was a pivotal aspect of our project. To generate diverse data for various entities, we utilized the Faker and Random libraries in Python. These libraries enabled us to create synthetic data, which we then organized into CSV files. Subsequently, we imported these CSV files into MySQL Workbench to populate the corresponding tables.

This process presented its challenges, as we had to meticulously handle various constraints. Ensuring the uniqueness of primary keys across tables was imperative, alongside managing different types of relationships such as total/partial, one-one, many-one, and one-many participation. Additionally, we paid close attention to using appropriate data types for each table.

To streamline this task, we devised individual scripts for each table. Initially, we focused on populating the entity tables. Once completed, we leveraged these entity tables to populate the relationship tables while adhering to their specific constraints. Finally, we imported all the CSV files containing the populated tables into MySQL.

➔ How do we ensure the primary key is not repeated?

Let's say we are generating company entity data:
I have used the below code where we keep track of the ids/company_ids which is the primary key for the company table. If the generated id is company_ids_ we generate it again till it is not in company_ids_

➔ How do we ensure constraints are being followed?

In the below code, if we see for generating, we have used faker, and we generate the name of the company till it follows the mentioned constraint i.e. len(compnay_name)<255.

```python
company_ids_ = []
def generate_company_data():
    company_id = random.randint(0, 9999)
    while company_id in company_ids_:
      company_id = random.randint(0, 9999)
    company_ids_.append(company_id)
    company_name = fake.company()
    if len(company_name)>=255:
      while(len(company_name)>=255):
        company_name = fake.company()
    company_website = company_name.replace(' ', '').lower() + '.com'
    organization_type = random.choice(["Public","Private","Non-Profit"])
    company_field = random.choice(["Tech", "Data", "Finance", "Design"])
    company_brochure = company_name.replace(' ', '').lower() + '_brochure.pdf'
    country_of_origin = random.choice(["USA", "Canada", "UK", "Australia", "Germany", "France", "Japan", "India", "China"])
    return [company_id, company_name, company_website, organization_type, company_field, company_brochure, country_of_origin]


company_entries = []

for _ in range(50):
    cur = generate_company_data()
    if cur in company_entries:
      while cur in company_entries:
        cur = generate_company_data()
    company_entries.append(cur)
```

Similarly, we generated data for all the entities.
Once that is done, we synthesize data for the relationships.

➔ How do we ensure cardinality is being followed?

Let us take the example below. In this, the relationship between company and opportunity is one to many while opportunity is having total participation.
To ensure the above condition of total participation, we iterated through every available opportunity and assigned randomly to any company

```python
opportunity_company_entries = []
for j in range(len(opportunity_entries)):
  comp_id = random.sample(company_entries,1)[0]
  opportunity_company_entries.append([comp[0],opportunity_entries[j][0]])
  comp_opp_dict[comp[0]].append(opportunity_entries[j][0])
```

➔ In the below case to ensure the relationship between the company and POC is one to many with total participation on both sides.

To do the above we iterated through every POC and assigned POC to every company till every company is assigned POC and the remaining POC is randomly assigned.

```
poc_company_entries = []
for j in range(len(person_of_contact_entries)):
  if j<len(company_entries):
    poc_company_entries.append([company_entries[j][0],person_of_contact_entries[j][0]])
    comp_poc_dict[company_entries[j][0]].append(person_of_contact_entries[j][0])
  else:
    comp = random.sample(company_entries)
    poc_company_entries.append([comp[0],person_of_contact_entries[j][0]])
    comp_poc_dict[comp[0]].append(person_of_contact_entries[j][0])
```

(Note: To ensure the number of POC entries generated are greater than company entries we added this check)

```
if len(company_entries) < len(person_of_contact_entries):
    raise ValueError("Less number of POC than requrired")
```

- To connect POC and Opportunity, we took the same company Opportunity and POC and randomly assigned opportunity to POC

```
poc_company_entries = []
for j in range(len(person_of_contact_entries)):
  if j<len(company_entries):
    poc_company_entries.append([company_entries[j][0],person_of_contact_entries[j][0]])
    comp_poc_dict[company_entries[j][0]].append(person_of_contact_entries[j][0])
  else:
    comp = random.sample(company_entries)
    poc_company_entries.append([comp[0],person_of_contact_entries[j][0]])
    comp_poc_dict[comp[0]].append(person_of_contact_entries[j][0])
```

Here is the complete code for making the below :

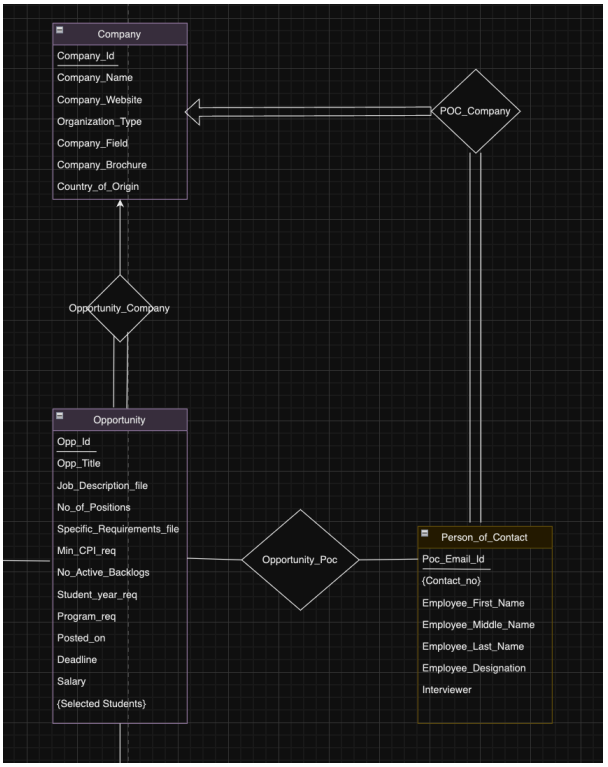
```
opportunity_poc_entries = []
comp_opp_dict = {}
comp_poc_dict = {}
for comp in company_entries:
  comp_opp_dict[comp[0]] = []
  comp_poc_dict[comp[0]] = []

opportunity_company_entries = []
for j in range(len(opportunity_entries)):
  comp_id = random.sample(company_entries,1)[0]
  opportunity_company_entries.append([comp[0],opportunity_entries[j][0]])
  comp_opp_dict[comp[0]].append(opportunity_entries[j][0])



poc_company_entries = []
for j in range(len(person_of_contact_entries)):
  if j<len(company_entries):
    poc_company_entries.append([company_entries[j][0],person_of_contact_entries[j][0]])
    comp_poc_dict[company_entries[j][0]].append(person_of_contact_entries[j][0])
  else:
    comp = random.sample(company_entries)
    poc_company_entries.append([comp[0],person_of_contact_entries[j][0]])
    comp_poc_dict[comp[0]].append(person_of_contact_entries[j][0])


for comp,opps in comp_opp_dict.items():
  pocs = comp_poc_dict[comp]
  for opp in opps:
    k = random.randint(1,4)
    poc = random.sample(pocs,min(k,len(pocs)-1))
    opportunity_poc_entries.append([opp,poc])
```



Here is the link for generated data : [Link]

INDEXING

2. Please explain and implement the indexing over one of the columns (where the search needs to be optimized), user-defined data types, and table extensions.

- We've performed indexing on schemas to facilitate extensive querying. This indexing aims to slash query times and enhance the overall efficiency of our database system. We've selected a set of frequently queried attributes within our database to serve as our indices.

Factors on which we selected indexing attribute-

1. **Most accessed attributes:**
    a. Attributes that are frequently accessed in queries, especially in WHERE, ORDER BY, and GROUP BY clauses should be considered for indexing. This is because an index on these columns will speed up data retrieval.
    b. Some examples include columns used to filter rows like status, type, date columns used to sort output like created_at, id columns used to group aggregation functions like user_id, category_id.
2. **Attributes used most as foreign keys in other tables:**
    a. Columns which are often used as foreign keys to join multiple tables should be indexed. Foreign key columns are queried frequently during joins. Indexes will make these joins faster.
    b. For example, if a 'user_id' column exists in many tables, it should be indexed since WHERE clauses are likely to filter on 'user_id' during joins.
3. **Columns declared as UNIQUE:**
    a. Unique columns are good candidates for indexing. Unique indexes enforce uniqueness and also speed up lookups of individual rows.

Indexing Results

1. **On student table**

Query = SELECT * FROM Student WHERE Student_ID<2000;
Time required before indexing : 0.0039s

Indexing => CREATE INDEX student_idx ON Student(Student_ID,Department);
Time required after indexing : 0.0013s

2. **Company**

SELECT * FROM Company WHERE Company_Field="Tech";
Time required before indexing : 0.00062s

CREATE INDEX company_indx ON Company(Company_ID,Company_Field);
Time required after indexing : 0.00048s

3. **Opportunity**

SELECT * FROM Opportunity WHERE Salary>=65000;
Time required before indexing: 0.0049s

CREATE INDEX opp_indx ON Opportunity(Opp_ID,Salary);
Time required after indexing: 0.0012s

4. **Placement**

SELECT * FROM Placement WHERE Company_Name="IBM";
Time required before indexing: 0.0076s

CREATE INDEX plac_indx ON Placement(Placement_Id,Company_Name);
Time required after indexing: 0.0010s

5. **Academic_Info**

SELECT * FROM Academic_Info WHERE CPI>=8.0;
Time required before indexing: 0.0056s

CREATE INDEX acad_indx ON Academic_Info(Academic_Info_ID,CPI);
Time required after indexing: 0.0019s

6. **Person Of Contact**

SELECT * FROM Person_of_Contact WHERE Employee_Designation="Director";
Time required before indexing: 0.00083s

CREATE INDEX poc_indx ON Person_of_Contact(Poc_Email_ID);
Time required after indexing: 0.00063s

User-Defined Data Type:

The MySQL implementation lacks support for user-defined data types. Nevertheless, we've utilized ENUM to designate certain attributes that can only accept specific values.

1. Gender ENUM('Male', 'Female', 'Other') NOT NULL : (in table ‘student’)

The gender column in the student table is defined as an ENUM data type that restricts the values allowed to only 'Male', 'Female', or 'Other'. This provides domain constraints to ensure standardized gender data compared to using an open VARCHAR or TEXT type.

The ENUM specifies that any record inserted or updated in the student table can only have a gender value of 'Male', 'Female', or 'Other'. No other gender values can be provided.

In addition, the NOT NULL constraint requires that a gender value must be provided and cannot be left as NULL. This ensures that every student record has an associated gender value defined as one of the permitted ENUM values.

Overall, the ENUM data type and NOT NULL constraint standardize the gender data while also making sure it is validated against a predefined set of allowed string values. This supports data integrity and uniformity.

2. Organization_Type ENUM('Public', 'Private', 'Non-Profit') NOT NULL : (in table ‘company’’)

The Organization_Type column in the company table is defined as an ENUM data type that restricts the values allowed to only 'Public', 'Private', or 'Non-Profit'. This standardizes the types of organizations that can be specified in this system.

The ENUM enforces that company records can have an Organization_Type value of either 'Public', 'Private', or 'Non-Profit'. No other organizational type values are allowed through this data definition.

Table Extensions

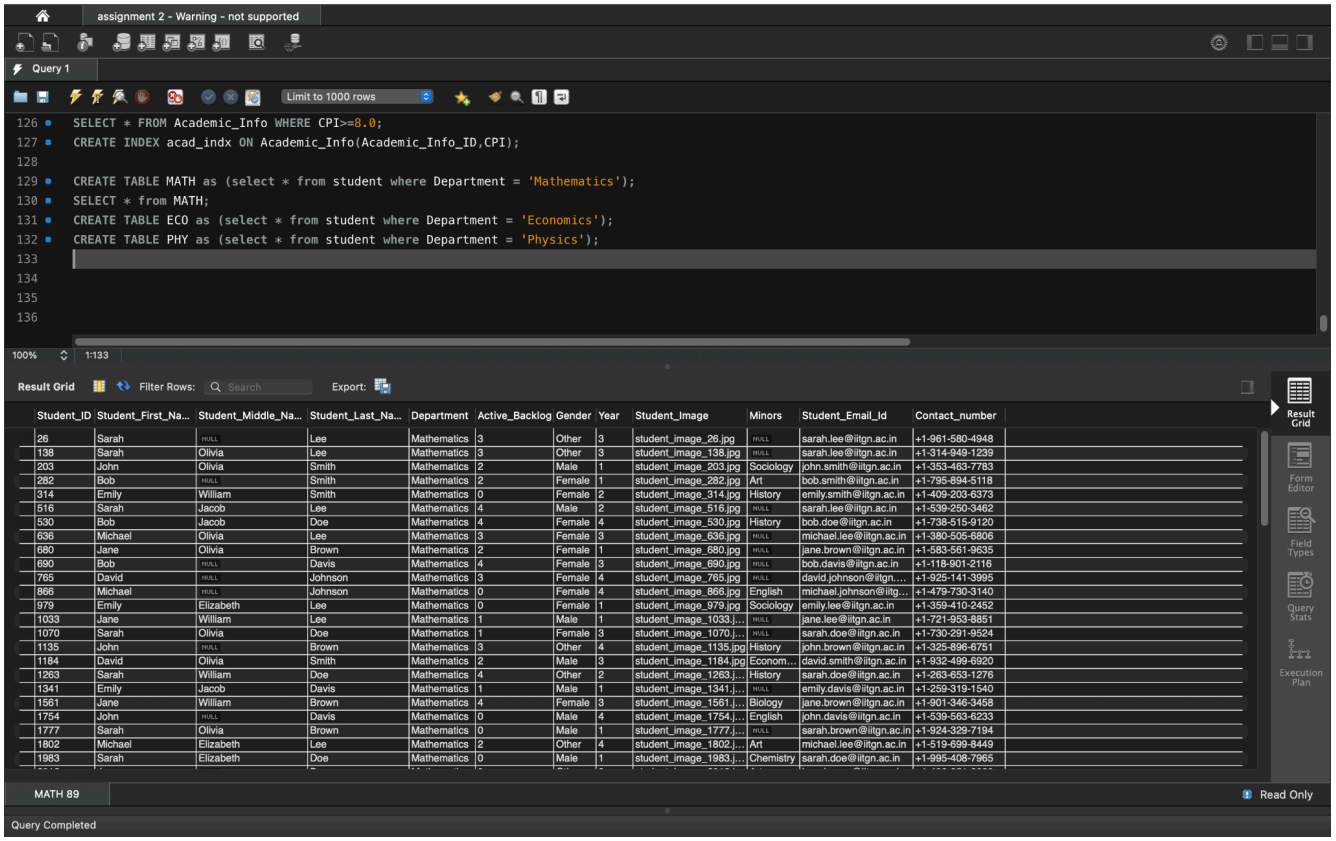
Key reasons why table extensions can be useful in database design-

1. **Support Evolving Requirements**
    a. Table extensions allow the database model to gracefully evolve over time as new requirements and attributes emerge.
    b. Instead of continually altering existing tables which can be disruptive, new columns can simply be added to extensions.
    c. The utilization of table extensions for the serves to enhance the database's performance, uphold data consistency, and facilitate efficient organization and management of data.
2. **Avoid Massive Wide Tables**
    a. As requirements change, tables often keep having more and more columns added.
    b. This leads to very wide, cumbersome tables that get difficult to maintain. Extensions let you keep the core table lean.

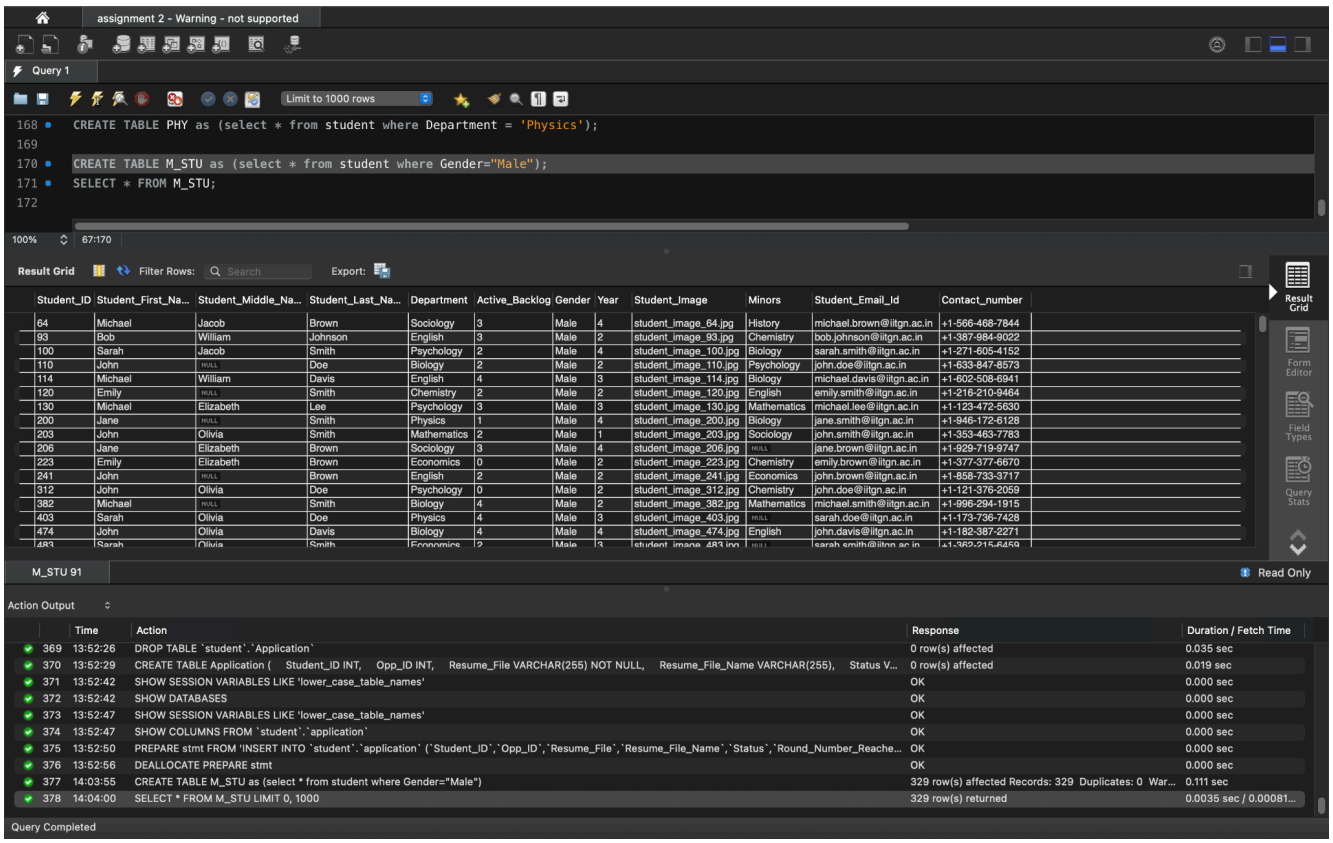
Let’s say once we build the database, but then we feel the following requirements:

1. To separate the data of students on the basis of their department, we use table extension

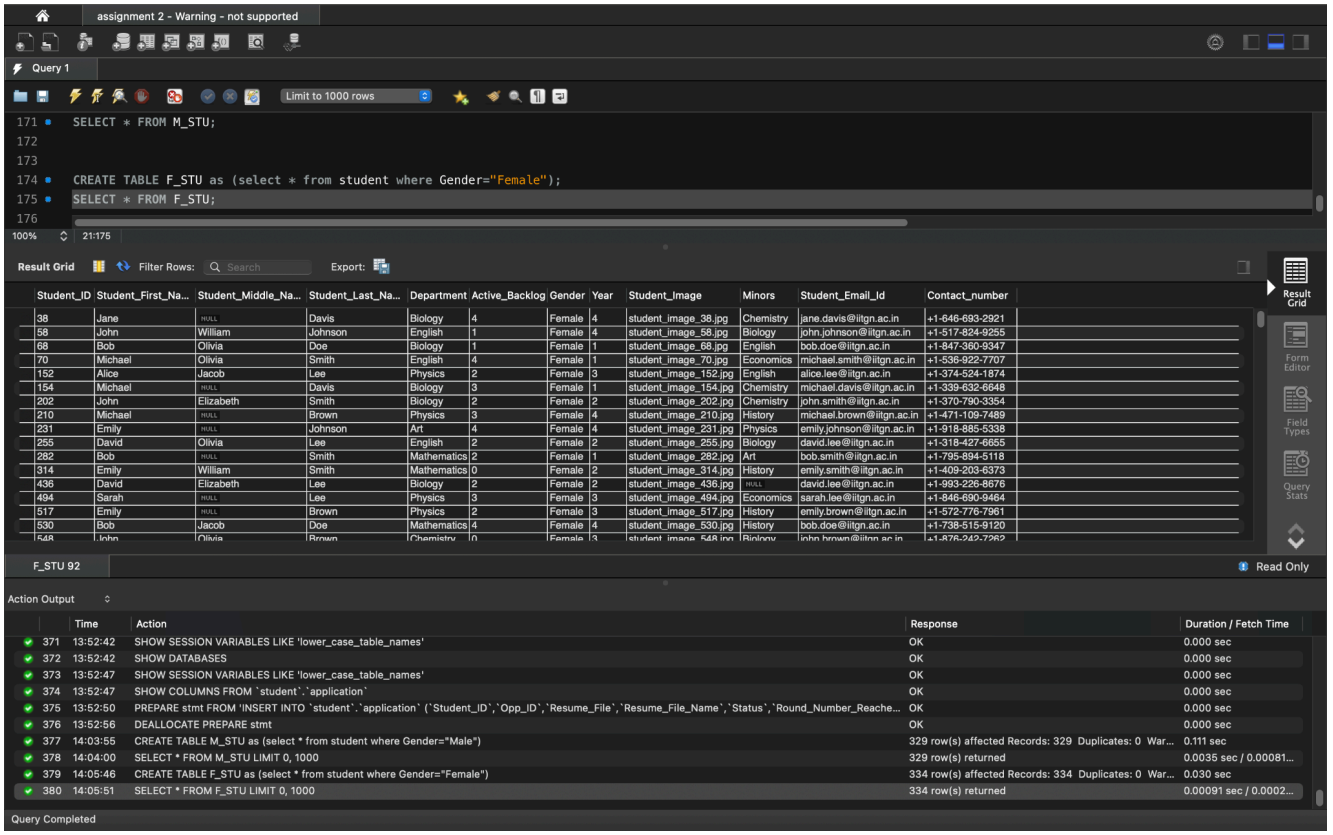
CREATE TABLE MATH as (select * from student where Department = 'Mathematics');
CREATE TABLE ECO as (select * from student where Department = 'Economics');
CREATE TABLE PHY as (select * from student where Department = 'Physics');



2. Also we can create separate tables from student table on the basis of gender
    a. Male Students



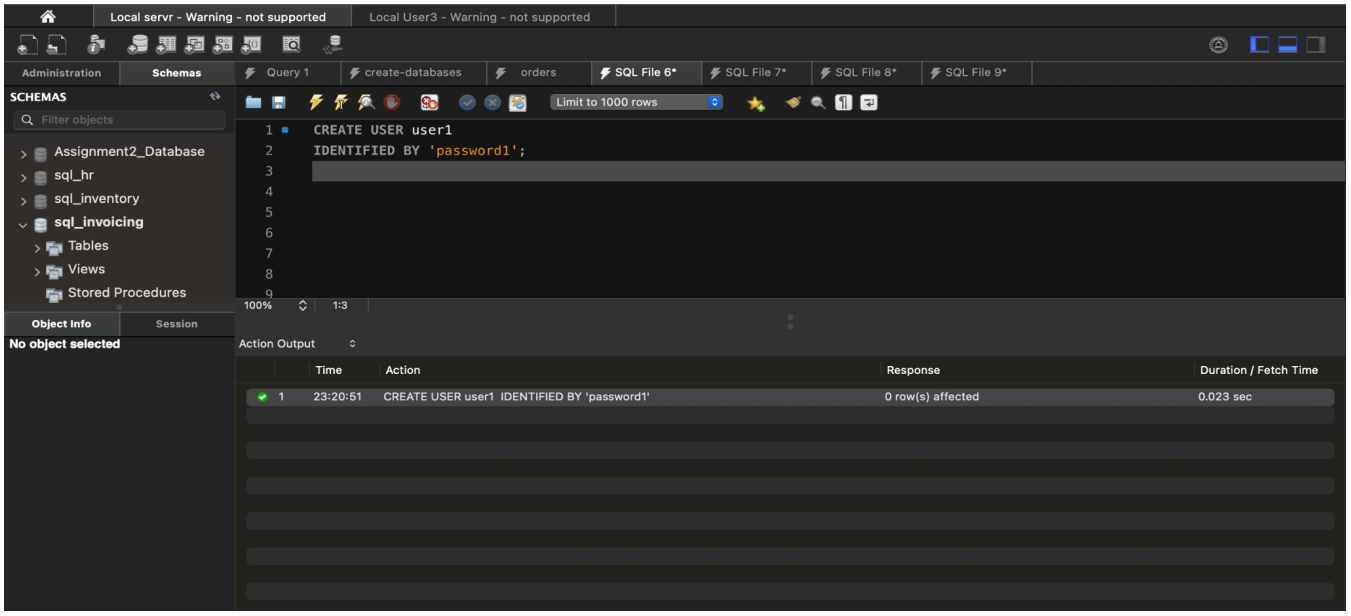
b. Female Students



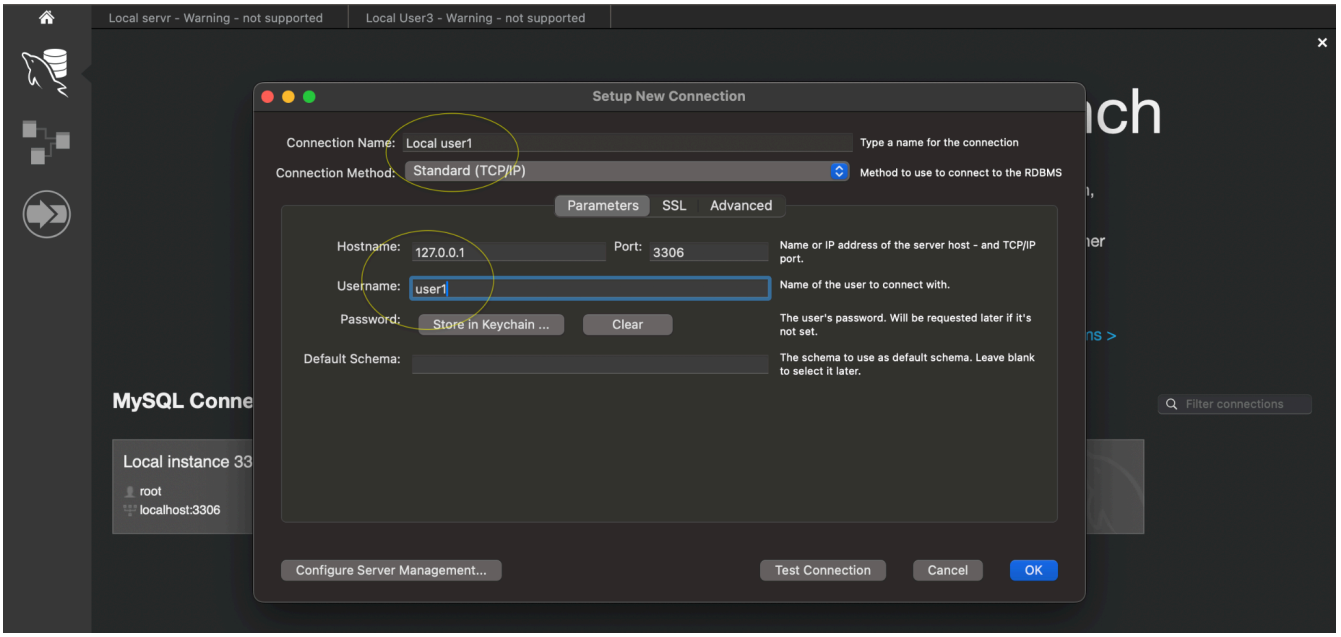
13.2 Responsibility of G2:

1. *Create a user named "user1" with the password "password1"*

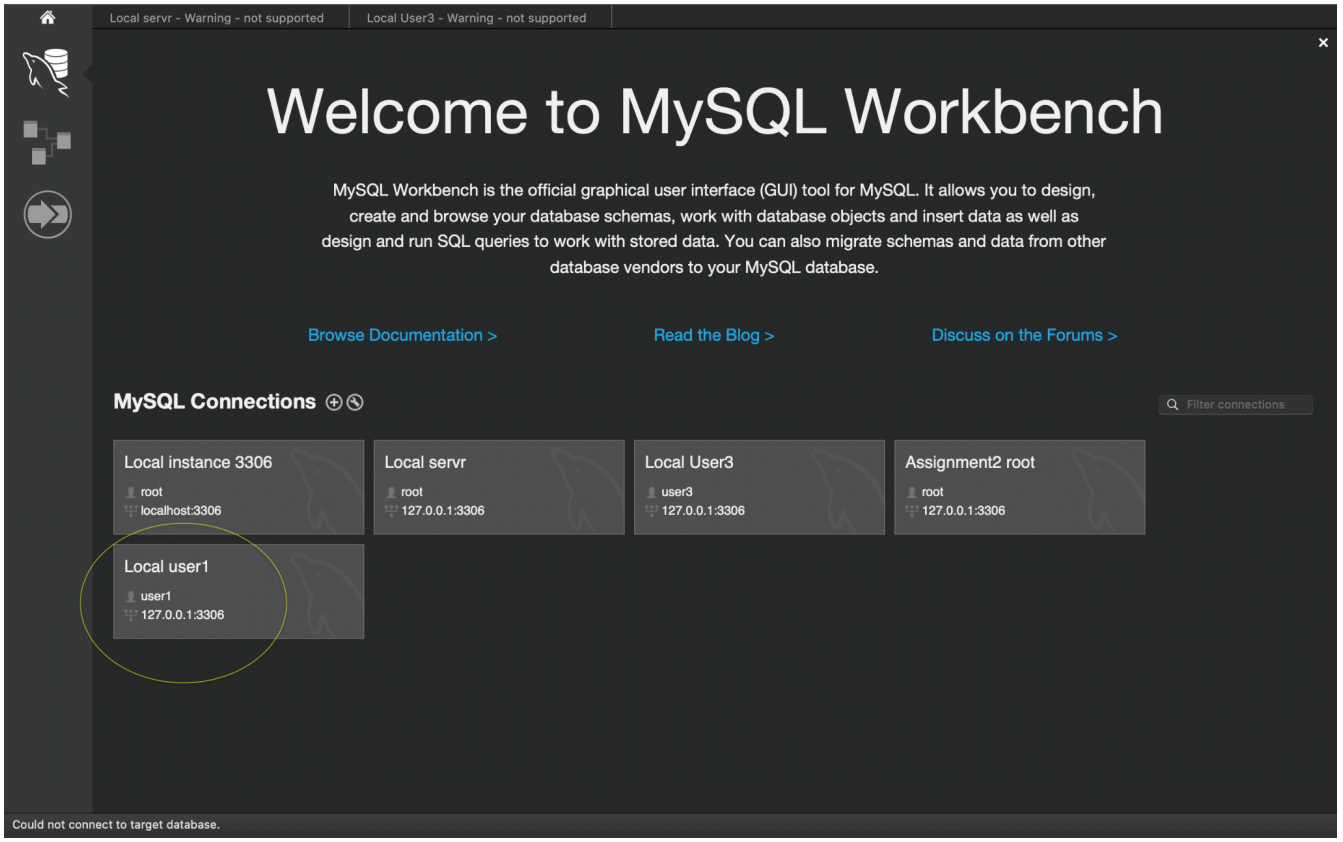
To create a new user in the database, we access the database using the root and create a new user.



This creates a new user “user1”, but we have to activate a connection.



And with this, we have created a new user, which we can access with our password.



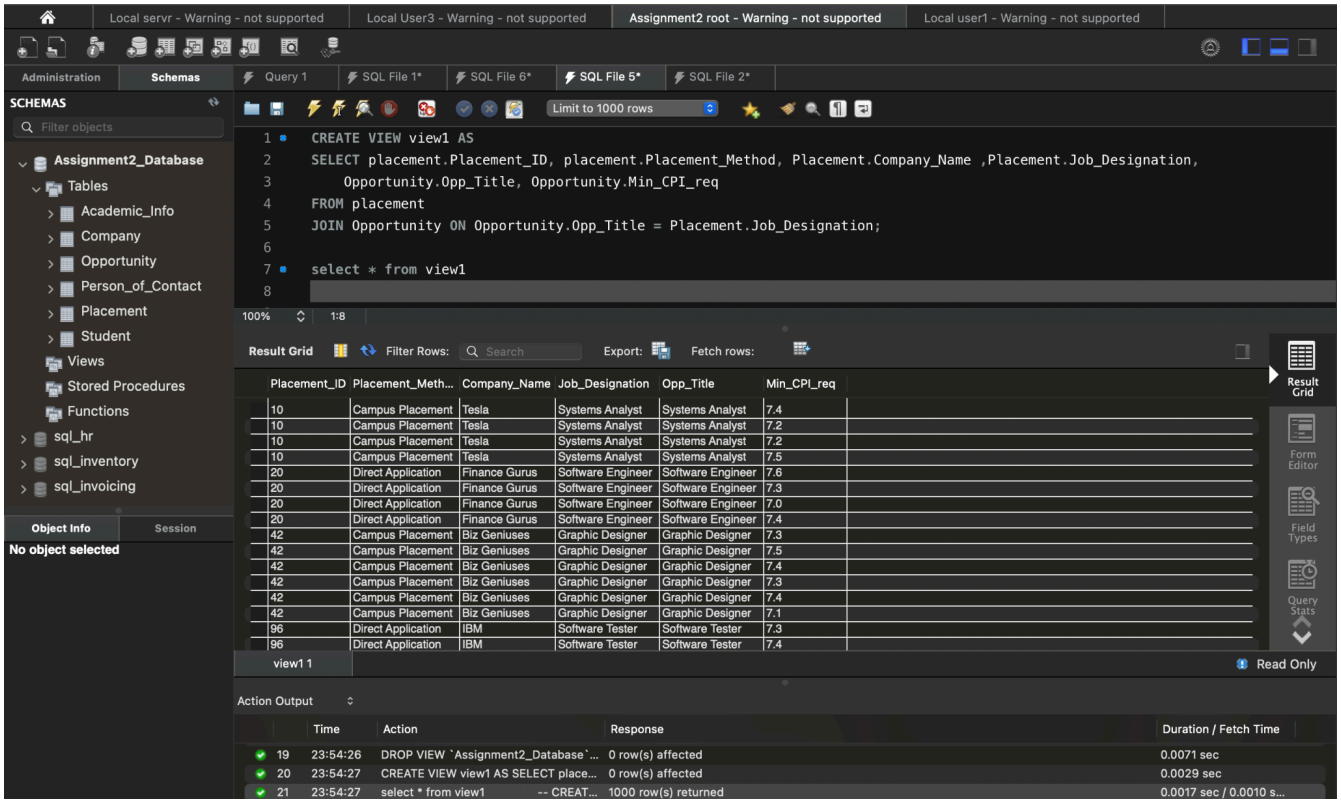
2. *Create Views on any of the two tables formed by G1 as view1 and view2. And make sure that views contain columns from at least two tables and one additional column with the user-defined data type*

VIEW1

This view is created by using the placement and Opportunity entity sets. This view does not have an enum value, as the new one have and this one has interesting attributes in it.

Here, we have provided the code and how the View looks like when selected.
From the Placement Entity, it has:- Placement_ID, Placement_Method, Company_Name, Job_Designation
From the Opportunity Entity it has:- Opp_Title, Min_CPI_req

It is a joint view, where we select those columns that have same Opportunity Title, and Placement Job Description and add them.



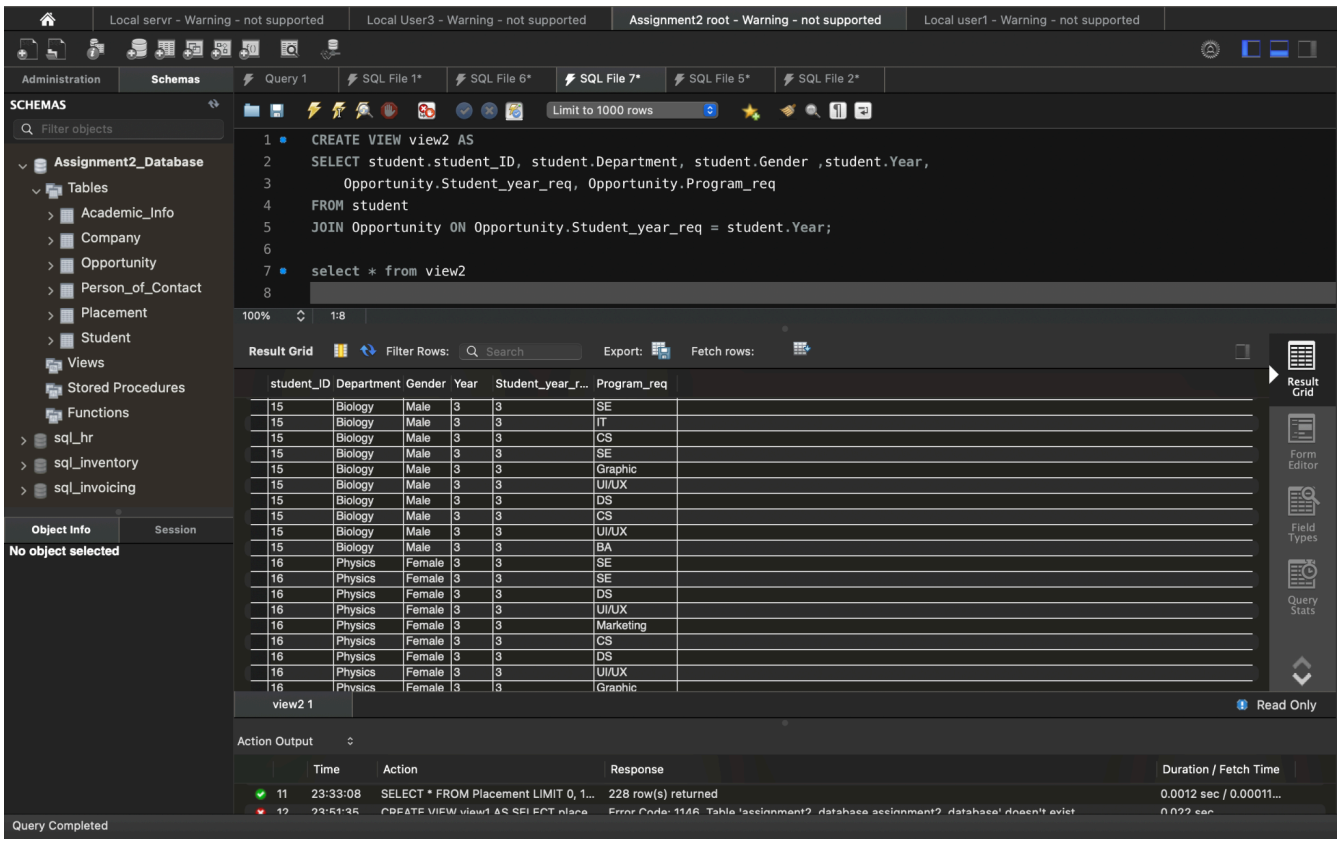
VIEW2

This view is created by using the Student and Opportunity entity sets.
This view does has an user-specified data type (given by enum value), It is Student.Gender, which can be Male, Female, Other.

Here, we have provided the code and what the View looks like when selected.

From the Student Entity, it has:- student.student_ID, student.Department, student.Gender ,student.Year,
From the Opportunity Entity it has:- Opportunity.Student_year_req, Opportunity.Program_req

It is a joint view, where we select those columns that have same Opportunity Student_year_req, and Student year and add them.



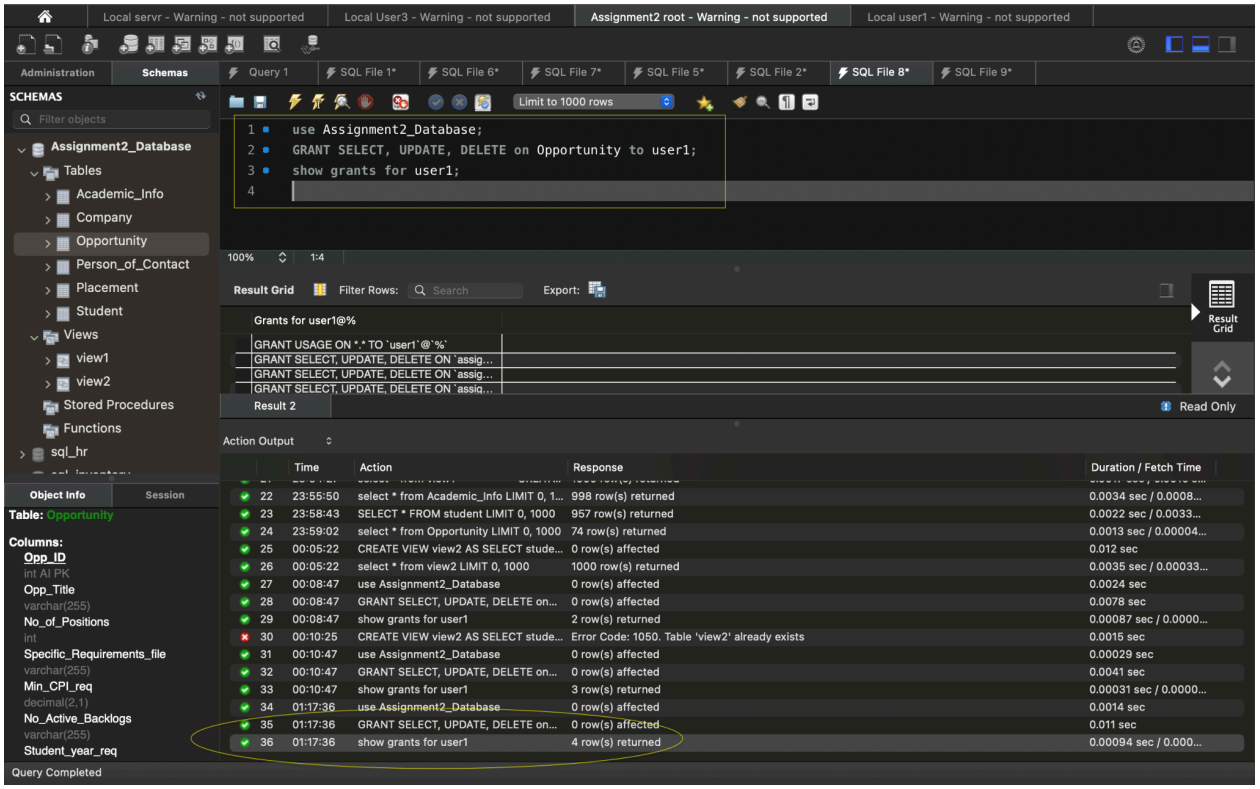
---

3. *Grant "user1" the following permissions on "table1": SELECT, UPDATE, DELETE*

To grant the new user “user1” permissions, we will go back to the root connection, and from there, update and give the permissions for the new user

Table1

Here, we will be giving these permissions to the User1 for the “table1”, which we have selected to be the opportunity table.

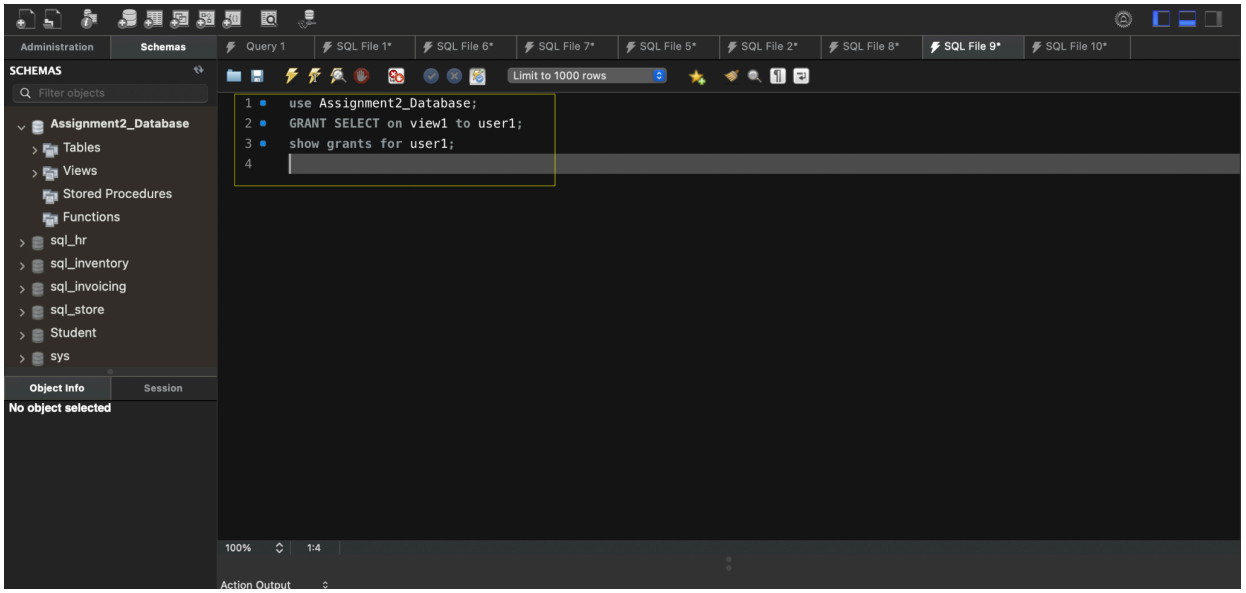


---

4. Grant "user1" the following permissions on "view1": SELECT

View1

Here, we will be giving these permissions to the User1 for the “view1”.

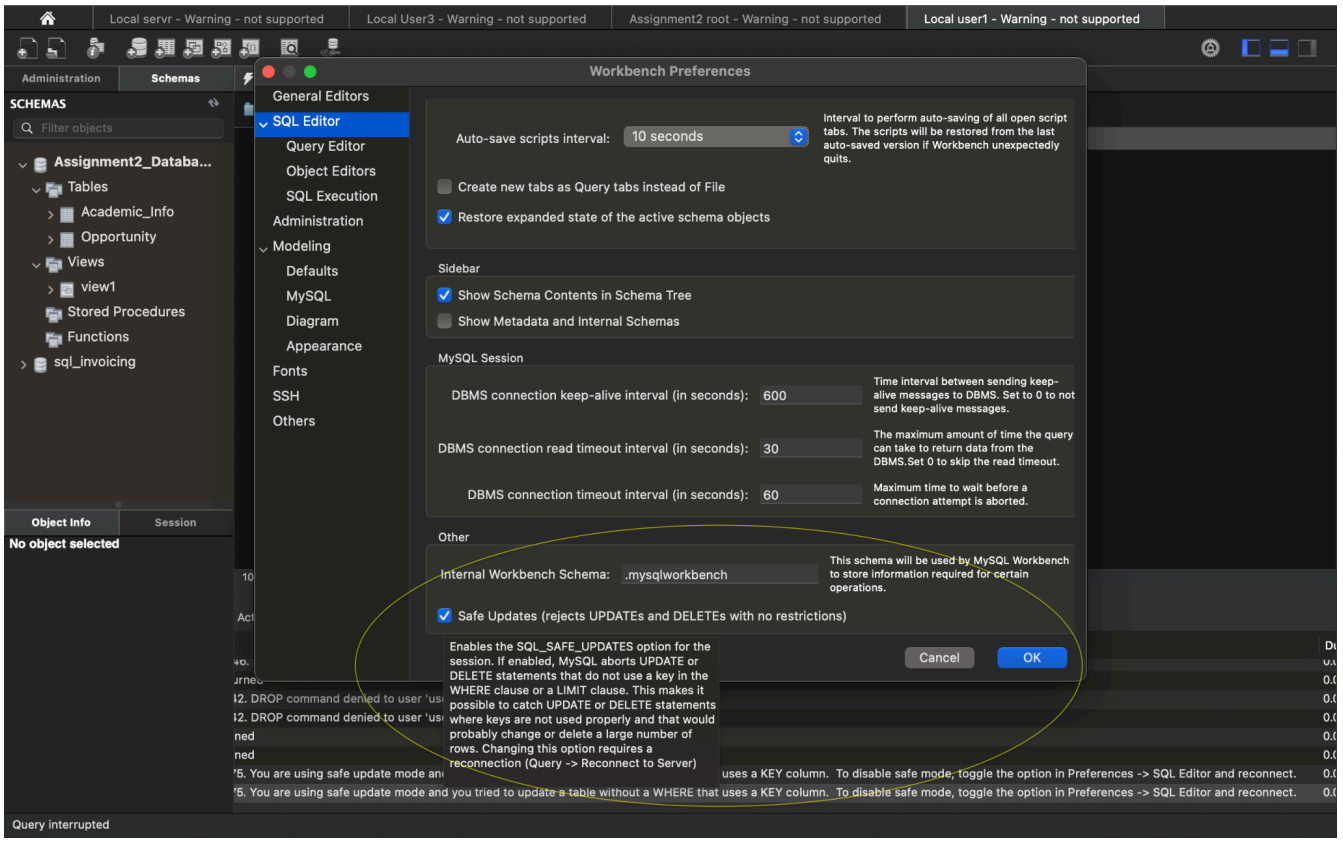


5. *Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" and report your findings.*

TABLE1

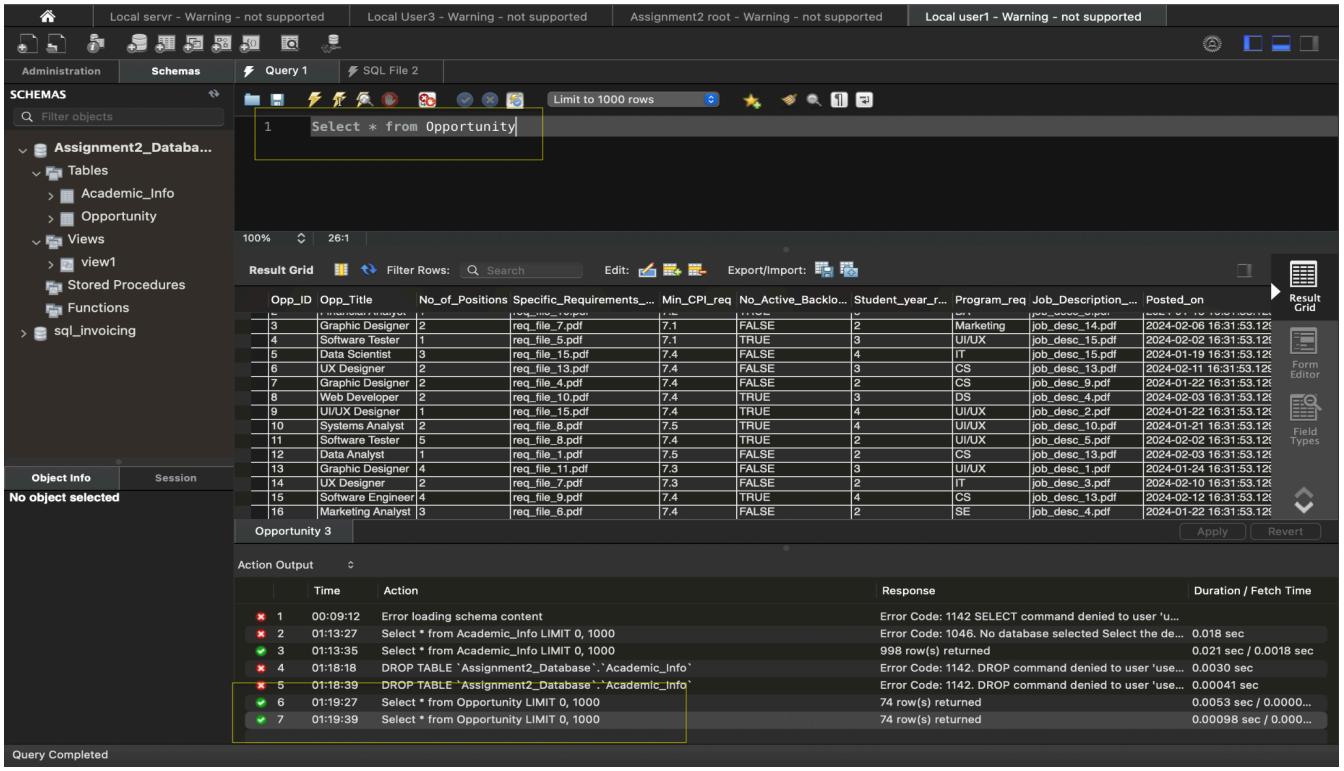
Here, we will perform some operations on the DATA in table, but before that, We will change the setting of the safe update mode, where the Database disable editing without using a where clause.

This can be done by changing the preferences and disselecting safe update mode.



Now, we can see the Operations we perform:-

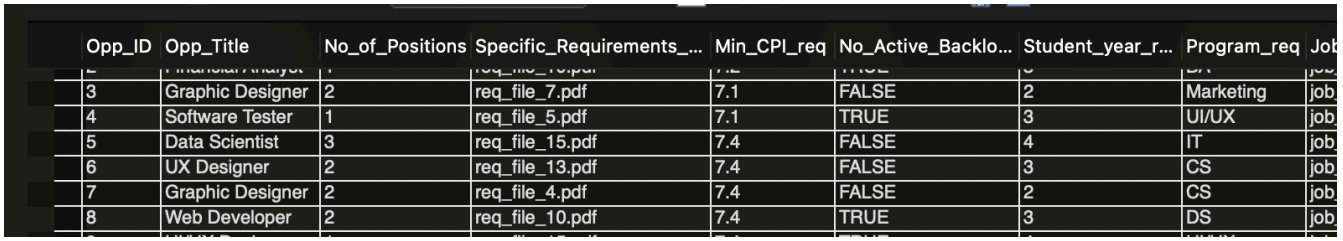
SELECT



UPDATE

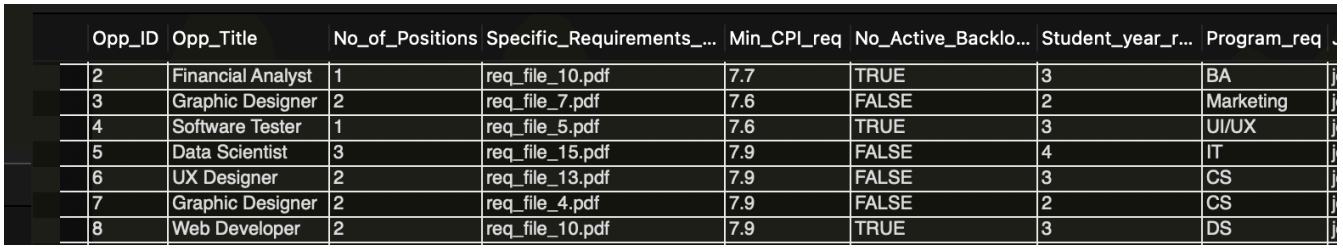
Here, we will update the value of the CPI to +0.5 than before, as one can clearly in the image.

BEFORE UPDATION



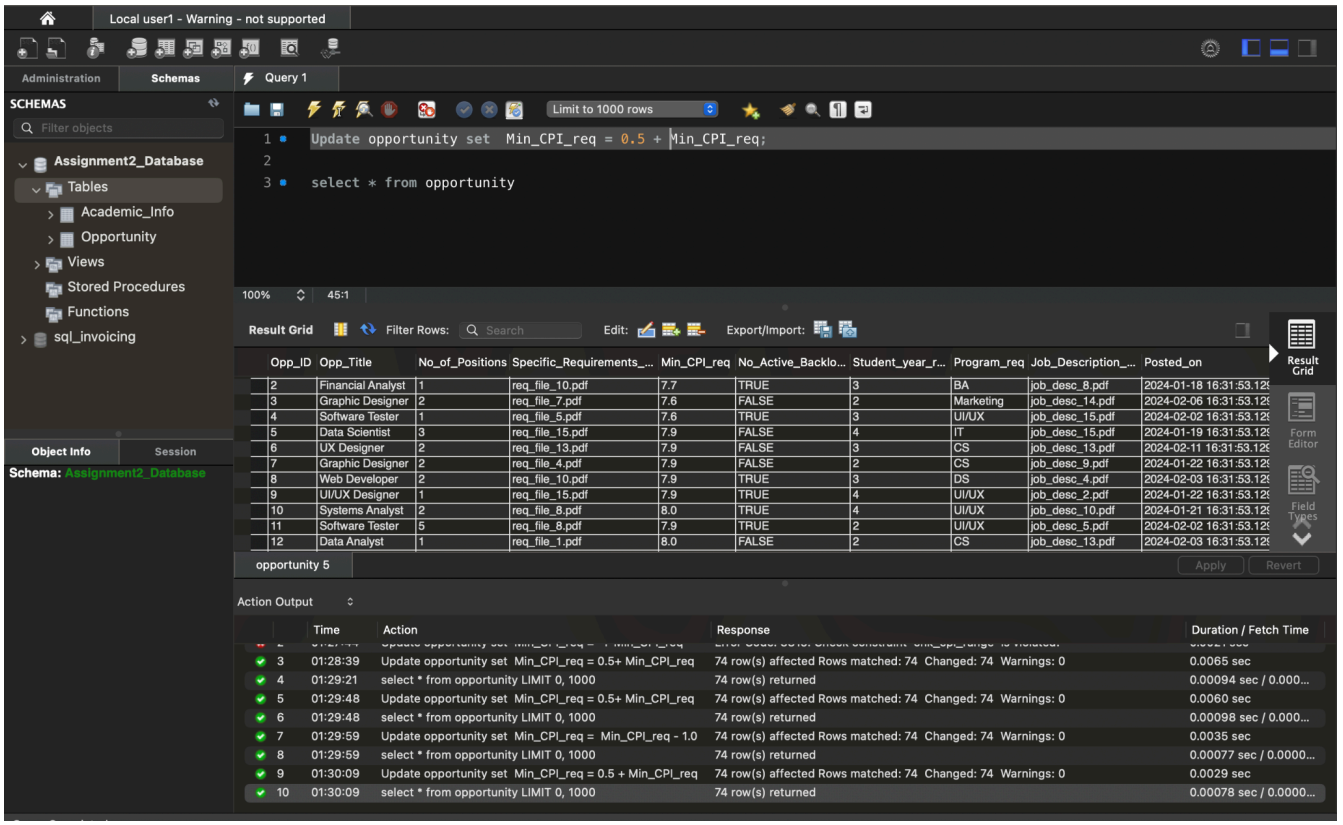
Opp_ID	Opp_Title	No_of_Positions	Specific_Requirements_...	Min_CPI_req	No_Active_Backlo...	Student_year_r...	Program_req	Job
3	Graphic Designer	2	req_file_7.pdf	7.1	FALSE	2	Marketing	job
4	Software Tester	1	req_file_5.pdf	7.1	TRUE	3	UI/UX	job
5	Data Scientist	3	req_file_15.pdf	7.4	FALSE	4	IT	job
6	UX Designer	2	req_file_13.pdf	7.4	FALSE	3	CS	job
7	Graphic Designer	2	req_file_4.pdf	7.4	FALSE	2	CS	job
8	Web Developer	2	req_file_10.pdf	7.4	TRUE	3	DS	job

AFTER UPDATION



Opp_ID	Opp_Title	No_of_Positions	Specific_Requirements_...	Min_CPI_req	No_Active_Backlo...	Student_year_r...	Program_req
2	Financial Analyst	1	req_file_10.pdf	7.7	TRUE	3	BA
3	Graphic Designer	2	req_file_7.pdf	7.6	FALSE	2	Marketing
4	Software Tester	1	req_file_5.pdf	7.6	TRUE	3	UI/UX
5	Data Scientist	3	req_file_15.pdf	7.9	FALSE	4	IT
6	UX Designer	2	req_file_13.pdf	7.9	FALSE	3	CS
7	Graphic Designer	2	req_file_4.pdf	7.9	FALSE	2	CS
8	Web Developer	2	req_file_10.pdf	7.9	TRUE	3	DS

The code and screen for the updation process



DELETION

In deletion, we are deleting the Opportunities which have their job description in "job_desc_15.pdf". The jobs with opp_id 4 and 5 have been deleted.

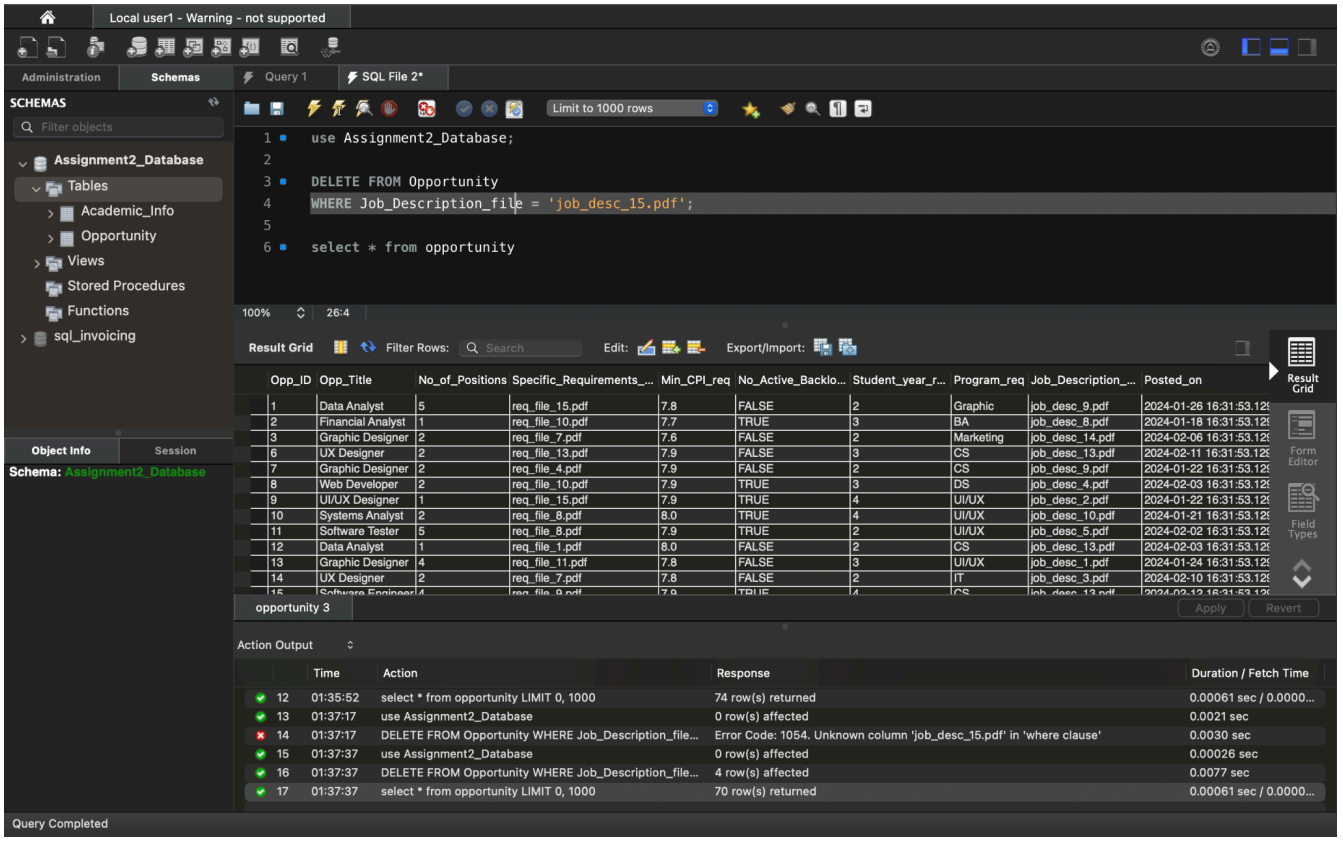
BEFORE DELETION

Opp_ID	Opp_Title	No_of_Positions	Specific_Requirements_...	Min_CPI_req	No_Active_Backlo...	Student_year_r...	Program_req	Job_Description_...
1	Data Analyst	5	req_file_15.pdf	7.8	FALSE	2	Graphic	job_desc_9.pdf
2	Financial Analyst	1	req_file_10.pdf	7.7	TRUE	3	BA	job_desc_8.pdf
3	Graphic Designer	2	req_file_7.pdf	7.6	FALSE	2	Marketing	job_desc_14.pdf
4	Software Tester	1	req_file_5.pdf	7.6	TRUE	3	UI/UX	job_desc_15.pdf
5	Data Scientist	3	req_file_15.pdf	7.9	FALSE	4	IT	job_desc_15.pdf
6	UX Designer	2	req_file_13.pdf	7.9	FALSE	3	CS	job_desc_13.pdf
7	Graphic Designer	2	req_file_4.pdf	7.9	FALSE	2	CS	job_desc_9.pdf

AFTER DELETION

Opp_ID	Opp_Title	No_of_Positions	Specific_Requirements_...	Min_CPI_req	No_Active_Backlo...	Student_year_r...	Program_req	Job_Description_
1	Data Analyst	5	req_file_15.pdf	7.8	FALSE	2	Graphic	job_desc_9.pdf
2	Financial Analyst	1	req_file_10.pdf	7.7	TRUE	3	BA	job_desc_8.pdf
3	Graphic Designer	2	req_file_7.pdf	7.6	FALSE	2	Marketing	job_desc_14.pdf
6	UX Designer	2	req_file_13.pdf	7.9	FALSE	3	CS	job_desc_13.pdf
7	Graphic Designer	2	req_file_4.pdf	7.9	FALSE	2	CS	job_desc_9.pdf
8	Web Developer	2	req_file_10.pdf	7.9	TRUE	3	DS	job_desc_4.pdf

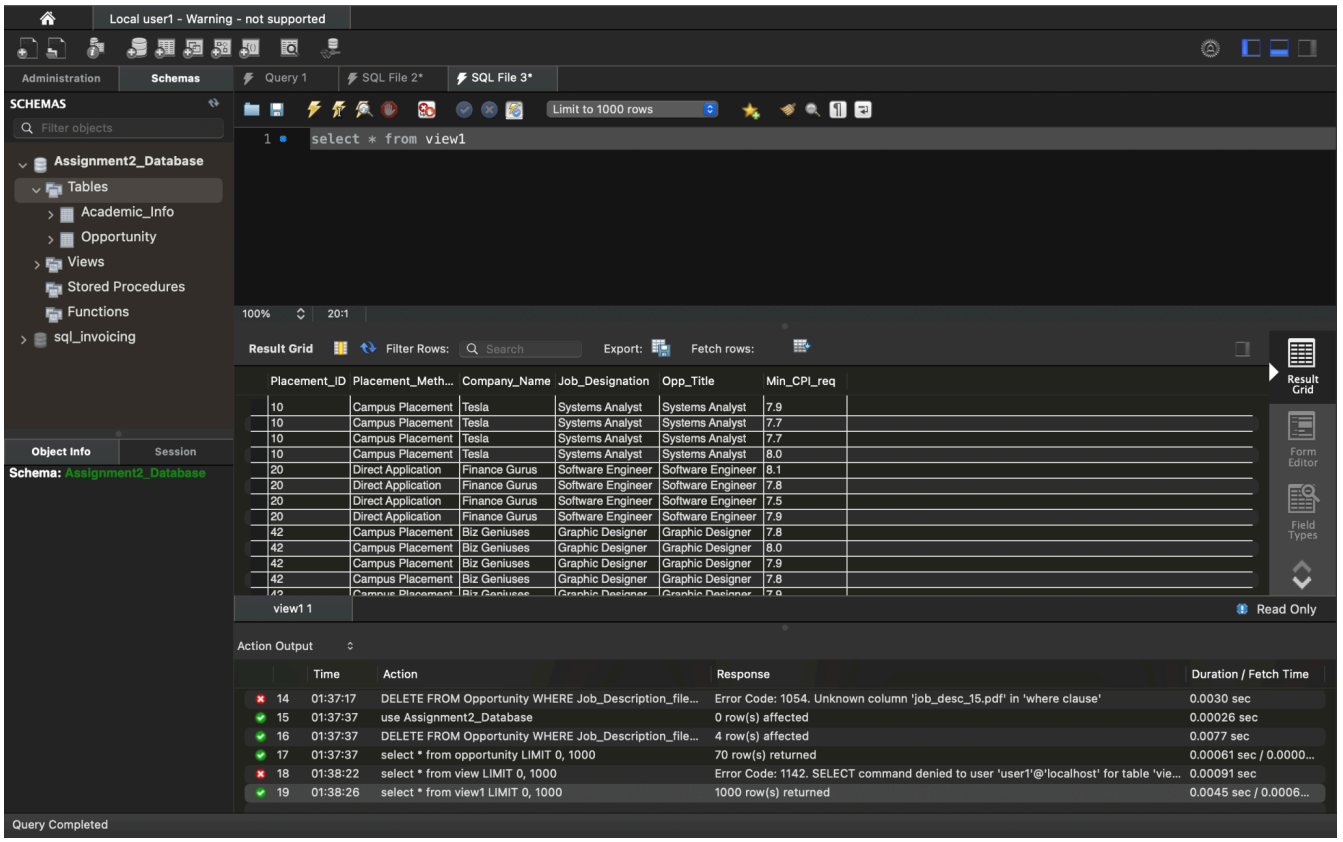
Here is the full code and screen for deletion process.



VIEW1

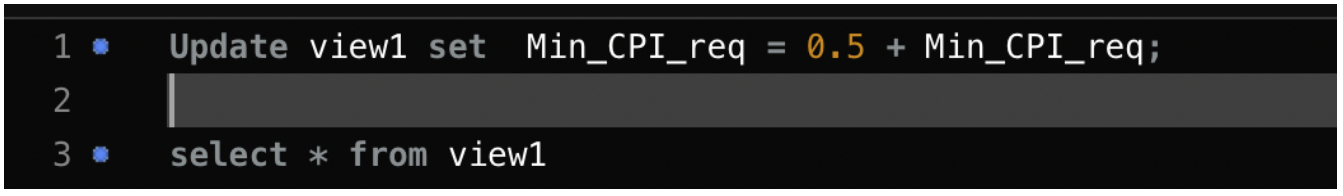
Here, we will perform the same operation, but on view1.

SELECT

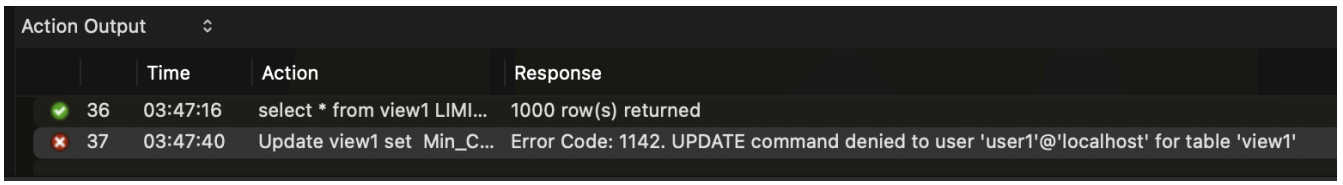


UPDATE

Here, we will update the value of the CPI to +0.5 than before, as one can see in the image,



The updation fails, as we did not provide the user with the necessary permissions.



DELETION

Here, we will update the value of the CPI to +0.5 than before, as one can see in the image,

```
use Assignment2_Database;

DELETE FROM view1
WHERE Placement_Method = 'Direct Application';

select * from view1
```

The updation fails, as we did not provide the user with the necessary permissions.

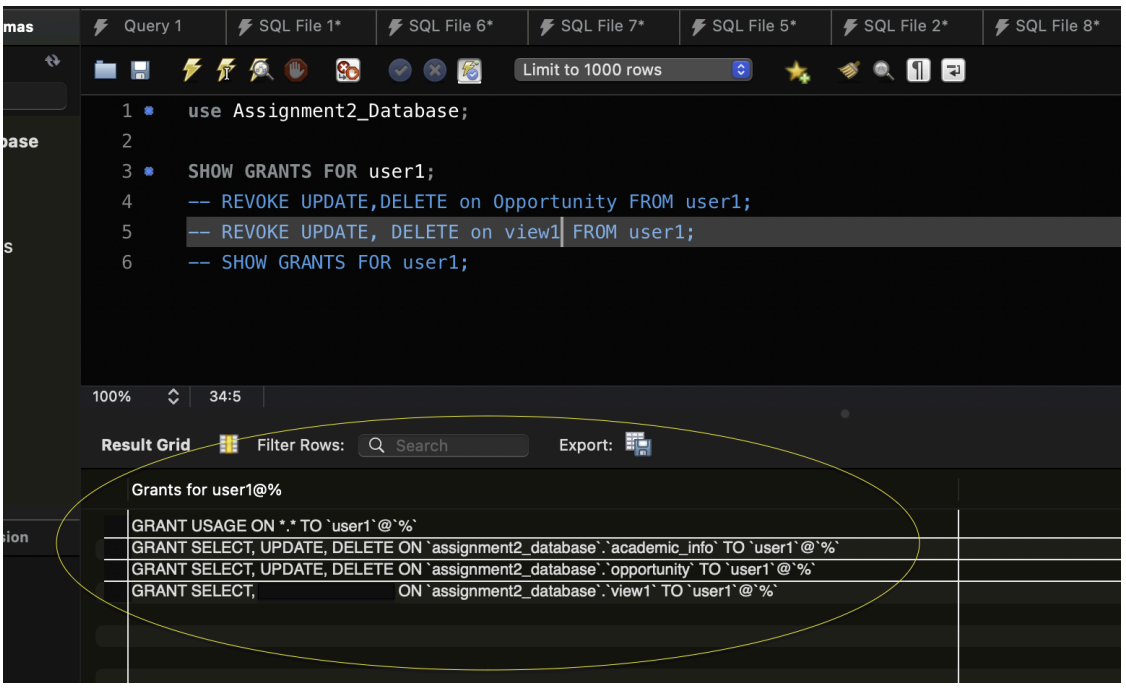
Action Output

	Time	Action	Response
38	03:50:16	use Assignment2_Datab...	0 row(s) affected
39	03:50:16	DELETE FROM view1 WH...	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'

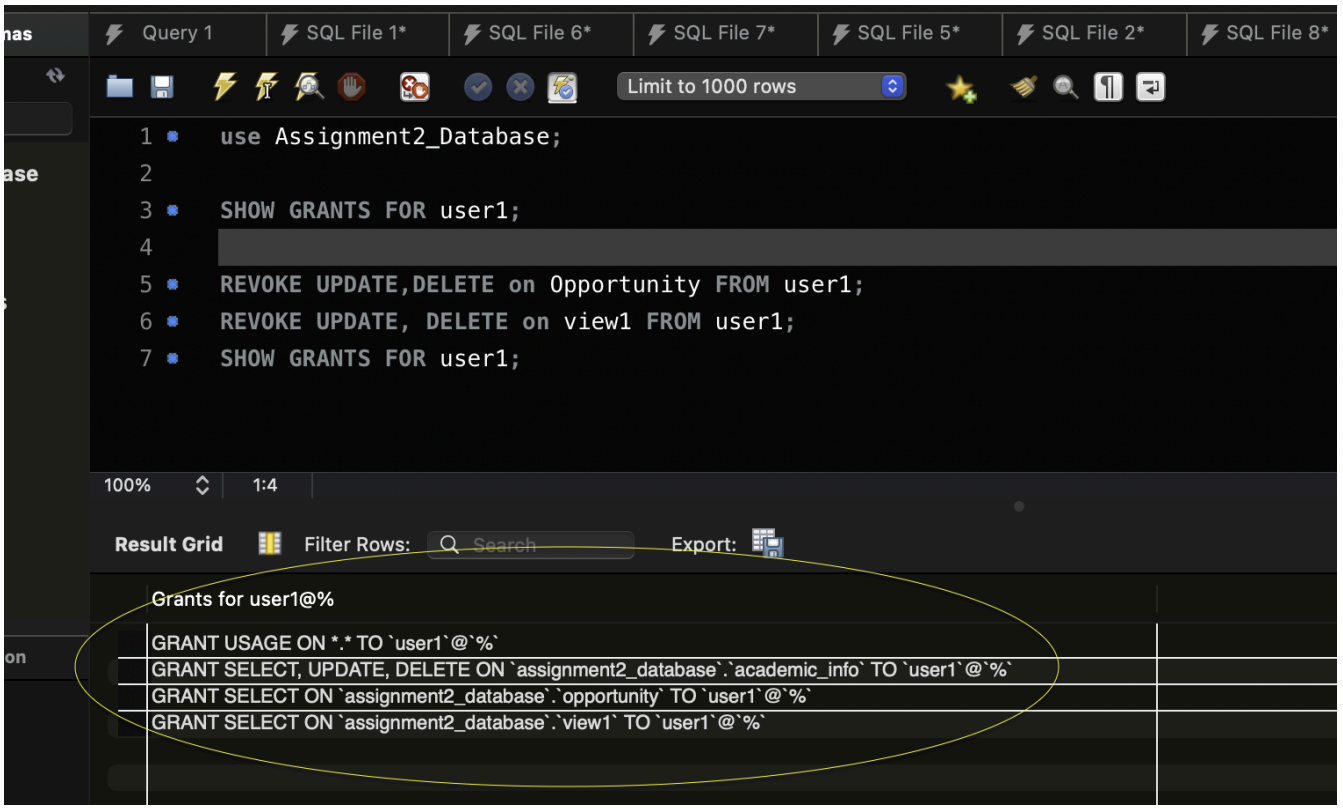
---

6. Revoke the UPDATE and DELETE permissions on "table1" for "user1" and report your findings.

BEFORE REVOKING THE PERMISSIONS:-



AFTER REVOKING PERMISSIONS



(NOTE: The permission to Academic_Info table, was only put for testing purposes, please ignore it)
Also, to remove the new user “user1” ’s permissions, we will go back to the root connection, and from there, revoke the permissions of the new user

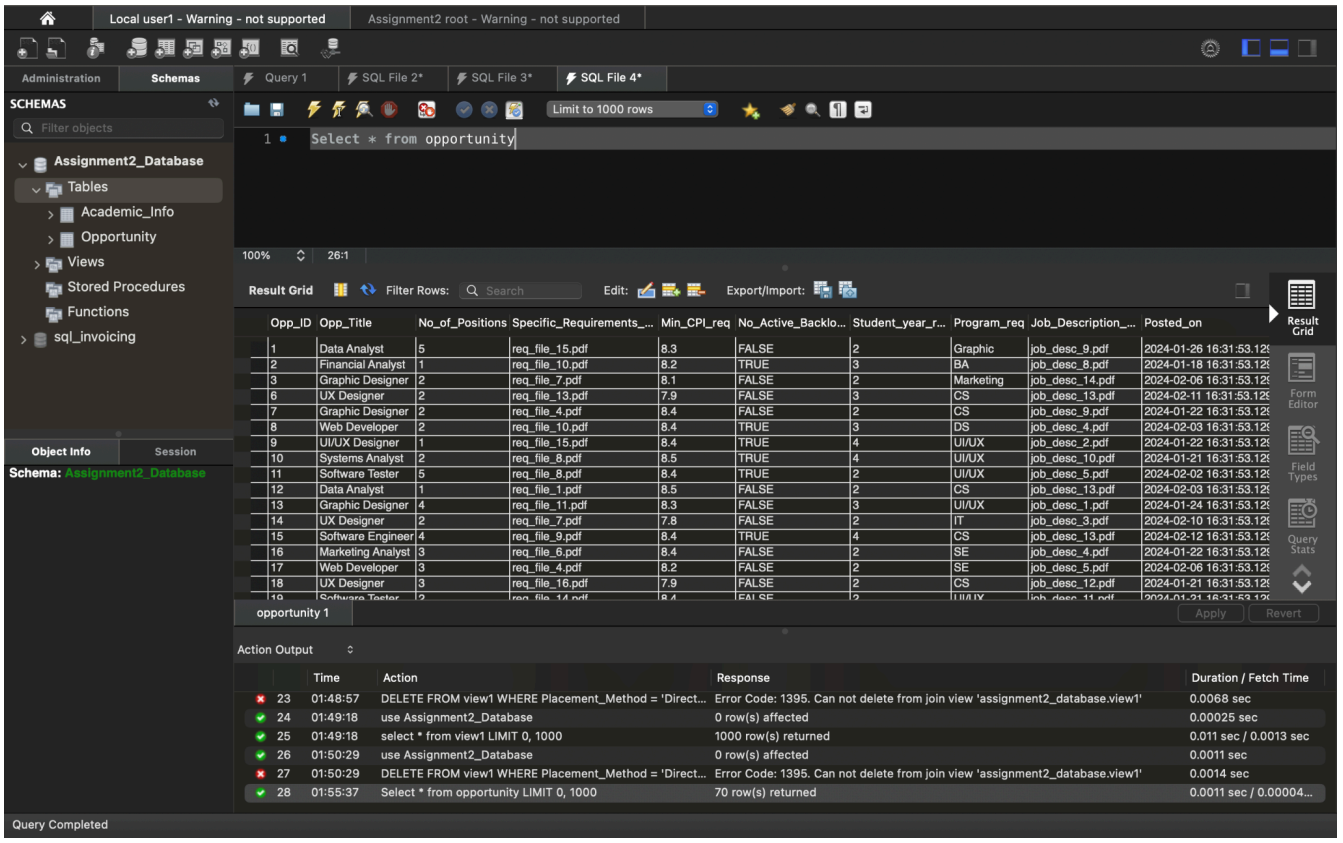
---

7. Revoke the UPDATE and DELETE permissions on "table1" for "user1" and report your findings.

As expected, we are only able to perform SELECT, as that is the only permission remaining with the new user.

TABLE1

SELECT



UPDATE

```
1  Update opportunity set  Min_CPI_req = 0.5 + Min_CPI_req;
2
3  select * from opportunity
```

29 01:56:17 Update opportunity set... Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'opportunity'

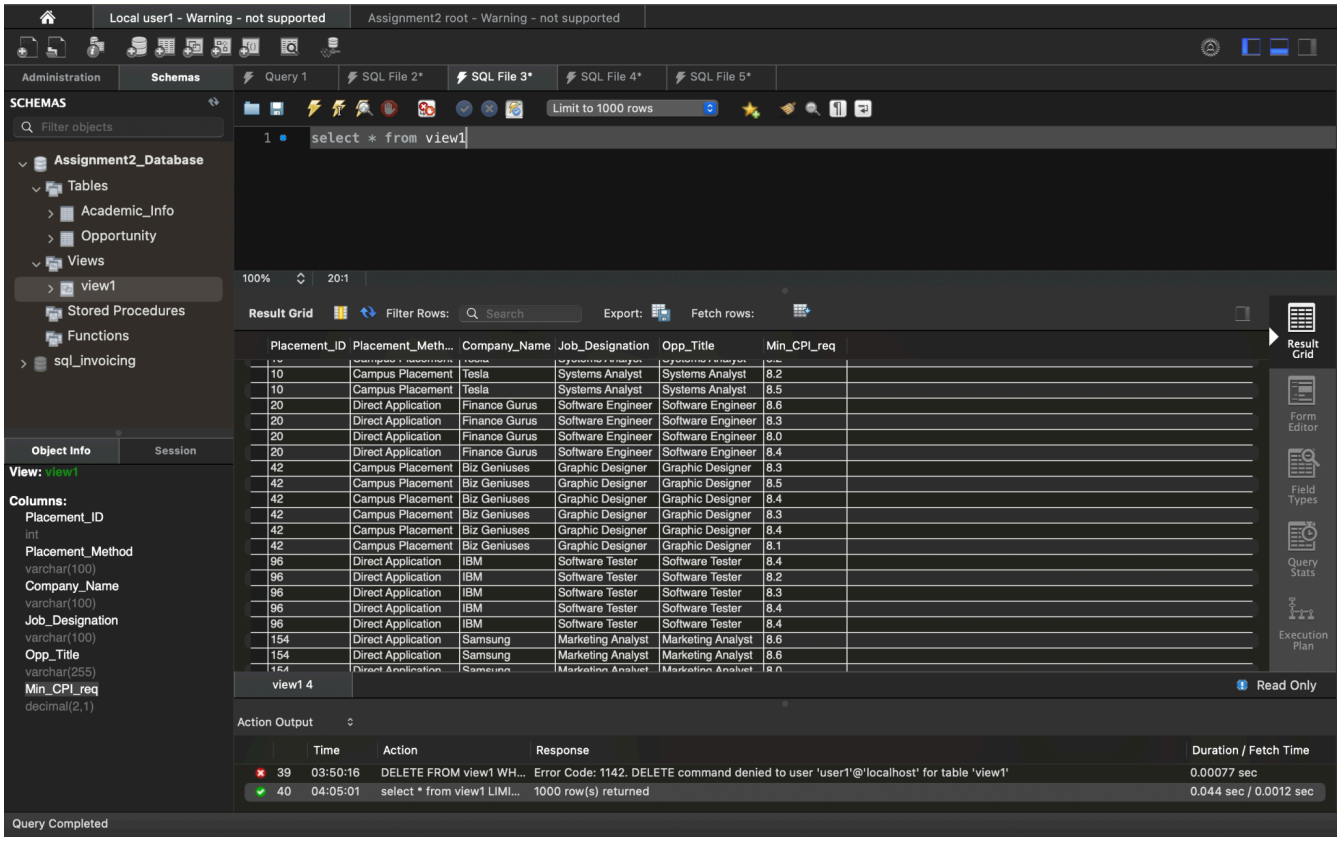
DELETION

```
1  use Assignment2_Database;
2
3  DELETE FROM Opportunity
4  WHERE Job_Description_file = 'job_desc_15.pdf';
5
6  select * from opportunity
```

```
34  01:57:56  DELETE FROM Opportun...  Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'opportunity'
```

VIEW1

SELECT



UPDATION

```
1    Update view1 set  Min_CPI_req = 0.5 + Min_CPI_req;
2
```

```
30  01:57:03  Update view1 set  Min_C...  Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'
```

DELETION

```
1    use Assignment2_Database;
2
3    DELETE FROM view1
4    WHERE Placement_Method = 'Direct Application';
5
6    select * from view1
```

```
32  01:57:22  DELETE FROM view1 WH...  Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'
```

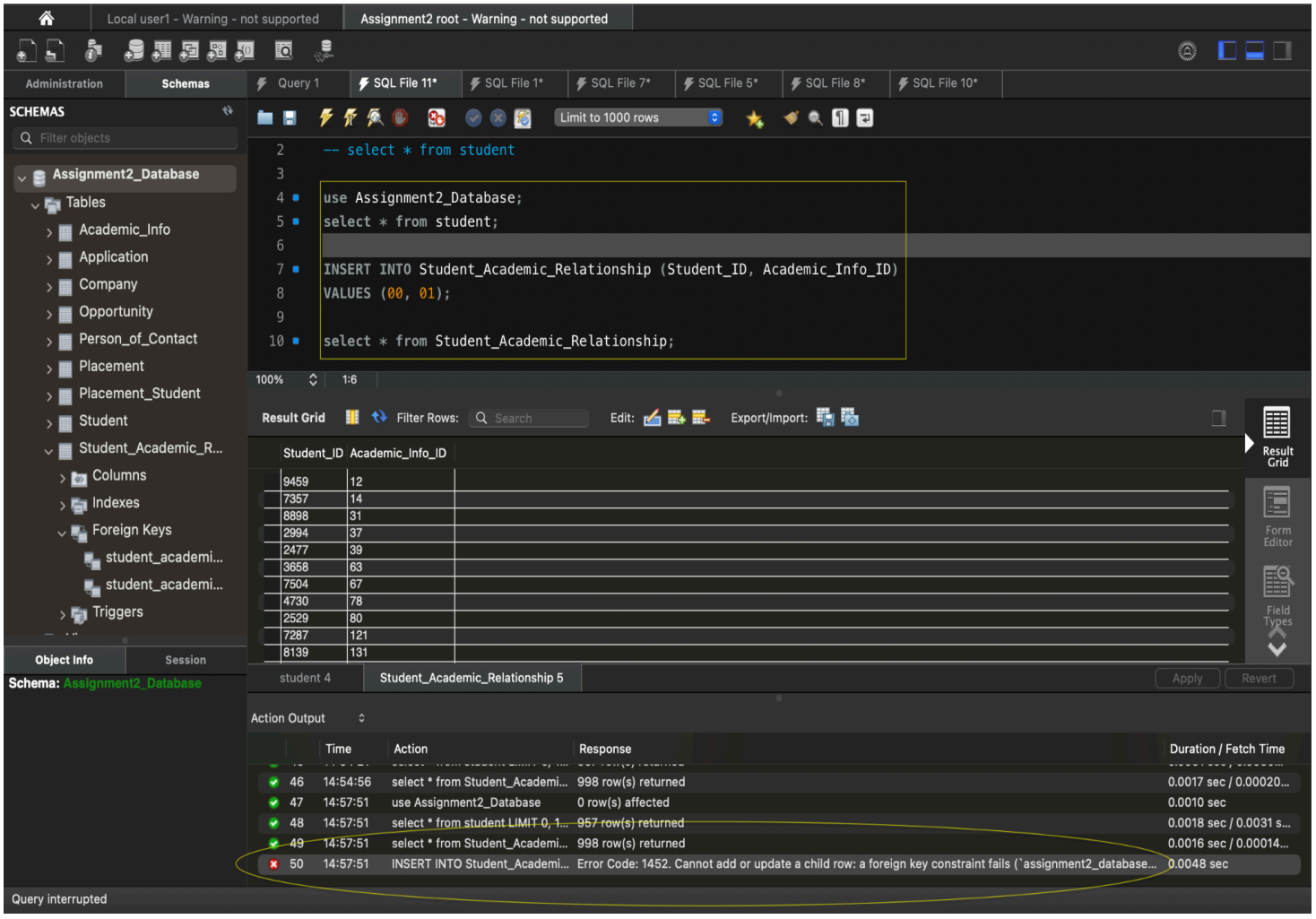
2. Mention the situation that violates the referential integrity, show the updates in the table, and how we can solve such problems

There are four main ways in which we can check the violations of referential integrity in DATBASEs, and it mostly has to do with the Foreign Key, which makes reference to the primary keys of the other tables.

1) **INSERTION**: If Table A contains the foreign key referencing the primary key of Table B, and we add a new record to Table A with a specified value for the foreign key (let's call it 'x'), an error will occur if 'x' does not correspond to any primary key value in Table B. This error indicates that the referenced value does not exist.

ERROR

Here, we are trying to insert a new value in the table "Student_Academic_Relationship", which has a foreign key (Student_ID for ID of Student dataset), If the new Tuple does not have an already existing ID, then we get an error.



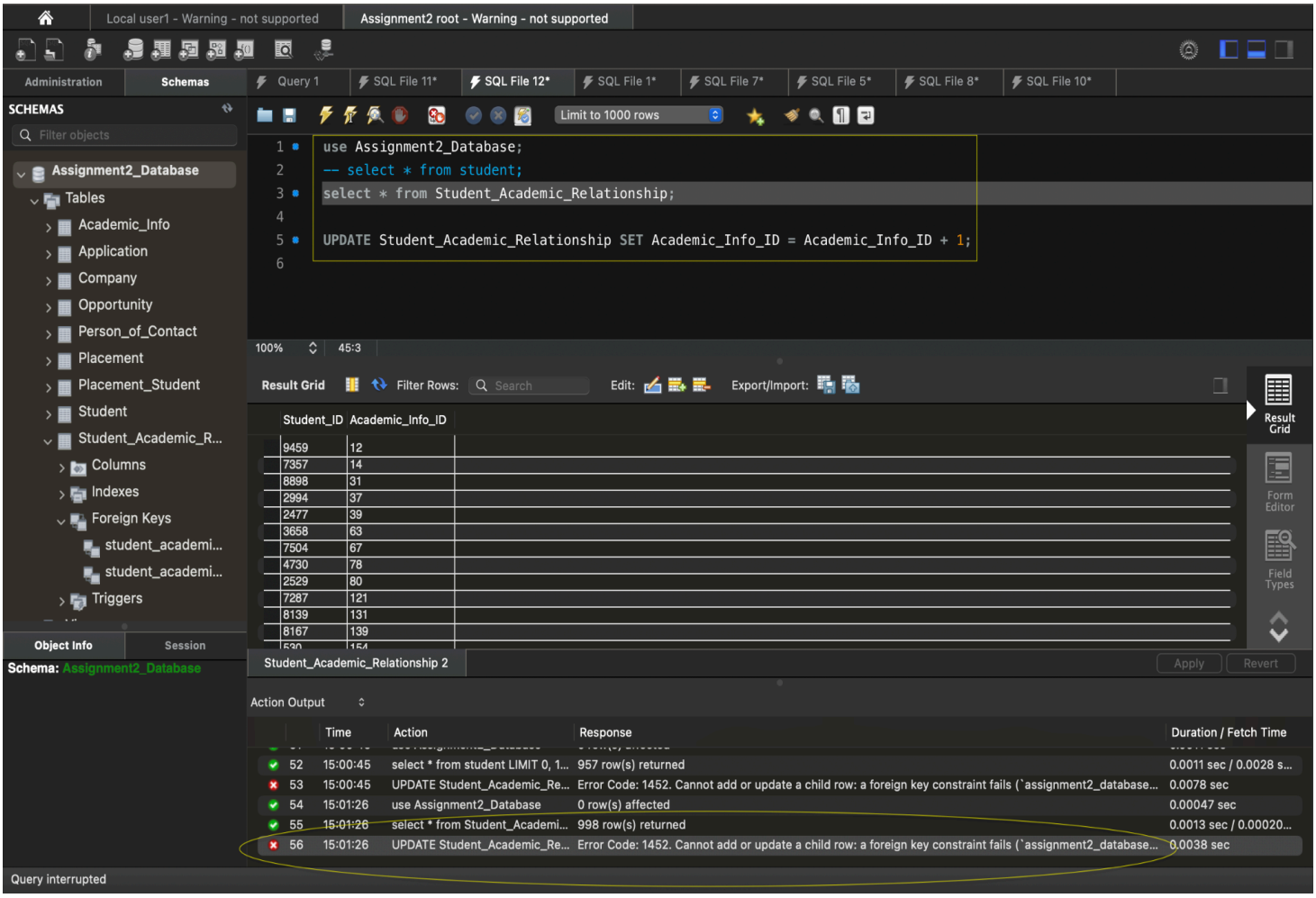
SOLUTION

This problem, does not have a particular solution, as the user must check if the ID already exists, before applying it.

2) **MODIFICATION**: If Table A includes a foreign key that references the primary key of Table B, and we update a record (along with its foreign key value) in Table A with a new value for the foreign key (let's call it 'x'), an error will occur if 'x' does not correspond to any primary key value in Table B. This error indicates the absence of the referenced value.

ERROR

Here, we are trying to modify a value in the table “Student_Academic_Relationship”, which has a foreign key (Student_ID for ID of Student dataset), but as the data is random, the incremented Value of Student_ID is not in the student table’s ID attribute, we get and error.



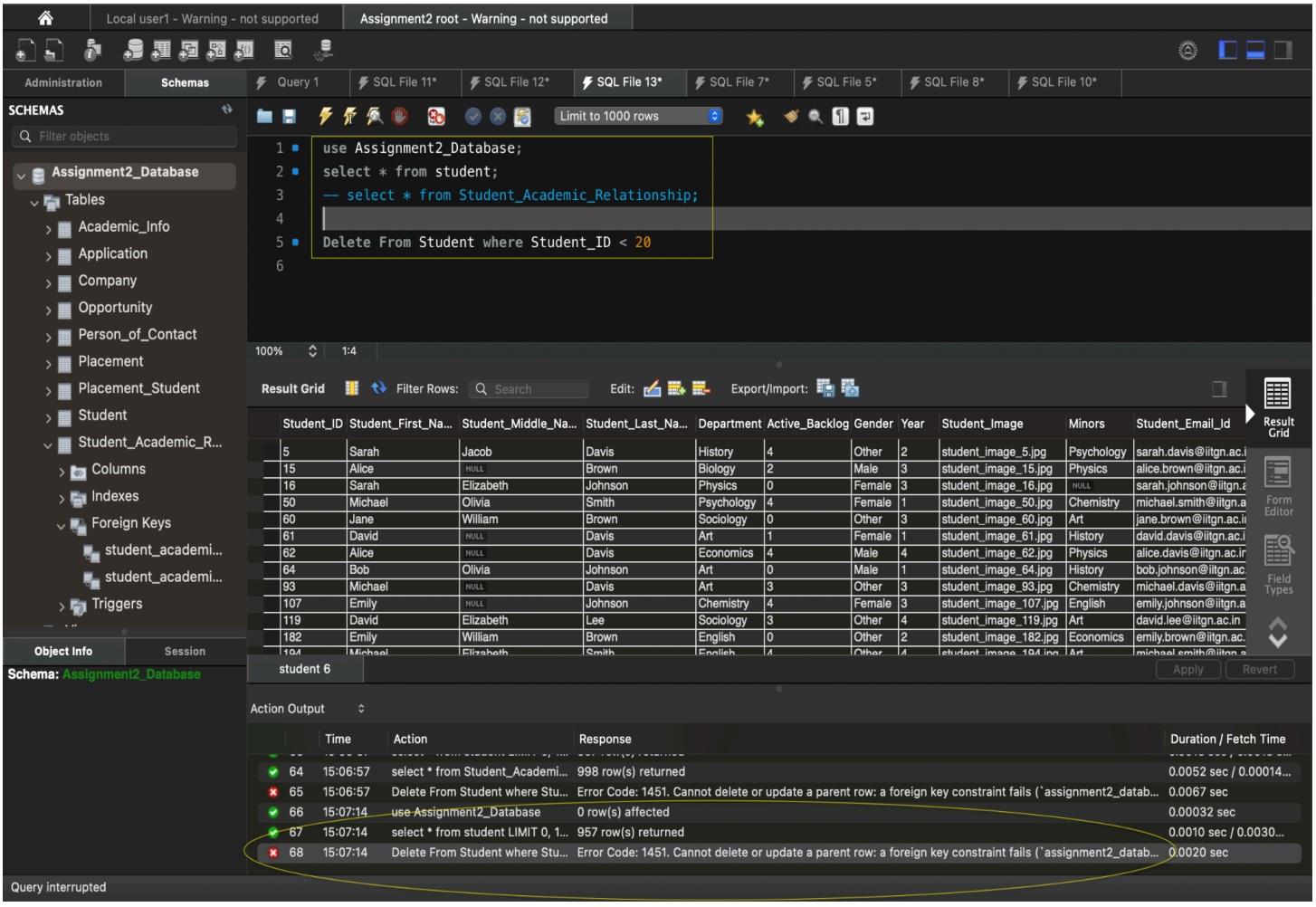
SOLUTION

This problem, does not have a particular solution, as the user must check if the modified ID already exists, before changing it.

3) **DELETION**: If Table A includes a foreign key that references the primary key of Table B, and we delete a specific row solely from Table B (without affecting Table A), and if that row's primary key is referenced elsewhere in Table A, an error will occur indicating that the parent row's value cannot be deleted because it is still referenced by Table A. Deleting such a row from Table B without removing the corresponding referencing rows from Table A would result in Table A referencing a non-existent value from Table B, thereby violating referential integrity

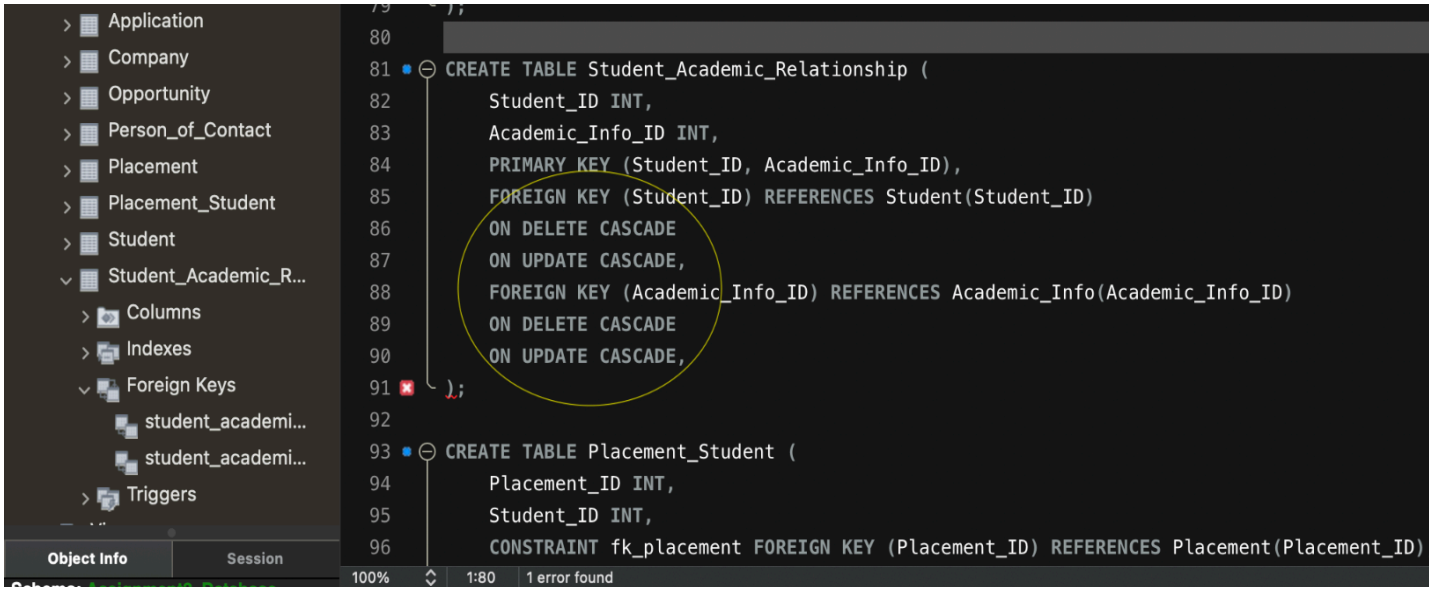
ERROR

Here, we are trying to delete the columns (tuples) of the students, whose STudent ID is less than 20. This will give an error as the Table “Student Academic Relationship” is also using it.



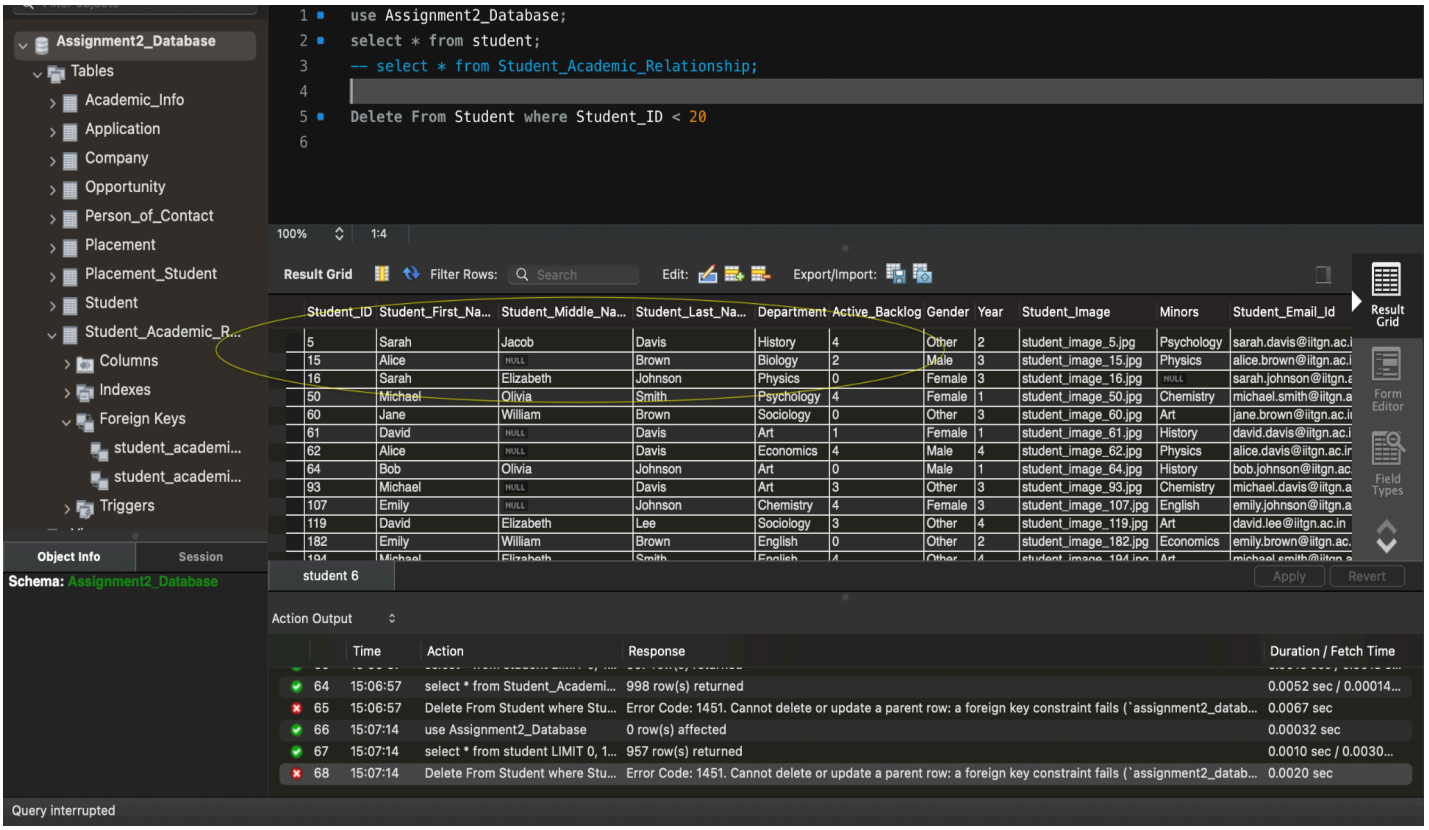
SOLUTION

We can address these concerns by implementing referential actions such as ON DELETE CASCADE and ON UPDATE CASCADE. Essentially, if a record is deleted or updated in Table B, all corresponding records in Table A will be automatically deleted or updated accordingly. This mechanism guarantees the maintenance of referential integrity, preventing any violations.

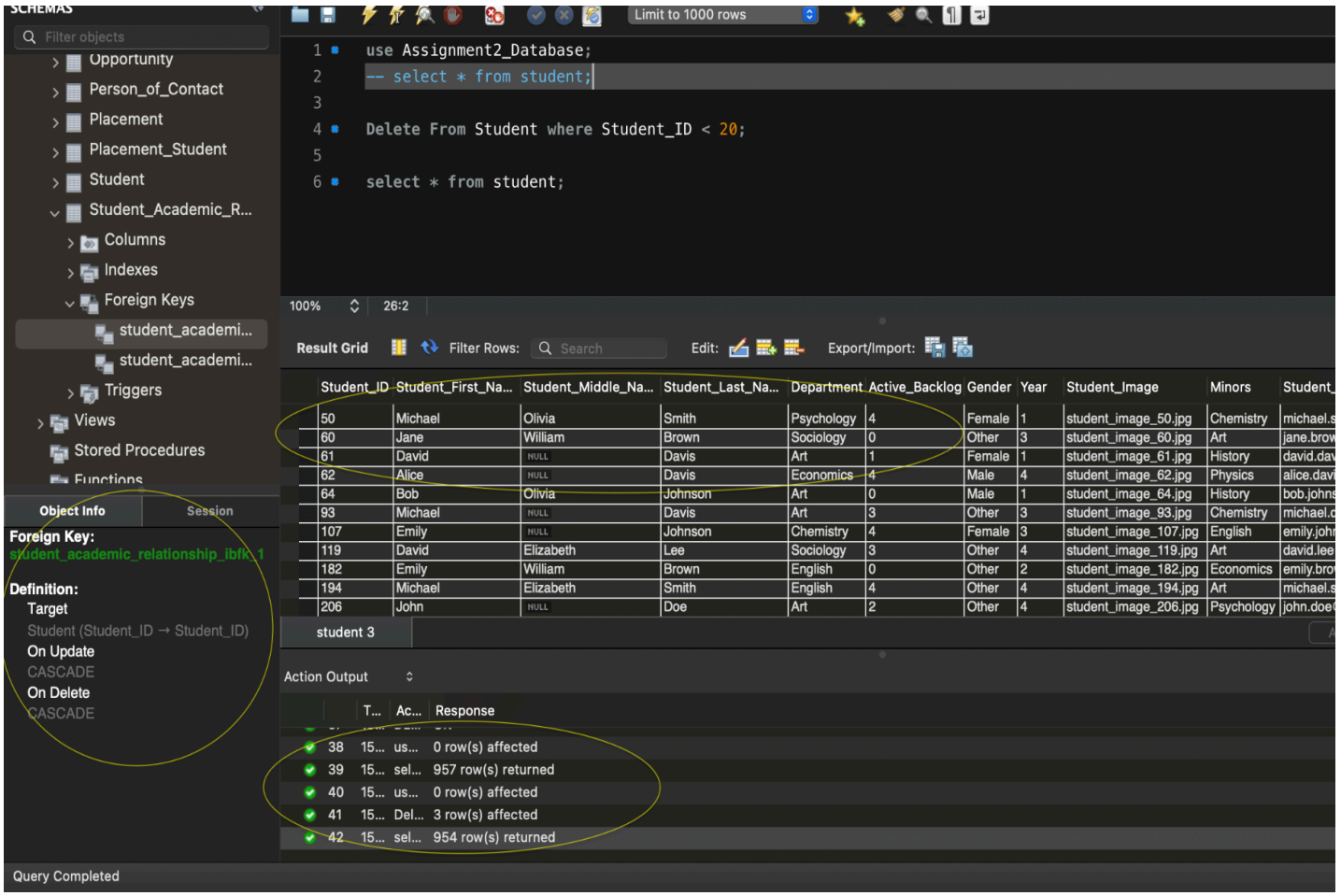


After applying Cascading, the Delete command worked well. You can see, that after the update, the columns with student ID 5, 15, 16 (<20) have been deleted, and in the left corner, it shows that the Foreign key has CASCADE applied.

BEFORE CASCADING



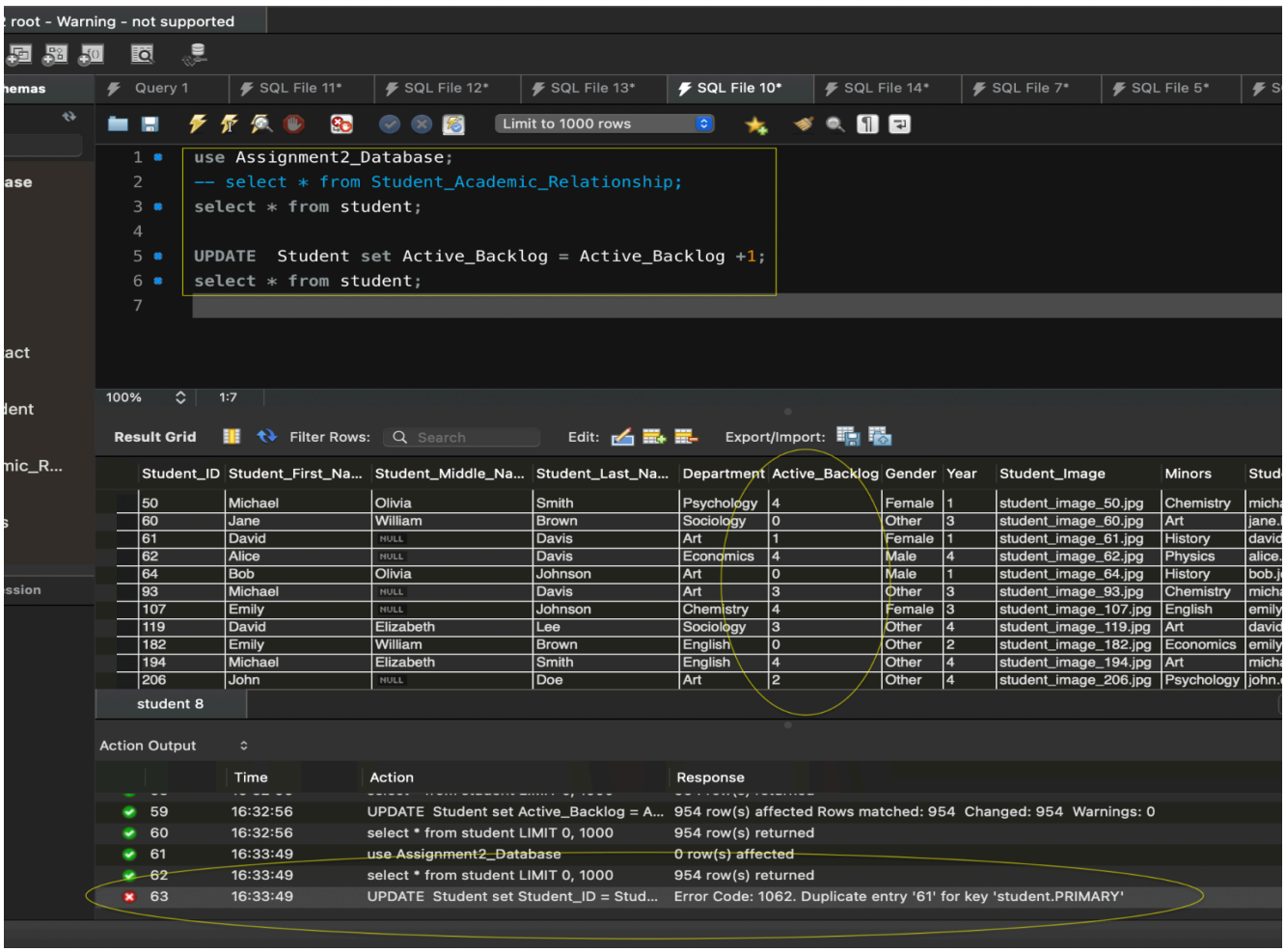
AFTER CASCADING



4) **UPDATION**: If Table A contains a foreign key referencing the primary key of Table B, and we update a specific row solely within Table B (without affecting Table A), where the primary key is referenced elsewhere in Table A, an error will occur indicating that the parent row's value cannot be updated because it is still referenced by Table A with its original value. Updating such a row in Table B without updating the corresponding referencing rows in Table A would result in Table A referencing an outdated value from Table B, thereby violating referential integrity

ERROR

Here, we are trying to UPDATE the columns (tuples) of the students,to increase the ACTIVE backlog to +1. This will give an error as the Table “Student Academic Relationship” is also using it.



SOLUTION

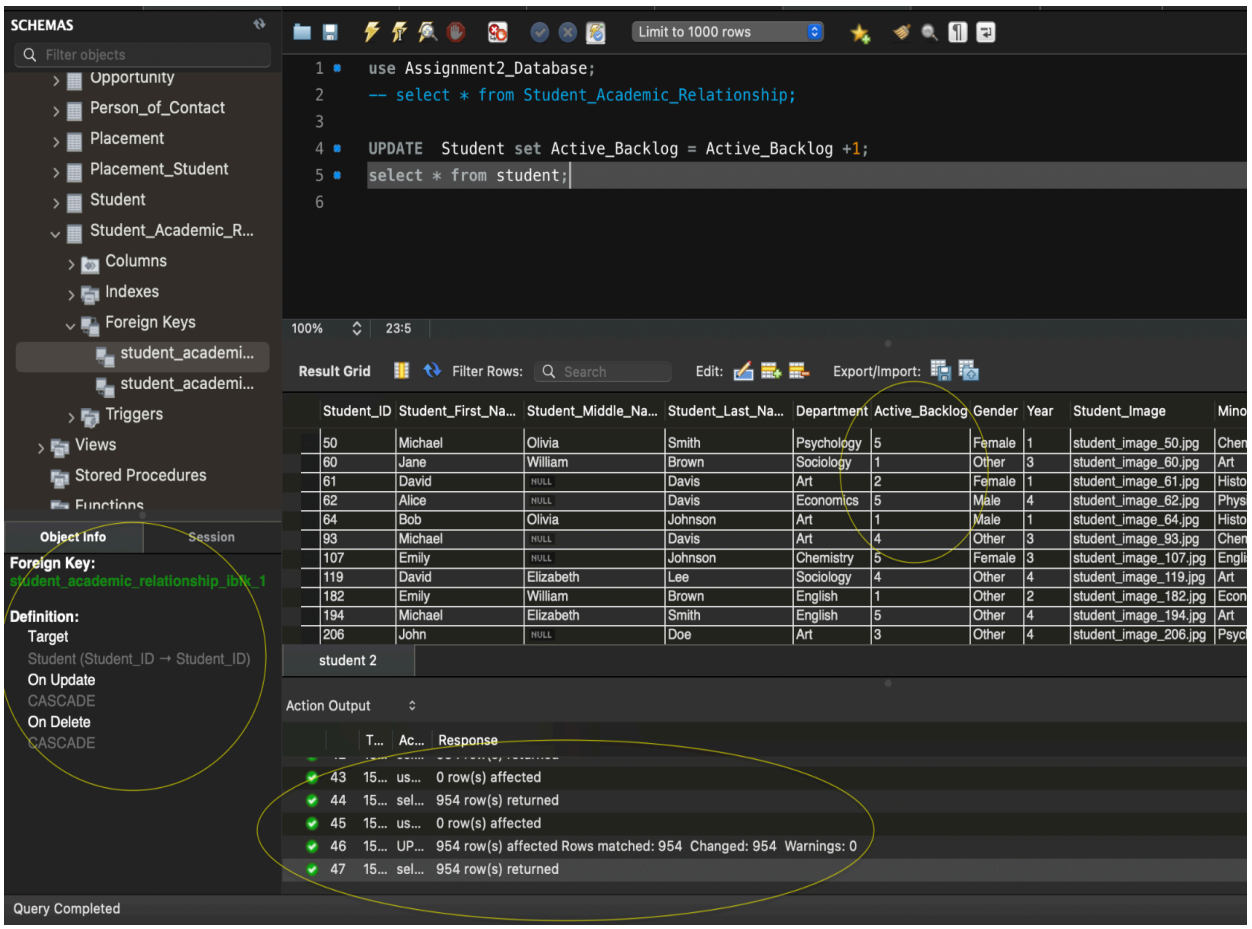
We can address these concerns by implementing referential actions such as ON DELETE CASCADE and ON UPDATE CASCADE. Essentially, if a record is deleted or updated in Table B, all corresponding records in Table A will be automatically deleted or updated accordingly. This mechanism guarantees the maintenance of referential integrity, preventing any violations.

BEFORE CASCADING

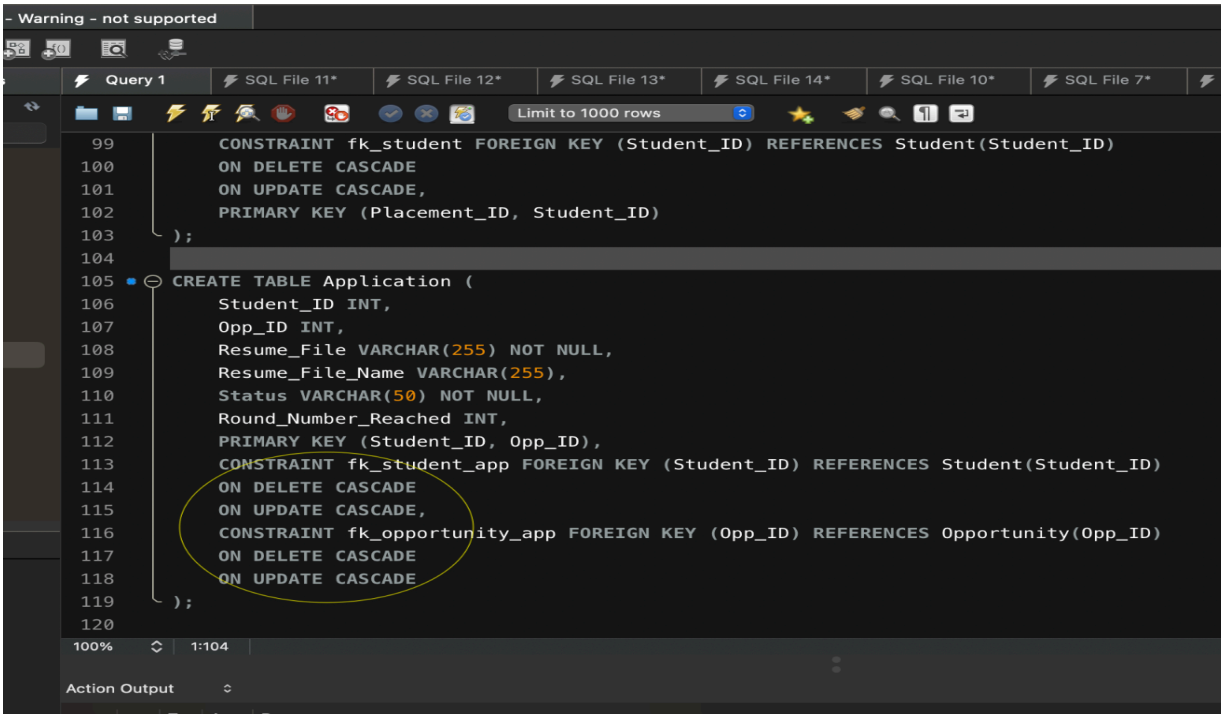
You can look at the figure above, the backlogs read (4, 0, 1, 4, 0, 3)

AFTER CASCADING

Now, the backlogs have incremented by 1 to (5, 1, 2, 5, 1, 4)



Note: In doing this question, we were getting repeated errors, even after CASCADING, and this was due to some other Tables, also having Foreign Keys (Like Application), hence we had to CASCADE them also.



3.3 Responsibility of G1 & G2:

Using the optimized ER diagram, write five operations, their corresponding nested SQL queries, and tuple relational calculus/relational algebra with the following specifications:

1. *Two queries must throw an error due to a violation of constraints specified by G1.*
2. *One of the functions should involve the storage of an image along with a caption in the database.*
3. *Include cases of natural join, outer join, renaming, two or more different kinds of aggregate functions, and case statements in one or more queries.*

Clearly mention the specifications each operation satisfies. For, e.g., if operation f1 involves storing an image, write "f1 satisfies specification 2". Also, submit screenshots of the results.

---

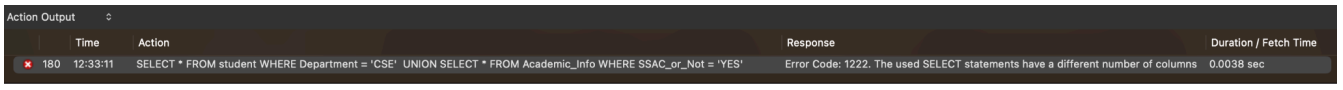
1. Operation: Here, we are attempting to use the UNION operator to combine the results of two SELECT statements. The first SELECT statement selects columns from the student table, while the second SELECT statement selects columns from the Academic_info table

Relational Algebra:

$\sigma_{\text{dept="CSE"}}$ (student) $\cup$ $\sigma_{\text{SSAC\_or\_Not="YES"}}$ (Academic_Info)

SQL Query:

```
SELECT * FROM student WHERE Department = 'CSE'
UNION
SELECT * FROM Academic_Info WHERE SSAC_or_Not = 'YES';
```



'This Operation satisfies specification 1'

This operation gives an error due to violating constraints specified by G1.

Reasoning: The two SELECT statements have different column names and data types, which will result in a type mismatch error when the UNION operator tries to combine the two result sets. Specifically, the student table columns are Student_id, Student_First_Name, Student_Middle_Name, Student_Last_Name, dept, Department, Active_Backlog, Gender, Year, Student_Image, Minor, Student_Email_Id, Contact_number.

The Academic_Info table columns are Academic_Info_Id, Transcript_File, CPI, SSAC_or_Not. The two tables have different column names and column numbers.

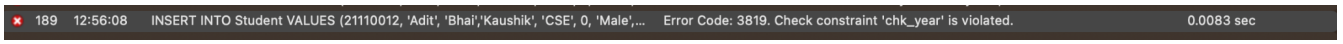
---

2. Operation: Inserting an entry into the Student table with having an invalid year.

Relational Algebra/Calculus:
Student ← Student ∪ {21110012, 'Adit', 'Bhai','Kaushik', 'CSE', 0, 'Male', 7, 'student_image_5.jpg', 'Business', 'adit@iitgn.ac.in', 1234567891}

SQL Query: INSERT INTO Student VALUES (21110012, 'Adit', 'Bhai','Kaushik', 'CSE', 0, 'Male', 7, 'student_image_5.jpg', 'Business', 'adit@iitgn.ac.in', 1234567891)

Error Code: 3819. Check constraint 'chk_year' is violated.



'This Operation satisfies specification 1'

Reasoning: We have set the constraint such that Student.year can only take values from set {1, 2, 3, 4}. But in the above SQL Query we are trying to insert Student with year 7, which is violating the constraint.

---

3. Operation: Here we are performing join over placement and opportunity, on the column Job_designation of placement and Opp_title in Opportunity.

SQL Query:

```
SELECT
  Placement.Placement_ID AS ID,
  Placement.Placement_Method AS Method,
  Placement.Company_Name AS Company,
  Placement.Job_Designation AS Designation,
  Opportunity.Opp_Title AS Opportunity_Title,
  Opportunity.Min_CPI_req AS Minimum_CPI_Requirement
FROM
```

Placement
LEFT OUTER JOIN
Opportunity ON Opportunity.Opp_Title = Placement.Job_Designation;

Relational Algebra/Calculus:

$\pi_{\text{Placement\_ID} \rightarrow \text{ID, Placement\_Method} \rightarrow \text{Method, Company\_Name} \rightarrow \text{Company, Job\_Designation} \rightarrow \text{Designation, Opp\_Title} \rightarrow \text{Opportunity\_Title, Min\_CPI\_req} \rightarrow \text{Minimum\_CPI\_Requirement}}$
$(\sigma_{\text{Opportunity.Opp\_Title = Placement.Job\_Designation}}(\text{Placement} \bowtie_{\text{Job\_Designation=Opp\_Title}} \text{Opportunity}))$

ID	Method	Company	Designation	Opportunity_Title	Minimum_CPI_Requireme...
10	Campus Placement	Tesla	Systems Analyst	Systems Analyst	7.4
10	Campus Placement	Tesla	Systems Analyst	Systems Analyst	7.2
10	Campus Placement	Tesla	Systems Analyst	Systems Analyst	7.2
10	Campus Placement	Tesla	Systems Analyst	Systems Analyst	7.5
20	Direct Application	Finance Gurus	Software Engineer	Software Engineer	7.6
20	Direct Application	Finance Gurus	Software Engineer	Software Engineer	7.3
20	Direct Application	Finance Gurus	Software Engineer	Software Engineer	7.0
20	Direct Application	Finance Gurus	Software Engineer	Software Engineer	7.4
42	Campus Placement	Biz Geniuses	Graphic Designer	Graphic Designer	7.3
42	Campus Placement	Biz Geniuses	Graphic Designer	Graphic Designer	7.5
42	Campus Placement	Biz Geniuses	Graphic Designer	Graphic Designer	7.4
42	Campus Placement	Biz Geniuses	Graphic Designer	Graphic Designer	7.3
42	Campus Placement	Biz Geniuses	Graphic Designer	Graphic Designer	7.4
42	Campus Placement	Biz Geniuses	Graphic Designer	Graphic Designer	7.1
96	Direct Application	IBM	Software Tester	Software Tester	7.3
96	Direct Application	IBM	Software Tester	Software Tester	7.4
96	Direct Application	IBM	Software Tester	Software Tester	7.2
96	Direct Application	IBM	Software Tester	Software Tester	7.3
96	Direct Application	IBM	Software Tester	Software Tester	7.4
96	Direct Application	IBM	Software Tester	Software Tester	7.4
96	Direct Application	IBM	Software Tester	Software Tester	7.1
154	Direct Application	Samsung	Marketing Analyst	Marketing Analyst	7.6
154	Direct Application	Samsung	Marketing Analyst	Marketing Analyst	7.6

'This Operation satisfies specification 3'

Reasoning: We are performing join over placement and opportunity, on the column Job_designation of placement and Opp_title in Opportunity. We are also renaming all of the columns in this SQL query.

4. Operation: We are categorizing the salary into three categories in Placement using case statement. If salary is >= 95000, then it is High and if <= 70000 then it is low else medium

Relational Algebra/Calculus:

$\pi_{\text{Company\_Name, Job\_Designation, Salary, Salary\_Category}}(\sigma_{\text{Salary} \geq 95000 \vee (\text{Salary} \leq 70000) \vee (\text{Salary} \geq 70000 \wedge \text{Salary} \leq 95000)}(\text{Placement}))$

SQL Query:

```
SELECT
  Placement.Company_Name,
  Placement.Job_Designation,
  Placement.Salary,
  CASE
    WHEN Placement.Salary >= 95000 THEN 'High'
    WHEN Placement.Salary <= 70000 THEN 'Low'
    ELSE 'Medium'
  END AS Salary_Category
FROM Placement;
```

Company_Name	Job_Designation	Salary	Salary_Category
Tesla	Systems Analyst	117244	High
Finance Gurus	Software Engineer	62849	Low
Biz Geniuses	Graphic Designer	90496	Medium
IBM	Software Tester	80156	Medium
Samsung	Marketing Analyst	67288	Low
Adobe	Software Engineer	118261	High
Code Crafters	Business Analyst	68305	Low
Network Pros	Web Developer	90740	Medium
Google	Business Analyst	69565	Low
Oracle	Software Engineer	64304	Low
Tesla	Data Analyst	106758	High
Code Innovate	Graphic Designer	68384	Low
Biz Geniuses	Network Engineer	100069	High
Samsung	Network Engineer	71964	Medium
Facebook	Systems Analyst	98386	High
Amazon	Data Analyst	75255	Medium
Microsoft	Web Developer	61156	Low
Soft Creations	Systems Analyst	115424	High
Data Insights	Graphic Designer	103596	High

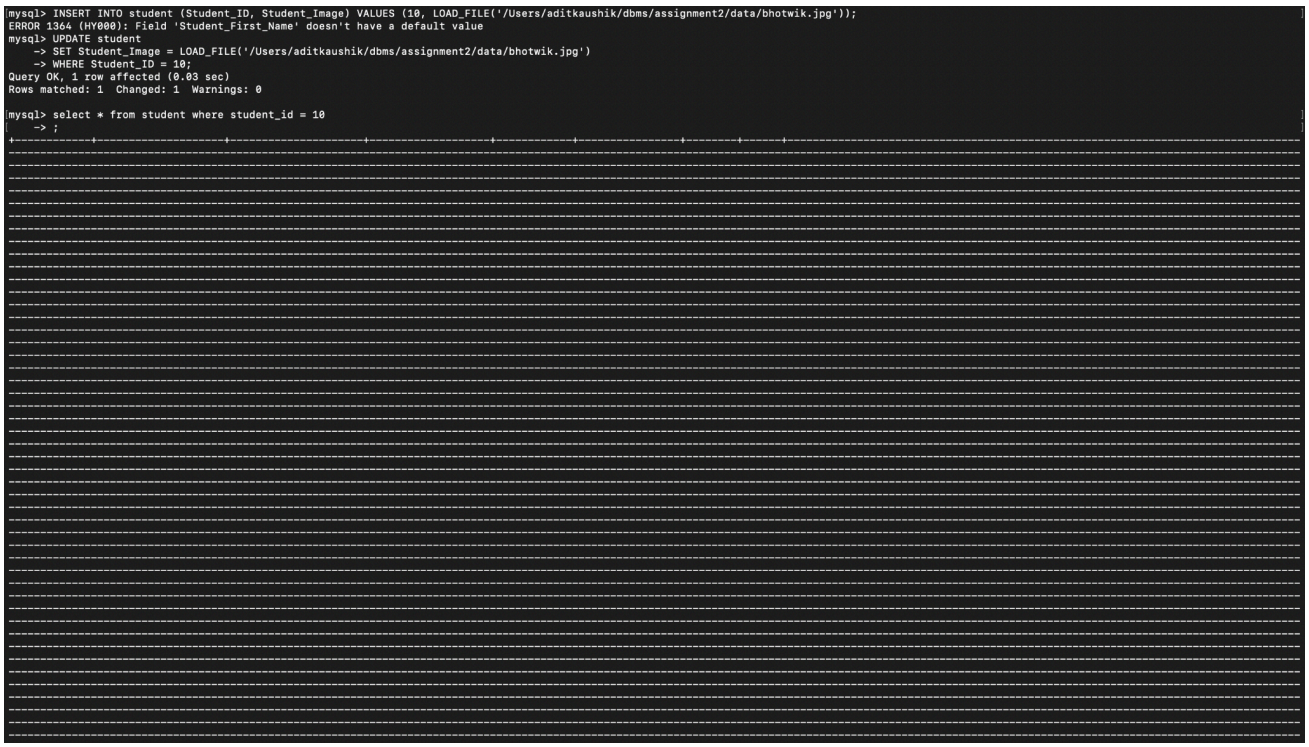
'This Operation satisfies specification 3'

Reasoning: Here we are making use of Case statement. We are categorizing the salary into three categories in Placement using Case statement. If salary is >= 95000, then it is High, and if <= 70000 then it is Low, else Medium

5. Operation: We are inserting image into "student" table. Then we are adding additional attribute "student_image-caption" into the table and inserting image and caption for the image for student with student_id = 10.

Relational Algebra/Calculus:
Student ← Student ∪ { (10, LOAD_FILE('bhotwik.jpg')) }

SQL Query:
UPDATE student SET Student_Image = LOAD_FILE('bhotwik.jpg')
WHERE Student_ID = 10;



ALTER TABLE student ADD COLUMN Student_Image_Caption varchar(100);
UPDATE student SET Student_Image_Caption = "student_image"
WHERE Student_ID = 10;

Student ← Student ∪ { student_image }

```
mysql> UPDATE student SET Student_Image_Caption = "student_image" WHERE Student_ID = 10;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select student_image_caption from student where student_id = 10;
+-----------------------+
| student_image_caption |
+-----------------------+
| student_image         |
+-----------------------+
1 row in set (0.05 sec)
```

'This Operation satisfies specification 2'

Important Links

5) ER Diagram: Draw.io file: https://app.diagrams.net/?client=1#G1yCpFEy0jyzqNLsKFg6Lim8za2PPASyPd#%7B%22pageId%22%3A%22n4RG28VcgyLvL5a97sLb%22%7D

6) Code for the Data Generation process: Google Colab notebook: https://colab.research.google.com/drive/1thD8FtA0pBjFwbtnWxz85L4WhKXOa4s-

7) Generated Dataset for the Database: Google Drive: https://drive.google.com/drive/folders/1lr40jKZI_w58t7_DY7AVztuXKtXEc2A2

8) SQL Files, and data dump: https://drive.google.com/drive/folders/1_MqzNJfXfNdrbH-PaVhR7fRd7r1Yt9vL

Contributions:

Name	Group	Roll Number	Contribution
Anugu Arun Reddy	G1	21110029	• Contributed to generating data. • Brainstormed The working of indexing and user-defined properties. • Contributed to debugging the code.
Aditya Deshmukh	G1	21110014	• Created a data generation script to streamline processes. • Conducted a thorough analysis of errors in the previous assignment and implemented necessary revisions. • Played a key role in extending column and table indexing functionality.
Abhay Kumar Upparwal	G1	21110004	• Contributed in generating the data of all the tables following the constraint and relations between them. • Contributed in writing operation for part 3.3.
Aaryan Darad	G1	21110001	• Contributed to Generation of Data • Contributed to the indexing of columns and table extension. • Formatted the document and corrected missing points or errors.
Gaurav Joshi	G2	21110065	• Worked on G2 Question, making the user, giving permission and making queries. • Brainstorming changes based on the suggestions of the TA • Remaking the ER Diagram in Draw.io • Working on the Referential Integrity problem, making the Cascading updates and Getting snapshots of the Queries and SQL

			files.
Adit Kaushik	G2	21110010	• Wrote down the nested SQL queries for the operations which included queries such as inserting images along with captions to the database and case statements. • Contributed & checked relational algebra for all the operations performed in Q 3.3
More Rutwik	G2	21110133	• Listed down the operations that throw errors due to violation of constraints specified on attributes. • Wrote down the nested SQL queries for the operations which included queries such as joins, renaming, and using case statements. • Wrote relational algebra for all the operations performed in Q 3.3 • Worked on Q 3.3, making the user, giving permission, and making queries.
Ahaan Giriya	G2	21110015	• Contributed in indexing of user-defined data type • Helped in Question 3.3 • Formatted the document