Indian Institute of Technology Gandhinagar


# CS 432: Databases

## Assignment 2: Developing the DBMS

Outlet Database Management System

## Team : The Pluto

*Yajurvedh Bodala - 19110077*
*Krish Raj - 20110160*
*Dharavath Mahesh - 22110073*
*Charan Teja - 22110136*
*Harsh Kumar Keshri - 22110094*
*Susmita R - 22110265*
*Shipra Kethwalia - 23210091*
*Shubham Kshirsagar - 23210097*

**Under The Guidance Of**
*Prof. Mayank Singh*

# 3. Tasks

## 3.1 Responsibility of G1:

*1. Populate the tables 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.*

- SQL file is attached in the below link:

https://drive.google.com/drive/folders/1Sg3MjijAPD8Jvk4OMXGyMfOh38fuwlXh?usp=sharing

*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.*

### Indexing:

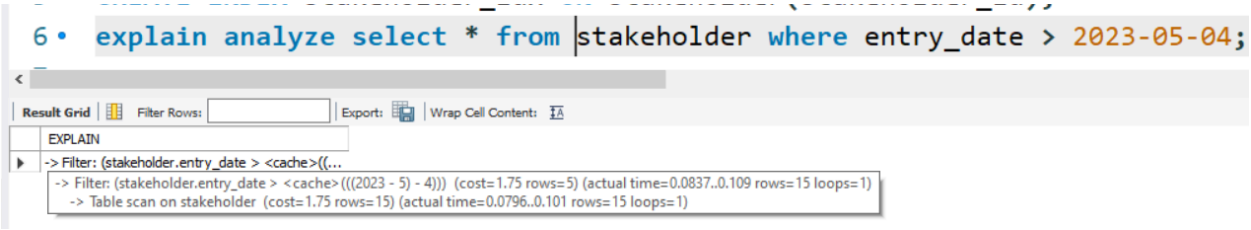
Indexing enables faster database searching by reducing the time we need to transfer data from the disk. Indexing is crucial for improving the speed of query processing, especially in large databases where searching through all the records sequentially can be time-consuming. The “Customer_Feedback” table contains a large number of samples or entities within the database. This observation suggests that the 'Customer_Feedback' table may significantly impact query performance and database operations due to its size and the number of records it contains.

```
use Outlet_Management;
```

First, we navigated to the 'Outlet_Management' database, which was previously created, to begin working with its data and schema.

**Stakeholder**

*Without Indexing*

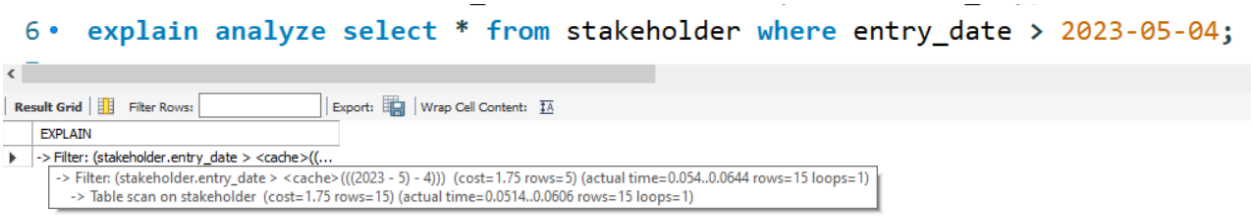



After, we implemented indexing in the “Stakeholder” table to optimize the search on “entry_date” for the following table, which scanned the entire 'stakeholder' table to find records with an 'entry_date' later than May 4, 2023.

```
4 •  show index in stakeholder;
5 •  CREATE INDEX stakeholder_idx ON stakeholder(stakeholder_id);
6 •  explain analyze select * from stakeholder where entry_date > 2023-05-04;
```

After verifying the index creation using the 'SHOW INDEX IN stakeholder' command, we reran the query to evaluate its performance post-indexing.

*With Indexing*

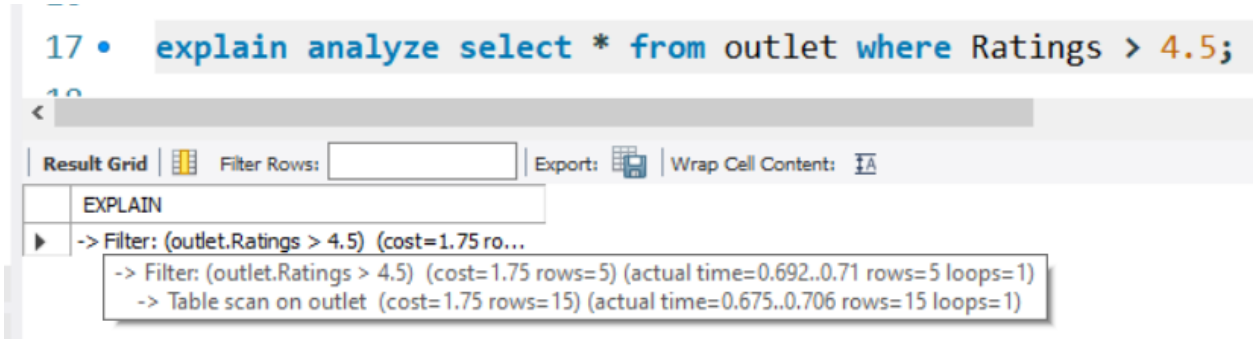



- Query execution time before indexing: 0.0837 s
- Query execution time after indexing: 0.054 s

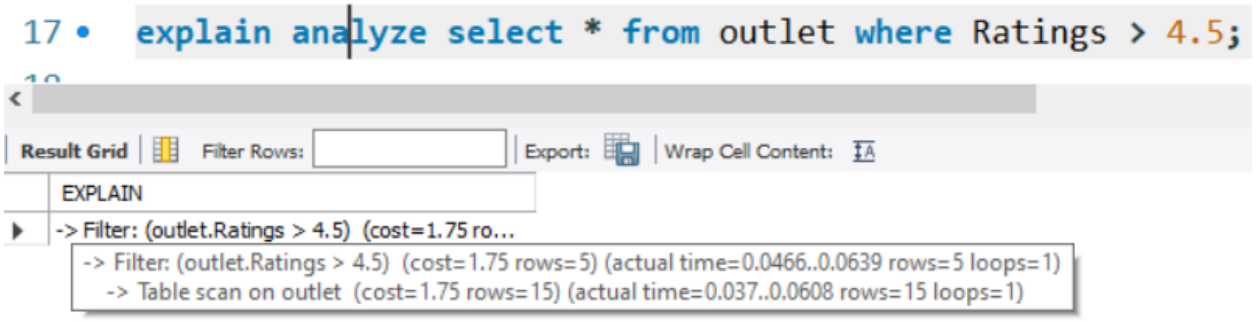
After executing the query to retrieve the schema information with indexing, we repeated the process after implementing indexing on all tables. By comparing the execution times before and after indexing.

## Outlet

*Without Indexing*



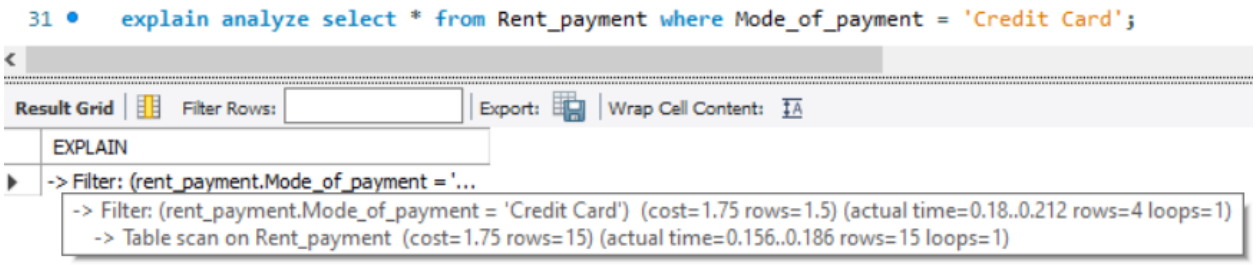

*With Indexing*




- Query execution time before indexing: 0.692 s
- Query execution time after indexing: 0.0466 s

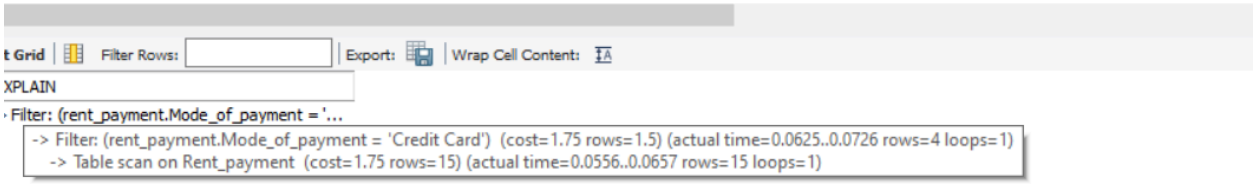
## Rent_payment

*Without Indexing*



*With Indexing*

```
explain analyze select * from Rent_payment where Mode_of_payment = 'Credit Card';
```

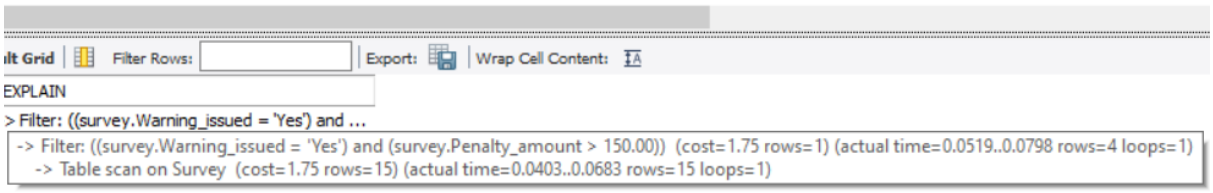



- Query execution time before indexing: 0.18 s
- Query execution time after indexing: 0.0625 s

**Survey**

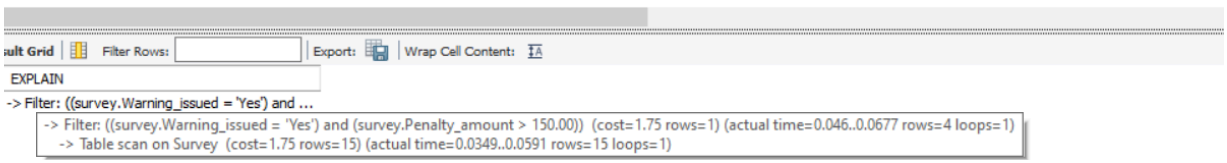
*Without Indexing*

```
explain analyze select * from Survey where Warning_issued = 'Yes' and Penalty_amount > 150;
```




*With Indexing*

```
explain analyze select * from Survey where Warning_issued = 'Yes' and Penalty_amount > 150;
```

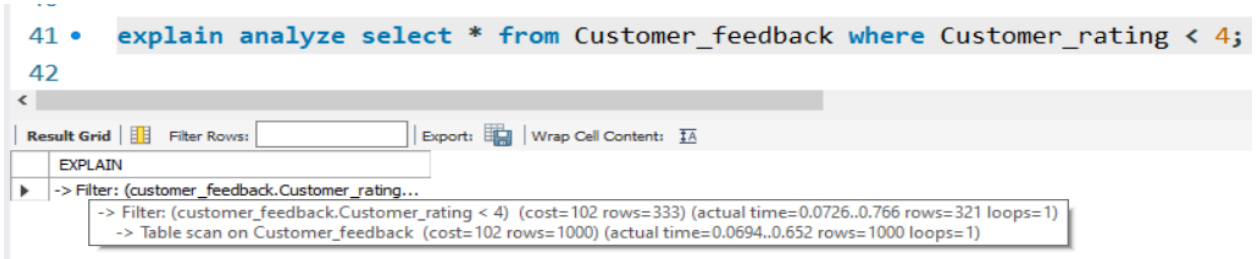



- Query execution time before indexing: 0.0519 s
- Query execution time after indexing: 0.046 s

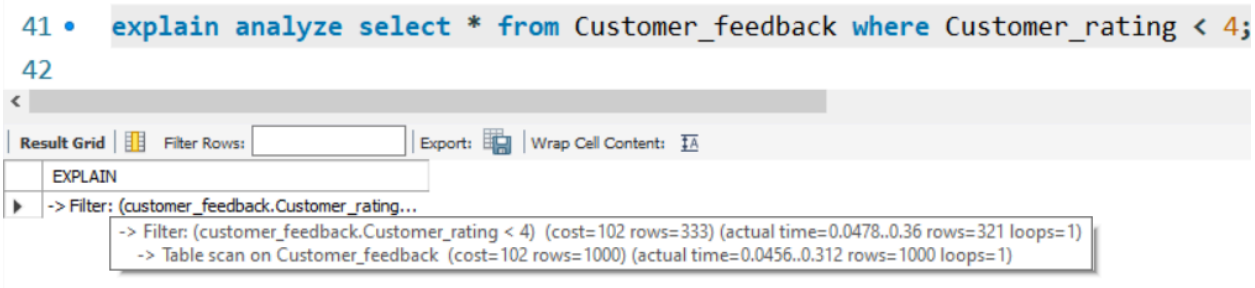
**Customer_feedback**

*Without Indexing*

```
explain analyze select * from Customer_feedback where Customer_rating < 4;
```



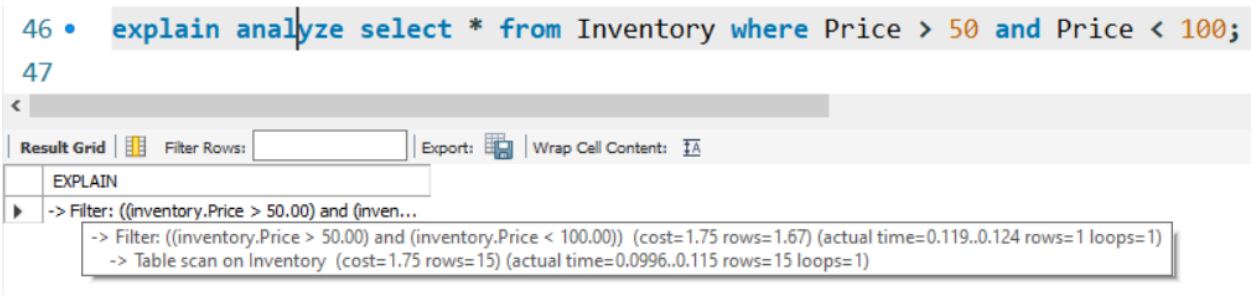
*With Indexing*



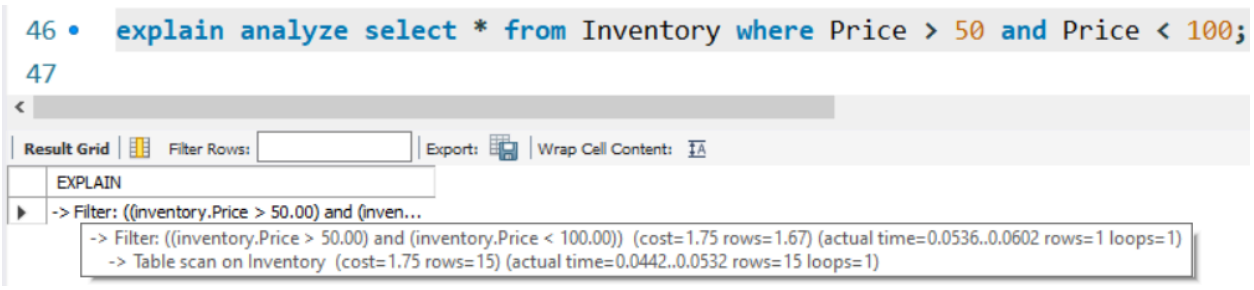

Query execution time before indexing: 0.0726 s
Query execution time after indexing: 0.0478 s

**Inventory**

*Without Indexing*



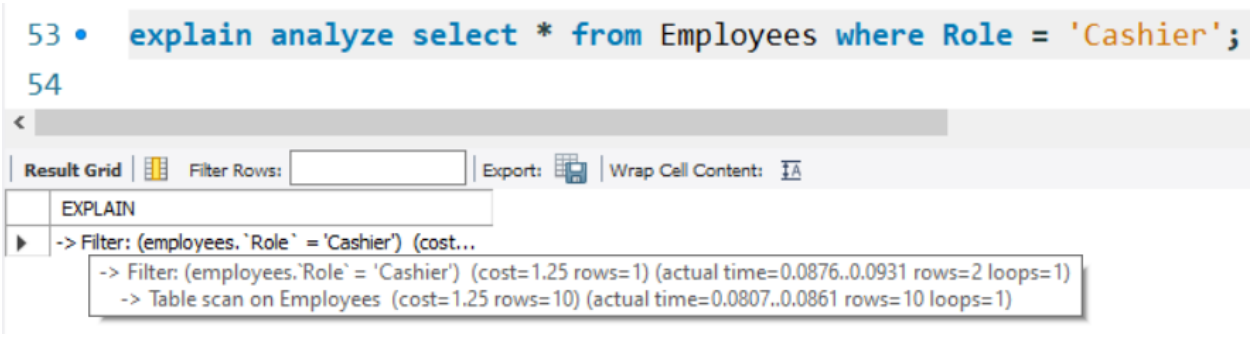

*With Indexing*



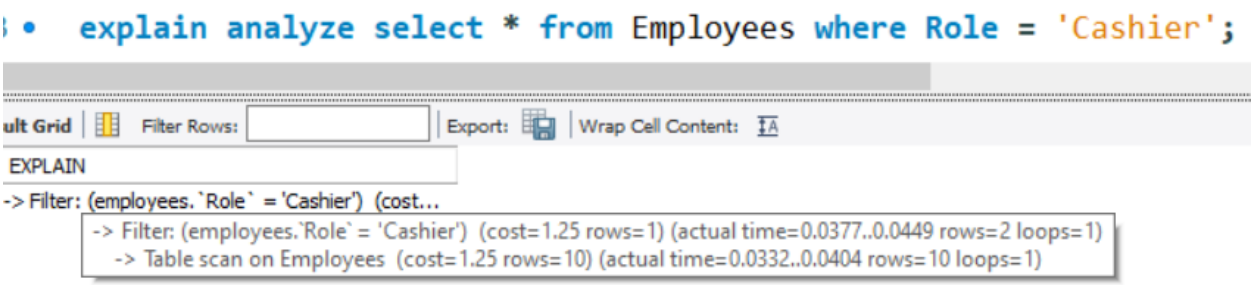

Query execution time before indexing: 0.119 s
Query execution time after indexing: 0.0536 s

**Employees**

*Without Indexing*




*With Indexing*

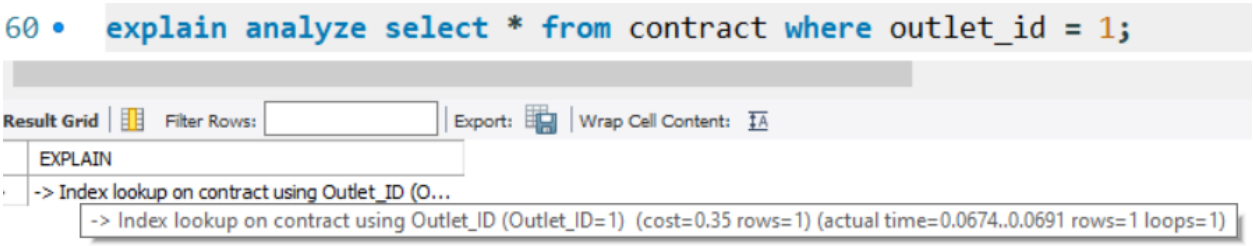



Query execution time before indexing: 0.0876 s

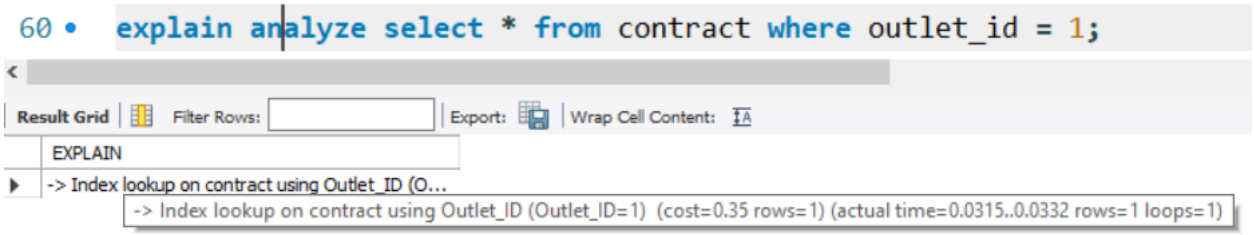
Query execution time after indexing: 0.0377 s

**Contract**

*Without Indexing*



*With Indexing*

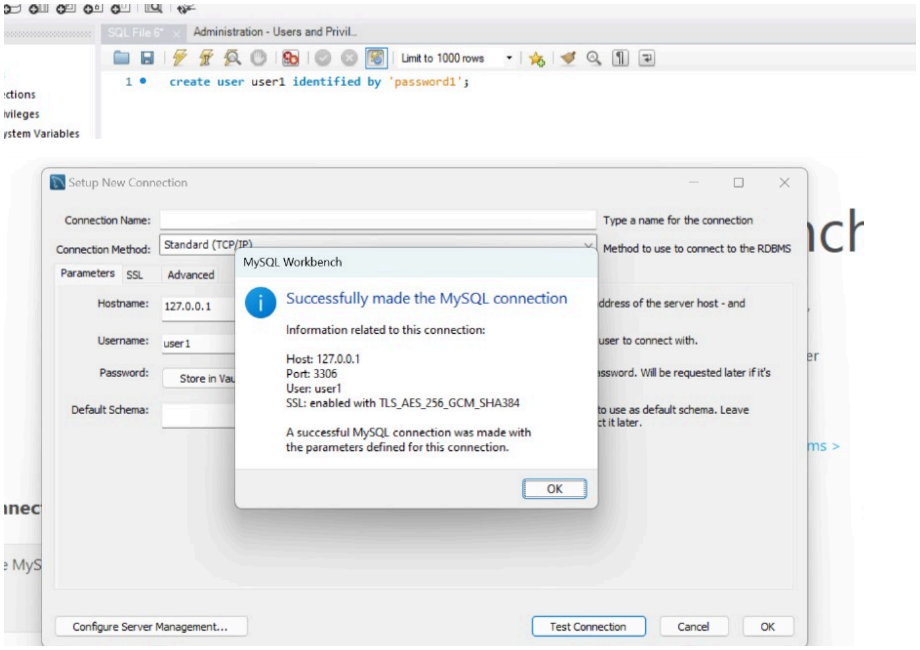



Query execution time before indexing: 0.0674 s
Query execution time after indexing: 0.0315 s

# 3.2 Responsibility of G2:

Part 1:

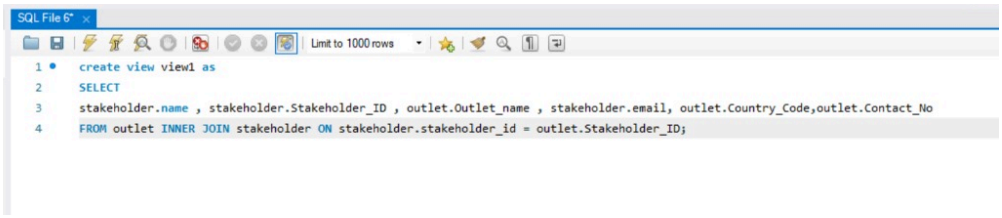
1. *Create a user named "user1" with the password "password1"*




**Setting up the connection**

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*

**Making View 1**




**View 1 Print**

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

| name | Stakeholder_ID | Outlet_name | email | Country_Code | Contact_No |
|---|---|---|---|---|---|
| Rahul Jain | 1 | Outlet 1 | jain.rahul@iitgn.ac.in | "+91" | 1234567890 |
| Sparsh Singh | 2 | Outlet 2 | singh.sparsh@iitgn.ac.in | "+91" | 2345678901 |
| Hrishav Gupta | 3 | Outlet 3 | gupta.hrishav@iitgn.ac.in | "+91" | 3456789012 |
| Rounak Mehta | 4 | Outlet 4 | mehta.rounak@iitgn.ac.in | "+91" | 4567890123 |
| Ayush Singh | 5 | Outlet 5 | singh.ayush@iitgn.ac.in | "+91" | 5678901234 |
| Ayush Kumar | 6 | Outlet 6 | kumar.ayush@iitgn.ac.in | "+91" | 6789012345 |
| Darsh Rungta | 7 | Outlet 7 | rungta.darsh@iitgn.ac.in | "+91" | 7890123456 |
| Yash Kumar | 8 | Outlet 8 | kumar.yash@iitgn.ac.in | "+91" | 8901234567 |
| Nakul Lal | 9 | Outlet 9 | lal.nakul@iitgn.ac.in | "+91" | 9012345678 |
| Himanshu Yadav | 10 | Outlet 10 | yadav.himanshu@iitgn.ac.in | "+91" | 1234567891 |
| Rahul Jain | 1 | Outlet 11 | jain.rahul@iitgn.ac.in | "+91" | 1234567890 |

view1 1

## Making View 2

SQL File 6*

Limit to 1000 rows

```
1  create view view2 as
2  SELECT
3  outlet.Outlet_name , outlet.Country_Code, outlet.Contact_No, rent_payment.Mode_of_payment , rent_payment.Paid_amount
4  FROM outlet INNER JOIN rent_payment where outlet.outlet_id = rent_payment.Outlet_ID;
5
```

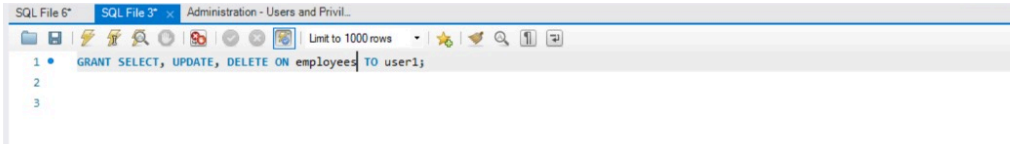
## Printing View 2

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

| Outlet_name | Country_Code | Contact_No | Mode_of_payment | Paid_amount |
|---|---|---|---|---|
| Outlet 1 | "+91" | 1234567890 | Cash | 8000.00 |
| Outlet 2 | "+91" | 2345678901 | Credit Card | 6200.00 |
| Outlet 3 | "+91" | 3456789012 | Debit Card | 9500.00 |
| Outlet 4 | "+91" | 4567890123 | Bank Transfer | 7000.00 |
| Outlet 5 | "+91" | 5678901234 | Cash | 7200.00 |
| Outlet 6 | "+91" | 6789012345 | Credit Card | 4500.00 |
| Outlet 7 | "+91" | 7890123456 | Debit Card | 5800.00 |
| Outlet 8 | "+91" | 8901234567 | Bank Transfer | 6500.00 |
| Outlet 9 | "+91" | 9012345678 | Cash | 2300.00 |

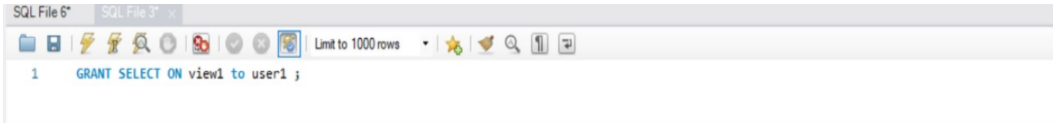
view2 2

3. *Grant "user1" the following permissions on "table1":*




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

**Permission to User1 on View1**

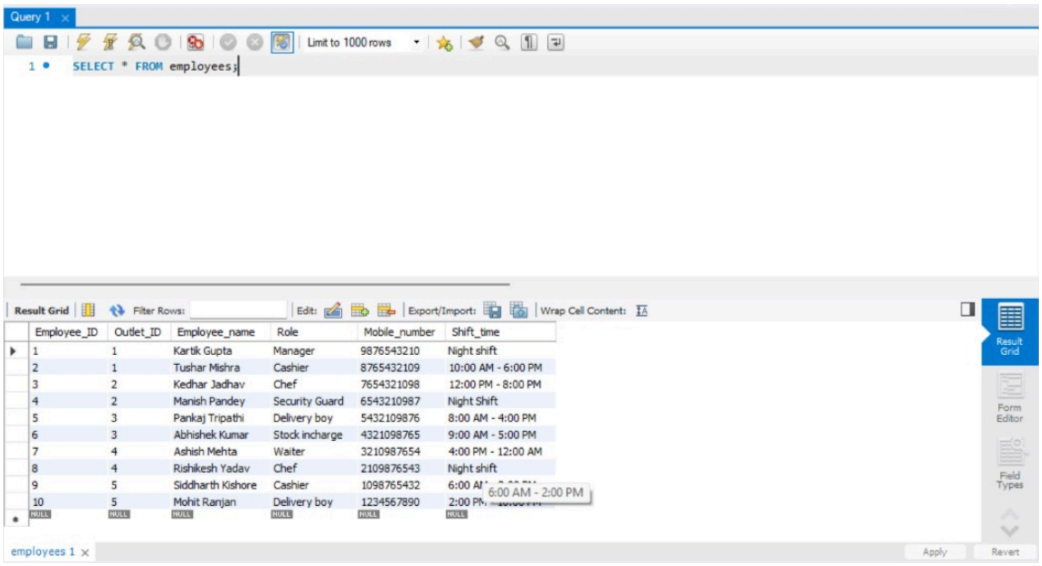



5. *Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" and report your findings:*

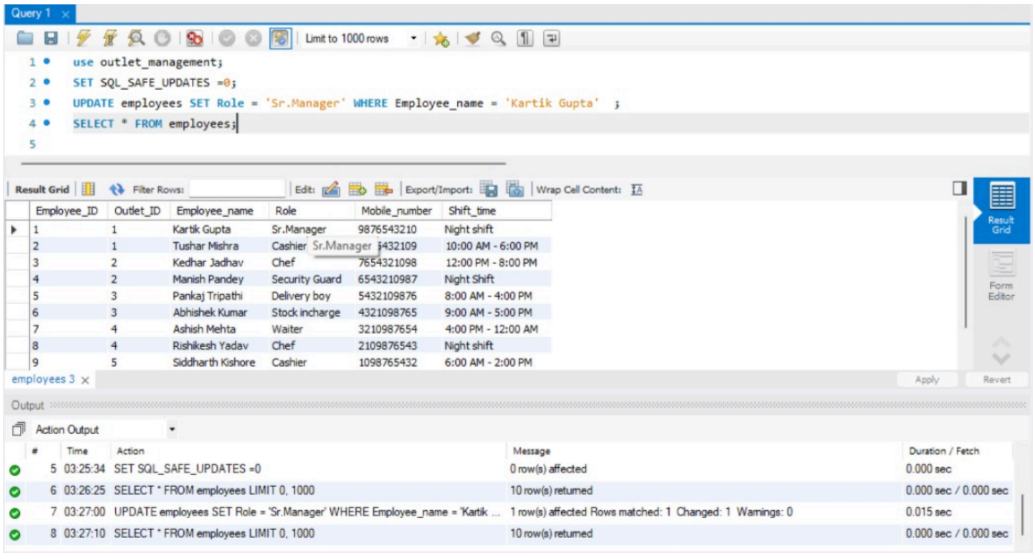
- **Select, Update and Delete operation on “table1”:**
  We have granted select, update and delete operation permission to “user1”. When the user tries to execute these operations, the SQL queries will run.

  The result is shown below for the above operations:

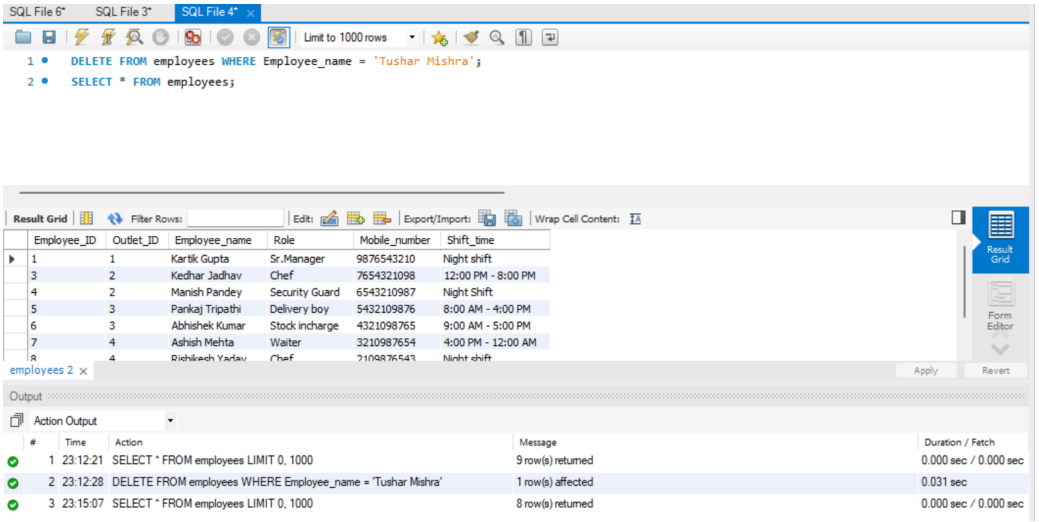
**SELECT: No Error**




**UPDATE: No Error**



**DELETE: No Error**

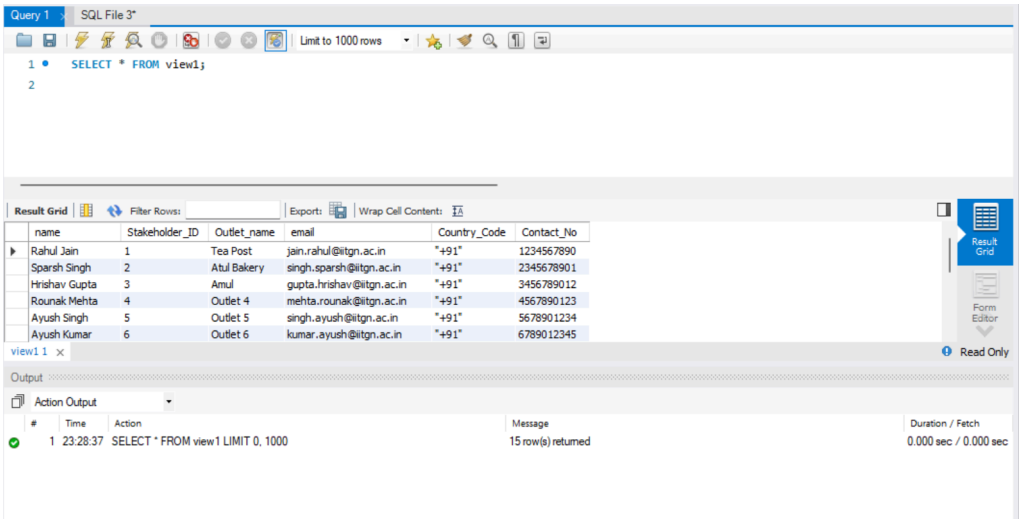



- **Select, Update and Delete operation on "view1":**
  We have granted only select permission to "user1" on "view1".

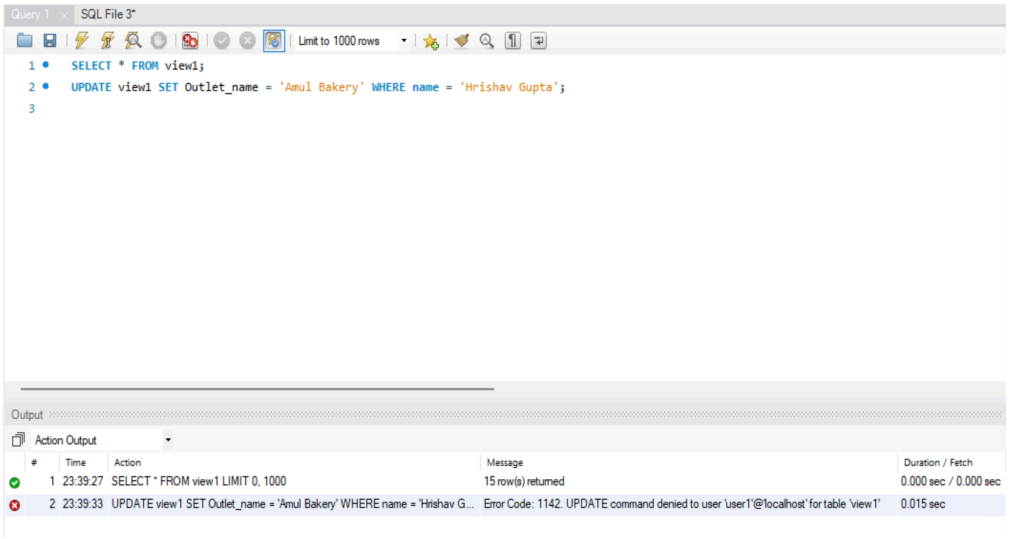
  When the user tries to execute the "Select" operation, the SQL query will run but for "Update" and "Delete" operation, we encountered an error.

  The result is shown below for the above operations:

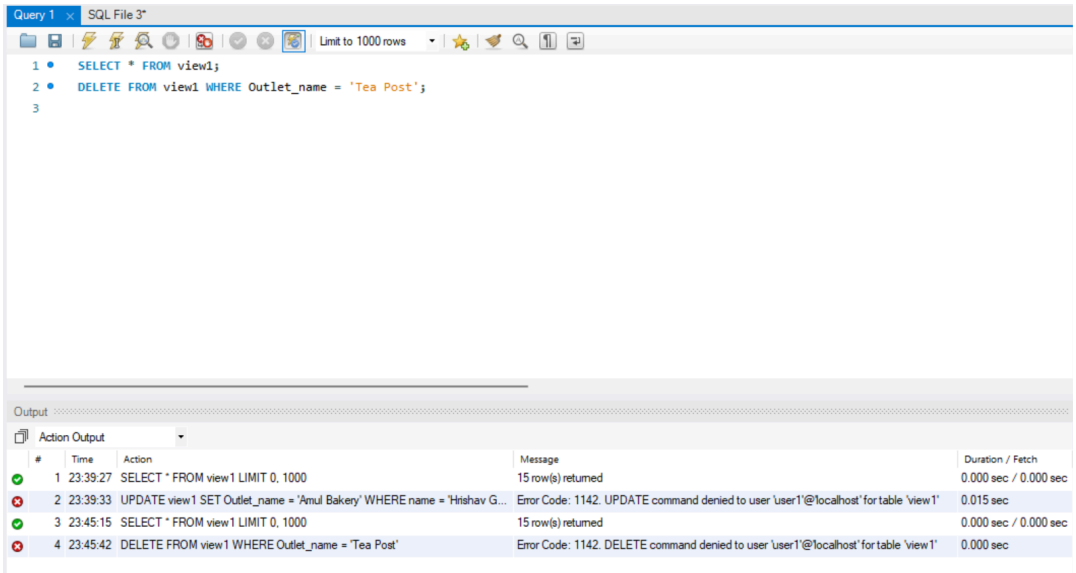
**SELECT: No Error**



**<u>UPDATE:</u> Encountered Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'**



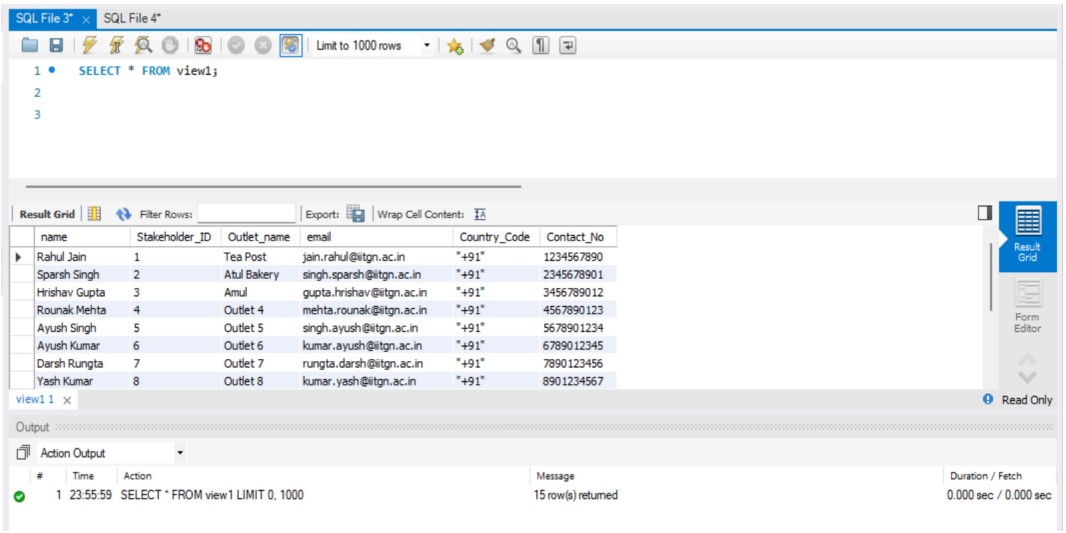

**<u>DELETE:</u> Encountered Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'.**



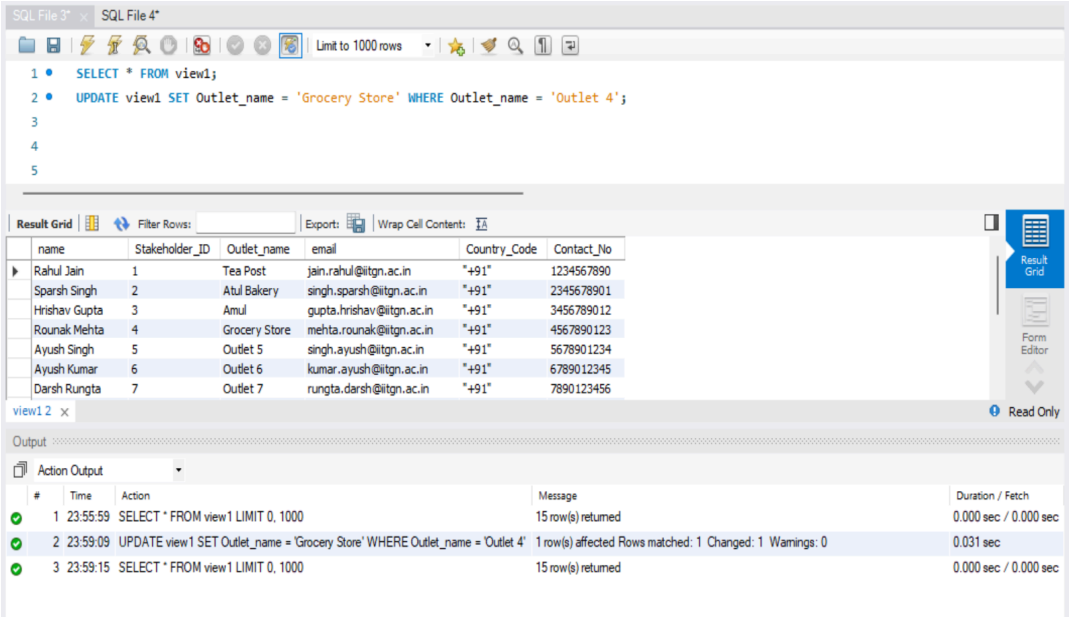

- **<u>If we perform the "Select" , "Update" and "Delete" operation in the localhost MySQL server, following are the observations:</u>**

The results are shown below for the operations on the localhost MySQL Server:

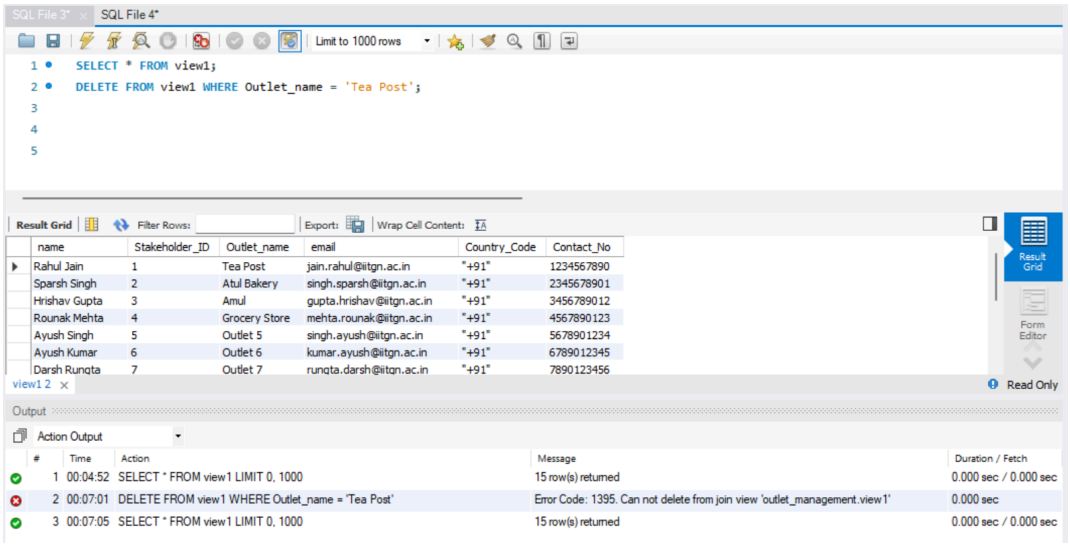
**SELECT: No error**




**UPDATE: No error**

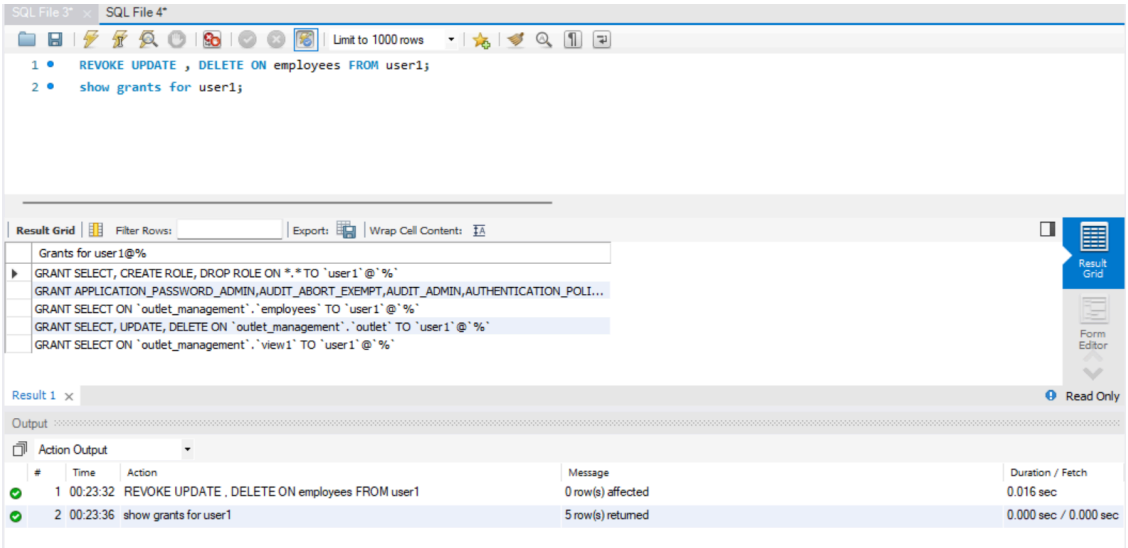


**<u>DELETE:</u> Encountered Error Code: 1395. Can not delete from join view 'outlet_manqagement.view1'.**




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

**We can see in the updated permissions that only SELECT operations are available to user1 on outlet_management and view1.**

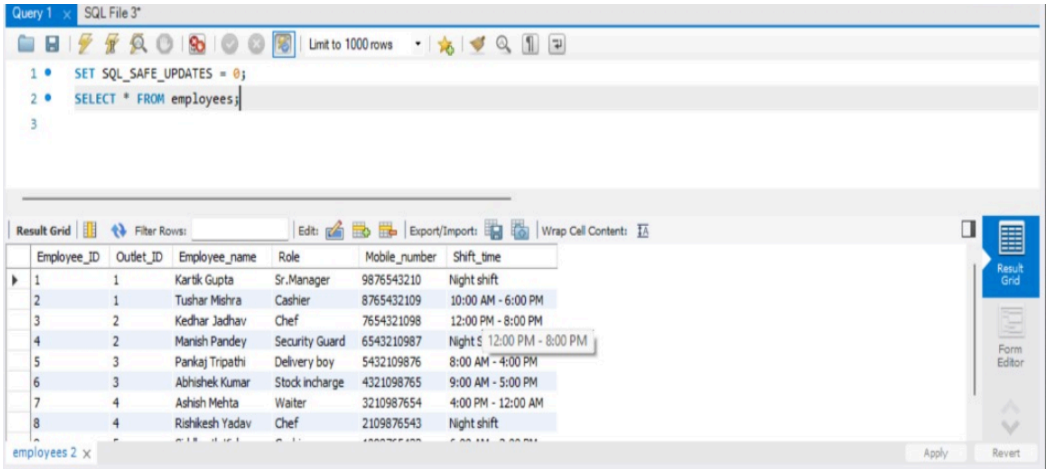


7. *Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" again.*

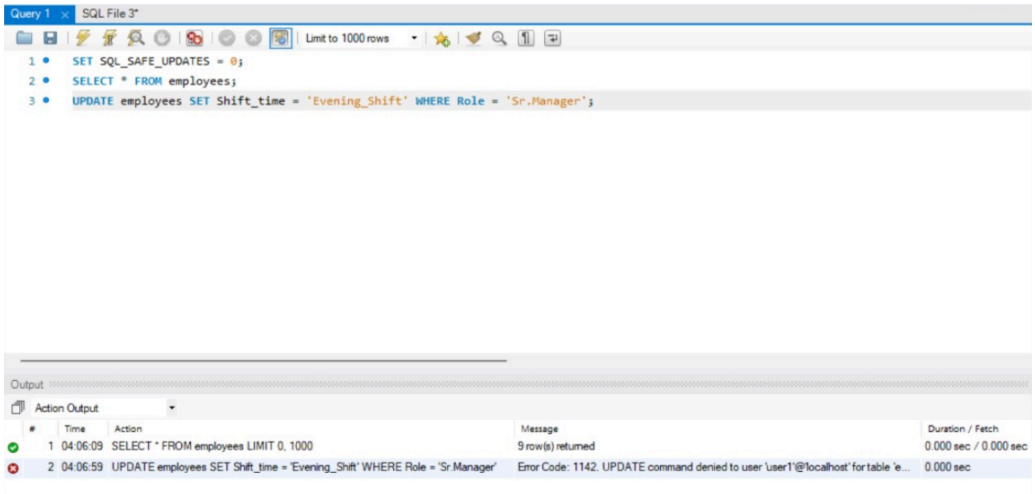
- **Select, Update and Delete operation on “table1”:**
  The “Select” Operation will run on “table1” as we did not revoke the Select Operation from “user1”.
  We encountered errors for “Update” and “Delete” Operation as we have revoked these permissions from “user1”.

  The result is shown below for above operations:

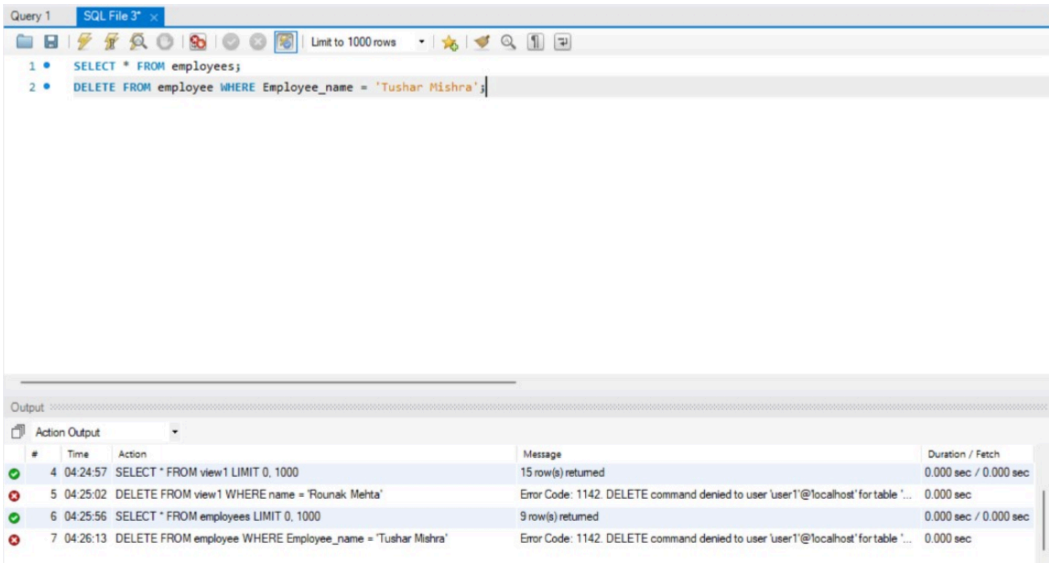
  **<u>SELECT:</u> No error**




  **<u>UPDATE:</u> Encountered Error Code: 1142 UPDATE command denied to user ‘user1’@’localhost’ for table ‘employees’.**



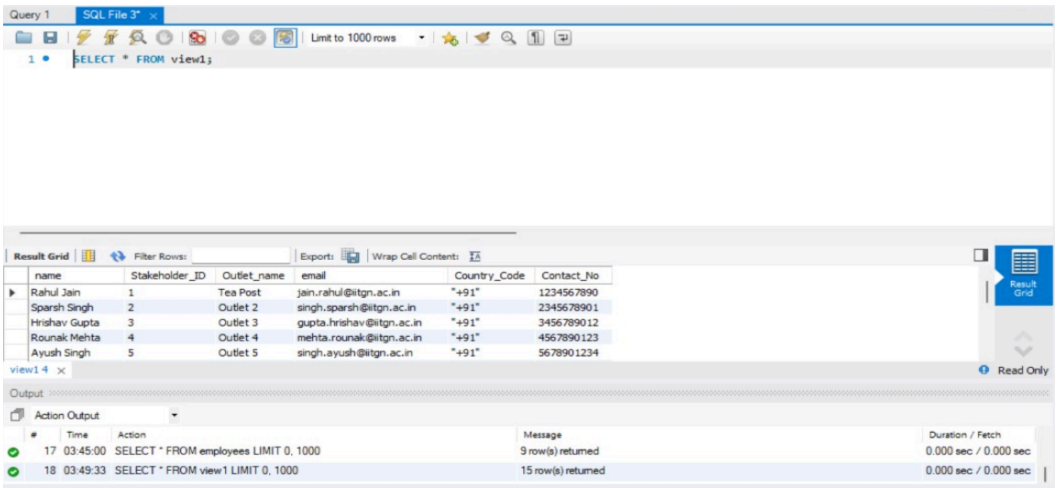
**DELETE: Encountered Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'employees'.**



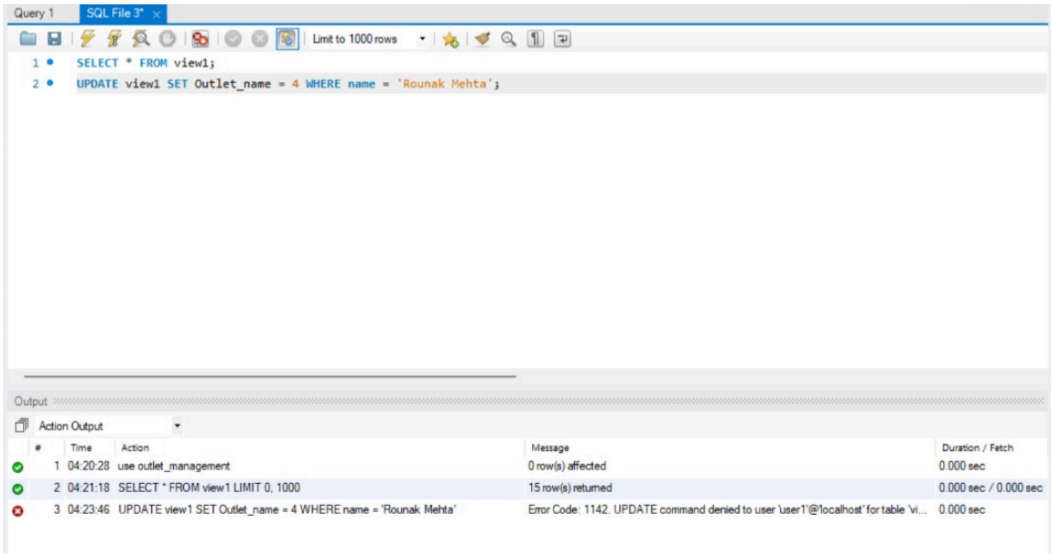

- **Select, Update and Delete operation on "view1":**
  We will again be able to run the "Select" operation on "view1" and encounter errors for "Update" and "Delete" Operation.

  The result is shown below for above operations:

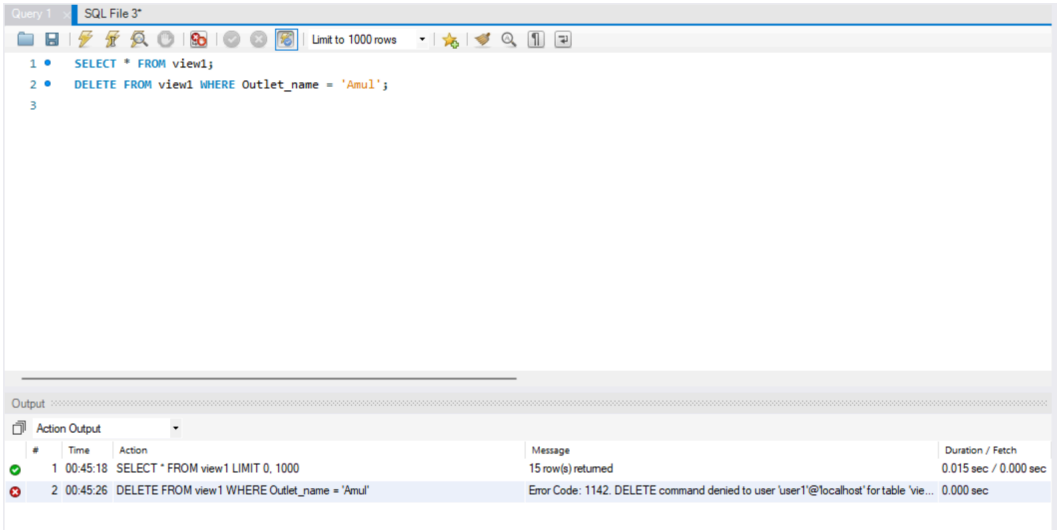
**SELECT: No error**



**<u>UPDATE:</u> Encountered Error Code: 1142 UPDATE command denied to user 'user1'@'localhost' for table 'view1'.**



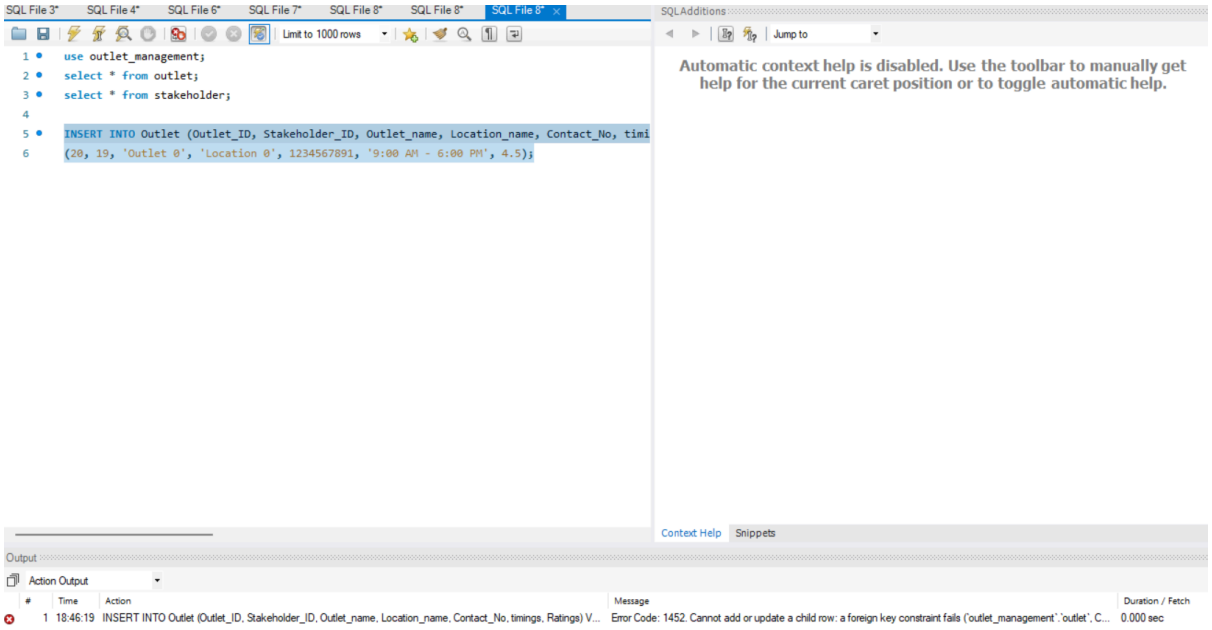

**<u>DELETE:</u> Encountered Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'.**



*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*

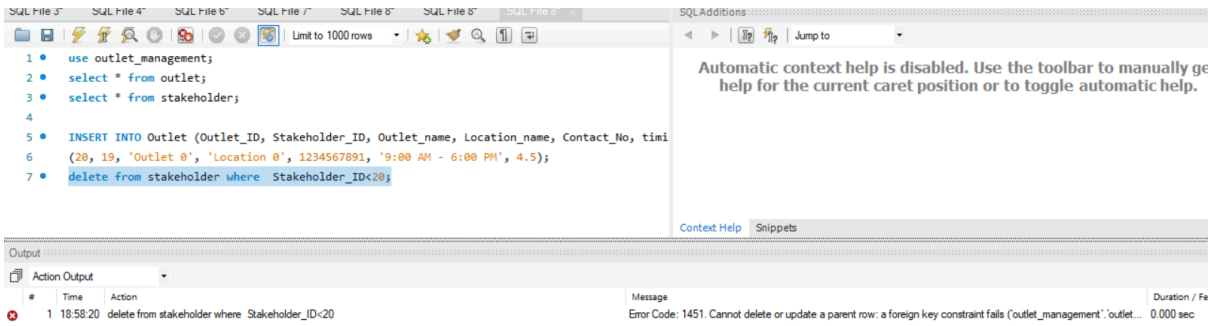
**Referential integrity violations** happen when the rules established by foreign key constraints are broken. The following are typical circumstances in which referential integrity could be broken:

**Inserting** a value in the foreign key column that does not exist in the referenced table: This violates referential integrity because the foreign key should always reference a valid primary key value in the referenced table.



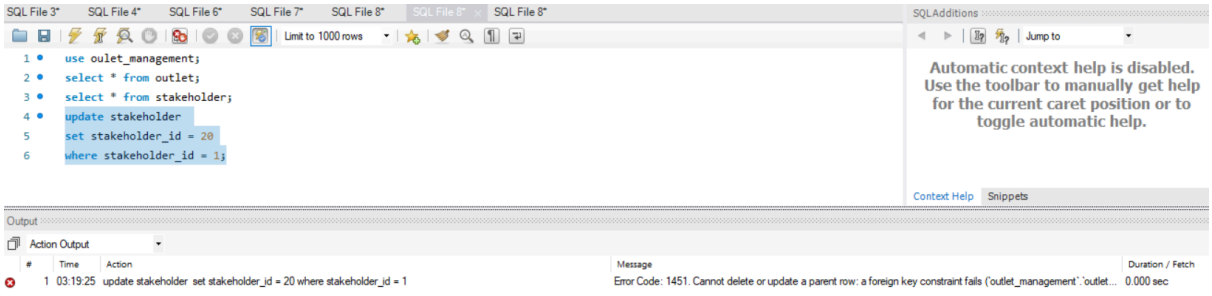

In the above case, we have added a new column i.e., inserting a value in the foreign key column. But , we are getting foreign key constraint error.

**Deleting** a primary key value in the referenced table that is referenced by foreign keys: If a primary key value is deleted in the referenced table, and there are related foreign key values in other tables, the deletion can lead to referential integrity violations. Depending on the database configuration, the deletion may be disallowed or cascade actions may be triggered to maintain integrity.




In the above case, we are deleting the 'stakeholder_id' primary key in the stakeholder table which is foreign key in the outlet table. But, we are getting an error 'cannot delete or update a parent row'.

**Updating** a primary key value in the referenced table without updating the corresponding foreign key values:To preserve referential integrity, the related foreign key values in other tables should be updated if a primary key value is changed in the referenced table and there are foreign key constraints in place.




In the above case, we are updating the 'stakeholder_id' primary key in the stakeholder table which is foreign key in the outlet table. But, we are getting an error 'cannot delete or update a parent row'.

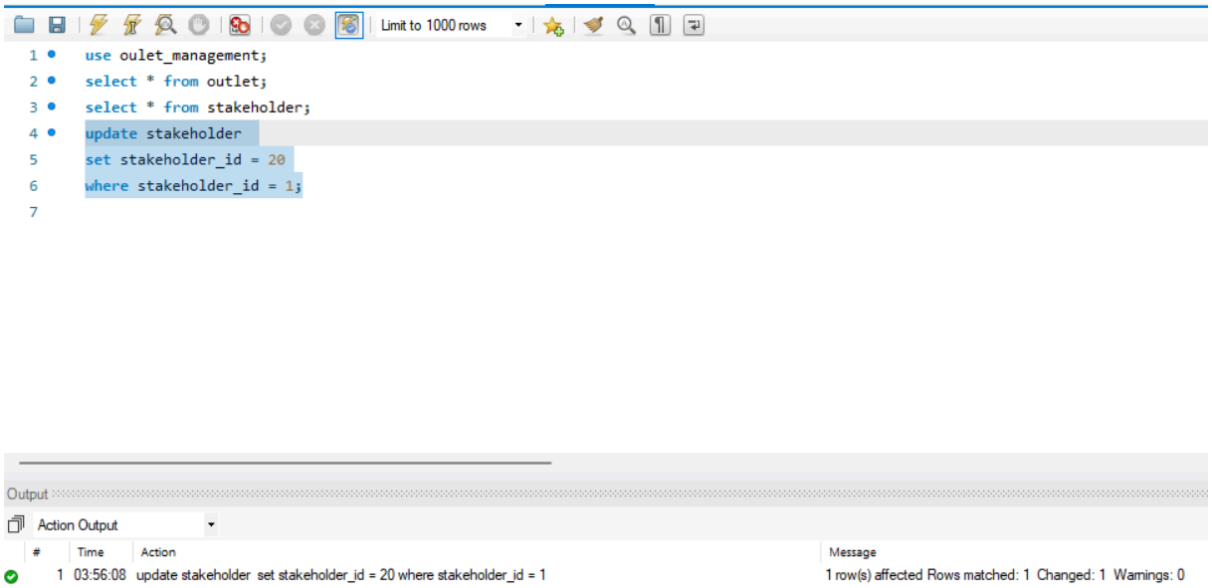
**Solving above problems**

By including referential operations like ON DELETE CASCADE and ON UPDATE CASCADE, we can address these problems.
This will handle the deleting and updating. Because of these two actions, each record that is modified or deleted in the referenced table will also automatically update or delete all associated records in the referencing table. This would guarantee that referential integrity is not affected. Database users are responsible for inserting errors; they should take care to avoid inserting tuples that reference nonexistent foreign keys and updating the referring values to nonexistent foreign keys.

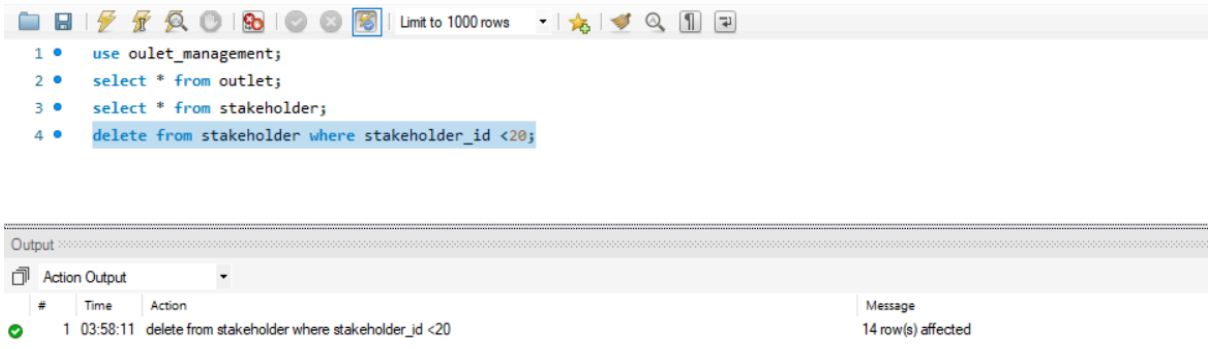
Below we have written ON DELETE CASCADE and ON UPDATE CASCADE

```
34
35      -- Create the Outlet table
36  •   CREATE TABLE Outlet (
37          Outlet_ID INT AUTO_INCREMENT PRIMARY KEY,
38          Stakeholder_ID  INT,
39          Outlet_name VARCHAR(50) NOT NULL,
40          Location_name VARCHAR(50) NOT NULL,
41          Contact_No BIGINT CHECK (Contact_No >= 1000000000 AND Contact_No < 10000000000),
42          timings VARCHAR(50),
43          Ratings FLOAT,
44          FOREIGN KEY (stakeholder_id) REFERENCES stakeholder(stakeholder_id)
45          on update cascade
46          on delete cascade
47      );
```

After implementing,we can observe below the updating operation working. There is no referential integrity error in this.



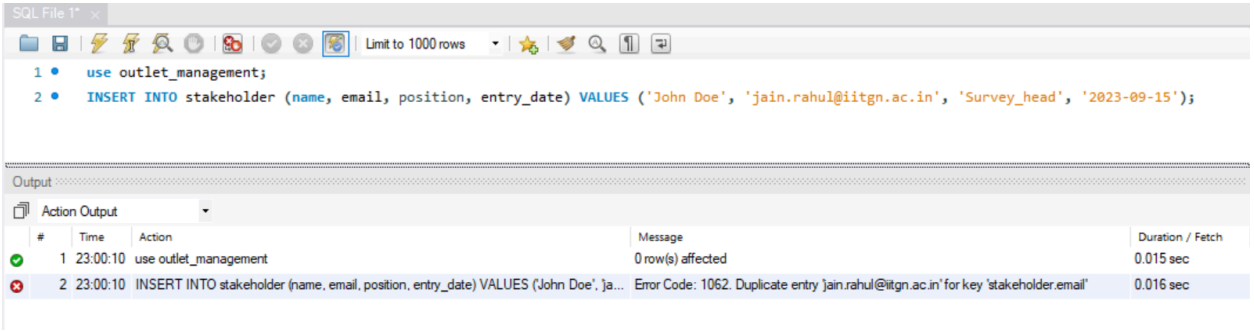

After implementing,we can observe below the deletion operation working. There is no referential integrity error in this.



## Responsibility of G1 and G2

1. *Two queries must throw an error due to violation of constraints specified by G1.*

**1st Query**




Duplicate entry '[jain.rahul@iitgn.ac.in](mailto:jain.rahul@iitgn.ac.in)' is inserted for the key stakeholder email.
This operation will throw an error due to the violation of the primary key constraint which ensures uniqueness.

To represent the given SQL query in relational algebra, the relational algebra expression for the given query would be:

$$\Pi_{name,\,email}(stakeholder) \cup \Pi_{name,\,email}(\sigma_{email = \text{"jain.rahul@iitgn.ac.in"}}(stakeholder))$$

This expression retrieves the name and email of all stakeholders in the Stakeholder relation and combines it with the name and email of the stakeholders who already have the email 'jain.rahul@iitgn.ac.in'.

**2nd Query**

```
4 •  INSERT INTO Contract (Outlet_ID, Document_number, Start_date, End_date, Contract_status) VALUES (100, 11, '2023-01-01', '2023-12-31', 'Active');
```

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 1 | 23:00:10 | use outlet_management | 0 row(s) affected | 0.015 sec |
| 2 | 23:00:10 | INSERT INTO stakeholder (name, email, position, entry_date) VALUES ('John Doe', 'jain.r... | Error Code: 1062. Duplicate entry 'jain.rahul@iitgn.ac.in' for key 'stakeholder.email' | 0.016 sec |
| 3 | 23:06:07 | INSERT INTO Contract (Outlet_ID, Document_number, Start_date, End_date, Contract_s... | Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('outlet_... | 0.015 sec |

Outlet ID ‘100’ doesn't exist.
Trying to insert a contract with an outlet ID that does not exist will violate the foreign key constraint.

To represent the given SQL query in relational algebra, the relational algebra expression for the given query would be:

$$f1 \leftarrow \pi_{outlet\_id}(Outlet)$$

$$f2 \leftarrow values(\{100, 11, '2023-01-01', '2023-12-3\,', 'Active'\})$$

$$result \leftarrow f2 - (f2 \bowtie f1)$$

Here, we first retrieve the set of existing Outlet IDs from the Outlet table (check_Outlet). Then, we create a tuple to be inserted (insert_tuple). Finally, we subtract the intersection of insert_tuple and check_Outlet from insert_tuple to get the result. If the result is not an empty set, it means the provided Outlet ID does not exist, violating the foreign key constraint.

2. *One of the functions should involve storage of an image along with a caption into the database.*

**Query 3 :** Storing image of the outlet image along with their location (place) in the table stroing_img, with the caption(column) showing name of the outlet and imge(column) storing the c drive path of the image and id_pic(column) showing the number of outlet image

are storing .

*3 satisfies specification 2*

```
CREATE TABLE storing_img (
    id_pic INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    caption VARCHAR(45) NOT NULL,
    imge LONGBLOB DEFAULT NULL,
    PRIMARY KEY (id_pic)
);
```

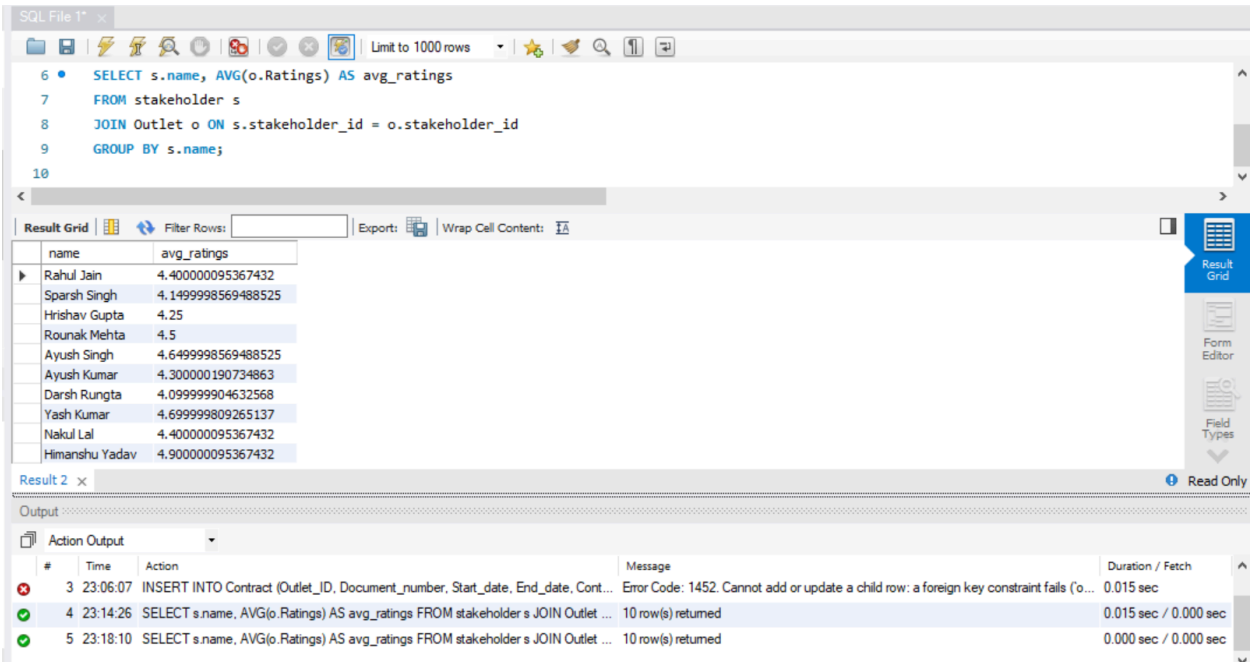
```
INSERT INTO storing_img(caption, imge) VALUES ('ATUL BAKERY', LOAD_FILE('C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ATUL_Bakeery.jpeg'));
INSERT INTO storing_img (caption, imge) VALUES ('DAWAT FOODS', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/DAWAT FOODS.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('MAHAVEER HEALTHY FOOD', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/MAHAVEER HEALTHY FOOD.jpeg"));
INSERT INTO storing_img (caption, imge) VALUES ('TEA POST', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Tea Post.jpeg"));
INSERT INTO storing_img (caption, imge) VALUES ('VS FAST FOOD', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/vs fast food.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('GO INSTA', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Go Insta.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES (' POOJA', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/POOJA.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('MONDEGO', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/MONDEGO.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('2 DEGREE', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/2DE.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES (' MINI 2 DEGREE', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/MIN2DE.jpeg"));

INSERT INTO storing_img(caption, imge) VALUES ('SAMYAK', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/SAMYAK.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('TFC', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/TFC.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('AMUL', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/AMU.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('KRUPA GENERALS', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/KR.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('JUST CHILL', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/JUST CHILL.jpeg"));
INSERT INTO storing_img(caption, imge) VALUES ('JK GROCERY', LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/JK.jpeg"));
```

```
SELECT * FROM outlet_management.storing_img;
```

| id_pic | caption | imge |
|---|---|---|
| 1 | ATUL BAKERY | BLOB |
| 2 | DAWAT FOODS | BLOB |
| 3 | MAHAVEER HEALTHY FOOD | BLOB |
| 4 | TEA POST | BLOB |
| 5 | VS FAST FOOD | BLOB |
| 6 | GO INSTA | BLOB |
| 7 | POOJA | BLOB |
| 8 | MONDEGO | BLOB |
| 9 | 2 DEGREE | BLOB |
| 10 | MINI 2 DEGREE | BLOB |
| 11 | SAMYAK | BLOB |
| 12 | TFC | BLOB |
| 13 | AMUL | BLOB |
| 14 | KRUPA GENERALS | BLOB |
| 15 | JUST CHILL | BLOB |
| 16 | JK GROCERY | BLOB |

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.*

**Query with natural join and aggregate function:**
Retrieve the average ratings of outlets managed by each stakeholder.



```
SELECT s.name, AVG(o.Ratings) AS avg_ratings
FROM stakeholder s
JOIN Outlet o ON s.stakeholder_id = o.stakeholder_id
GROUP BY s.name;
```

| name | avg_ratings |
|---|---|
| Rahul Jain | 4.400000095367432 |
| Sparsh Singh | 4.1499998569488525 |
| Hrishav Gupta | 4.25 |
| Rounak Mehta | 4.5 |
| Ayush Singh | 4.6499998569488525 |
| Ayush Kumar | 4.300000190734863 |
| Darsh Rungta | 4.099999904632568 |
| Yash Kumar | 4.699999809265137 |
| Nakul Lal | 4.400000095367432 |
| Himanshu Yadav | 4.900000095367432 |

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 3 | 23:06:07 | INSERT INTO Contract (Outlet_ID, Document_number, Start_date, End_date, Cont... | Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('o... | 0.015 sec |
| 4 | 23:14:26 | SELECT s.name, AVG(o.Ratings) AS avg_ratings FROM stakeholder s JOIN Outlet ... | 10 row(s) returned | 0.015 sec / 0.000 sec |
| 5 | 23:18:10 | SELECT s.name, AVG(o.Ratings) AS avg_ratings FROM stakeholder s JOIN Outlet ... | 10 row(s) returned | 0.000 sec / 0.000 sec |

Relational Algebra:

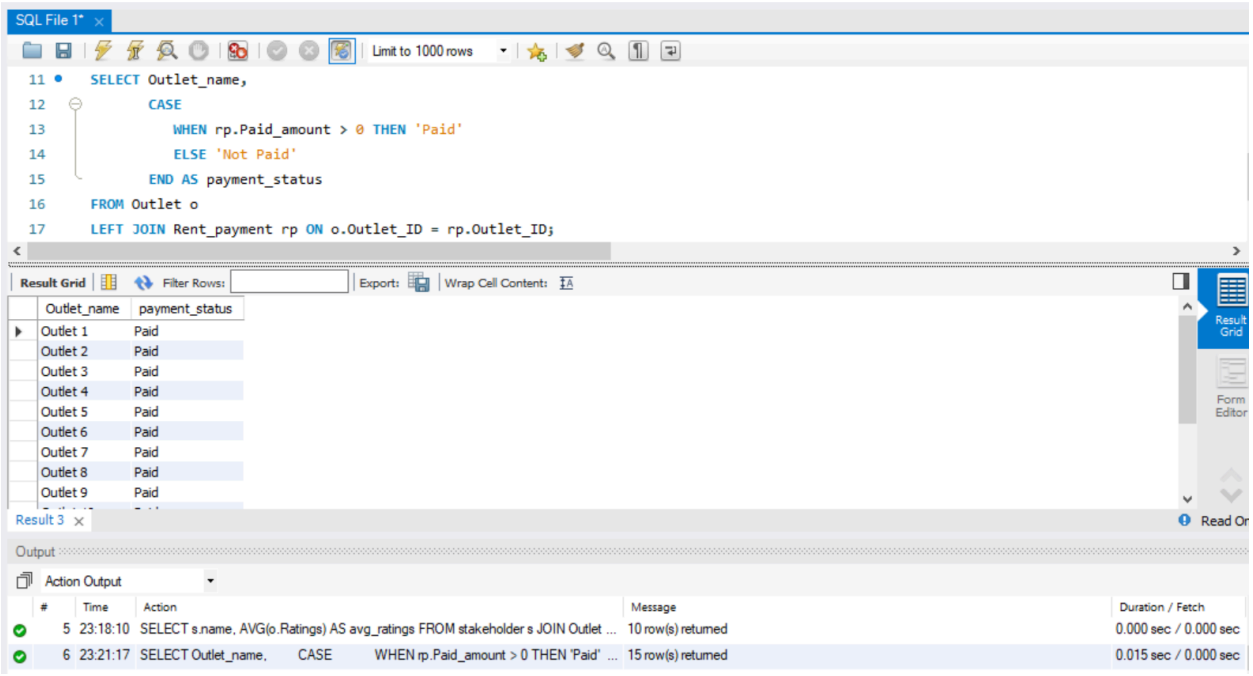
$$join \leftarrow stakeholder \bowtie_{stakeholder.stakeholder\_id = outlet.stakeholder\_id} Outlet$$

$$avg \leftarrow \gamma_{stakeholder\_name,\ avg\_ratings}(join)$$

$$result \leftarrow \Pi_{stakeholder\_name,\ avg\_ratings}(avg)$$

Here, we first perform a natural join between the stakeholder and Outlet tables on the common attribute stakeholder_id. Then, we compute the average ratings for each stakeholder using the aggregation operator. Finally, we project the stakeholder name and average ratings from the result to get the final output.

**Query with outer join and case statement:**
Retrieve all outlets and their corresponding rent payment status (paid or not paid).




$$\Pi_{Outlet\_name,\ payment\_status}\left(\rho_{1}\left(Outlet \times Rent\_payment\right)\right) \bowtie_{Outlet.Outlet\_ID = Rent\_payment.Outlet\_ID} \sigma_{Rent\_payment.Paid\_amount > 0}$$

Perform a natural join between the Outlet and Rent_payment relations on the Outlet_ID attribute.
Apply a selection (σ) operation to filter rows where the Paid_amount in the Rent_payment relation is greater than 0.
Project (π) the Outlet_name and payment_status attributes from the resulting relation.

**Query with Aggregate function:**
Retrieve the total number of employees in each outlet.

```
19 •  SELECT o.Outlet_name, COUNT(*) AS total_employees
20    FROM Outlet AS o
21    JOIN Employees AS e ON o.Outlet_ID = e.Outlet_ID
22    GROUP BY o.Outlet_name;
```

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

| Outlet_name | total_employees |
|---|---|
| Outlet 1 | 2 |
| Outlet 2 | 2 |
| Outlet 3 | 2 |
| Outlet 4 | 2 |
| Outlet 5 | 2 |

Result 4

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 4 | 23:14:26 | SELECT s.name, AVG(o.Ratings) AS avg_ratings FROM stakeholder s JOIN Outlet ... | 10 row(s) returned | 0.015 sec / 0.000 sec |
| 5 | 23:18:10 | SELECT s.name, AVG(o.Ratings) AS avg_ratings FROM stakeholder s JOIN Outlet ... | 10 row(s) returned | 0.000 sec / 0.000 sec |
| 6 | 23:21:17 | SELECT Outlet_name, CASE WHEN rp.Paid_amount > 0 THEN 'Paid' ... | 15 row(s) returned | 0.015 sec / 0.000 sec |
| 7 | 23:26:41 | SELECT o.Outlet_name, COUNT(*) AS total_employees FROM Outlet AS o JOIN E... | 5 row(s) returned | 0.000 sec / 0.000 sec |

$$\Pi_{Outlet\_name,\ total\text{-}employees}\left(\gamma_{Outlet\_name,\ COUNT(*)\ \rightarrow\ total\_employees}\left(\sigma_{Outlet.Outlet\_ID\ =\ Employees.Outlet\_ID}\left(Outlet \times Employees\right)\right)\right)$$

Perform the cross product (X) between the Outlet and Employees relations.
Apply a selection (σ) operation to filter rows where the Outlet_ID in the Outlet relation matches the Outlet_ID in the Employees relation.
Perform an aggregation (γ) operation to group the resulting relation by Outlet_name and calculate the count of employees for each group.
Project (π) the Outlet_name and total_employees attributes from the resulting relation.

**Query with Rename function:**

```
34 •  alter table outlet
35    RENAME COLUMN Outlet_ID TO ID;
36
37 •  select ID from outlet;
```

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

| ID |
|---|
| 1 |
| 11 |
| 2 |
| 12 |
| 3 |
| 13 |

outlet 5

Apply | Revert

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 14 | 23:56:44 | alter table outlet RENAME COLUMN Outlet_ID TO ID | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.078 sec |
| 15 | 23:59:13 | select Outlet_ID from outlet LIMIT 0, 1000 | Error Code: 1054. Unknown column 'Outlet_ID' in 'field list' | 0.000 sec |
| 16 | 23:59:19 | select ID from outlet LIMIT 0, 1000 | 15 row(s) returned | 0.000 sec / 0.000 sec |

$\rho_{outlet\_ID,\ ID}(outlet)$

## Contribution details by each group member:

**DHARAVATH MAHESH(22110073):** Group(G2)

- Brainstormed and wrote the Referential integrity violations and solved them via SQL
- Wrote SQL queries for storage of an image along with a caption into the database
- Helped in creating SQL queries which involves creating users, views
- contributions in the documentation

**SHUBHAM KSHIRSAGAR :** Group(G2)

- Brainstorming for indexing and population of the database
- Wrote SQL queries which involved, Creating users, views, and accessing and revoking permissions for different views and tables.
- Helped in creating user defined data type, and to display it using joins.
- Brainstorming to display errors on violation of constraints.
- Brainstormed and wrote the Relational Algebra Queries for Part G1 and G2.

**MALLEPOGULA CHARAN TEJA(22110136):** Group(G2)

- Brainstormed and wrote the Referential integrity violations and solved them via SQL
- Brainstormed and wrote some part of the Q2 of Responsibility G1&G2
- Helped in creating SQL queries which involves creating users, views
- Helped in the documentation of assignment

**SHIPRA KETHWALIA(23210091):** Group(G2)

- Brainstorming in the modification of SQL queries for the database and implementation of indexing.
- Helped in solving the scope identity and primary key violation during formation of the tables in MySQL.
- Wrote SQL queries for setting up the user, tables, views.
- Wrote SQL queries to implement and test the granting permissions and revoking permission component of the assignment.
- Helped in compiling the overall database and ensuring sanity checks for different tables and report documentation.

**Harsh Kumar Keshri (22110094)**: Group(G1)

- Populated all the entities with the desired data with the constraints mentioned in the previous assignment with all the ACID properties. [ Answer 1 of G1]
- Created user-defined data types and also helped in table extension. [Answer 2 of G2]
- Helped in G1 and G2 parts in writing the queries regarding the natural join, outer join and renaming.
- Also helped in writing the relational algebra for all the queries given in the G1 and G2 parts.

**Krish Raj (20110160)**: Group(G1)

- Implemented indexing and explained the reasons wherever used. [Answer 2 of G1]
- Helped in the population of the database.
- Helped in G1 and G2 parts in writing the queries regarding the natural join, outer join and renaming.
- Helped on nested queries, queries that give errors, and conversion of queries into relational algebra.
- Helping in the documentation for assignment

**Yajurvedh Bodala (19110077)** : Group(G1)

- Brainstorming to find appropriate columns and queries to implement indexing.
- Brainstorming to find appropriate SQL queries in the G1&G2 part to throw up an error.
- Brainstorming to find appropriate SQL queries with different kinds of aggregate functions, and case statements.

**Susmita** (22110265) :Group (G1)

- Helped in solving questions in G2 by finding out appropriate queries.
- Brainstorming of ideas in G1 and researching on queries and MySql

- Helped in researching about indexing

## Reference:

1 *MySQL forums :: General* (no date a) *MySQL*. Available at: https://forums.mysql.com/read.php?20%2C17671%2C27914

2 *Lecture 6: Formal_relational_query_languages.PDF* (no date) *Google Drive*. Available at: https://drive.google.com/file/d/19QgWCbxgwVKiK7-oe5l_6pSWe_hNjb9J/view