

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

```
6 • explain analyze select * from stakeholder where entry_date > 2023-05-04;
```

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

EXPLAIN

```

-> Filter: (stakeholder.entry_date > <cache>((...))
-> Filter: (stakeholder.entry_date > <cache>(((2023 - 5) - 4))) (cost=1.75 rows=5) (actual time=0.0837..0.109 rows=15 loops=1)
-> Table scan on stakeholder (cost=1.75 rows=15) (actual time=0.0796..0.101 rows=15 loops=1)

```

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

```
6 • explain analyze select * from stakeholder where entry_date > 2023-05-04;
```

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

EXPLAIN

```

-> Filter: (stakeholder.entry_date > <cache>((...
-> Filter: (stakeholder.entry_date > <cache>(((2023 - 5) - 4))) (cost=1.75 rows=5) (actual time=0.054..0.0644 rows=15 loops=1)
-> Table scan on stakeholder (cost=1.75 rows=15) (actual time=0.0514..0.0606 rows=15 loops=1)

```

- 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

```
17 • explain analyze select * from outlet where Ratings > 4.5;
```

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

EXPLAIN

```

-> Filter: (outlet.Ratings > 4.5) (cost=1.75 rows=5)
  -> Filter: (outlet.Ratings > 4.5) (cost=1.75 rows=5 actual time=0.692..0.71)
    -> Table scan on outlet (cost=1.75 rows=15 actual time=0.675..0.706)

```

With Indexing

```
17 • explain analyze select * from outlet where Ratings > 4.5;
```

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

EXPLAIN

```

-> Filter: (outlet.Ratings > 4.5) (cost=1.75 rows=5)
  -> Filter: (outlet.Ratings > 4.5) (cost=1.75 rows=5 actual time=0.0466..0.0639)
    -> Table scan on outlet (cost=1.75 rows=15 actual time=0.037..0.0608)

```

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

Rent_payment

Without Indexing

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

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

EXPLAIN

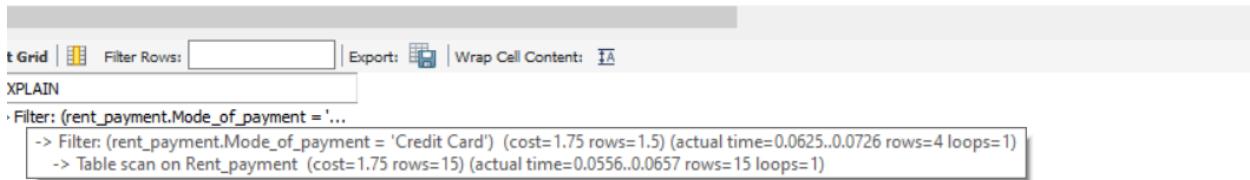
```

-> Filter: (rent_payment.Mode_of_payment = 'Credit Card')
  -> Filter: (rent_payment.Mode_of_payment = 'Credit Card') (cost=1.75 rows=1.5 actual time=0.18..0.212)
    -> Table scan on Rent_payment (cost=1.75 rows=15 actual time=0.156..0.186)

```

With Indexing

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



The screenshot shows the Explain Grid interface. The query is `explain analyze select * from Rent_payment where Mode_of_payment = 'Credit Card';`. The EXPLAIN output shows the following plan:

```

EXPLAIN
Filter: (rent_payment.Mode_of_payment = '...
-> Filter: (rent_payment.Mode_of_payment = 'Credit Card') (cost=1.75 rows=15) (actual time=0.0625..0.0726 rows=4 loops=1)
-> Table scan on Rent_payment (cost=1.75 rows=15) (actual time=0.0556..0.0657 rows=15 loops=1)

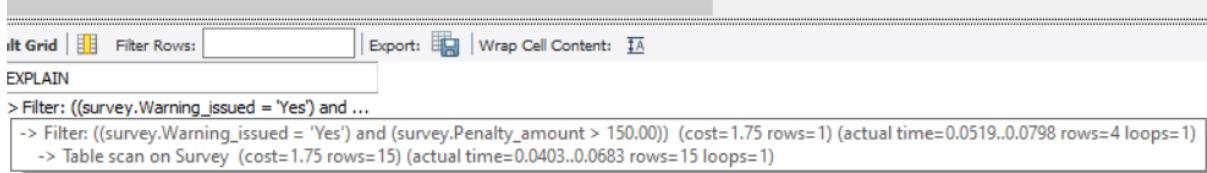
```

- 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;`



The screenshot shows the Explain Grid interface. The query is `explain analyze select * from Survey where Warning_issued = 'Yes' and Penalty_amount > 150;`. The EXPLAIN output shows the following plan:

```

EXPLAIN
> Filter: ((survey.Warning_issued = 'Yes') and ...
-> Filter: ((survey.Warning_issued = 'Yes') and (survey.Penalty_amount > 150.00)) (cost=1.75 rows=1) (actual time=0.0519..0.0798 rows=4 loops=1)
-> Table scan on Survey (cost=1.75 rows=15) (actual time=0.0403..0.0683 rows=15 loops=1)

```

With Indexing

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



The screenshot shows the Explain Grid interface. The query is `explain analyze select * from Survey where Warning_issued = 'Yes' and Penalty_amount > 150;`. The EXPLAIN output shows the following plan:

```

EXPLAIN
-> Filter: ((survey.Warning_issued = 'Yes') and ...
-> Filter: ((survey.Warning_issued = 'Yes') and (survey.Penalty_amount > 150.00)) (cost=1.75 rows=1) (actual time=0.046..0.0677 rows=4 loops=1)
-> Table scan on Survey (cost=1.75 rows=15) (actual time=0.0349..0.0591 rows=15 loops=1)

```

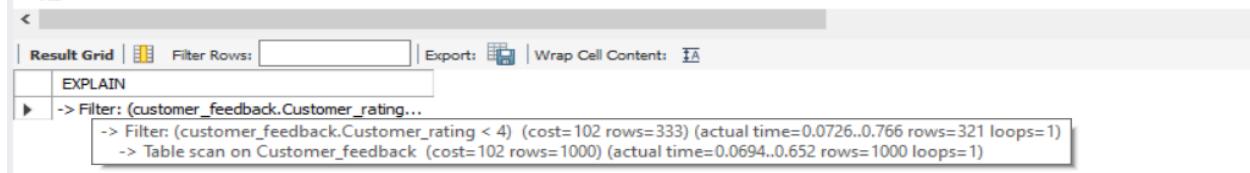
- Query execution time before indexing: 0.0519 s
- Query execution time after indexing: 0.046 s

Customer_feedback

Without Indexing

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

42



The screenshot shows the Explain Grid interface. The query is `explain analyze select * from Customer_feedback where Customer_rating < 4;`. The EXPLAIN output shows the following plan:

```

EXPLAIN
-> Filter: (customer_feedback.Customer_rating...
-> Filter: (customer_feedback.Customer_rating < 4) (cost=102 rows=333) (actual time=0.0726..0.766 rows=321 loops=1)
-> Table scan on Customer_feedback (cost=102 rows=1000) (actual time=0.0694..0.652 rows=1000 loops=1)

```

With Indexing

```
41 • explain analyze select * from Customer_feedback where Customer_rating < 4;
42
<
| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
| EXPLAIN     |
| -> Filter: (customer_feedback.Customer_rating...
    -> Filter: (customer_feedback.Customer_rating < 4) (cost=102 rows=333) (actual time=0.0478..0.36 rows=321 loops=1)
        -> Table scan on Customer_feedback (cost=102 rows=1000) (actual time=0.0456..0.312 rows=1000 loops=1)
```

Query execution time before indexing: 0.0726 s

Query execution time after indexing: 0.0478 s

InventoryWithout Indexing

```
46 • explain analyze select * from Inventory where Price > 50 and Price < 100;
47
<
| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
| EXPLAIN     |
| -> Filter: ((inventory.Price > 50.00) and (inven...
    -> Filter: ((inventory.Price > 50.00) and (inventory.Price < 100.00)) (cost=1.75 rows=1.67) (actual time=0.119..0.124 rows=1 loops=1)
        -> Table scan on Inventory (cost=1.75 rows=15) (actual time=0.0996..0.115 rows=15 loops=1)
```

With Indexing

```
46 • explain analyze select * from Inventory where Price > 50 and Price < 100;
47
<
| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
| EXPLAIN     |
| -> Filter: ((inventory.Price > 50.00) and (inven...
    -> Filter: ((inventory.Price > 50.00) and (inventory.Price < 100.00)) (cost=1.75 rows=1.67) (actual time=0.0536..0.0602 rows=1 loops=1)
        -> Table scan on Inventory (cost=1.75 rows=15) (actual time=0.0442..0.0532 rows=15 loops=1)
```

Query execution time before indexing: 0.119 s

Query execution time after indexing: 0.0536 s

Employees

Without Indexing

```
53 • explain analyze select * from Employees where Role = 'Cashier';
54
```

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

EXPLAIN
-> Filter: (employees.'Role' = 'Cashier') (cost...
-> Filter: (employees.'Role' = 'Cashier') (cost=1.25 rows=1) (actual time=0.0876..0.0931 rows=2 loops=1)
-> Table scan on Employees (cost=1.25 rows=10) (actual time=0.0807..0.0861 rows=10 loops=1)

With Indexing

```
1 • explain analyze select * from Employees where Role = 'Cashier';
```

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

EXPLAIN
-> Filter: (employees.'Role' = 'Cashier') (cost...
-> Filter: (employees.'Role' = 'Cashier') (cost=1.25 rows=1) (actual time=0.0377..0.0449 rows=2 loops=1)
-> Table scan on Employees (cost=1.25 rows=10) (actual time=0.0332..0.0404 rows=10 loops=1)

Query execution time before indexing: 0.0876 s

Query execution time after indexing: 0.0377 s

Contract

Without Indexing

```
60 • explain analyze select * from contract where outlet_id = 1;
```

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

EXPLAIN
-> Index lookup on contract using Outlet_ID (O... -> Index lookup on contract using Outlet_ID (Outlet_ID=1) (cost=0.35 rows=1) (actual time=0.0674..0.0691 rows=1 loops=1)

With Indexing

```
60 • explain analyze select * from contract where outlet_id = 1;
```

The screenshot shows the PostgreSQL Explain Analyze output for the query `explain analyze select * from contract where outlet_id = 1;`. The output is as follows:

```
EXPLAIN
-> Index lookup on contract using Outlet_ID (O...
-> Index lookup on contract using Outlet_ID (Outlet_ID=1) (cost=0.35 rows=1) (actual time=0.0315..0.0332 rows=1 loops=1)
```

The EXPLAIN output indicates that the query uses an index lookup on the `contract` table, specifically using the `Outlet_ID` index. The cost is 0.35, and the actual time taken is between 0.0315 and 0.0332 seconds for one row.

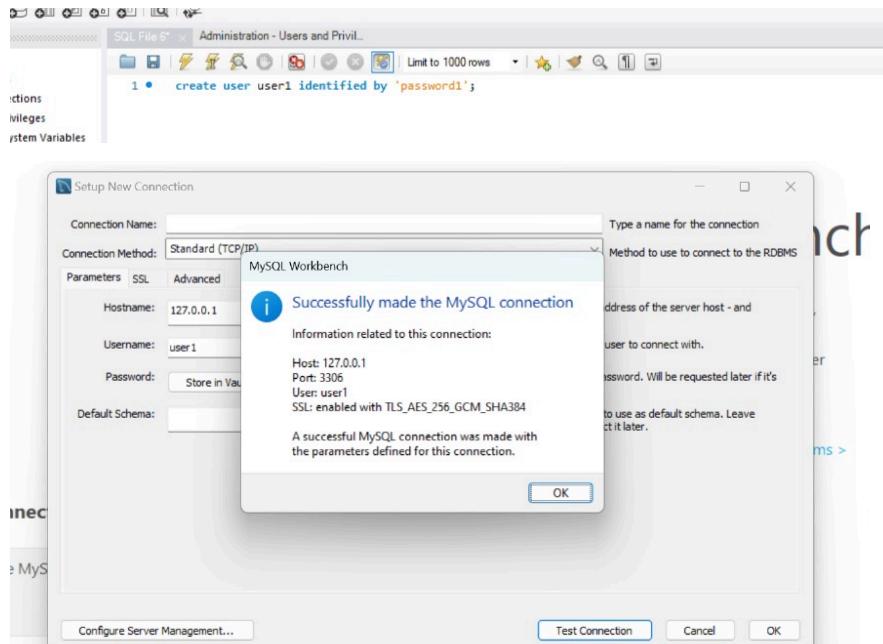
Query execution time before indexing: 0.0674 s

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

The screenshot shows the MySQL Workbench interface with a SQL editor titled 'SQL File 6'. The code entered is:

```

1 * create view view1 as
2 SELECT
3 stakeholder.name , stakeholder.Stakeholder_ID , outlet.Outlet_name , stakeholder.email, outlet.Country_Code,outlet.Contact_No
4 FROM outlet INNER JOIN stakeholder ON stakeholder.stakeholder_id = outlet.Stakeholder_ID;

```

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.rahu@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.rahu@iitgn.ac.in	+91	1234567890

view1 1 ×

Making View 2

SQL File 6*

```

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

```

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	7300.00

view2 2 ×

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

```
SQL File 6* SQL File 3* Administration - Users and Privileges
1 • GRANT SELECT, UPDATE, DELETE ON employees TO user1;
2
3
```

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

SELECT:

Permission to User1 on View1

```
SQL File 6* SQL File 3* Administration - Users and Privileges
1 GRANT SELECT ON view1 TO user1;
```

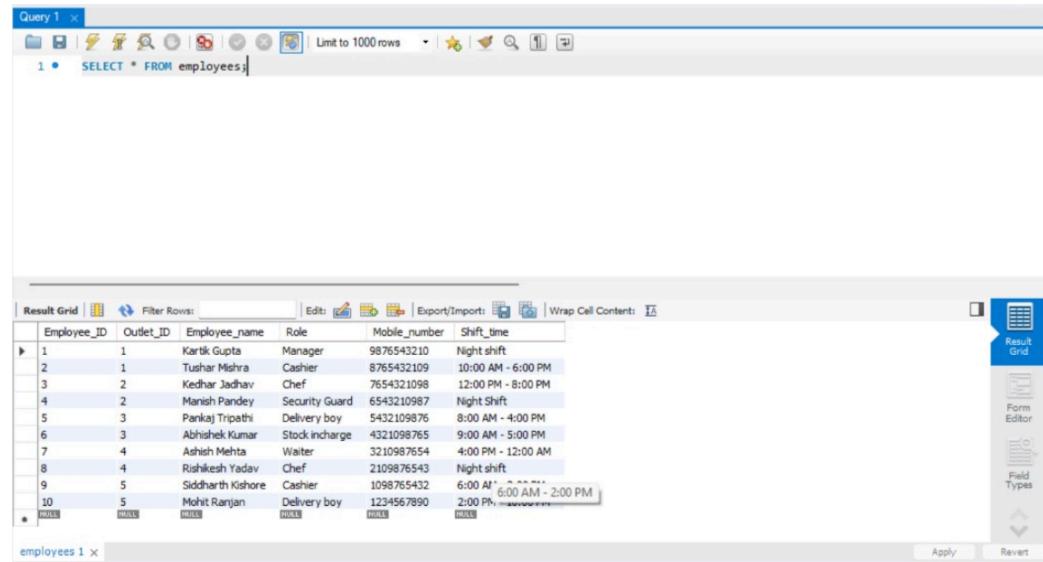
5. Try to perform SELECT, UPDATE, and DELETE operations on "table1" and "view1" as "user1" and report your findings:

- 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



The screenshot shows the MySQL Workbench interface with a query editor and a result grid.

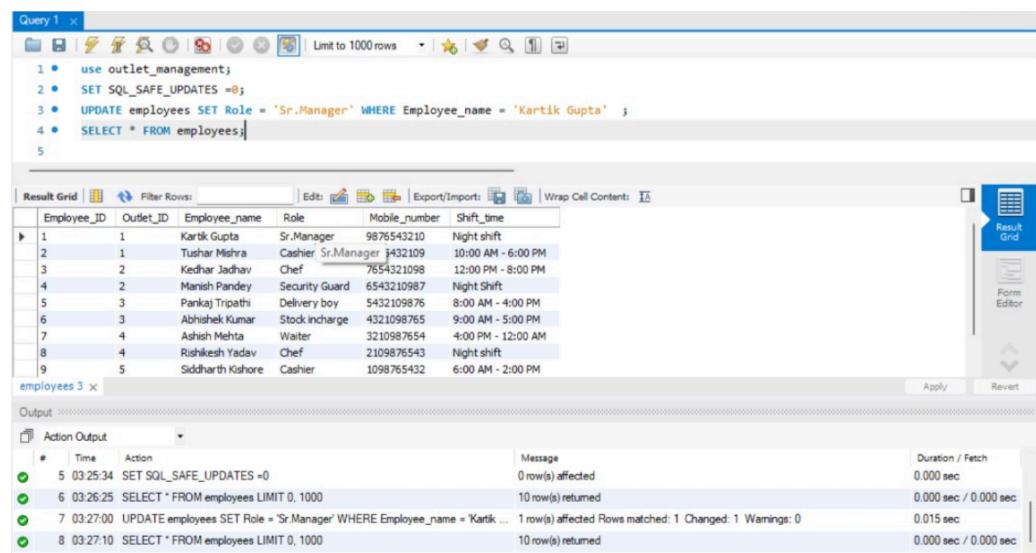
Query Editor:

```
Query 1
1 •  SELECT * FROM employees;
```

Result Grid:

Employee_ID	Outlet_ID	Employee_name	Role	Mobile_number	Shift_time
1	1	Kartik Gupta	Manager	9876543210	Night shift
2	1	Tushar Mishra	Cashier	8765432109	10:00 AM - 6:00 PM
3	2	Kedhar Jadhav	Chef	7654321098	12:00 PM - 8:00 PM
4	2	Manish Pandey	Security Guard	6543210987	Night Shift
5	3	Pankaj Tripathi	Delivery boy	5432109876	8:00 AM - 4:00 PM
6	3	Abhishek Kumar	Stock Incharge	4321098765	9:00 AM - 5:00 PM
7	4	Ashish Mehta	Waiter	3210987654	4:00 PM - 12:00 AM
8	4	Rishikesh Yadav	Chef	2109876543	Night shift
9	5	Siddharth Kishore	Cashier	1098765432	6:00 AM - 12:00 PM
10	5	Mohit Ranjan	Delivery boy	1234567890	2:00 PM - 6:00 AM

UPDATE: No Error



The screenshot shows the MySQL Workbench interface with a query editor and a result grid.

Query Editor:

```
Query 1
1 •  use outlet_management;
2 •  SET SQL_SAFE_UPDATES = 0;
3 •  UPDATE employees SET Role = 'Sr.Manager' WHERE Employee_name = 'Kartik Gupta';
4 •  SELECT * FROM employees;
```

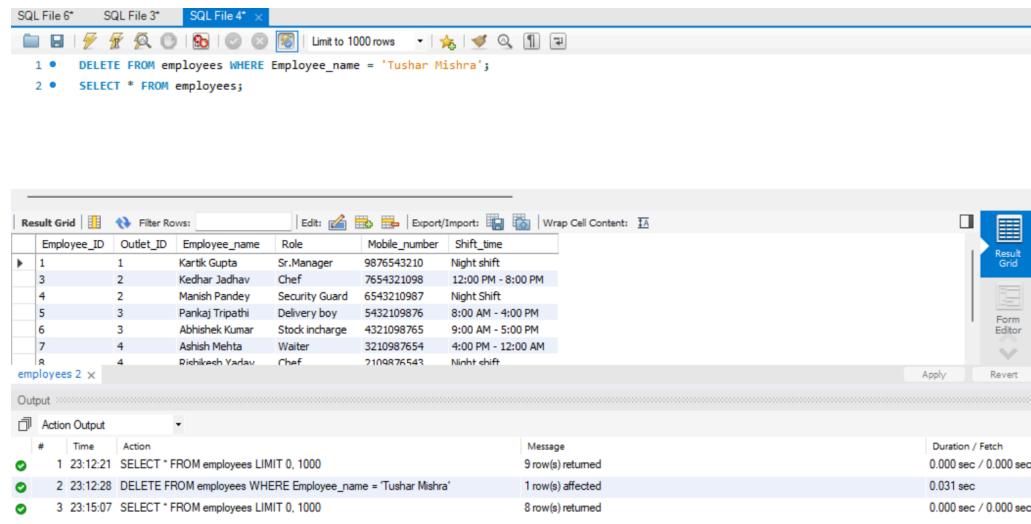
Result Grid:

Employee_ID	Outlet_ID	Employee_name	Role	Mobile_number	Shift_time
1	1	Kartik Gupta	Sr.Manager	9876543210	Night shift
2	1	Tushar Mishra	Sr.Manager	8765432109	10:00 AM - 6:00 PM
3	2	Kedhar Jadhav	Chef	7654321098	12:00 PM - 8:00 PM
4	2	Manish Pandey	Security Guard	6543210987	Night Shift
5	3	Pankaj Tripathi	Delivery boy	5432109876	8:00 AM - 4:00 PM
6	3	Abhishek Kumar	Stock Incharge	4321098765	9:00 AM - 5:00 PM
7	4	Ashish Mehta	Waiter	3210987654	4:00 PM - 12:00 AM
8	4	Rishikesh Yadav	Chef	2109876543	Night shift
9	5	Siddharth Kishore	Cashier	1098765432	6:00 AM - 12:00 PM

Action Output:

#	Time	Action	Message	Duration / Fetch
5	03:25:34	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.000 sec
6	03:26:25	SELECT * FROM employees LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec
7	03:27:00	UPDATE employees SET Role = 'Sr.Manager' WHERE Employee_name = 'Kartik ...	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.015 sec
8	03:27:10	SELECT * FROM employees LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec

DELETE: No Error



The screenshot shows a SQL Server Management Studio window with four tabs at the top: SQL File 6*, SQL File 3*, SQL File 4*, and the active tab SQL File 2*. The SQL File 2* tab contains the following SQL code:

```

1 •  DELETE FROM employees WHERE Employee_name = 'Tushar Mishra';
2 •  SELECT * FROM employees;

```

Below the code, the Results pane displays a table titled "employees 2" with 8 rows of data. The columns are Employee_ID, Outlet_ID, Employee_name, Role, Mobile_number, and Shift_time. The data includes entries for Kartik Gupta, Kedhar Jadhav, Manish Pandey, Pankaj Tripathi, Abhishek Kumar, Ashish Mehta, and Rishabh Yardi.

The Output pane shows the execution log with three entries:

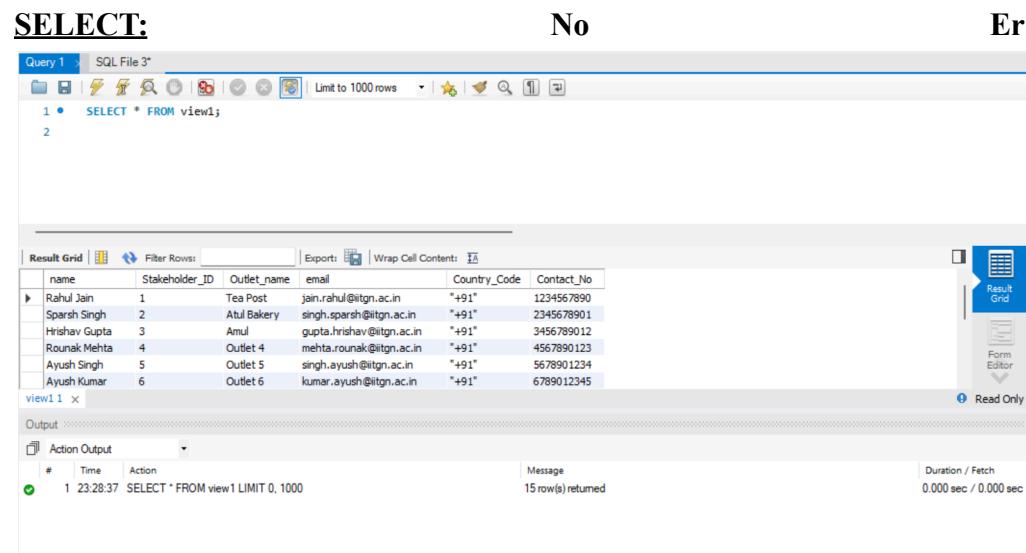
#	Time	Action	Message	Duration / Fetch
1	23:12:21	SELECT * FROM employees LIMIT 0, 1000	9 row(s) returned	0.000 sec / 0.000 sec
2	23:12:28	DELETE FROM employees WHERE Employee_name = 'Tushar Mishra'	1 row(s) affected	0.031 sec
3	23:15:07	SELECT * FROM employees LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec

- **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:

<u>SELECT:</u>		No	Error
Query 1	SQL File 3*		
 <pre> 1 • SELECT * FROM view1; 2 </pre>			

The screenshot shows a SQL Server Management Studio window with two tabs at the top: Query 1 and SQL File 3*. The Query 1 tab contains the SQL code:

```

1 •  SELECT * FROM view1;
2

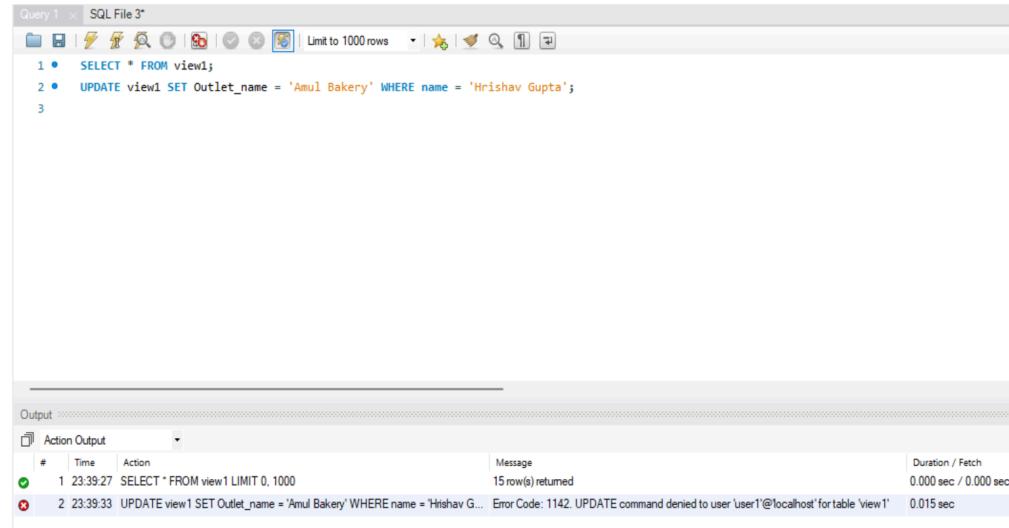
```

Below the code, the Results pane displays a table titled "view1 1" with 6 rows of data. The columns are name, Stakeholder_ID, Outlet_name, email, Country_Code, and Contact_No. The data includes entries for Rahul Jain, Sparsh Singh, Hrishav Gupta, Rounak Mehta, Ayush Singh, and Ayush Kumar.

The Output pane shows the execution log with one entry:

#	Time	Action	Message	Duration / Fetch
1	23:28:37	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec

UPDATE: Encountered Error Code: 1142. UPDATE command denied to user ‘user1’@‘localhost’ for table ‘view1’



The screenshot shows a MySQL Workbench interface with two tabs: 'Query 1' and 'SQL File 3'. The 'Query 1' tab contains the following SQL code:

```

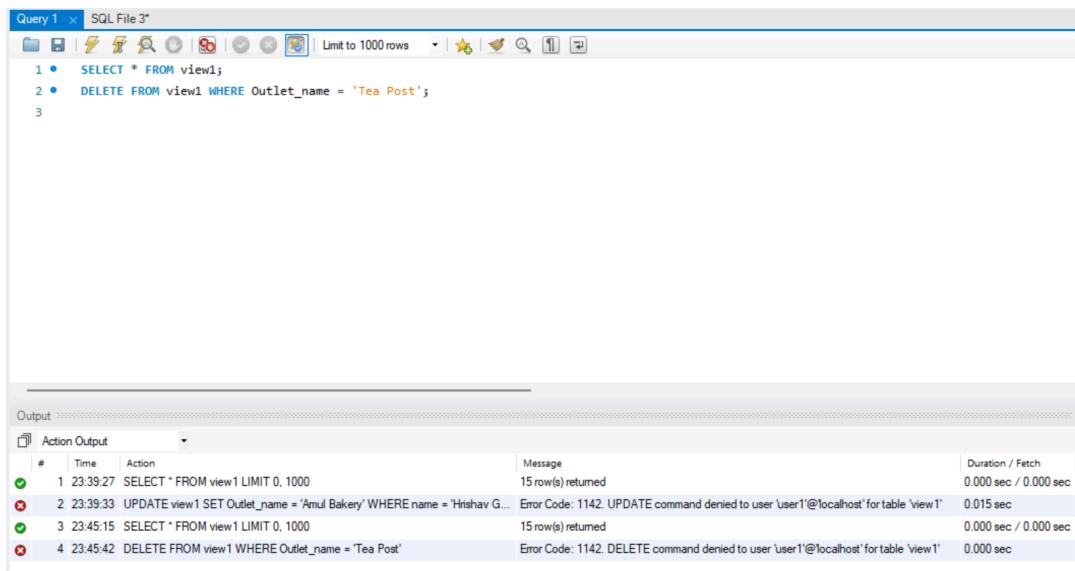
1 •  SELECT * FROM view1;
2 •  UPDATE view1 SET Outlet_name = 'Amul Bakery' WHERE name = 'Hrishav Gupta';
3

```

The 'Output' tab displays the results of the queries. It has a header row with columns: #, Time, Action, Message, and Duration / Fetch. The data shows:

#	Time	Action	Message	Duration / Fetch
1	23:39:27	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec
2	23:39:33	UPDATE view1 SET Outlet_name = 'Amul Bakery' WHERE name = 'Hrishav G...' Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	0.015 sec

DELETE: Encountered Error Code: 1142. DELETE command denied to user ‘user1’@‘localhost’ for table ‘view1’.



The screenshot shows a MySQL Workbench interface with two tabs: 'Query 1' and 'SQL File 3'. The 'Query 1' tab contains the following SQL code:

```

1 •  SELECT * FROM view1;
2 •  DELETE FROM view1 WHERE Outlet_name = 'Tea Post';
3

```

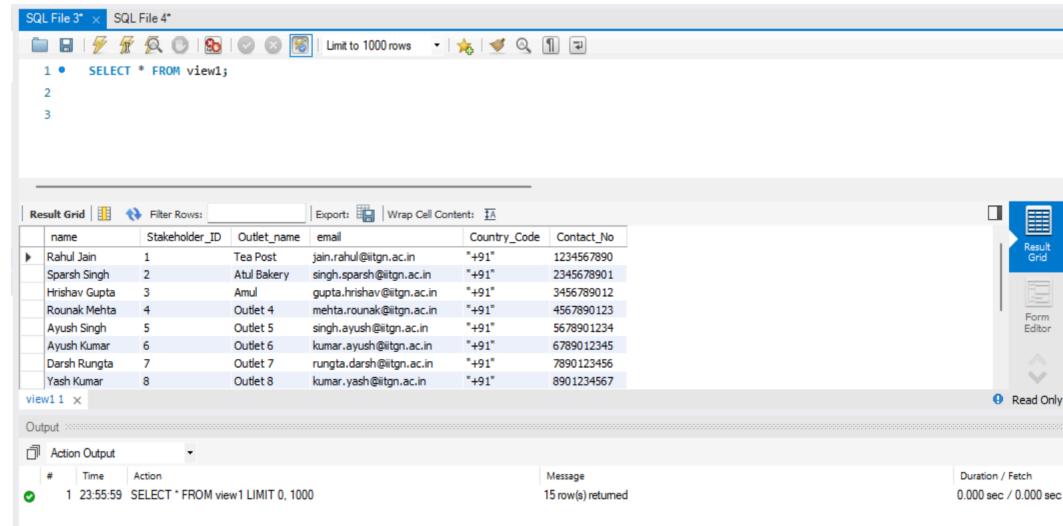
The 'Output' tab displays the results of the queries. It has a header row with columns: #, Time, Action, Message, and Duration / Fetch. The data shows:

#	Time	Action	Message	Duration / Fetch
1	23:39:27	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec
2	23:39:33	UPDATE view1 SET Outlet_name = 'Amul Bakery' WHERE name = 'Hrishav G...' Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'	0.015 sec
3	23:45:15	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec
4	23:45:42	DELETE FROM view1 WHERE Outlet_name = 'Tea Post'	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'	0.000 sec

- If we perform the “Select”, “Update” and “Delete” operation in the localhost MySQL server, following are the observations:

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

SELECT: No error



SQL File 3* × SQL File 4*

```
1 •  SELECT * FROM view1;
```

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

name	Stakeholder_ID	Outlet_name	email	Country_Code	Contact_No
Rahul Jain	1	Tea Post	jain.rahu@itgn.ac.in	+91	1234567890
Sparsh Singh	2	Atul Bakery	singh.sparsh@itgn.ac.in	+91	2345678901
Hrishav Gupta	3	Amul	gupta.hrishav@itgn.ac.in	+91	3456789012
Rounak Mehta	4	Outlet 4	mehta.rounak@itgn.ac.in	+91	4567890123
Ayush Singh	5	Outlet 5	singh.ayush@itgn.ac.in	+91	5678901234
Ayush Kumar	6	Outlet 6	kumar/ayush@itgn.ac.in	+91	6789012345
Darsh Rungta	7	Outlet 7	rungta.darsh@itgn.ac.in	+91	7890123456
Yash Kumar	8	Outlet 8	kumar.yash@itgn.ac.in	+91	8901234567

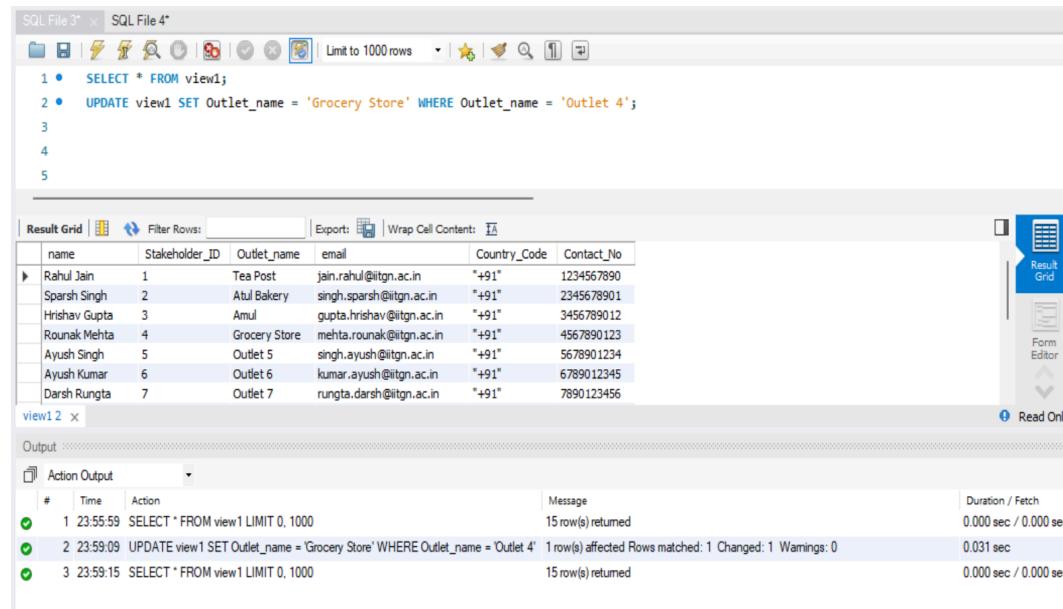
view1 1 ×

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	23:55:59	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec

UPDATE: No error



SQL File 3* × SQL File 4*

```
1 •  SELECT * FROM view1;
2 •  UPDATE view1 SET Outlet_name = 'Grocery Store' WHERE Outlet_name = 'Outlet 4';
3
4
5
```

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

name	Stakeholder_ID	Outlet_name	email	Country_Code	Contact_No
Rahul Jain	1	Tea Post	jain.rahu@itgn.ac.in	+91	1234567890
Sparsh Singh	2	Atul Bakery	singh.sparsh@itgn.ac.in	+91	2345678901
Hrishav Gupta	3	Amul	gupta.hrishav@itgn.ac.in	+91	3456789012
Rounak Mehta	4	Grocery Store	mehta.rounak@itgn.ac.in	+91	4567890123
Ayush Singh	5	Outlet 5	singh.ayush@itgn.ac.in	+91	5678901234
Ayush Kumar	6	Outlet 6	kumar/ayush@itgn.ac.in	+91	6789012345
Darsh Rungta	7	Outlet 7	rungta.darsh@itgn.ac.in	+91	7890123456

view1 2 ×

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	23:55:59	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec
2	23:59:09	UPDATE view1 SET Outlet_name = 'Grocery Store' WHERE Outlet_name = 'Outlet 4'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.031 sec
3	23:59:15	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec

DELETE: Encountered Error Code: 1395. Can not delete from join view 'outlet_management.view1'.

The screenshot shows a MySQL Workbench interface with two tabs: 'SQL File 3*' and 'SQL File 4*'. The SQL tab contains the following code:

```

1 •  SELECT * FROM view1;
2 •  DELETE FROM view1 WHERE Outlet_name = 'Tea Post';
3
4
5

```

The Result Grid displays data from view1:

name	Stakeholder_ID	Outlet_name	email	Country_Code	Contact_No
Rahul Jain	1	Tea Post	jain.rahu@itgn.ac.in	+91	1234567890
Sparsh Singh	2	Atul Bakery	singh.sparsh@itgn.ac.in	+91	2345678901
Hrishav Gupta	3	Amul	gupta.hrishav@itgn.ac.in	+91	3456789012
Rounak Mehta	4	Grocery Store	mehta.rounak@itgn.ac.in	+91	4567890123
Ayush Singh	5	Outlet 5	singh.ayush@itgn.ac.in	+91	5678901234
Ayush Kumar	6	Outlet 6	kumar.ayush@itgn.ac.in	+91	6789012345
Darsh Rungta	7	Outlet 7	rungta.darsh@itgn.ac.in	+91	7890123456

The Output pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	00:04:52	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec
2	00:07:01	DELETE FROM view1 WHERE Outlet_name = 'Tea Post'	Error Code: 1395. Can not delete from join view 'outlet_management.view1'	0.000 sec
3	00:07:05	SELECT * FROM view1 LIMIT 0, 1000	15 row(s) returned	0.000 sec / 0.000 sec

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.

The screenshot shows a MySQL Workbench interface with two tabs: 'SQL File 3*' and 'SQL File 4*'. The SQL tab contains the following code:

```

1 •  REVOKE UPDATE , DELETE ON employees FROM user1;
2 •  show grants for user1;

```

The Result Grid displays grants for user1@%:

Grants for user1@%
GRANT SELECT, CREATE ROLE, DROP ROLE ON *.* TO 'user1'@'%'
GRANT APPLICATION_PASSWORD_ADMIN,AUDIT_ABORT_EXEMPT,AUDIT_ADMIN,AUTHENTICATION_POLI...
GRANT SELECT ON `outlet_management`.`employees` TO 'user1'@'%'
GRANT SELECT, UPDATE, DELETE ON `outlet_management`.`outlet` TO 'user1'@'%'
GRANT SELECT ON `outlet_management`.`view1` TO 'user1'@'%'

The Output pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	00:23:32	REVOKE UPDATE , DELETE ON employees FROM user1	0 row(s) affected	0.016 sec
2	00:23:36	show grants for user1	5 row(s) returned	0.000 sec / 0.000 sec

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

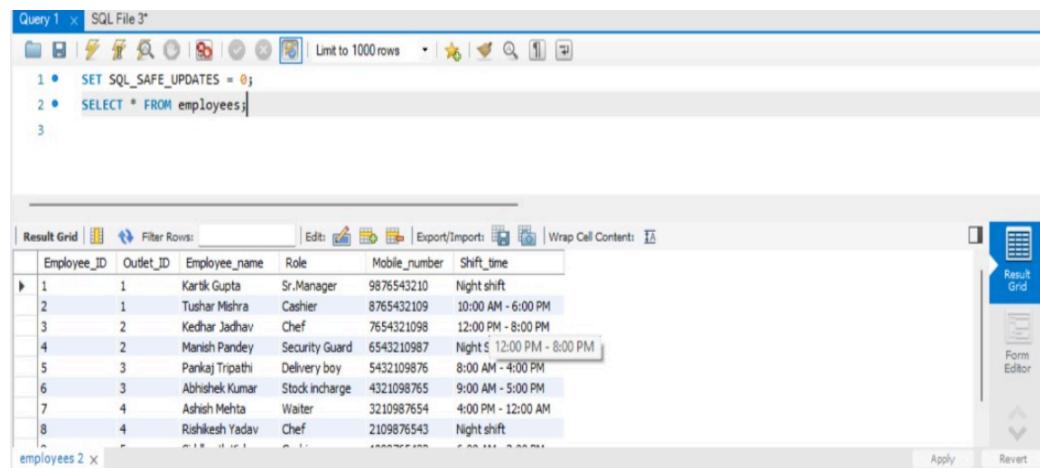
- **Select, Update and Delete operation on “table1”:**

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:

SELECT: No error



Query 1 × SQL File 3*

```

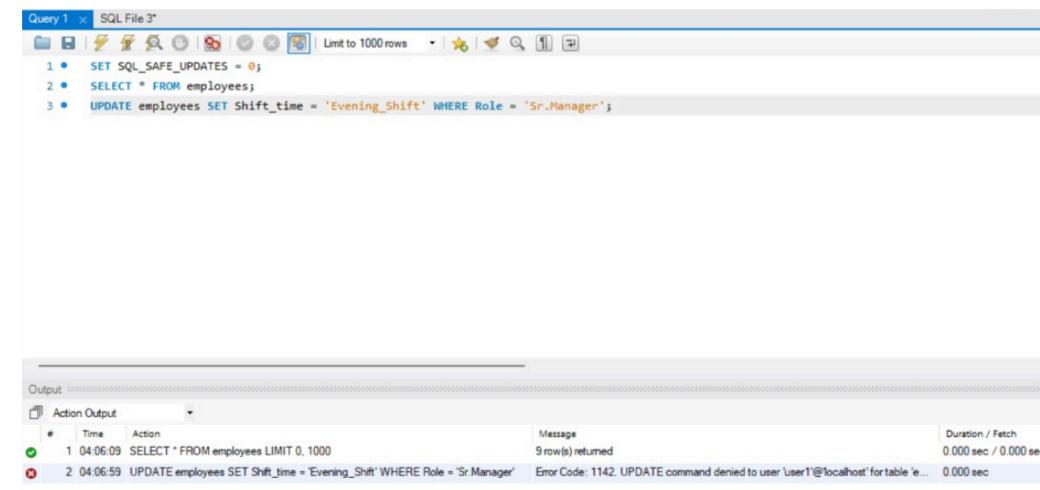
1 • SET SQL_SAFE_UPDATES = 0;
2 • SELECT * FROM employees;
3

```

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

Employee_ID	Outlet_ID	Employee_name	Role	Mobile_number	Shift_time
1	1	Kartik Gupta	Sr.Manager	9876543210	Night shift
2	1	Tushar Mishra	Cashier	8765432109	10:00 AM - 6:00 PM
3	2	Kedhar JadHAV	Chef	7654321098	12:00 PM - 8:00 PM
4	2	Manish Pandey	Security Guard	6543210987	Night Shift 12:00 PM - 8:00 PM
5	3	Pankaj Tripathi	Delivery boy	5432109876	8:00 AM - 4:00 PM
6	3	Abhishek Kumar	Stock Incharge	4321098765	9:00 AM - 5:00 PM
7	4	Ashish Mehta	Waiter	3210987654	4:00 PM - 12:00 AM
8	4	Rishikesh Yadav	Chef	2109876543	Night shift

UPDATE: Encountered Error Code: 1142 UPDATE command denied to user ‘user1’@‘localhost’ for table ‘employees’.



Query 1 × SQL File 3*

```

1 • SET SQL_SAFE_UPDATES = 0;
2 • SELECT * FROM employees;
3 • UPDATE employees SET Shift_time = 'Evening_Shift' WHERE Role = 'Sr.Manager';

```

Action Output

#	Time	Action	Message	Duration / Fetch
1	04:06:09	SELECT * FROM employees LIMIT 0, 1000	9 row(s) returned	0.000 sec / 0.000 sec
2	04:06:59	UPDATE employees SET Shift_time = 'Evening_Shift' WHERE Role = 'Sr.Manager'	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'employees'	0.000 sec

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

```

Query 1 SQL File 3* ×
1 • SELECT * FROM employees;
2 • DELETE FROM employee WHERE Employee_name = 'Tushar Mishra';

Output
Action Output
# Time Action
4 04:24:57 SELECT * FROM view1 LIMIT 0, 1000
5 04:25:02 DELETE FROM view1 WHERE name = 'Rounak Mehta'
6 04:25:56 SELECT * FROM employees LIMIT 0, 1000
7 04:26:13 DELETE FROM employee WHERE Employee_name = 'Tushar Mishra'

Message
15 row(s) returned
Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table ...
9 row(s) returned
Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table ...
Duration / Fetch
0.000 sec / 0.000 sec
0.000 sec
0.000 sec / 0.000 sec
0.000 sec
0.000 sec

```

- **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

```

Query 1 SQL File 3* ×
1 • SELECT * FROM view1;

Result Grid Filter Rows: Export: Wrap Cell Contents: 
name Stakeholder_ID Outlet_name email Country_Code Contact_No
Rahul Jain 1 Tea Post Jain.rehul@itgn.ac.in +91 1234567890
Sparsh Singh 2 Outlet 2 Singh.sparsh@itgn.ac.in +91 2345678901
Hrishav Gupta 3 Outlet 3 Gupta.Hrishav@itgn.ac.in +91 3456789012
Rounak Mehta 4 Outlet 4 Mehta.rounak@itgn.ac.in +91 4567890123
Ayush Singh 5 Outlet 5 Singh.ayush@itgn.ac.in +91 567890123

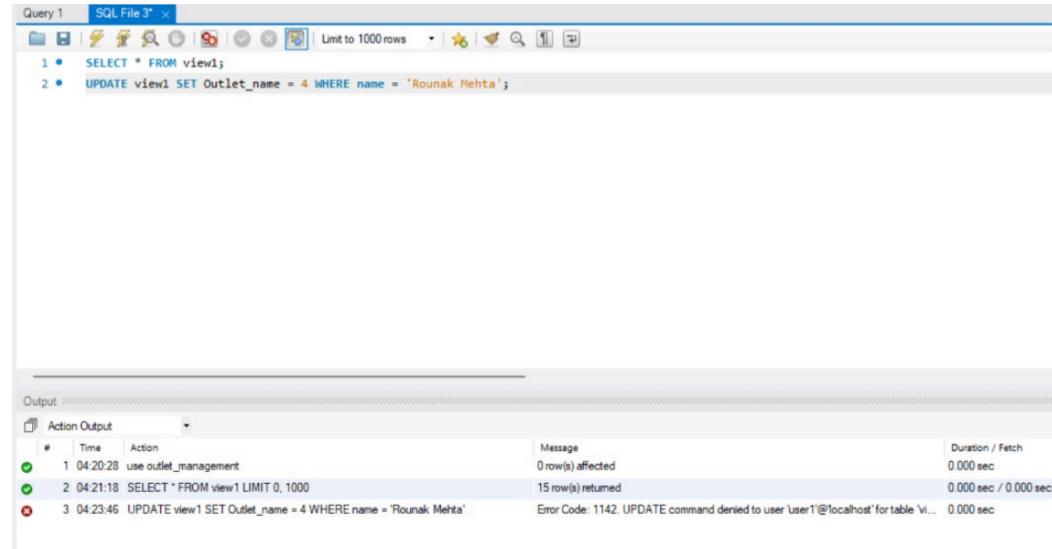
view1 4 ×

Output
Action Output
# Time Action
17 03:45:00 SELECT * FROM employees LIMIT 0, 1000
18 03:49:33 SELECT * FROM view1 LIMIT 0, 1000

Message
9 row(s) returned
15 row(s) returned
Duration / Fetch
0.000 sec / 0.000 sec
0.000 sec / 0.000 sec

```

UPDATE: Encountered Error Code: 1142 UPDATE command denied to user ‘user1’@‘localhost’ for table ‘view1’.



The screenshot shows a MySQL Workbench interface with two tabs: 'Query 1' and 'SQL File 3'. The 'Query 1' tab contains the following SQL code:

```

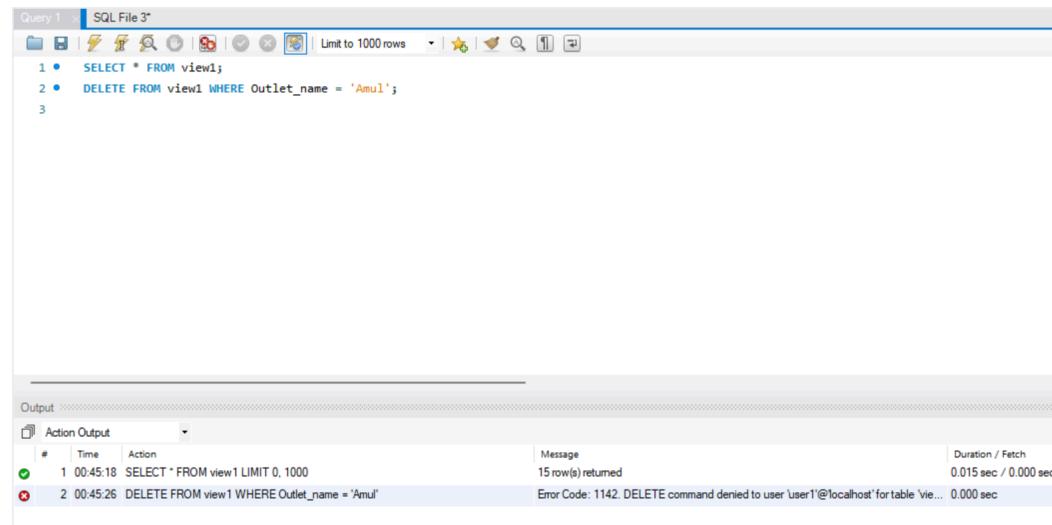
1 •  SELECT * FROM view1;
2 •  UPDATE view1 SET Outlet_name = 4 WHERE name = 'Rounak Mehta';

```

The 'Output' tab displays the execution log:

Action	Time	Message	Duration / Fetch
use outlet_management	04:20:28	0 row(s) affected	0.000 sec
SELECT * FROM view1 LIMIT 0, 1000	04:21:18	15 row(s) returned	0.000 sec / 0.000 sec
UPDATE view1 SET Outlet_name = 4 WHERE name = 'Rounak Mehta'	04:23:46	Error Code: 1142. UPDATE command denied to user 'user1'@'localhost' for table 'view1'.	0.000 sec

DELETE: Encountered Error Code: 1142. DELETE command denied to user ‘user1’@‘localhost’ for table ‘view1’.



The screenshot shows a MySQL Workbench interface with two tabs: 'Query 1' and 'SQL File 3'. The 'Query 1' tab contains the following SQL code:

```

1 •  SELECT * FROM view1;
2 •  DELETE FROM view1 WHERE Outlet_name = 'Amul';
3

```

The 'Output' tab displays the execution log:

Action	Time	Message	Duration / Fetch
SELECT * FROM view1 LIMIT 0, 1000	00:45:18	15 row(s) returned	0.015 sec / 0.000 sec
DELETE FROM view1 WHERE Outlet_name = 'Amul'	00:45:26	Error Code: 1142. DELETE command denied to user 'user1'@'localhost' for table 'view1'.	0.000 sec

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.

The screenshot shows the SQL Server Management Studio interface. In the top-left pane, there is a code editor window containing the following SQL script:

```

SQL File 3* SQL File 4* SQL File 5* SQL File 6* SQL File 7* SQL File 8* SQL File 8* X
1 • use outlet_management;
2 • select * from outlet;
3 • select * from stakeholder;
4
5 • INSERT INTO Outlet (Outlet_ID, Stakeholder_ID, Outlet_name, Location_name, Contact_No, timi
(20, 19, 'Outlet 0', 'Location 0', 1234567891, '9:00 AM - 6:00 PM', 4.5);

```

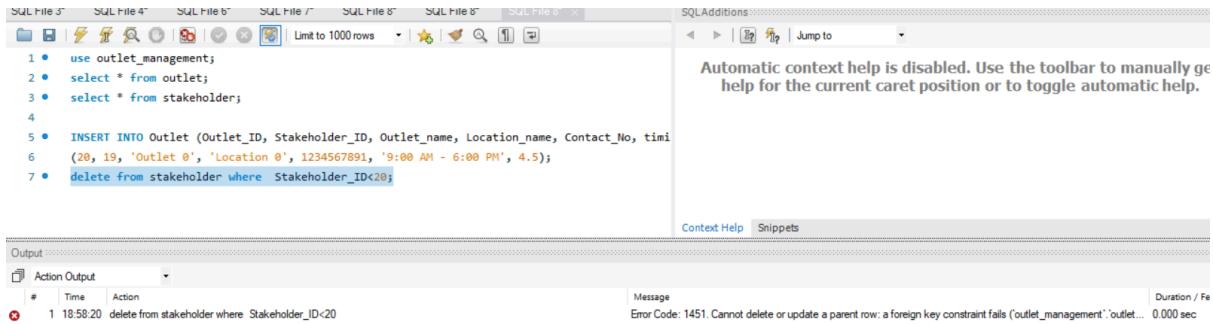
In the top-right pane, there is a message box stating: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

At the bottom, the Output window displays the following log entry:

Action	Time	Message	Duration / Fetch
Action Output	1 18:46:19	INSERT INTO Outlet (Outlet_ID, Stakeholder_ID, Outlet_name, Location_name, Contact_No, timings, Ratings) V... Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('outlet_management'.outlet', C...	0.000 sec

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.



The screenshot shows the SQL Server Management Studio interface. In the top pane, several tabs are open, and the current tab contains the following SQL code:

```

1 • use outlet_management;
2 • select * from outlet;
3 • select * from stakeholder;
4
5 • INSERT INTO Outlet (Outlet_ID, Stakeholder_ID, Outlet_name, Location_name, Contact_No, timi
(20, 19, 'Outlet 0', 'Location 0', 1234567891, '9:00 AM - 6:00 PM', 4.5);
6 • delete from stakeholder where Stakeholder_ID<20;

```

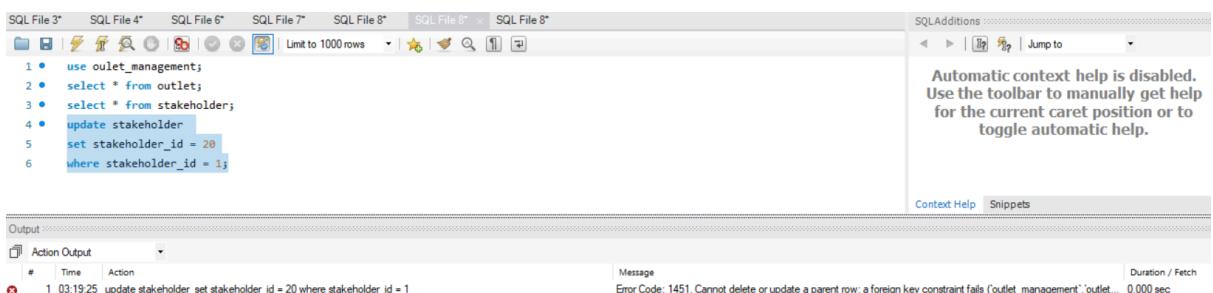
In the bottom pane, the 'Output' tab is selected, showing the execution results:

#	Time	Action
1	18:58:20	delete from stakeholder where Stakeholder_ID<20

Message: Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('outlet_management'.outlet...'). Duration / Fetch: 0.000 sec

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.



The screenshot shows the SQL Server Management Studio interface. In the top pane, several tabs are open, and the current tab contains the following SQL code:

```

1 • use outlet_management;
2 • select * from outlet;
3 • select * from stakeholder;
4 • update stakeholder
5   set stakeholder_id = 20
6   where stakeholder_id = 1;

```

In the bottom pane, the 'Output' tab is selected, showing the execution results:

#	Time	Action
1	03:19:25	update stakeholder set stakeholder_id = 20 where stakeholder_id = 1

Message: Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('outlet_management'.outlet...'). Duration / Fetch: 0.000 sec

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.

The screenshot shows the MySQL Workbench interface. In the SQL editor tab, the following SQL code is displayed:

```
1 • use outlet_management;
2 • select * from outlet;
3 • select * from stakeholder;
4 • update stakeholder
5     set stakeholder_id = 20
6     where stakeholder_id = 1;
7
```

The screenshot shows the MySQL Workbench interface. In the Output tab, the Action Output section displays the following log entry:

#	Time	Action
1	03:56:08	update stakeholder set stakeholder_id = 20 where stakeholder_id = 1

Message: 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

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

The screenshot shows the MySQL Workbench interface. In the SQL editor tab, the following SQL code is displayed:

```
1 • use outlet_management;
2 • select * from outlet;
3 • select * from stakeholder;
4 • delete from stakeholder where stakeholder_id < 20;
```

The screenshot shows the MySQL Workbench interface. In the Output tab, the Action Output section displays the following log entry:

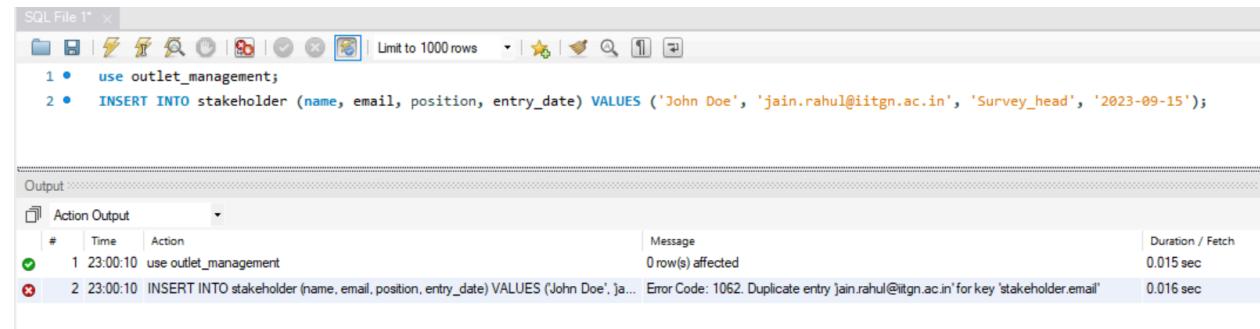
#	Time	Action
1	03:58:11	delete from stakeholder where stakeholder_id < 20

Message: 14 row(s) affected

Responsibility of G1 and G2

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

1st Query



```

SQL File 1* ×
1 • use outlet_management;
2 • INSERT INTO stakeholder (name, email, position, entry_date) VALUES ('John Doe', 'jain.rahal@iitgn.ac.in', 'Survey_head', '2023-09-15');

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', ja... Error Code: 1062. Duplicate entry 'jain.rahal@iitgn.ac.in' for key 'stakeholder.email' 0.016 sec

```

Duplicate entry 'jain.rahal@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:

$$\begin{array}{c}
 \overline{\Pi}_{\text{name}, \text{email}} (\text{stakeholder}) \\
 \cup \\
 \overline{\Pi}_{\text{name}, \text{email}} \left(\sigma_{\text{email} = "jain.rahal@iitgn.ac.in"} (\text{stakeholder}) \right)
 \end{array}$$

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.rahal@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				
#	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.rahu...', 'Error Code: 1062. Duplicate entry 'jain.rahu@itgn.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:

$$\begin{aligned}
 f_1 &\leftarrow \Pi_{\text{Outlet_id}}(\text{Outlet}) \\
 f_2 &\leftarrow \text{values}(\{100, 11, '2023-01-01', '2023-12-31', 'Active'\}) \\
 \text{result} &\leftarrow f_2 - (f_2 \bowtie f_1)
 \end{aligned}$$

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 storing_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/AMUL.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.

The screenshot shows the SQL Server Management Studio interface. In the top pane, there is a query editor window titled "SQL File 1" containing the following SQL code:

```

6 •    SELECT s.name, AVG(o.Ratings) AS avg_ratings
7     FROM stakeholder s
8     JOIN Outlet o ON s.stakeholder_id = o.stakeholder_id
9     GROUP BY s.name;
10

```

The bottom pane displays the results of the query in a "Result Grid". The columns are "name" and "avg_ratings". The data is as follows:

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	4.09999904632568
Yash Kumar	4.699999809265137
Nakul Lal	4.400000095367432
Himanshu Yadav	4.900000095367432

Below the result grid is a "Result 2" pane showing the execution history:

#	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:

$$\begin{aligned}
 \text{join} &\leftarrow \text{stakeholder} \bowtie \text{stakeholder}.stakeholder_id = \text{outlet}.stakeholder_id \text{ Outlet} \\
 \text{avg} &\leftarrow \forall_{\text{stakeholder_name}, \text{avg_ratings}} (\text{join}) \\
 \text{result} &\leftarrow \Pi_{\text{stakeholder_name}, \text{avg_ratings}} (\text{avg})
 \end{aligned}$$

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).

```

SQL File 1* 
11 •  SELECT Outlet_name,
12     CASE
13         WHEN rp.Paid_amount > 0 THEN 'Paid'
14         ELSE 'Not Paid'
15     END AS payment_status
16   FROM Outlet o
17   LEFT JOIN Rent_payment rp ON o.Outlet_ID = rp.Outlet_ID;
  
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result Grid | Form Editor | Read Only

Outlet_name	payment_status
Outlet 1	Paid
Outlet 2	Paid
Outlet 3	Paid
Outlet 4	Paid
Outlet 5	Paid
Outlet 6	Paid
Outlet 7	Paid
Outlet 8	Paid
Outlet 9	Paid

Output

#	Time	Action	Message	Duration / Fetch
5	23:18:10	SELECT s.name, AVG(o.Ratings) AS avg_ratings FROM stakeholders s JOIN Outlet o ON s.Outlet_ID = o.Outlet_ID GROUP BY s.name;	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' ELSE 'Not Paid' END AS payment_status FROM Outlet o LEFT JOIN Rent_payment rp ON o.Outlet_ID = rp.Outlet_ID;	15 row(s) returned	0.015 sec / 0.000 sec

$$\Pi_{\text{Outlet_name}, \text{payment_status}} (\sigma_{(\text{outlet} \times \text{Rent_payment})} \bowtie_{\text{Outlet.Outlet_ID} = \text{Rent_payment.Outlet_ID}} \sigma_{\text{Rent_payment.Paid_amount} > 0})$$

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: | Result Grid | Read Only

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

Output:

#	Time	Action	Message	Duration / Fetch
4	23:14:26	SELECT s.name, AVG(o.Ratings) AS avg_ratings FROM stakeholder s JOIN Outlet o ON s.Outlet_ID = o.Outlet_ID GROUP BY o.Outlet_name;	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 o ON s.Outlet_ID = o.Outlet_ID GROUP BY o.Outlet_name;	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' ELSE 'Unpaid' END AS Paid_Status FROM Outlet o JOIN Employee e ON o.Outlet_ID = e.Outlet_ID;	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 Employee e ON o.Outlet_ID = e.Outlet_ID GROUP BY o.Outlet_name;	5 row(s) returned	0.000 sec / 0.000 sec

$\Pi_{\text{Outlet_name}}, \text{total_employees} (\gamma_{\text{Outlet_name}, \text{COUNT}(*)} \rightarrow \text{total_employees} (\sigma_{\text{Outlet.Outlet_ID} = \text{Employees.outlet_ID}} (\text{Outlet} \times \text{Employees})))$

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: | Result Grid | Form Editor | Revert

ID
1
11
2
12
3
13

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

$\rho_{\text{outlet_id}, \text{ID}} (\text{outlet})$

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