Indian Institute of Technology Gandhinagar

# CS 432 Databases

Assignment 2

---

## Labs Management

---

January 29, 2024


SQL Dummies:
Kaushal Kothiya, 21110107
Sachin Jalan, 21110183
Manav Jain, 22110140
Anish Karnik, 21110098
Malwat Husain, 21110117
Srujan Shetty, 21110214
Ameya Tajne, 21110220

# 1. Responsibility of G1:

After having discussion with TA, we made some modifications in our database. As Contact entity had a one to one relation with other entities, it would be better to have a separate attribute of contact for respective entities rather than making a new table for it. Therefore, the relations involved with Contact entity were also removed and replaced with Email_ID and Contact for every user.

## Population of tables with data :

- Utilized the Faker and Random libraries in Python for generating diverse data for various entities.
- The generated data was written into CSV files for further use.
- The generated CSV files were imported into MySQL Workbench to populate corresponding tables.
- Ensured that primary keys were unique to prevent repetitions in the tables.
- Addressed total/partial/one-one/many-one/one-many/one-one participation constraints.
- Implemented appropriate data types for accurate table filling.

<Link To Code> Notebook written in Python script developed to generate data for each entity.

<Link To SQL Code> SQL file written in MySQL to create a database along with queries.

<Link To Github Repo > Github Repository containing all files

**References:**


- Faker Library Documentation. (https://faker.readthedocs.io/en/master/)
- Relational calculus (LectureSlides)

- ## Indexing:

- Indexing in databases enhances search efficiency by creating a data structure that accelerates data retrieval.
- However, it comes with trade-offs, including increased storage and potential write performance reduction, as indexes need updating when the indexed columns are modified.
- We applied indexing where it was deemed beneficial for improving search efficiency, taking into account the trade-off of increased storage space due to indexing.

**Note:** It's important to note that this indexing decision aligns with the user experience and query patterns expected in the application. For instance, we created indexing for Staff_name in the entity Staff, where the staff members are predominantly identified and accessed by their names rather than numerical identifiers. However, it's always essential to strike a balance and consider the overall query workload and data modification operations when deciding on indexing strategies.

- **First_Name(Professor entity):**

  **Reason:** Indexing the Name(First Name) attribute in the Professor entity can be beneficial for queries that involve searching for professors by their names. For example, queries like "List all professors in alphabetical order by name" can benefit from the index. Without an index, the database might need to scan the entire Professor table to find matching rows, leading to slower query execution times.

```
create index Prof_name on professor(First_Name);
```

  **SQL Query:**

```
select * from staff where First_Name='Melissa';
```

| | |
|---|---|
| **Before Indexing:** | 0.004268 s |
| **After Indexing:** | .0012745 s |

## Prof Name (Before Indexing):

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00426800
Table lock wait time: 0:00:0.00001800

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 1
Rows examined: 30

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
No Index used

**Other Info:**
Event Id: 1376
Thread Id: 50

## Prof Name (After Indexing):

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00147450
Table lock wait time: 0:00:0.00000400

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 1
Rows examined: 1

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 1780
Thread Id: 50

- **Roll_Number (Student entity):**

**Reason:** Indexing the Roll number attribute of the Student entity can significantly improve query performance when filtering or sorting data based on this attribute. For example, queries like “Show student data for a student with Roll number X" or "List all students in ascending order of Roll numbers" can benefit from the index. Without an index, the database might have to scan the entire table to find matching rows, leading to slower query execution times.

```
create index student_name on students(Roll_Number);
```

**SQL Query:**

```
select * from students where First_Name='Ameya';
```

| | |
|---|---|
| **Before_Indexing:** | 0.00132 s |
| **After_Indexing:** | 0.00033 s |

**Student Roll Number (Before Indexing)**

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00132430
Table lock wait time: 0:00:0.00000700

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 1
Rows examined: 1

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 1379
Thread Id: 50

**Student Roll Number (After Indexing)**

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00033450
Table lock wait time: 0:00:0.00000300

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 1
Rows examined: 1

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 1787
Thread Id: 50

- **First_Name (Staff entity):**

**Reason:** Staff members are more likely to be identified with their names rather than numerical identifiers like Employee IDs. Indexing the staff_name attribute in the Staff entity improves optimizing queries that involve filtering or sorting based on the staff member's name.

```
create index staff_name_idx on staff(First_Name);
```

**SQL Query:**

```
select * from staff where First_Name='Melissa';
```

| | |
|---|---|
| **Before_Indexing:** | 0.00101 s |
| **After_Indexing:** | 0.000476 s |

## Staff Name (Before Indexing)

**Timing (as measured at client side):**
Execution time: 0:00:0.01500000

**Timing (as measured by the server):**
Execution time: 0:00:0.00101450
Table lock wait time: 0:00:0.00000700

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 1
Rows examined: 50

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
No Index used

**Other Info:**
Event Id: 1382
Thread Id: 50

## Staff Name (After Indexing):

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00047600
Table lock wait time: 0:00:0.00000300

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 1
Rows examined: 1

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 1783
Thread Id: 50

- **Equipment_Name (Inventory entity)**

**Reason:** Indexing the Equipment_Name attribute is chosen to optimize queries where users search for tools or equipment by their names rather than their unique numerical identifiers (Item ID). It is user-friendly to search for a tool using its name rather than remembering or looking up its specific ID. Queries like "Get total number of beakers available" or “Know how much stock is remaining of a particular tool” can quickly locate records that match the specified name conditions.

```
create index Eqp_name on inventory(Equipment_Name);
```

**SQL Query:**

```
select * from inventory where Equipment_Name='Centrifuge';
```

| | |
|---|---|
| **Before_Indexing:** | 0.00369 s |
| **After_Indexing:** | 0.00286 s |

**Equipment Name(Before Indexing):**

**Timing (as measured at client side):**
Execution time: 0:00:0.00000000

**Timing (as measured by the server):**
Execution time: 0:00:0.00369440
Table lock wait time: 0:00:0.00000300

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 9
Rows examined: 120

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
No Index used

**Other Info:**
Event Id: 1194
Thread Id: 88

**Equipment Name (After Indexing):**

**Timing (as measured at client side):**
Execution time: 0:00:0.01600000

**Timing (as measured by the server):**
Execution time: 0:00:0.00286470
Table lock wait time: 0:00:0.00000500

**Errors:**
Had Errors: NO
Warnings: 0

**Rows Processed:**
Rows affected: 0
Rows sent to client: 9
Rows examined: 9

**Temporary Tables:**
Temporary disk tables created: 0
Temporary tables created: 0

**Joins per Type:**
Full table scans (Select_scan): 0
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

**Sorting:**
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

**Index Usage:**
At least one Index was used

**Other Info:**
Event Id: 1201
Thread Id: 88

- # User Defined Data Types

1. **ENUM:**

   The ENUM data type is used to define a column that can have one of a predefined set of values. It allows you to specify a list of possible values, and any column of that type can only store one of those values.

- The ENUM type enforces data validation by allowing only specific values of the specified set. If you try to insert a value outside the specified set, MySQL will generate an error:

We have implemented ENUM for below attributes:

- **Lab_Name in Lab, Course Slot, Grants Entities and lab related relations** :
  - It can take values from: ( 'Anatomy Lab', 'Biochemistry Lab', 'Biology Lab', 'Botany Lab', 'Chemistry Lab', 'Computer Lab', 'Ecology Lab', 'Engineering Lab', 'Genetics Lab', 'Geology Lab', 'Microbiology Lab', 'Neuroscience Lab', 'Physics Lab', 'Psychology Lab', 'Zoology Lab' )

  - Labs names need to be user defined as there are limited labs and we don't want our data to have any other lab than ours.

```
DROP TABLE IF EXISTS `course_slot`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `course_slot` (
  `Time_ID` varchar(255) NOT NULL,
    `Lab_Name` ENUM('Anatomy Lab', 'Biochemistry Lab', 'Biology Lab', 'Botany Lab', 'Chemistry Lab',
    'Computer Lab', 'Ecology Lab', 'Engineering Lab', 'Genetics Lab', 'Geology Lab', 'Microbiology Lab',
    'Neuroscience Lab', 'Physics Lab', 'Psychology Lab', 'Zoology Lab') NOT NULL,
  PRIMARY KEY (`Time_ID`,`Lab_Name`),
  KEY `Lab_Name` (`Lab_Name`),
  CONSTRAINT `course_slot_fk_1` FOREIGN KEY (`Time_ID`) REFERENCES `time_slot` (`Time_ID`),
  CONSTRAINT `course_slot_fk_2` FOREIGN KEY (`Lab_Name`) REFERENCES `lab` (`Lab_Name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- **Dept_name in Department, Students entities and department related relations**:
  - It can take values from the set: ('CE', 'CL', 'CSE', 'EE', 'ME', 'MSE')
  - Department names need to be user defined as there are limited departments and we don't want our data to have any other departments.

```
DROP TABLE IF EXISTS `department`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `department` (
  Dept_Name ENUM('CE', 'CL', 'CSE', 'EE', 'ME', 'MSE') NOT NULL,
  `No_of_Faculties` int DEFAULT '0',
  PRIMARY KEY (`Dept_Name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

As mentioned above, it throws an error when we insert a department other than in our list.

```
INSERT INTO `students` VALUES (11637579,'Ameya',NULL,'Tajne',2021,'MA',0,'Project','CSE','tajne@iitgn.com',1234567890,9.99);
```

```
475  17:02:53  INSERT INTO `students` VALUES (11637579,'Ameya',NULL,'Tajne',2021,'MA',0,'P...  Error Code: 1265. Data truncated for column 'Degree' at row 1
```

- **Role in Staff entity**
  - It can take values from the set: 'Assistant', 'Researcher', or 'Technician'
  - You can easily filter or search for employees with a specific role without dealing with arbitrary codes or strings.

```
DROP TABLE IF EXISTS `staff`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `staff` (
  `Employee_ID` bigint NOT NULL,
  `First_Name` varchar(255) NOT NULL,
  `Middle_Name` varchar(255) DEFAULT NULL,
  `Last_Name` varchar(255) DEFAULT NULL,
  `Salary` float DEFAULT NULL,
  `Role` enum('Assistant','Researcher','Technician') DEFAULT NULL,
   `Lab_Name` ENUM('Anatomy Lab', 'Biochemistry Lab', 'Biology Lab', 'Botany Lab', 'Chemistry Lab', 'Computer Lab',
   'Ecology Lab', 'Engineering Lab', 'Genetics Lab', 'Geology Lab', 'Microbiology Lab', 'Neuroscience Lab',
   'Physics Lab','Psychology Lab', 'Zoology Lab') NOT NULL,
  `Email_ID` VARCHAR(255) CHECK (Email_ID REGEXP'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),
  `Contact` bigint DEFAULT NULL,
  PRIMARY KEY (`Employee_ID`),
  UNIQUE KEY `Employee_ID` (`Employee_ID`),
  CONSTRAINT `staff_chk_2` CHECK ((((`Contact` >= 1000000000) and (`Contact` < 10000000000))),
  CONSTRAINT `staff_fk_2` FOREIGN KEY (`Lab_Name`) REFERENCES `lab` (`Lab_Name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

## 2. User Defined DataType for Email:

```
`Email_ID` VARCHAR(255) CHECK (Email_ID REGEXP'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),
```

To verify that an email address is correct, we have used regular expressions. These will give errors if we don't specify email in a particular structure. These needs to be user defined as we don't need wrong email IDs in our database which would create problems going ahead in future.

Below is an example of inserting a faulty email along with the console giving us an error to do that.

```
INSERT INTO `students` VALUES(11637579,'Ameya',NULL,'Tajne',2021,'BTech',0,'Investigation','CSE','anthony.marquezgmail.com',1234567890,9);
```

```
477  17:26:27  INSERT INTO `students` VALUES(11637579,'Ameya',NULL,'Tajne',2021,'BTech',0...  Error Code: 3819. Check constraint 'students_chk_1' is violated.
```

Violated Constraint for Email ID.

3. **JSON data type:**

- JSON data type is beneficial while dealing with semi-structured data where the schema might vary and you want to store and query data in a flexible way.
- The JSON data type in MySQL allows you to store and manipulate JSON data in a structured way.You can also query and extract data from the JSON column using various JSON functions provided by MySQL.

- Vendor_Address in inventory entity:

```
DROP TABLE IF EXISTS `inventory`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
SET character_set_client = utf8mb4 ;
CREATE TABLE `inventory` (
  `ID` bigint NOT NULL,
  `Equipment_Name` varchar(255) NOT NULL,
  `Price` float DEFAULT NULL,
  `Vendor_Address` json DEFAULT NULL,
  `Vendor_Phone_Number` varchar(255) DEFAULT NULL,
  `Manufacturer_Name` varchar(255) DEFAULT NULL,
  `Status` varchar(255) DEFAULT NULL,
   `Lab_Name` ENUM('Anatomy Lab', 'Biochemistry Lab', 'Biology Lab', 'Botany Lab', 'Chemistry Lab',
   'Computer Lab', 'Ecology Lab', 'Engineering Lab', 'Genetics Lab', 'Geology Lab', 'Microbiology Lab',
   'Neuroscience Lab', 'Physics Lab', 'Psychology Lab', 'Zoology Lab') NOT NULL,
  `Quantity` int DEFAULT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `ID` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- ## Table Extension

  We can perform table extension in many ways. We have two examples to show table extensions for our database. These are:

  - Creating a sub table from an existing table
  - Adding new columns which extends the table

  First we create sub tables for students in different branches. The goal of this table addition is to improve database performance. The table extension makes it easier to organize and handle data. For instance, we may simply access a list of all students in a specific department or the number of students enrolled in each. This form of data analysis can be useful for making employment and statistical judgements.

  Here are the queries for creating sub tables:

```
CREATE TABLE CSE as (select * from students where Dept_Name = 'CSE');
CREATE TABLE EE as (select * from students where Dept_Name = 'EE');
CREATE TABLE ME as (select * from students where Dept_Name = 'ME');
CREATE TABLE CE as (select * from students where Dept_Name = 'CE');
CREATE TABLE CL as (select * from students where Dept_Name = 'CL');
CREATE TABLE MSE as (select * from students where Dept_Name = 'MSE');
```

  Here, is an example table from CSE sub-table:

```
680 • select * from CSE;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 21321656 | Jorge | Shawn | Sloan | 2012 | BTech | 0 | Examination | CSE | jorge.sloan@example.com | 1234567902 |
| 26160676 | Elizabeth | NULL | Marshall | 2017 | BTech | 0 | Development | CSE | elizabeth.marshall@example.com | 1234567908 |
| 29952133 | Christopher | NULL | Mann | 2023 | MTech | 0 | Training | CSE | christopher.mann@example.com | 1234567914 |
| 34114128 | Nicole | Cory | Wilkinson | 2021 | MTech | 0 | Demonstration | CSE | nicole.wilkinson@example.com | 1234567920 |
| 39953828 | Richard | Jessica | Daniels | 2012 | MTech | 0 | Sampling | CSE | richard.daniels@example.com | 1234567926 |
| 44059805 | Robert | NULL | Park | 2012 | MTech | 0 | Investigation | CSE | robert.park@example.com | 1234567932 |
| 48882413 | Breanna | George | Brown | 2015 | MTech | 0 | Experiment | CSE | breanna.brown@example.com | 1234567938 |
| 53733286 | Kaitlyn | Kaitlyn | Andrade | 2011 | PhD | 0 | Calibration | CSE | kaitlyn.andrade@example.com | 1234567944 |
| 59731192 | Nathan | NULL | Morris | 2012 | PhD | 0 | Testing | CSE | nathan.morris@example.com | 1234567950 |
| 73781167 | Steven | NULL | Maynard | 2009 | MTech | 0 | Field Work | CSE | steven.maynard@example.com | 1234567956 |

  Second way to extend tables is to add new columns. The query to add a columns is:

```
ALTER TABLE students
ADD COLUMN GPA float;
```

  Here, we add a new column for the GPA of students.

```
INSERT INTO `students` VALUES (11637579,'Ameya',NULL,'Tajne',2021,'BTech',0,'Project','CSE','tajne@iitgn.com',1234567890,9.99);
```

```
649 •  select * from students where First_Name='Ameya';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact | GPA |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11637579 | Ameya | NULL | Tajne | 2021 | BTech | 0 | Project | CSE | tajne@iitgn.com | 1234567890 | 9.99 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

The new column GPA has been added.

## Responsibility of G2:

Q1. Create a user named "user1" with the password "password1".

A user with the name ‘user1’ with password ‘password1’ was created using the following command.

To connect to as user1, in the Home Page of MySQL workbench > Add a new Connection and fill the necessary fields.

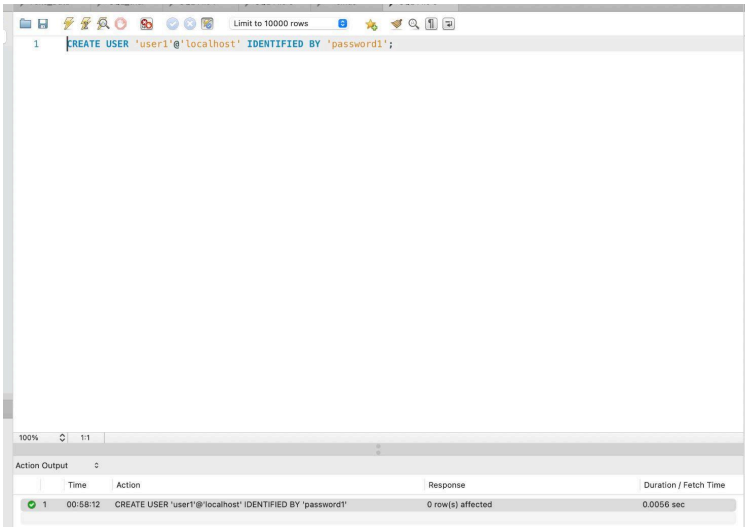
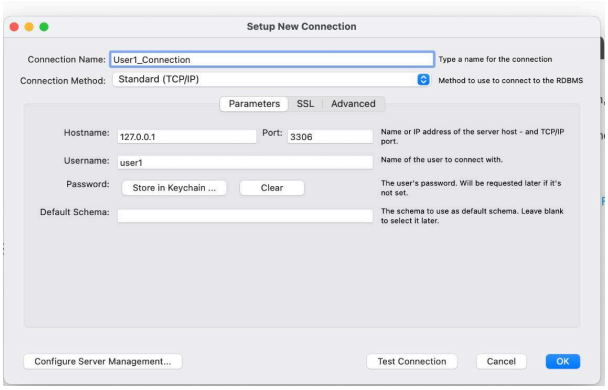



Fig. Adding new user[user1] through the root user.



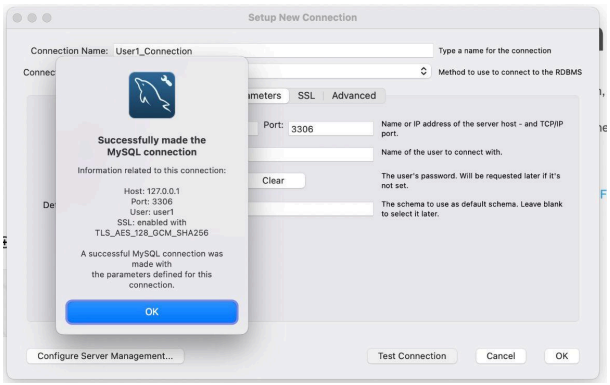



Fig. Connecting through the new User

Fig. Successfully connected as `user1`

Q2. 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

**View 1:**

View1 was created by joining 2 tables:

1. Inventory
2. accessed_tool

Here, vendor_Address is of the JSON type, a user-defined datatype.

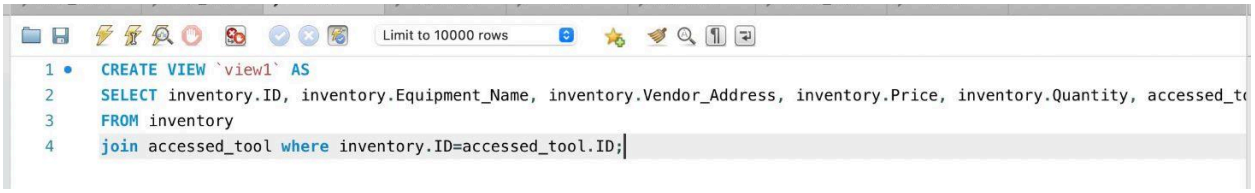



Fig. Query to create view1

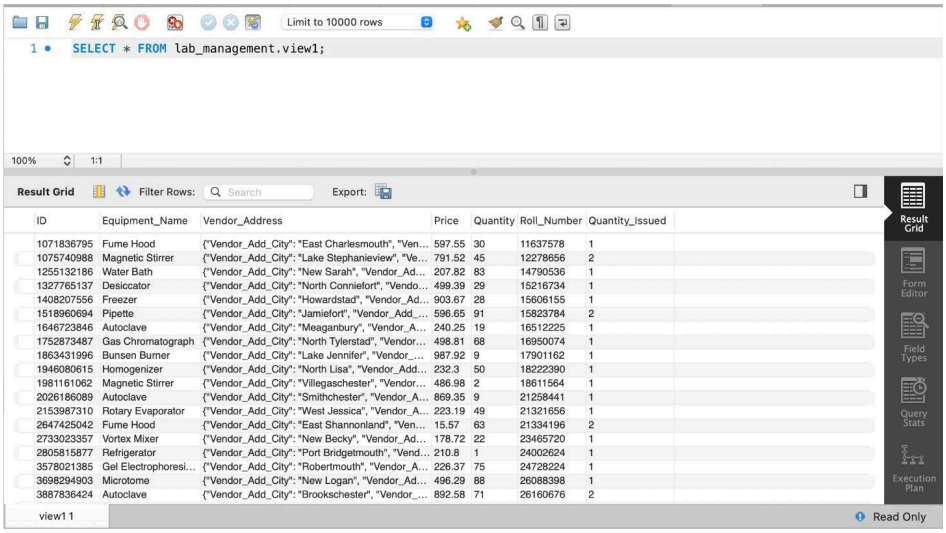



Fig. Viewing the data in View1

**View 2:**

View2 was created by joining 2 tables:

1. Lab
2. staff

Here, Role is an ENUM datatype, which is a user-defined datatype.

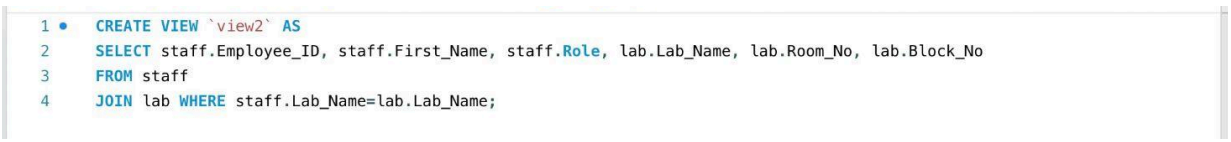
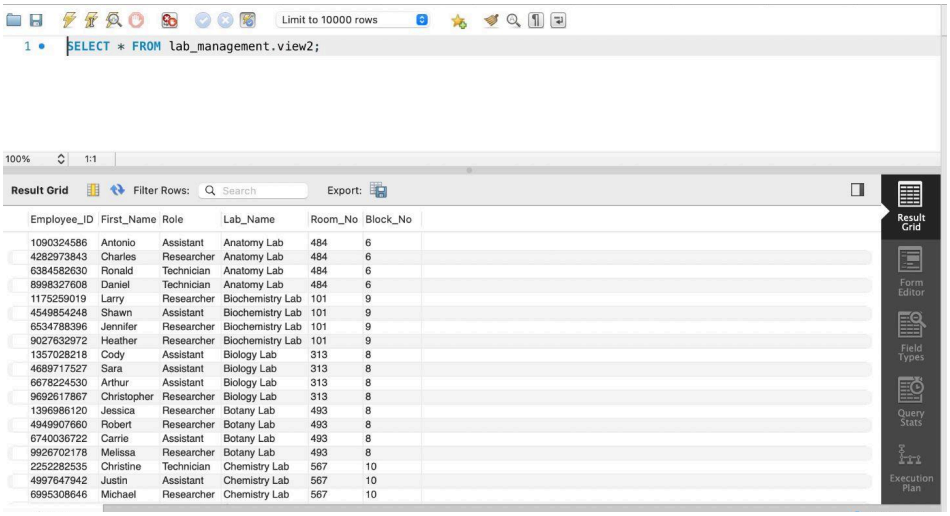



Fig. Query to create view2




| Employee_ID | First_Name | Role | Lab_Name | Room_No | Block_No |
|---|---|---|---|---|---|
| 1090324586 | Antonio | Assistant | Anatomy Lab | 484 | 6 |
| 4282973843 | Charles | Researcher | Anatomy Lab | 484 | 6 |
| 6384582630 | Ronald | Technician | Anatomy Lab | 484 | 6 |
| 8998327608 | Daniel | Technician | Anatomy Lab | 484 | 6 |
| 1175259019 | Larry | Researcher | Biochemistry Lab | 101 | 9 |
| 4549854248 | Shawn | Assistant | Biochemistry Lab | 101 | 9 |
| 6534788396 | Jennifer | Researcher | Biochemistry Lab | 101 | 9 |
| 9027632972 | Heather | Researcher | Biochemistry Lab | 101 | 9 |
| 1357028218 | Cody | Assistant | Biology Lab | 313 | 8 |
| 4689717527 | Sara | Assistant | Biology Lab | 313 | 8 |
| 6678224530 | Arthur | Assistant | Biology Lab | 313 | 8 |
| 9692617867 | Christopher | Researcher | Biology Lab | 313 | 8 |
| 1396986120 | Jessica | Researcher | Botany Lab | 493 | 8 |
| 4949907660 | Robert | Researcher | Botany Lab | 493 | 8 |
| 6740036722 | Carrie | Assistant | Botany Lab | 493 | 8 |
| 9926702178 | Melissa | Researcher | Botany Lab | 493 | 8 |
| 2252282535 | Christine | Technician | Chemistry Lab | 567 | 10 |
| 4997647942 | Justin | Assistant | Chemistry Lab | 567 | 10 |
| 6995308646 | Michael | Researcher | Chemistry Lab | 567 | 10 |

Fig. Viewing the data in View2

```
CREATE TABLE `staff` (
  `Employee_ID` bigint NOT NULL,
  `First_Name` varchar(255) NOT NULL,
  `Middle_Name` varchar(255) DEFAULT NULL,
  `Last_Name` varchar(255) DEFAULT NULL,
  `Salary` float DEFAULT NULL,
  `Role` enum('Assistant','Researcher','Technician') DEFAULT NULL,
  `Lab_Name` varchar(255) DEFAULT NULL,
  `Email_id` varchar(255) NOT NULL,
  `Contact` bigint DEFAULT NULL,
  PRIMARY KEY (`Employee_ID`),
  UNIQUE KEY `Employee_ID` (`Employee_ID`),
  CONSTRAINT `staff_chk_1` CHECK ((`Contact` >= 1000000000) and (`Contact` < 10000000000)),
  CONSTRAINT `staff_fk_1` FOREIGN KEY (`Lab_Name`) REFERENCES `lab` (`Lab_Name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Fig. The SQL schema for **staff** table.

Q3. Grant "user1" the following permissions on "table1":

- SELECT, UPDATE, DELETE

We grant the permission using the root.

**Note:** The permissions need to be when logged in as the root user.

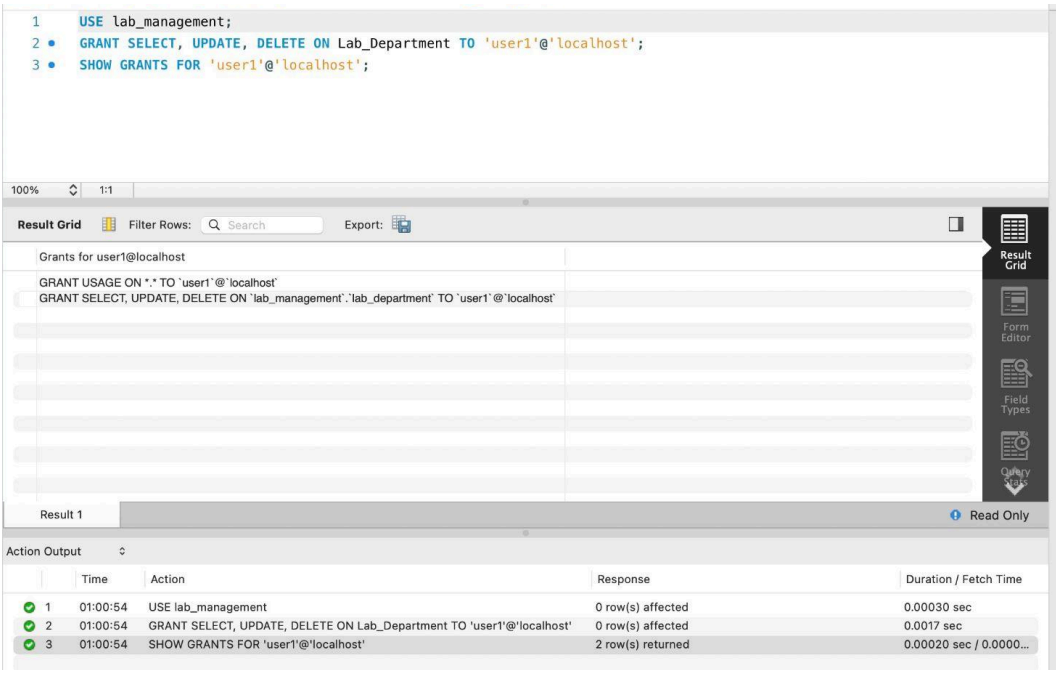



Fig. Query to Grant permissions to **user1** on **Lab_Departments** Table

Q4. Grant "user1" the following permissions on "view1":

- SELECT

We grant the permission using the root.

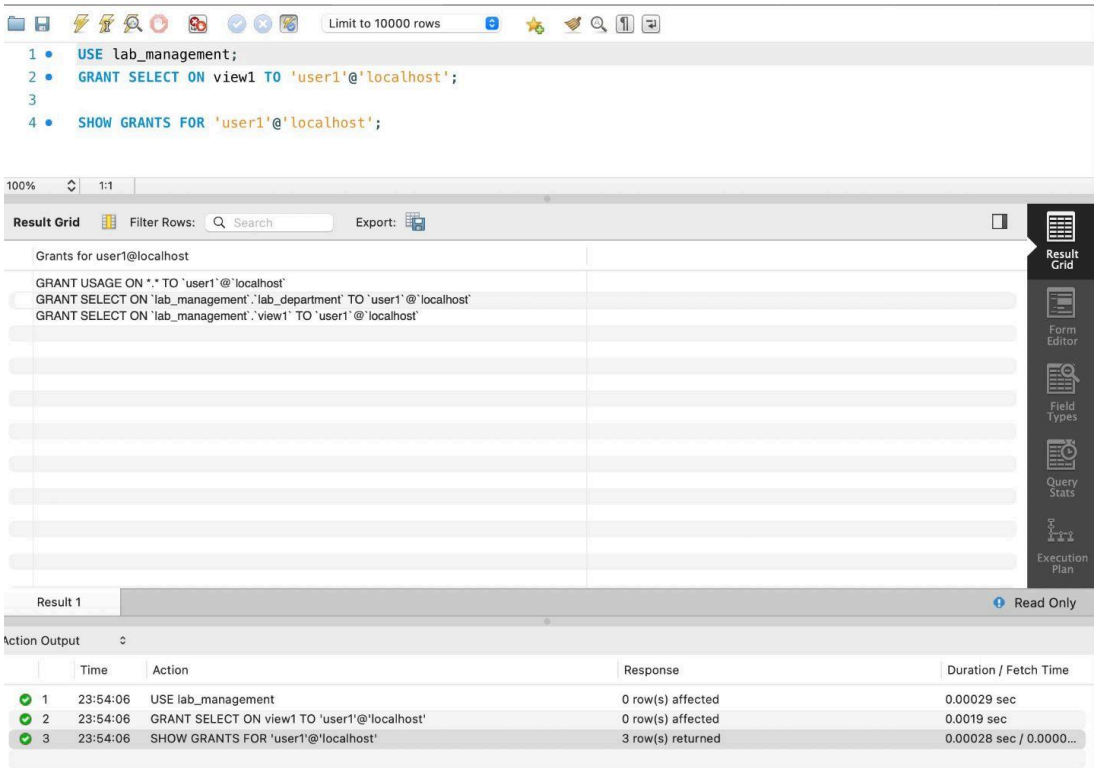



Fig. Query to Grant permissions to **user1** on **view1**

Q5. Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" and report your findings

**Note:** Before running the queries, connect as **user1** through the method demonstrated in Q1[G2].

**SELECT:**

We granted SELECT access of view1 and table1 to the user1 for selection so, as seen below, we could run the queries.

**Table1:**

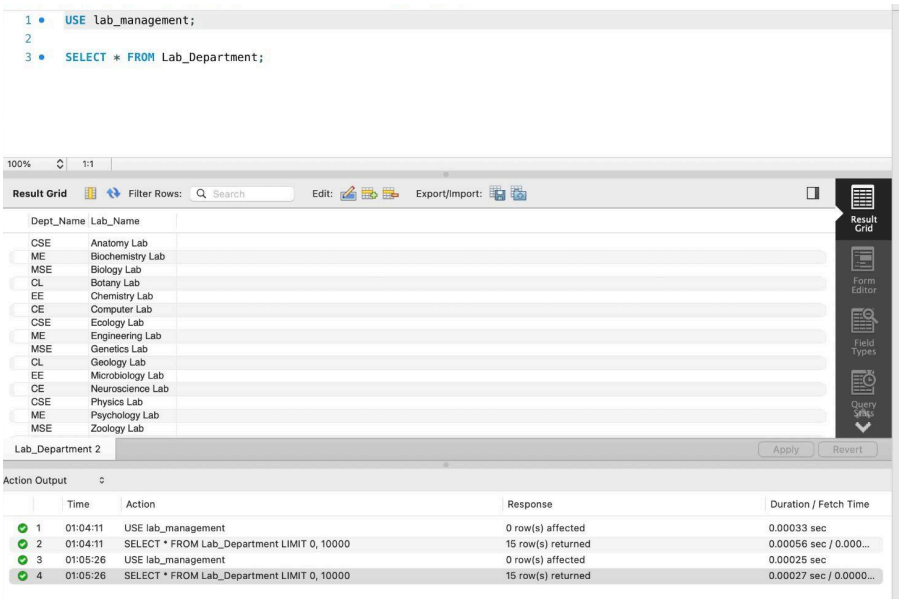



Fig. Select Query on **Lab_Department** as **user1**

**View1:**

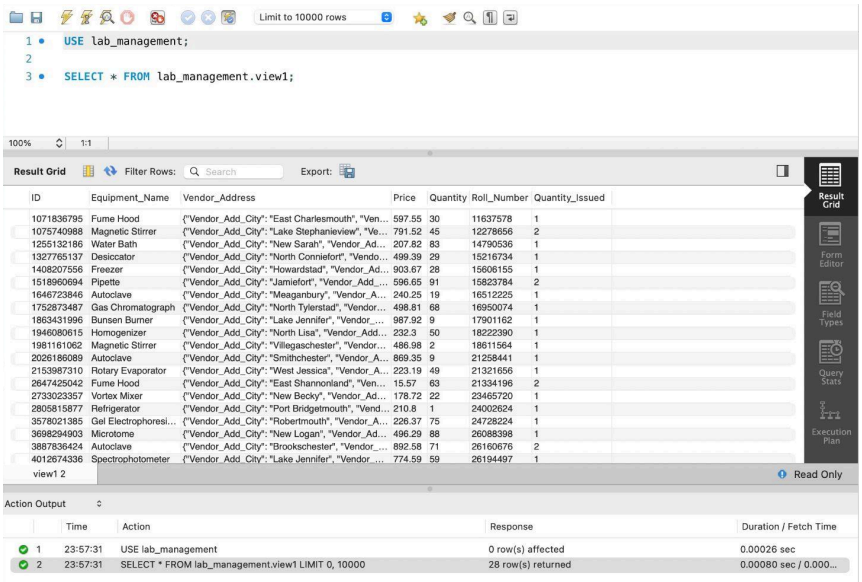



Fig. Select Query on **view1** as **user1**

## UPDATE:

1. **Table1:**

   Since user1 has permission to update the table, all the rows with Dept_Name=MSE have been changed with Dept_Name=CSE.

### Before running the Query:

```
1 • USE lab_management;
2
3 • SELECT * FROM Lab_Department;
```

Result Grid

| Dept_Name | Lab_Name |
|---|---|
| CSE | Anatomy Lab |
| ME | Biochemistry Lab |
| MSE | Biology Lab |
| CL | Botany Lab |
| EE | Chemistry Lab |
| CE | Computer Lab |
| CSE | Ecology Lab |
| ME | Engineering Lab |
| MSE | Genetics Lab |
| CL | Geology Lab |
| EE | Microbiology Lab |
| CE | Neuroscience Lab |
| CSE | Physics Lab |
| ME | Psychology Lab |
| MSE | Zoology Lab |

Lab_Department 2

Action Output

| | Time | Action | Response | Duration / Fetch Time |
|---|---|---|---|---|
| 1 | 01:04:11 | USE lab_management | 0 row(s) affected | 0.00033 sec |
| 2 | 01:04:11 | SELECT * FROM Lab_Department LIMIT 0, 10000 | 15 row(s) returned | 0.00056 sec / 0.000... |
| 3 | 01:05:26 | USE lab_management | 0 row(s) affected | 0.00025 sec |
| 4 | 01:05:26 | SELECT * FROM Lab_Department LIMIT 0, 10000 | 15 row(s) returned | 0.00027 sec / 0.0000... |

Fig. Data tuples present in **Lab_Department**.

### After:

```
1 • USE lab_management;
2
3 • UPDATE Lab_Department SET Dept_Name='CSE' WHERE Dept_Name='MSE';
4
5 • SELECT * FROM Lab_Department;
```

Result Grid

| Dept_Name | Lab_Name |
|---|---|
| CSE | Anatomy Lab |
| ME | Biochemistry Lab |
| CSE | Biology Lab |
| CL | Botany Lab |
| EE | Chemistry Lab |
| CE | Computer Lab |
| CSE | Ecology Lab |
| ME | Engineering Lab |
| CSE | Genetics Lab |
| CL | Geology Lab |
| EE | Microbiology Lab |
| CE | Neuroscience Lab |
| CSE | Physics Lab |
| ME | Psychology Lab |
| CSE | Zoology Lab |
| NULL | NULL |

Lab_Department 3

Action Output

| | Time | Action | Response | Duration / Fetch Time |
|---|---|---|---|---|
| 1 | 01:04:11 | USE lab_management | 0 row(s) affected | 0.00033 sec |
| 2 | 01:04:11 | SELECT * FROM Lab_Department LIMIT 0, 10000 | 15 row(s) returned | 0.00056 sec / 0.000... |
| 3 | 01:05:26 | USE lab_management | 0 row(s) affected | 0.00025 sec |
| 4 | 01:05:26 | SELECT * FROM Lab_Department LIMIT 0, 10000 | 15 row(s) returned | 0.00027 sec / 0.0000... |
| 5 | 01:06:32 | USE lab_management | 0 row(s) affected | 0.00026 sec |
| 6 | 01:06:32 | UPDATE Lab_Department SET Dept_Name='CSE' WHERE Dept_Name='MSE' | 3 row(s) affected Rows matched: 3 Changed: 3 War... | 0.0022 sec |
| 7 | 01:06:32 | SELECT * FROM Lab_Department LIMIT 0, 10000 | 15 row(s) returned | 0.00038 sec / 0.000... |

Fig. In **Lab_Department**, data tuples previously containing Dept_Name= MSE are updated to CSE.

2. **View1:**
   Since user1 doesn't have UPDATE permission on view1, so user1 couldn't change the "Quantity" column.

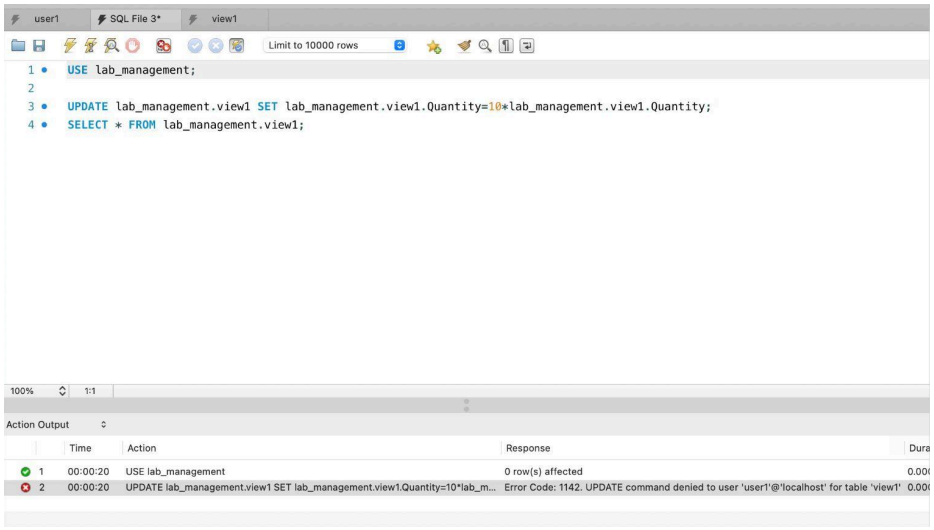



Fig. Failed UPDATE query on **view1** as **user1.**

**DELETE:**

1. **Table:**
   Since user1 has permission to update the table, all the rows with Dept_Name=CSE has been deleted.

   **Before:**

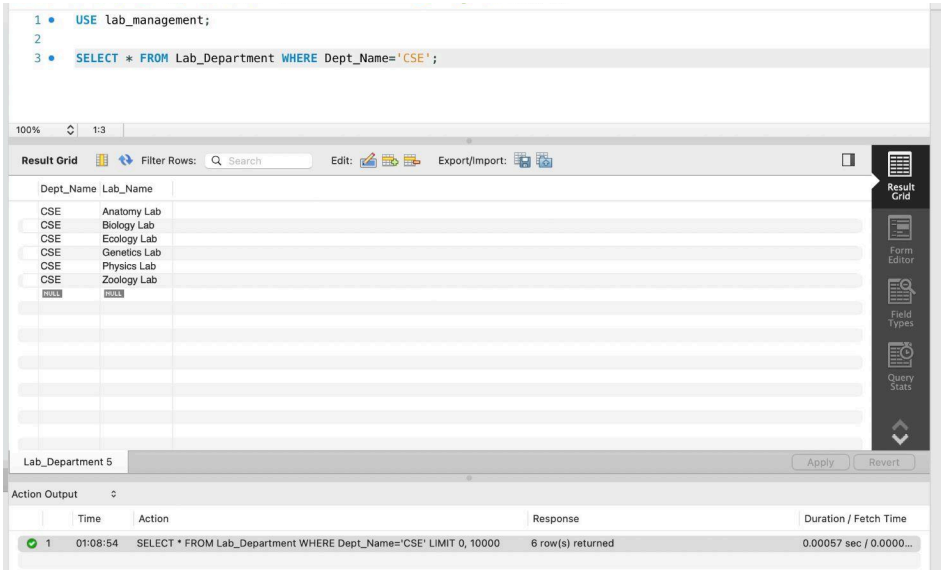



Fig. Data tuples containing Dept_Name=CSE, in **Lab_Department**.

   **After:**

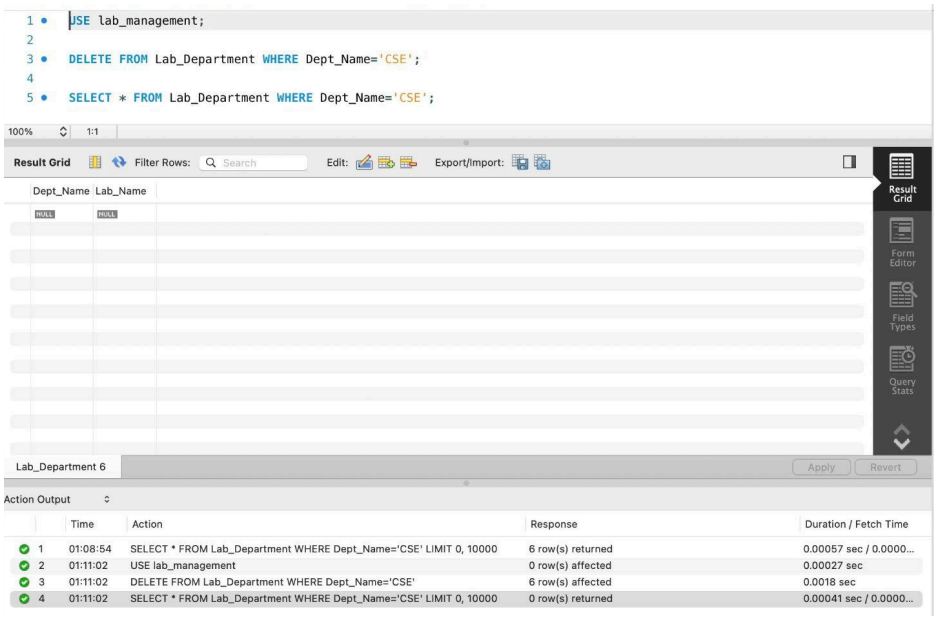



Fig. Data tuples containing Dept_Name=CSE, in **Lab_Department** after deletion.

2. **View1:**
   Since user1 doesn't have DELETE permission on view1, user1 couldn't delete from the view.

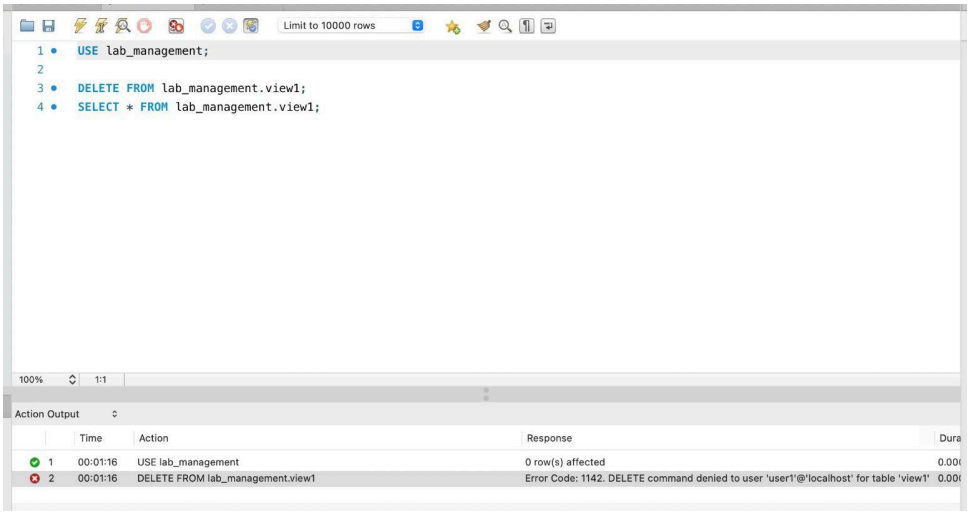



Fig. Failed DELETE query on **view1** as **user1.**

Q6. Revoke the UPDATE and DELETE permissions on "table1" for "user1" and report your findings.

The following command was used from the **root user** to revoke permissions from the Lab_Department table.

**Note:** The permissions need to be revoked when logged in as the root user.

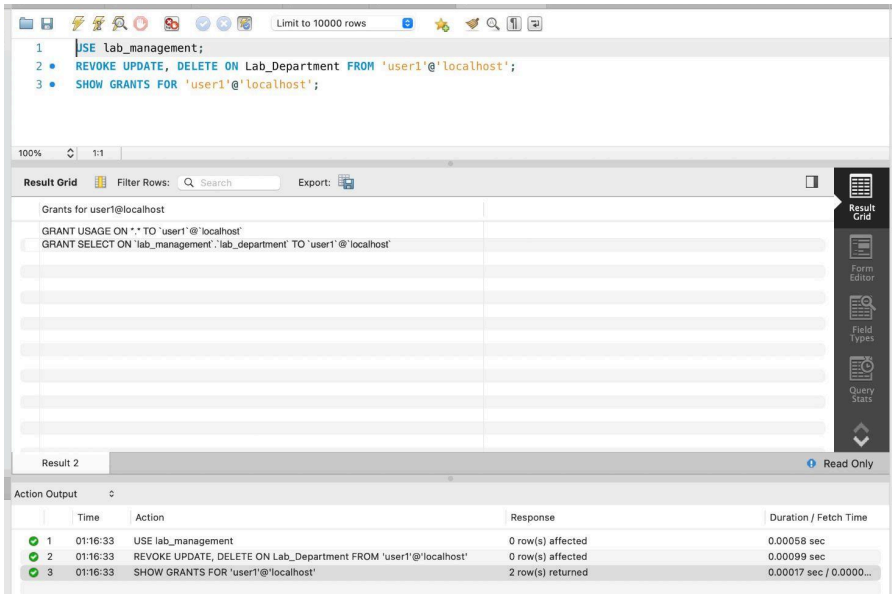



Fig. Revoking UPDATE and DELETE permissions on **Lab_Department** from **user1.**

Q7. Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" again.

**SELECT:**

1. **Table1:**
   Since SELECT permission was not REVOKED, user1 can still SELECT from table1(Lab_Department) and view1.

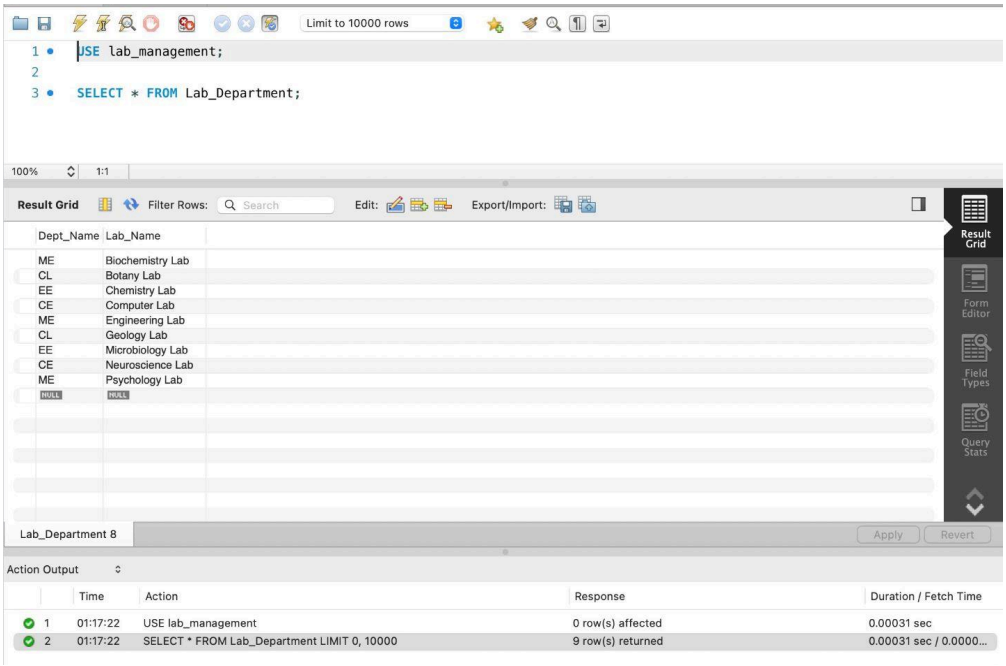



Fig. SELECT Query on **Lab_Department** as **user1.**

2. **View1:**

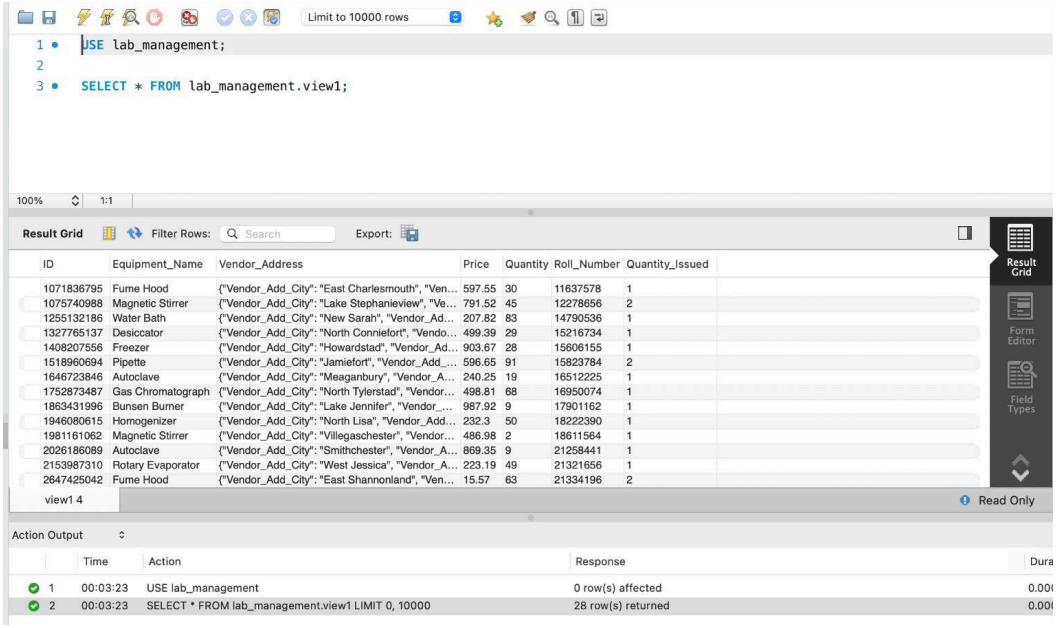



Fig. SELECT Query on **view1** as **user1.**

**UPDATE:**

- Since UPDATE permission was REVOKED, user1 can't UPDATE Entry(Dept_Name) into table1(Lab_Department).
- User1 doesn't have UPDATE permission for view1, so he/she can't update view1.

1. **Table1:**

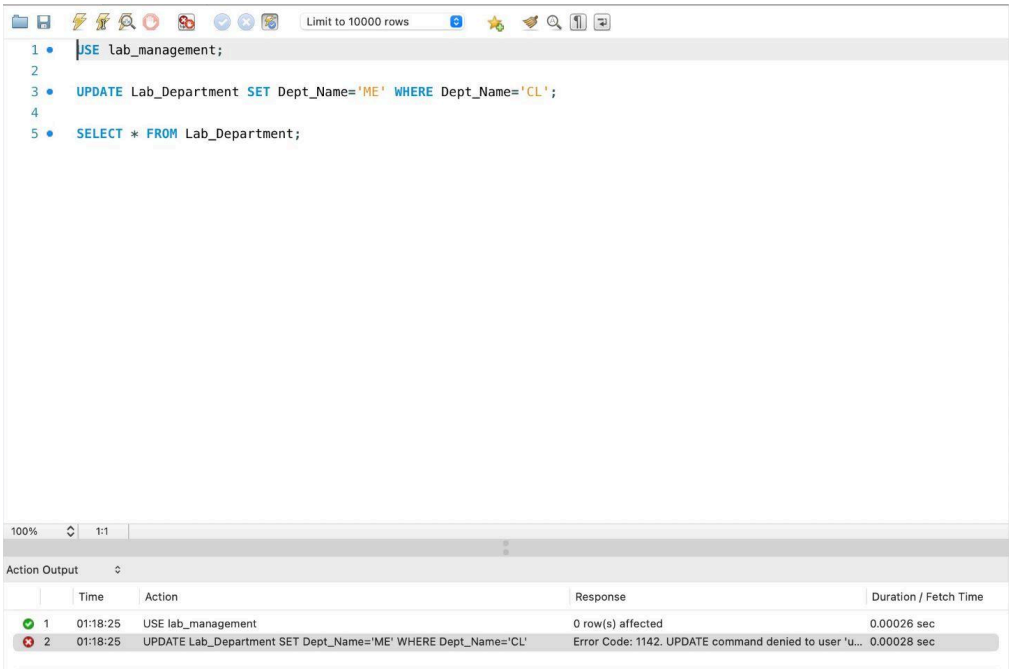



Fig. Failed UPDATE query on **Lab_Departement** as **user1.**

2. **View1:**

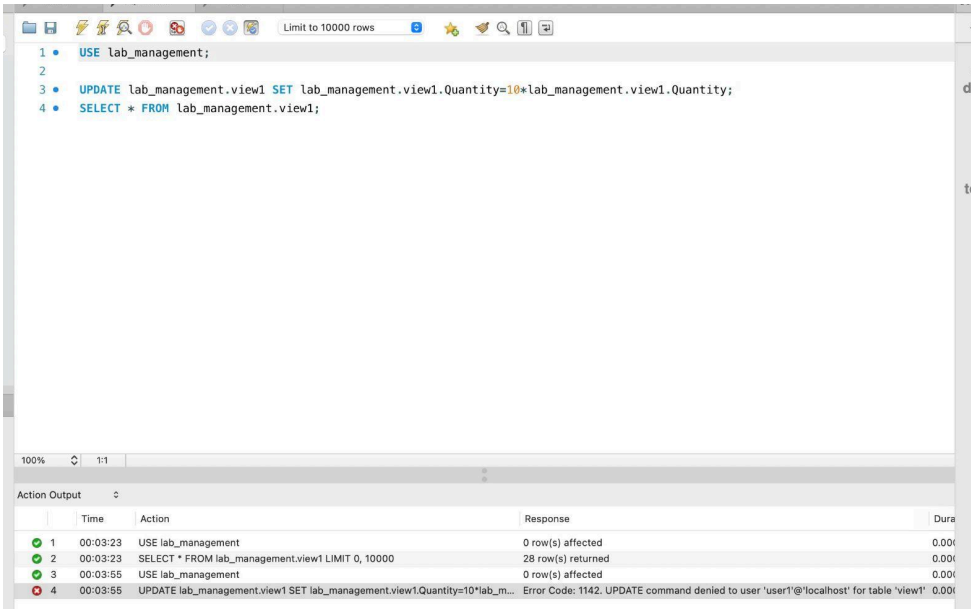



Fig. Failed UPDATE query on **view1** as **user1.**

DELETE:

- Since DELETE permission was REVOKED, user1 can't DELETE from table1(Lab_Department).
- User1 doesn't have DELETE permission for view1, so he/she can't delete from view1.

1. **Table1:**

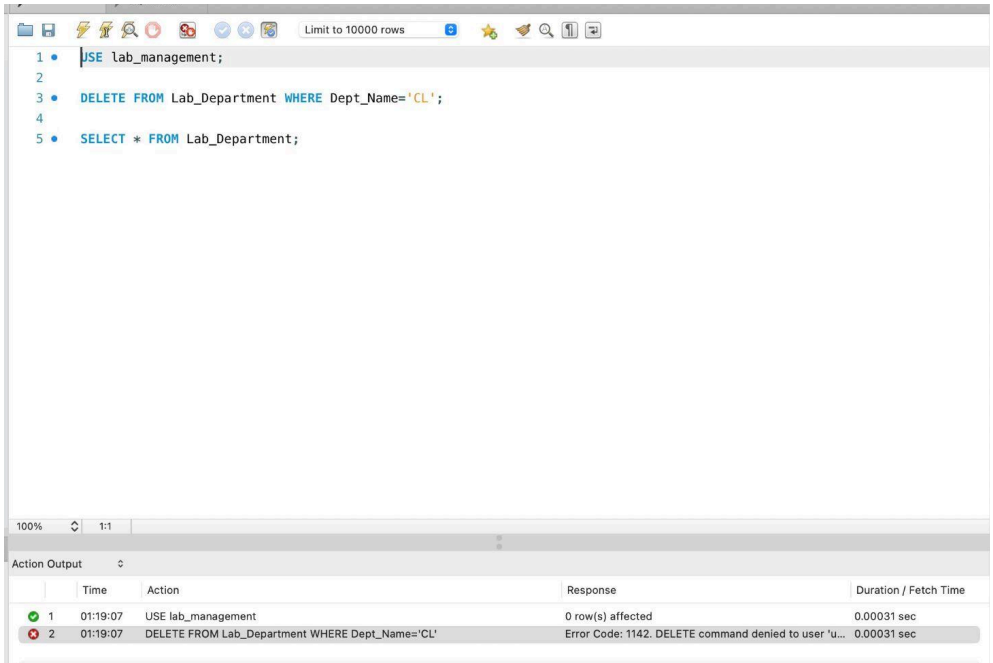



Fig. Failed DELETE query on **Lab_Departement** as **user1.**

2. **View1:**

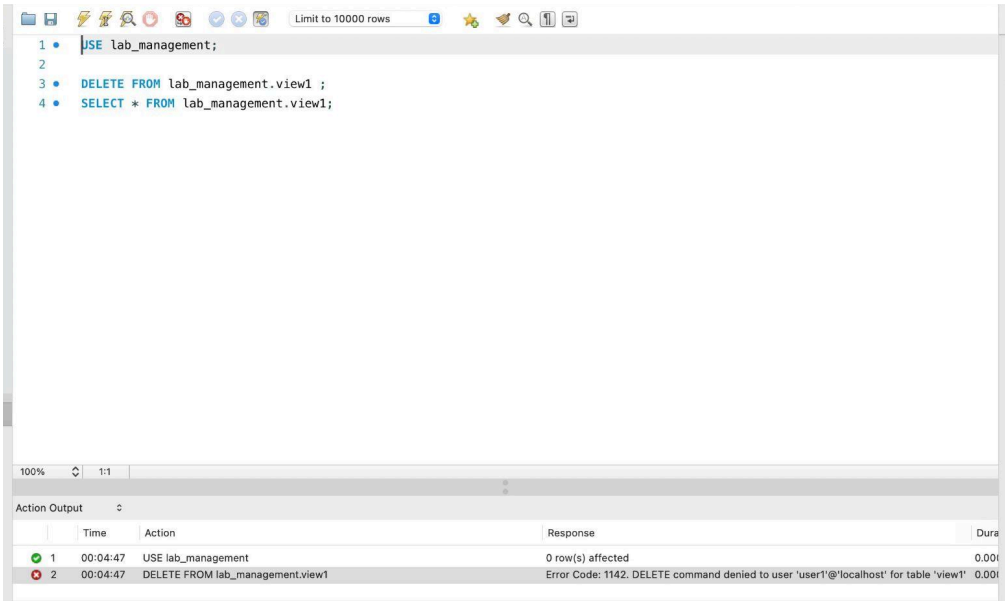



Fig. Failed DELETE query on **view1** as **user1.**

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

## Referential Integrity:-

Referential integrity is the logical dependency of a foreign key on a primary key. Any foreign key used in a certain table refers to the primary key of some other table. When there are inconsistencies in this, such as a reference to a primary key that does not exist, we face an error. This is known as violating referential integrity. Referential integrity can be violated in multiple ways. Let there be a Table A, which contains the primary key of another Table B as a foreign key. In such a case, referential integrity can be violated in the following ways:

- INSERT
  When we INSERT a new tuple in Table A, which contains the foreign key as “x”. If this “x” was non-existent as a primary key in Table B, Table A refers to a value that does not exist in Table B. This causes a violation of referential integrity.

- DELETE
  When we DELETE a particular tuple from Table B, whose primary key was being referenced in Table A as a foreign key, referential integrity is violated. This is because Table A refers to a value in Table B that does not exist now.

- UPDATE

When we UPDATE a particular tuple in Table B, changing a primary key referenced as a foreign key in Table A, we face a violation of referential integrity. This is because the value being referenced by Table A as a foreign key has been changed and does not exist anymore. Another way UPDATE can cause a violation of referential integrity is when we UPDATE a tuple of Table A, changing the foreign key to a value which does not exist in Table B. Since the foreign key of Table A is referencing a non-existent primary key in Table B, we encounter a violation of referential integrity.

The referential integrity can still be ensured under such violations by using certain constraints on the foreign keys, such as:

- ON DELETE CASCADE
  This action helps ensure referential integrity by handling violations caused when we DELETE a certain tuple in Table B, containing a primary key being referenced as a foreign key in Table A. This is done by deleting every tuple in Table A, which contains a foreign key referencing the primary key in Table B, which was deleted initially. Since every tuple in Table A that referenced the deleted primary key in Table B has been deleted, there does not exist any foreign keys whose primary keys do not exist. Hence, referential integrity is maintained.

- ON UPDATE CASCADE
  This action helps maintain referential integrity by handling violations caused when we UPDATE a certain tuple in Table B, containing a primary key being referenced as a foreign key in Table A. This is done by updating every tuple in Table A, which contains a foreign key referencing the primary key in Table B, which was updated initially. Since every tuple in Table A which referenced the updated primary key in Table B has been updated as well, there do not exist any foreign keys whose primary keys have been changed. Hence, referential integrity is maintained.

Note: When using CASCADE we need to add this part to wherever the foreign key is referenced.

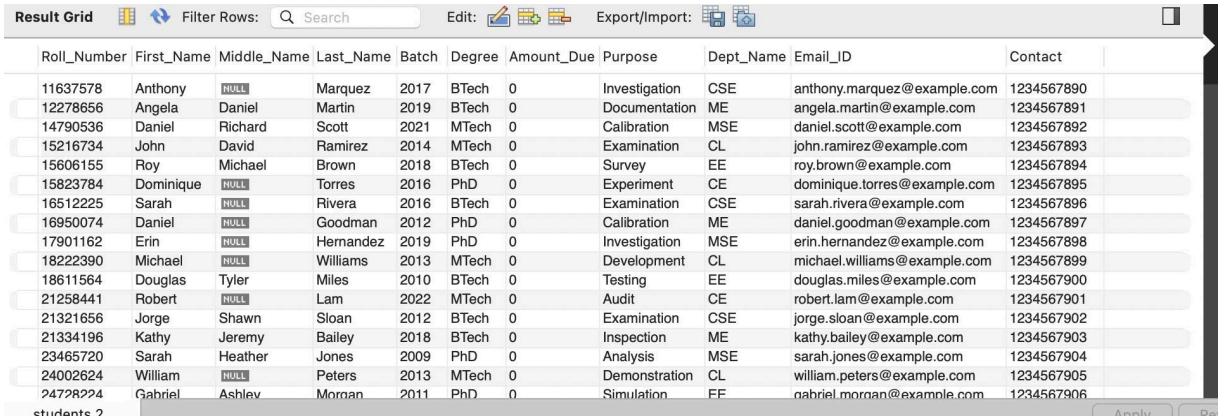
## Examples:

Case 1)
**UPDATE:**

Query: `UPDATE students SET students.Roll_Number=12000001 WHERE students.Roll_Number=12278656;`

In this case, we used “students” and “proj_taken” tables, here Roll_Number is a primary key for the “students” table and a foreign key for the “proj_taken” table. When we update the student’s roll number in the “students” table, the Roll_Number attribute in the “proj_taken” table is not updated.

**Students Table:**

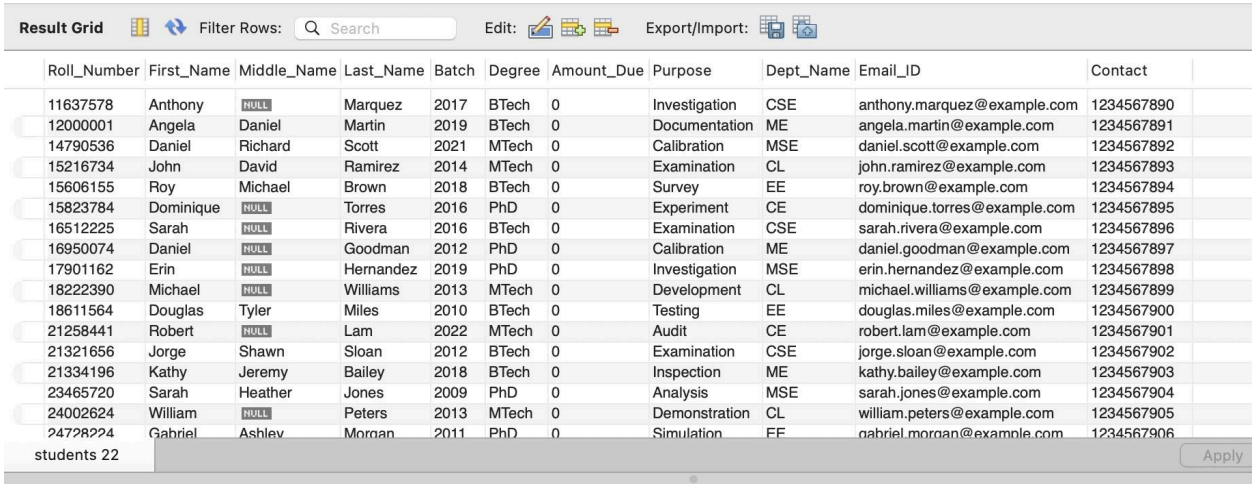


Result Grid  Filter Rows: Search  Edit:  Export/Import:

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 12278656 | Angela | Daniel | Martin | 2019 | BTech | 0 | Documentation | ME | angela.martin@example.com | 1234567891 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |
| 18611564 | Douglas | Tyler | Miles | 2010 | BTech | 0 | Testing | EE | douglas.miles@example.com | 1234567900 |
| 21258441 | Robert | NULL | Lam | 2022 | MTech | 0 | Audit | CE | robert.lam@example.com | 1234567901 |
| 21321656 | Jorge | Shawn | Sloan | 2012 | BTech | 0 | Examination | CSE | jorge.sloan@example.com | 1234567902 |
| 21334196 | Kathy | Jeremy | Bailey | 2018 | BTech | 0 | Inspection | ME | kathy.bailey@example.com | 1234567903 |
| 23465720 | Sarah | Heather | Jones | 2009 | PhD | 0 | Analysis | MSE | sarah.jones@example.com | 1234567904 |
| 24002624 | William | NULL | Peters | 2013 | MTech | 0 | Demonstration | CL | william.peters@example.com | 1234567905 |
| 24728224 | Gabriel | Ashley | Morgan | 2011 | PhD | 0 | Simulation | EE | gabriel.morgan@example.com | 1234567906 |

students 2

Fig. **Students** Table before the UPDATE query.



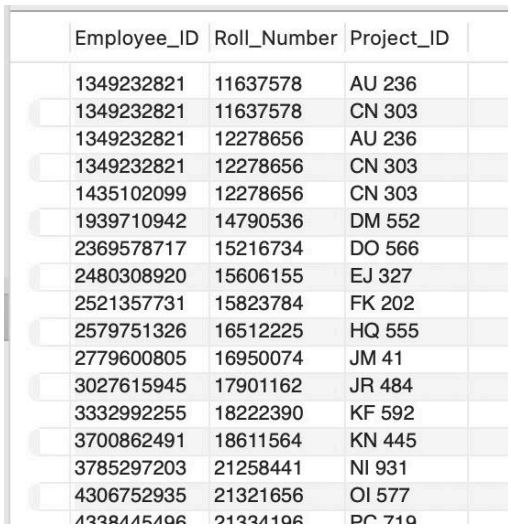
Result Grid  Filter Rows: Search  Edit:  Export/Import:

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 12000001 | Angela | Daniel | Martin | 2019 | BTech | 0 | Documentation | ME | angela.martin@example.com | 1234567891 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |
| 18611564 | Douglas | Tyler | Miles | 2010 | BTech | 0 | Testing | EE | douglas.miles@example.com | 1234567900 |
| 21258441 | Robert | NULL | Lam | 2022 | MTech | 0 | Audit | CE | robert.lam@example.com | 1234567901 |
| 21321656 | Jorge | Shawn | Sloan | 2012 | BTech | 0 | Examination | CSE | jorge.sloan@example.com | 1234567902 |
| 21334196 | Kathy | Jeremy | Bailey | 2018 | BTech | 0 | Inspection | ME | kathy.bailey@example.com | 1234567903 |
| 23465720 | Sarah | Heather | Jones | 2009 | PhD | 0 | Analysis | MSE | sarah.jones@example.com | 1234567904 |
| 24002624 | William | NULL | Peters | 2013 | MTech | 0 | Demonstration | CL | william.peters@example.com | 1234567905 |
| 24728224 | Gabriel | Ashley | Morgan | 2011 | PhD | 0 | Simulation | EE | gabriel.morgan@example.com | 1234567906 |

students 22  Apply

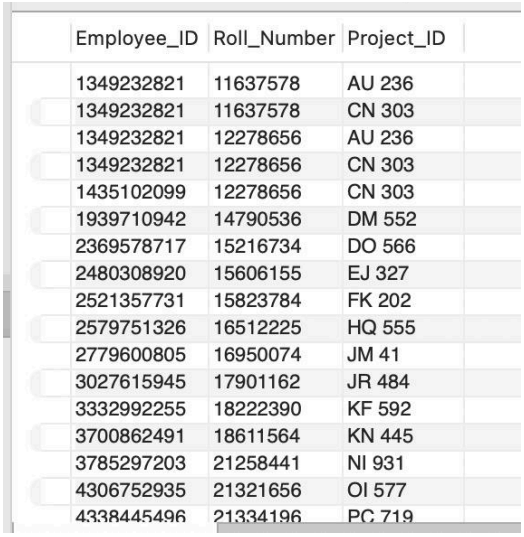
Fig. **Students** Table before the UPDATE query.

## Proj_taken table:



| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12278656 | AU 236 |
| 1349232821 | 12278656 | CN 303 |
| 1435102099 | 12278656 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |
| 3785297203 | 21258441 | NI 931 |
| 4306752935 | 21321656 | OI 577 |
| 4338445496 | 21334196 | PC 719 |

Fig. **Proj_Taken** Table before the UPDATE query.



| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12278656 | AU 236 |
| 1349232821 | 12278656 | CN 303 |
| 1435102099 | 12278656 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |
| 3785297203 | 21258441 | NI 931 |
| 4306752935 | 21321656 | OI 577 |
| 4338445496 | 21334196 | PC 719 |

Fig. **Proj_taken** Table after the UPDATE query.

**Solution:**

```
CREATE TABLE `proj_taken` (
  `Employee_ID` bigint NOT NULL,
  `Roll_Number` int NOT NULL,
  `Project_ID` varchar(255) NOT NULL,
  PRIMARY KEY (`Employee_ID`,`Roll_Number`,`Project_ID`),
  KEY `Roll_Number` (`Roll_Number`),
  KEY `Project_ID` (`Project_ID`),
  CONSTRAINT `proj_taken_fk_1` FOREIGN KEY (`Employee_ID`) REFERENCES `professor` (`Employee_ID`)
  CONSTRAINT `proj_taken_fk_2` FOREIGN KEY (`Roll_Number`) REFERENCES `students` (`Roll_Number`)
  ON UPDATE CASCADE,
  CONSTRAINT `proj_taken_fk_3` FOREIGN KEY (`Project_ID`) REFERENCES `project` (`Project_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Fig. Adding the ON UPDATE CASCADE in the **Proj_taken** Schema where foreign key is referenced.

We used ON UPDATE CASCADE to ensure that whenever the primary key is updated, the corresponding table entries, wherever that primary key is referenced as the foreign key, get updated as well.

**Note:** The "students" table was reset;

Query: `UPDATE students SET students.Roll_Number=12000001 WHERE students.Roll_Number=12278656;`

100% 1:1

Result Grid Filter Rows: Search

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12000001 | AU 236 |
| 1349232821 | 12000001 | CN 303 |
| 1435102099 | 12000001 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |

Fig. **Proj_taken** Table after the UPDATE query, this time the corresponding values are updated.

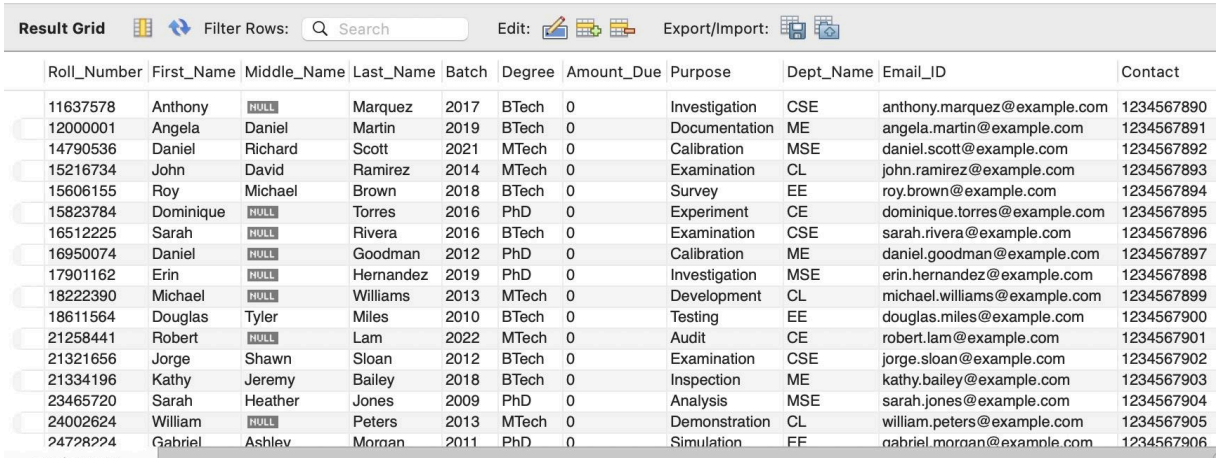
Here we can see Roll_Number in the "proj_taken" table is updated to 12000001.

Case 2)
**DELETE:**

In this case, we used “students” and “proj_taken” tables, here Roll_Number is a primary key for the “students” table and a foreign key for the “proj_taken” table. When we delete a row from the “students” table with a particular roll number, its respective entries from the “proj_taken” table are not deleted.

Query: `DELETE FROM students WHERE students.Roll_Number=12000001;`

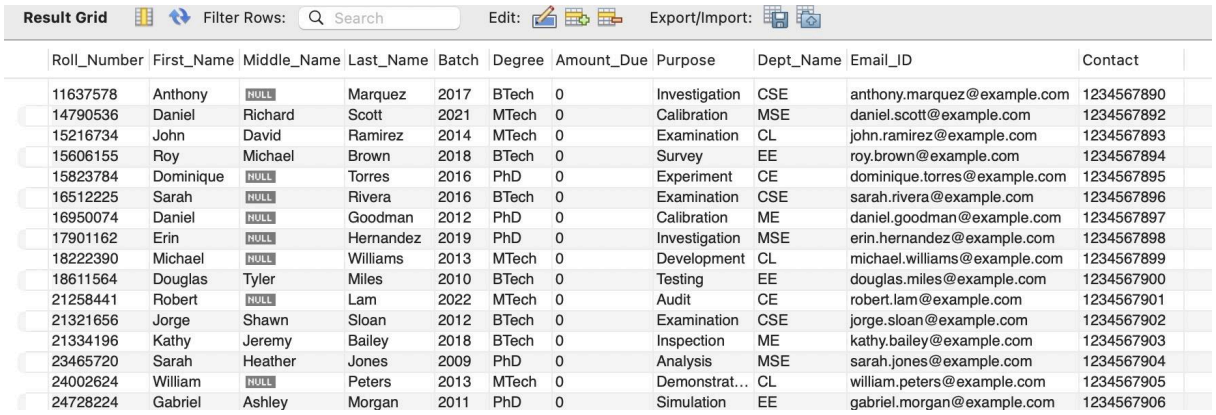
**Students Table:**



Result Grid  Filter Rows: Search  Edit:  Export/Import:

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 12000001 | Angela | Daniel | Martin | 2019 | BTech | 0 | Documentation | ME | angela.martin@example.com | 1234567891 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |
| 18611564 | Douglas | Tyler | Miles | 2010 | BTech | 0 | Testing | EE | douglas.miles@example.com | 1234567900 |
| 21258441 | Robert | NULL | Lam | 2022 | MTech | 0 | Audit | CE | robert.lam@example.com | 1234567901 |
| 21321656 | Jorge | Shawn | Sloan | 2012 | BTech | 0 | Examination | CSE | jorge.sloan@example.com | 1234567902 |
| 21334196 | Kathy | Jeremy | Bailey | 2018 | BTech | 0 | Inspection | ME | kathy.bailey@example.com | 1234567903 |
| 23465720 | Sarah | Heather | Jones | 2009 | PhD | 0 | Analysis | MSE | sarah.jones@example.com | 1234567904 |
| 24002624 | William | NULL | Peters | 2013 | MTech | 0 | Demonstration | CL | william.peters@example.com | 1234567905 |
| 24728224 | Gabriel | Ashley | Morgan | 2011 | PhD | 0 | Simulation | EE | gabriel.morgan@example.com | 1234567906 |

Fig. **Students** Table before the DELETE query.



Result Grid  Filter Rows: Search  Edit:  Export/Import:

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |
| 18611564 | Douglas | Tyler | Miles | 2010 | BTech | 0 | Testing | EE | douglas.miles@example.com | 1234567900 |
| 21258441 | Robert | NULL | Lam | 2022 | MTech | 0 | Audit | CE | robert.lam@example.com | 1234567901 |
| 21321656 | Jorge | Shawn | Sloan | 2012 | BTech | 0 | Examination | CSE | jorge.sloan@example.com | 1234567902 |
| 21334196 | Kathy | Jeremy | Bailey | 2018 | BTech | 0 | Inspection | ME | kathy.bailey@example.com | 1234567903 |
| 23465720 | Sarah | Heather | Jones | 2009 | PhD | 0 | Analysis | MSE | sarah.jones@example.com | 1234567904 |
| 24002624 | William | NULL | Peters | 2013 | MTech | 0 | Demonstrat... | CL | william.peters@example.com | 1234567905 |
| 24728224 | Gabriel | Ashley | Morgan | 2011 | PhD | 0 | Simulation | EE | gabriel.morgan@example.com | 1234567906 |

Fig. **Students** Table after the DELETE query.

**Proj_taken table:**

Result Grid | Filter Rows: Search

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12000001 | AU 236 |
| 1349232821 | 12000001 | CN 303 |
| 1435102099 | 12000001 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |
| 3785297203 | 21258441 | NI 931 |
| 4306752935 | 21321656 | OI 577 |

Fig. **Proj_Taken** Table before the DELETE query.

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12000001 | AU 236 |
| 1349232821 | 12000001 | CN 303 |
| 1435102099 | 12000001 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |
| 3785297203 | 21258441 | NI 931 |
| 4306752935 | 21321656 | OI 577 |

Fig. **Proj_taken** Table after the DELETE query.

## Solution:

```
DROP TABLE IF EXISTS `proj_taken`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `proj_taken` (
  `Employee_ID` bigint NOT NULL,
  `Roll_Number` int NOT NULL,
  `Project_ID` varchar(255) NOT NULL,
  PRIMARY KEY (`Employee_ID`,`Roll_Number`,`Project_ID`),
  KEY `Roll_Number` (`Roll_Number`),
  KEY `Project_ID` (`Project_ID`),
  CONSTRAINT `proj_taken_fk_1` FOREIGN KEY (`Employee_ID`) REFERENCES `professor` (`Employee_ID`),
  CONSTRAINT `proj_taken_fk_2` FOREIGN KEY (`Roll_Number`) REFERENCES `students` (`Roll_Number`)
  ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT `proj_taken_fk_3` FOREIGN KEY (`Project_ID`) REFERENCES `project` (`Project_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

Fig. Adding the ON DELETE CASCADE in the **Proj_taken** Schema where the foreign key is referenced.

Query: `DELETE FROM students WHERE students.Roll_Number=12000001;`

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |
| 3785297203 | 21258441 | NI 931 |
| 4306752935 | 21321656 | OI 577 |
| 4338445496 | 21334196 | PC 719 |
| 4374962421 | 23465720 | PH 783 |
| 4796673686 | 24002624 | SO 27 |

Fig. **Proj_taken** Table after the DELETE query, entries with Roll_Number=12000001 are deleted.

We used ON DELETE CASCADE to ensure that whenever the primary key is updated, the corresponding table entries, wherever that primary key is referenced as the foreign key, get updated as well.

Case 3)
**Update: with foreign key checks = 1**

When updating the roll number in the “proj_taken” table where the roll number is a foreign key, it throws an error due to a foreign key constraint failing.

Query: `UPDATE proj_taken SET proj_taken.Roll_Number=12000000 WHERE proj_taken.Roll_Number=16950074;`

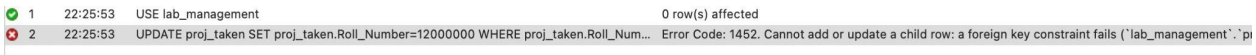



Fig. Failed UPDATE query on **Proj_taken** Table with foreign_key_checks set to 1.

Case 4)
**Delete: with foreign key checks = 1**

When we try to delete a row with a particular roll number from the “proj_taken” table, where Roll_Number is the foreign key, it also deletes the corresponding entry from the “students” table where Roll_Numer is the primary key.

Query: `DELETE FROM proj_taken WHERE proj_taken.Roll_Number=12278656;`

**Students Table:**

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 12278656 | Angela | Daniel | Martin | 2019 | BTech | 0 | Documentation | ME | angela.martin@example.com | 1234567891 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |

Fig. **Students** Table before the DELETE query on **Proj_taken**.

| Roll_Number | First_Name | Middle_Name | Last_Name | Batch | Degree | Amount_Due | Purpose | Dept_Name | Email_ID | Contact |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 12278656 | Angela | Daniel | Martin | 2019 | BTech | 0 | Documentation | ME | angela.martin@example.com | 1234567891 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |
| 18611564 | Douglas | Tyler | Miles | 2010 | BTech | 0 | Testing | EE | douglas.miles@example.com | 1234567900 |

Fig. **Students** Table after the DELETE query on **Proj_taken**.

**Proj_taken table:**

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12278656 | AU 236 |
| 1349232821 | 12278656 | CN 303 |
| 1435102099 | 12278656 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |

Fig. **Proj_Taken** Table before the DELETE query.

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |
| 3785297203 | 21258441 | NI 931 |
| 4306752935 | 21321656 | OI 577 |

Fig. **Proj_taken** Table after the DELETE query.

Case 5)

**Update: Foreign key checks = 0;**

When foreign key checks are set to 0, if we update the roll number in the “proj_taken” table, It only changes in the “proj_taken” table, and entries of the “students” table remains same.

Query: `UPDATE proj_taken SET proj_taken.Roll_Number=12000000 WHERE proj_taken.Roll_Number=12278656;`

**Students Table:**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 12278656 | Angela | Daniel | Martin | 2019 | BTech | 0 | Documentation | ME | angela.martin@example.com | 1234567891 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |
| 18611564 | Douglas | Tyler | Miles | 2010 | BTech | 0 | Testing | EE | douglas.miles@example.com | 1234567900 |

Fig. **Students** Table before the UPDATE query on **Proj_taken**.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 11637578 | Anthony | NULL | Marquez | 2017 | BTech | 0 | Investigation | CSE | anthony.marquez@example.com | 1234567890 |
| 12278656 | Angela | Daniel | Martin | 2019 | BTech | 0 | Documentation | ME | angela.martin@example.com | 1234567891 |
| 14790536 | Daniel | Richard | Scott | 2021 | MTech | 0 | Calibration | MSE | daniel.scott@example.com | 1234567892 |
| 15216734 | John | David | Ramirez | 2014 | MTech | 0 | Examination | CL | john.ramirez@example.com | 1234567893 |
| 15606155 | Roy | Michael | Brown | 2018 | BTech | 0 | Survey | EE | roy.brown@example.com | 1234567894 |
| 15823784 | Dominique | NULL | Torres | 2016 | PhD | 0 | Experiment | CE | dominique.torres@example.com | 1234567895 |
| 16512225 | Sarah | NULL | Rivera | 2016 | BTech | 0 | Examination | CSE | sarah.rivera@example.com | 1234567896 |
| 16950074 | Daniel | NULL | Goodman | 2012 | PhD | 0 | Calibration | ME | daniel.goodman@example.com | 1234567897 |
| 17901162 | Erin | NULL | Hernandez | 2019 | PhD | 0 | Investigation | MSE | erin.hernandez@example.com | 1234567898 |
| 18222390 | Michael | NULL | Williams | 2013 | MTech | 0 | Development | CL | michael.williams@example.com | 1234567899 |
| 18611564 | Douglas | Tyler | Miles | 2010 | BTech | 0 | Testing | EE | douglas.miles@example.com | 1234567900 |

Fig. **Students** Table after the UPDATE query on **Proj_taken**.

**Proj_taken table:**

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12278656 | AU 236 |
| 1349232821 | 12278656 | CN 303 |
| 1435102099 | 12278656 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |

Fig. **Proj_Taken** Table before the UPDATE query.

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12000000 | AU 236 |
| 1349232821 | 12000000 | CN 303 |
| 1435102099 | 12000000 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |

Fig. **Proj_taken** Table after the UPDATE query.

Case 6)

**DELETE** with **Foreign key checks = 0** behaves similarly to case 4, as demonstrated above.

Case 7)

**INSERT**

When we insert a new tuple/entry in the table “proj_taken” where the student’s roll number is a foreign key, then it inserts a new entry in the table, but no such student is present in the “students” table. So it violates the referential integrity. The solution for this is to insert with foreign key checks=1. After doing that, it will now show an error message(“cannot add or update a child row, a foreign key constraint fails”).

**INSERT** with **Foreign key checks = 1**

Query: `INSERT INTO `proj_taken` VALUES (1349232821,12000001,'CS 799');`

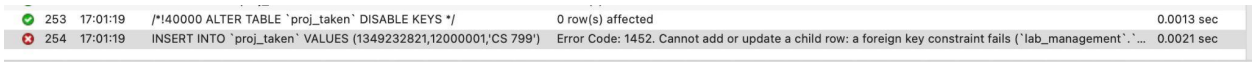



Fig. INSERT query failed in **Proj_Taken** Table due to the Foreign Key constraints.

**INSERT** with **Foreign key checks = 0**

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12278656 | AU 236 |
| 1349232821 | 12278656 | CN 303 |
| 1435102099 | 12278656 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |
| 3332992255 | 18222390 | KF 592 |
| 3700862491 | 18611564 | KN 445 |

Fig. **Proj_Taken** Table before the INSERT query.

| Employee_ID | Roll_Number | Project_ID |
|---|---|---|
| 1349232821 | 11637578 | AU 236 |
| 1349232821 | 11637578 | CN 303 |
| 1349232821 | 12000001 | CS 799 |
| 1349232821 | 12278656 | AU 236 |
| 1349232821 | 12278656 | CN 303 |
| 1435102099 | 12278656 | CN 303 |
| 1939710942 | 14790536 | DM 552 |
| 2369578717 | 15216734 | DO 566 |
| 2480308920 | 15606155 | EJ 327 |
| 2521357731 | 15823784 | FK 202 |
| 2579751326 | 16512225 | HQ 555 |
| 2779600805 | 16950074 | JM 41 |
| 3027615945 | 17901162 | JR 484 |

Fig. **Proj_taken** Table after the INSERT query.

## 3. Responsibility of G1 & G2:

**Operation 1:**

Here we update the roll number of a student by a roll number of already registered students. We use the UPDATE operator to update the query, SET operator to assign the roll number.

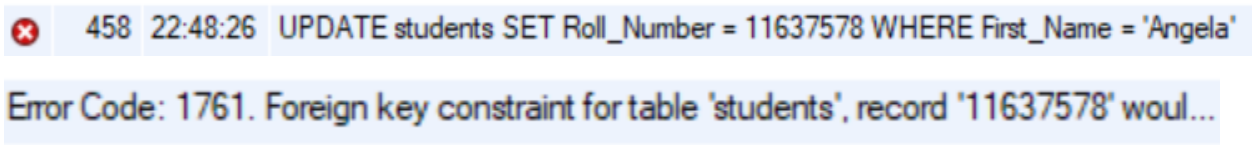
**SQL Query:**

```
UPDATE    students
SET Roll_Number  =  11637578
WHERE   First_Name  =  'Angela';
```

**Relational Algebra:**

$$\text{students} \leftarrow (\text{students} - \sigma_{\text{First\_Name}} = \text{'Angela'}(\text{students})) \cup (\sigma_{\text{First\_Name}} = \text{'Angela'}(\text{students}))[\text{Roll\_Number} := 11637578]$$

**Error Message:**




**The operation satisfies 1 (error due to violation of constraints specified by G1)**

Reason for Error:
We have kept a constraint on the student's roll number to be unique. Here, we are updating a student with a specific name's roll number with an already existing roll number. This violates the constraint of roll number to be unique. Hence, this throws an error.

**Operation 2:**
Here we update the email of a student by a wrong email. We use the UPDATE operator to update the query, SET operator to assign the email.

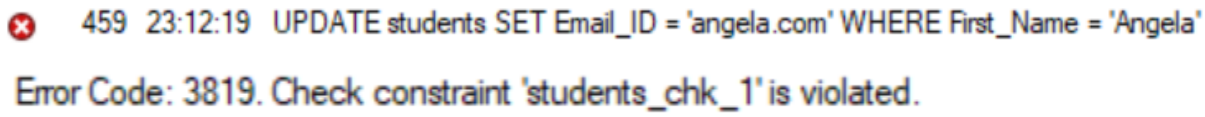
**SQL Query:**

```
UPDATE students
SET Email_ID = 'angela.com'
WHERE First_Name = 'Angela';
```

**Relational Algebra:**

$$\text{students} \leftarrow (\text{students} - \sigma_{\text{First\_Name}} = \text{'Angela'}(\text{students}) \cup (\sigma_{\text{First\_Name}} = \text{'Angela'}(\text{students}))[\text{Email\_ID} := \text{angela.com}]$$

**Error Message:**




**The operation satisfies 1 (error due to violation of constraints specified by G1)**
Reason for Error:
The email attribute is specified with a regular expression. As we updated the email value with an erroneous one( without an @ symbol), we got an error message.

**Operation 3:**
Here we are finding the maximum salary for staff members in the lab. For this query we use the CROSS JOIN, and NOT operator. We create a cross join table and select the salaries which are lower than salaries

in the second table. Atlast, we do a not operation to filter out the maximum salary which would not be present in the table.

**The operation satisfies 3 ( Aggregate Functions, Comparison, Use of NOT and Cross Join).**

List of Salaries in Descending order:

```
686 •  select salary from staff order by salary desc;
687
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| salary |
|---|
| 99937.1 |
| 96318.4 |
| 95889.6 |
| 95749.9 |
| 90618 |

**SQL Query:**

```
select s.salary from staff s
where s.salary not in
(select s2.salary from staff s2 cross join (select * from staff) x
where s2.salary < x.salary );
```

**Result of Query:**

```
681 •  select s.salary from staff s
682    where s.salary not in
683    (select s2.salary from staff s2 cross join (select * from staff) x where s2.salary < x.salary );
684
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| salary |
|---|
| 99937.1 |

**Relational Algebra:**

$$\prod_{\text{salary}}(staff) - \prod_{\text{staff.salary}}(\sigma_{staff.salary<x.salary}(staff \times \rho_x(staff))$$

**Operation 4:**

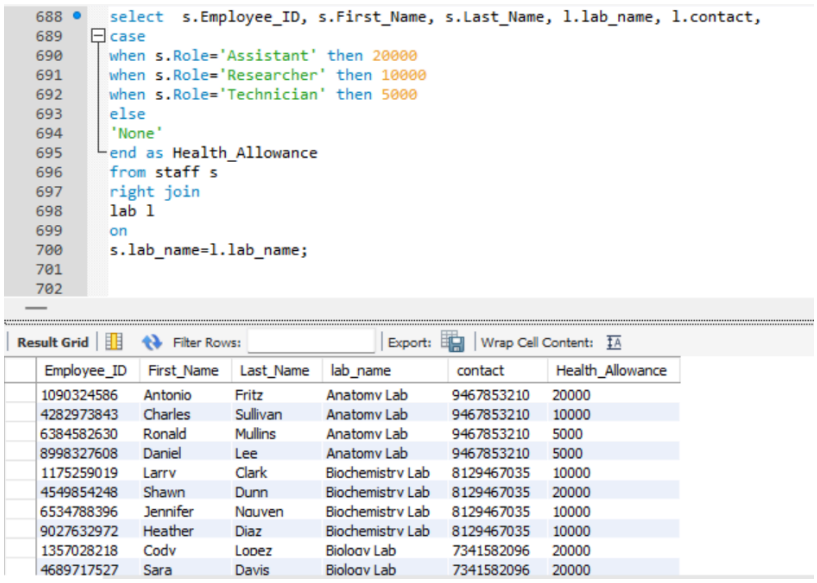
Here we are making a new column renaming it to Health Allowance using AS. We use CASE and ELSE statements to assign allowance to staff members having different roles. We also perform a RIGHT JOIN of staff with the lab to show data of lab contacts.

**The operation satisfies 3 (Right Join, Case Constructs, Renaming).**

**SQL Query:**

```
select  s.Employee_ID, s.First_Name, s.Last_Name, l.lab_name,
l.contact,
case
when s.Role='Assistant' then 20000
when s.Role='Researcher' then 10000
when s.Role='Technician' then 5000
else 'None'
end as Health_Allowance
from staff s
right join
lab l
on
s.lab_name = l.lab_name;
```

**Result of Query:**



```
select  s.Employee_ID, s.First_Name, s.Last_Name, l.lab_name, l.contact,
case
when s.Role='Assistant' then 20000
when s.Role='Researcher' then 10000
when s.Role='Technician' then 5000
else
'None'
end as Health_Allowance
from staff s
right join
lab l
on
s.lab_name=l.lab_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Employee_ID | First_Name | Last_Name | lab_name | contact | Health_Allowance |
|---|---|---|---|---|---|
| 1090324586 | Antonio | Fritz | Anatomy Lab | 9467853210 | 20000 |
| 4282973843 | Charles | Sullivan | Anatomy Lab | 9467853210 | 10000 |
| 6384582630 | Ronald | Mullins | Anatomy Lab | 9467853210 | 5000 |
| 8998327608 | Daniel | Lee | Anatomy Lab | 9467853210 | 5000 |
| 1175259019 | Larry | Clark | Biochemistry Lab | 8129467035 | 10000 |
| 4549854248 | Shawn | Dunn | Biochemistry Lab | 8129467035 | 20000 |
| 6534788396 | Jennifer | Nguyen | Biochemistry Lab | 8129467035 | 10000 |
| 9027632972 | Heather | Diaz | Biochemistry Lab | 8129467035 | 10000 |
| 1357028218 | Cody | Lopez | Biology Lab | 7341582096 | 20000 |
| 4689717527 | Sara | Davis | Biology Lab | 7341582096 | 20000 |

**Relational Algebra:**

$$\rho_{\text{Health\_Allowance}}\ (\textit{case when Role} = \textit{'Assistant' then 20,000,}$$
$$\textit{when Role} = \textit{'Researcher' then 10,000,}$$
$$\textit{when Role} = \textit{'Technician' then 5000 else 'Unknown' end})$$
$$\prod_{\text{Employee\_ID, First\_Name, Last\_Name, Lab\_Name Health\_Allowance}} (\text{Staff} \bowtie_{Lab\_Name=Lab\_Name} Lab]$$

**Operation 5:**
Here we insert a staff member which involves storage of an image along with a caption into the database.

**SQL Query:**

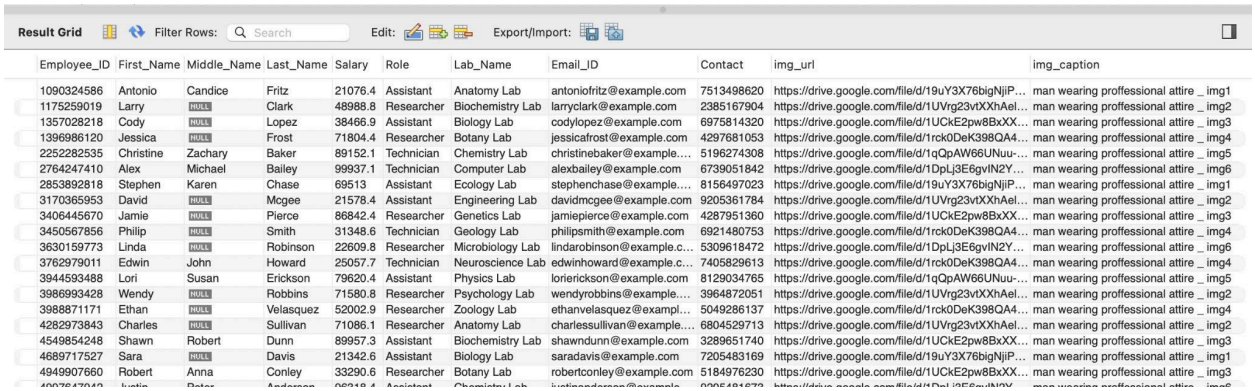
```
INSERT INTO staff VALUES
(1200000000,'Ameya',NULL,'Tajne',500000.8,'Researcher','Physics
Lab','tajneameyas@example.com',9090900089,
'https://drive.google.com/file/d/1UVrg23vtXXhAelbIsNwhuPQ60PapMhlK/vie
w?usp=sharing', 'man wearing professional attire _ img2');
```

**The operation satisfies 2 (Storage of an image along with a caption)**

**Result of Query:**

```
DROP TABLE IF EXISTS `staff`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `staff` (
  `Employee_ID` bigint NOT NULL,
  `First_Name` varchar(255) NOT NULL,
  `Middle_Name` varchar(255) DEFAULT NULL,
  `Last_Name` varchar(255) DEFAULT NULL,
  `Salary` float DEFAULT NULL,
  `Role` enum('Assistant','Researcher','Technician') DEFAULT NULL,
   `Lab_Name` ENUM('Anatomy Lab', 'Biochemistry Lab', 'Biology Lab', 'Botany Lab', 'Chemistry Lab', 'Co
  `Email_ID` VARCHAR(255) CHECK (Email_ID REGEXP'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),
  `Contact` bigint DEFAULT NULL,
  `img_url` VARCHAR(255) DEFAULT NULL,
  `img_caption` VARCHAR(255) DEFAULT NULL,
  PRIMARY KEY (`Employee_ID`),
  UNIQUE KEY `Employee_ID` (`Employee_ID`),
  CONSTRAINT `staff_chk_2` CHECK ((((`Contact` >= 1000000000) and (`Contact` < 10000000000))),
  CONSTRAINT `staff_fk_2` FOREIGN KEY (`Lab_Name`) REFERENCES `lab` (`Lab_Name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

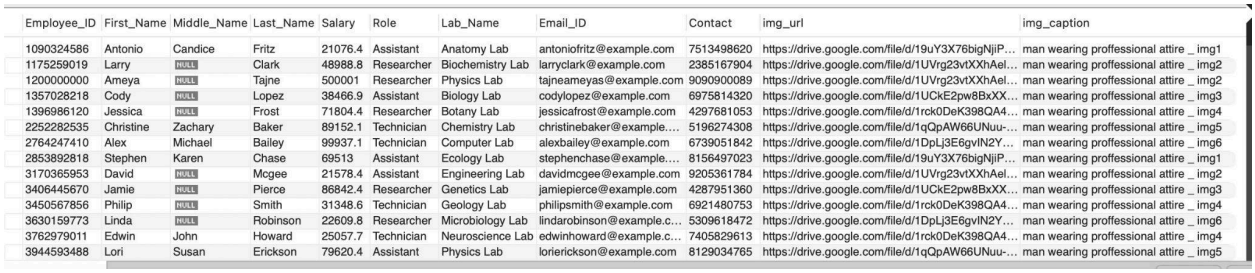
## Before Insertion :



Result Grid  Filter Rows: Search  Edit:  Export/Import:

| Employee_ID | First_Name | Middle_Name | Last_Name | Salary | Role | Lab_Name | Email_ID | Contact | img_url | img_caption |
|---|---|---|---|---|---|---|---|---|---|---|
| 1090324586 | Antonio | Candice | Fritz | 21076.4 | Assistant | Anatomy Lab | antoniofritz@example.com | 7513498620 | https://drive.google.com/file/d/19uY3X76bigNjiP… | man wearing proffessional attire _ img1 |
| 1175259019 | Larry | NULL | Clark | 48988.8 | Researcher | Biochemistry Lab | larryclark@example.com | 2385167904 | https://drive.google.com/file/d/1UVrg23vtXXhAel… | man wearing proffessional attire _ img2 |
| 1357028218 | Cody | NULL | Lopez | 38466.9 | Assistant | Biology Lab | codylopez@example.com | 6975814320 | https://drive.google.com/file/d/1UCkE2pw8BxXX… | man wearing proffessional attire _ img3 |
| 1396986120 | Jessica | NULL | Frost | 71804.4 | Researcher | Botany Lab | jessicafrost@example.com | 4297681053 | https://drive.google.com/file/d/1rck0DeK398QA4… | man wearing proffessional attire _ img4 |
| 2252282535 | Christine | Zachary | Baker | 89152.1 | Technician | Chemistry Lab | christinebaker@example.… | 5196274308 | https://drive.google.com/file/d/1qQpAW66UNuu-… | man wearing proffessional attire _ img5 |
| 2764247410 | Alex | Michael | Bailey | 99937.1 | Technician | Computer Lab | alexbailey@example.com | 6739051842 | https://drive.google.com/file/d/1DpLj3E6gvIN2Y… | man wearing proffessional attire _ img6 |
| 2853892818 | Stephen | Karen | Chase | 69513 | Assistant | Ecology Lab | stephenchase@example.… | 8156497023 | https://drive.google.com/file/d/19uY3X76bigNjiP… | man wearing proffessional attire _ img1 |
| 3170365953 | David | NULL | Mcgee | 21578.4 | Assistant | Engineering Lab | davidmcgee@example.com | 9205361784 | https://drive.google.com/file/d/1UVrg23vtXXhAel… | man wearing proffessional attire _ img2 |
| 3406445670 | Jamie | NULL | Pierce | 86842.4 | Researcher | Genetics Lab | jamiepierce@example.com | 4287951360 | https://drive.google.com/file/d/1UCkE2pw8BxXX… | man wearing proffessional attire _ img3 |
| 3450567856 | Philip | NULL | Smith | 31348.6 | Technician | Geology Lab | philipsmith@example.com | 6921480753 | https://drive.google.com/file/d/1rck0DeK398QA4… | man wearing proffessional attire _ img4 |
| 3630159773 | Linda | NULL | Robinson | 22609.8 | Researcher | Microbiology Lab | lindarobinson@example.c… | 5309618472 | https://drive.google.com/file/d/1DpLj3E6gvIN2Y… | man wearing proffessional attire _ img6 |
| 3762979011 | Edwin | John | Howard | 25057.7 | Technician | Neuroscience Lab | edwinhoward@example.c… | 7405829613 | https://drive.google.com/file/d/1rck0DeK398QA4… | man wearing proffessional attire _ img4 |
| 3944593488 | Lori | Susan | Erickson | 79620.4 | Assistant | Physics Lab | lorierickson@example.com | 8129034765 | https://drive.google.com/file/d/1qQpAW66UNuu-… | man wearing proffessional attire _ img5 |
| 3986993428 | Wendy | NULL | Robbins | 71580.8 | Researcher | Psychology Lab | wendyrobbins@example.… | 3964872051 | https://drive.google.com/file/d/1UVrg23vtXXhAel… | man wearing proffessional attire _ img2 |
| 3988871171 | Ethan | NULL | Velasquez | 52002.9 | Researcher | Zoology Lab | ethanvelasquez@exampl… | 5049286137 | https://drive.google.com/file/d/1rck0DeK398QA4… | man wearing proffessional attire _ img4 |
| 4282973843 | Charles | NULL | Sullivan | 71086.1 | Researcher | Anatomy Lab | charlessullivan@example.… | 6804529713 | https://drive.google.com/file/d/1UVrg23vtXXhAel… | man wearing proffessional attire _ img2 |
| 4549854248 | Shawn | Robert | Dunn | 89957.3 | Assistant | Biochemistry Lab | shawndunn@example.com | 3289651740 | https://drive.google.com/file/d/1UCkE2pw8BxXX… | man wearing proffessional attire _ img3 |
| 4689717527 | Sara | NULL | Davis | 21342.6 | Assistant | Biology Lab | saradavis@example.com | 7205483169 | https://drive.google.com/file/d/19uY3X76bigNjiP… | man wearing proffessional attire _ img1 |
| 4949907660 | Robert | Anna | Conley | 33290.6 | Researcher | Botany Lab | robertconley@example.com | 5184976230 | https://drive.google.com/file/d/1UCkE2pw8BxXX… | man wearing proffessional attire _ img3 |

## After Insertion :



| Employee_ID | First_Name | Middle_Name | Last_Name | Salary | Role | Lab_Name | Email_ID | Contact | img_url | img_caption |
|---|---|---|---|---|---|---|---|---|---|---|
| 1090324586 | Antonio | Candice | Fritz | 21076.4 | Assistant | Anatomy Lab | antoniofritz@example.com | 7513498620 | https://drive.google.com/file/d/19uY3X76bigNjiP… | man wearing proffessional attire _ img1 |
| 1175259019 | Larry | NULL | Clark | 48988.8 | Researcher | Biochemistry Lab | larryclark@example.com | 2385167904 | https://drive.google.com/file/d/1UVrg23vtXXhAel… | man wearing proffessional attire _ img2 |
| 1200000000 | Ameya | NULL | Tajne | 500001 | Researcher | Physics Lab | tajneameyas@example.com | 9090900089 | https://drive.google.com/file/d/1UVrg23vtXXhAel… | man wearing proffessional attire _ img2 |
| 1357028218 | Cody | NULL | Lopez | 38466.9 | Assistant | Biology Lab | codylopez@example.com | 6975814320 | https://drive.google.com/file/d/1UCkE2pw8BxXX… | man wearing proffessional attire _ img3 |
| 1396986120 | Jessica | NULL | Frost | 71804.4 | Researcher | Botany Lab | jessicafrost@example.com | 4297681053 | https://drive.google.com/file/d/1rck0DeK398QA4… | man wearing proffessional attire _ img4 |
| 2252282535 | Christine | Zachary | Baker | 89152.1 | Technician | Chemistry Lab | christinebaker@example.… | 5196274308 | https://drive.google.com/file/d/1qQpAW66UNuu-… | man wearing proffessional attire _ img5 |
| 2764247410 | Alex | Michael | Bailey | 99937.1 | Technician | Computer Lab | alexbailey@example.com | 6739051842 | https://drive.google.com/file/d/1DpLj3E6gvIN2Y… | man wearing proffessional attire _ img6 |
| 2853892818 | Stephen | Karen | Chase | 69513 | Assistant | Ecology Lab | stephenchase@example.… | 8156497023 | https://drive.google.com/file/d/19uY3X76bigNjiP… | man wearing proffessional attire _ img1 |
| 3170365953 | David | NULL | Mcgee | 21578.4 | Assistant | Engineering Lab | davidmcgee@example.com | 9205361784 | https://drive.google.com/file/d/1UVrg23vtXXhAel… | man wearing proffessional attire _ img2 |
| 3406445670 | Jamie | NULL | Pierce | 86842.4 | Researcher | Genetics Lab | jamiepierce@example.com | 4287951360 | https://drive.google.com/file/d/1UCkE2pw8BxXX… | man wearing proffessional attire _ img3 |
| 3450567856 | Philip | NULL | Smith | 31348.6 | Technician | Geology Lab | philipsmith@example.com | 6921480753 | https://drive.google.com/file/d/1rck0DeK398QA4… | man wearing proffessional attire _ img4 |
| 3630159773 | Linda | NULL | Robinson | 22609.8 | Researcher | Microbiology Lab | lindarobinson@example.c… | 5309618472 | https://drive.google.com/file/d/1DpLj3E6gvIN2Y… | man wearing proffessional attire _ img6 |
| 3762979011 | Edwin | John | Howard | 25057.7 | Technician | Neuroscience Lab | edwinhoward@example.c… | 7405829613 | https://drive.google.com/file/d/1rck0DeK398QA4… | man wearing proffessional attire _ img4 |
| 3944593488 | Lori | Susan | Erickson | 79620.4 | Assistant | Physics Lab | lorierickson@example.com | 8129034765 | https://drive.google.com/file/d/1qQpAW66UNuu-… | man wearing proffessional attire _ img5 |

## Relational Algebra:

$$\text{staff} \leftarrow (\text{staff}) \cup (\ 1200000000, \text{'Ameya'}, \text{NULL}, \text{'Tajne'}, 500000.8, \text{'Researcher'}, \text{'Physics Lab'}, \text{'tajneameyas@example.com'}, 9090900089, \text{'https://drive.google.com/file/d/1UVrg23vtXXhAelbIsNwhuPQ60PapMhlK/view?usp=sharing'}, \text{'man wearing professional attire img2'})$$

## Work Distribution:

Manav Jain (G2):

- Contributed in the solution of referential integrity question, covering possible violations, along with their solutions
- Helped in creating new user
- Helped in creating different views of the table

Anish Karnik(G1):

- Generated Fake data using python
- Created User defined data types
- Wrote SQL queries and relational algebra for them in Q3
- Wrote the queries throwing error and its reasons

Kaushal Kothiya(G2):

- Contributed to making users, tables and views for G2.
- Helped in discovering different examples in our database where referential integrity is violated and finding potential solutions.
- Contributed to the documentation aspect for G2.
- Helped in crafting tables incorporating images and formulating associated queries.

Tajne Ameya(G1):

- Wrote user defined data type Enum for lab_names and department
- Created data for the student and inventory
- Created many to many , one-to-one and one to many relationships between the entities.
- Implemented relevant indexing strategies to improve data retrieval efficiency.

Srujan Kumar Shetty(G2):

- Helped create tables with photos and formulate related queries.
- Contributed to the discovery of various examples of referential integrity violations in our database, as well as the identification of feasible solutions.
- Contributed to the documentation portion for G2.
- Helped to create users, tables, and views for G2.

Sachin Jalan(G1):

- Created table extensions
- Did indexing on tables and found the query execution time for them
- Contributed in writing sql queries
- Contributed in writing relational algebra for those queries

Husain Malwat(G1):

- Created SQL schema and contributed in generating fake data.
- Created many-to-many, one-to-one and one-to-many relationships between the entities.
- Contributed to the documentation of G1(Indexing over columns, user-defined data types, and table extension.)