Indian Institute of Technology Gandhinagar

Transport Management System

CS432 Assignment 2
GitHub: https://github.com/PechimuthuMithil/CS432_Assignments.git
16th February, 2024

Moon Macrosystems: Group Members

Ayush Modi, 21110039
Mithil Pechimuthu, 21110129
Vedant Kumbhar, 21110234
Shreesh Agarwal, 21110198
Anushk Bhana, 21110031
Abhishek Mandlik, 21110122

Contents

1. **Responsibility of Group 1**
2. **Responsibility of Group 2**
3. **Responsibility of both Group 1 and Group 2**
4. **Contributions**

1. Responsibility of Group 1

A1.

Ans: The ACID properties, i.e Atomicity, Consistency, Isolation and Durability are features of the database management system, but the tables must adhere to these properties. The tables that we have made are from the converted relational schemas that we had submitted previously in Assignment 1. We have ensured to add constraints such as Not Null, Primary Key and Foreign key to maintain the integrity and keep the data in the table to be coherent and consistent. We have populated dummy data that are of the following form.

1) Names of drivers and some users have been taken from here.
2) Names of Faculty follow the convention First_<alphabet string> Last_<alphabetic string>. For example the name of the faculty can be First_A Last_A, or First_AB Last_AB.
3) Names of the Staff are kept as Staff_A, Staff_b etc..
4) Relevant tables fields like routes, transportation logs etc have data pertaining to the transport scenario of IITGN. For example there are routes from IITGN to Kudasan, Visat Circle, Railway stations and Airport.
5) Additionally we have also included the shops in our DBMS and we have included the shops present in our campus like JK Grocery, Krupa Generals etc…

We have tried to keep the useful data wherever possible. At other places we have filled in dummy data from sources mentioned above and some text generating softwares.

A2.

Indexing allows us to speed up execution of queries by storing a temporary map of the tuples of relations in the memory (RAM). MySQL says, “Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.” Since we can’t store such mapping for all the tables in our relationship in the RAM (as the tables grow larger), we first identify a set of tables that will be frequently accessed during the practical scenario where our DBMS will be used. To optimize the execution of our database, we chose the following set of relations that needed indexing to speed up execution:

1) Users table
2) Transportation Log
3) Booking

Example: created an index named user_index to optimize search on the ‘username’ column in the users table.

```
CREATE INDEX user_index
 on Users (username);
```

Before adding index

```
581
582     -- Before adding index on username in the Users table
583 •   explain analyze
584     SELECT username from Users;
```

Output: Covering index scan on Users using password (cost=4.25 rows=40) (actual time=0.0517..0.0586 rows=40 loops=1)

After adding index

```
588
589     -- After adding index on username in the Users table
590 •   explain analyze
591     SELECT username from Users;
```

Output: Covering index scan on Users using user_index (cost=4.25 rows=40) (actual time=0.0376..0.0421 rows=40 loops=1)

User Defined Data Types (UDDTs):

- User-defined data types are custom data types given a name by the user, which may then be used in data definition language statements later.
- They are useful while dealing with complex data as they can club the different attributes under a single column. Therefore, reducing the complexity.
- As MySQL does not directly support the creation of UDDTs (through CREATE TYPE syntax), we have used JSON dictionaries and enums at some places to implement them.
    - Users Table:
        - data_ JSON: This was added to identify the username and password associated with a particular User as we did not put any foreign key in our Users table.
            - General structure is '{"first_name":"..","last_name":"..","email_id":".."}' in case of students, staff, or faculty

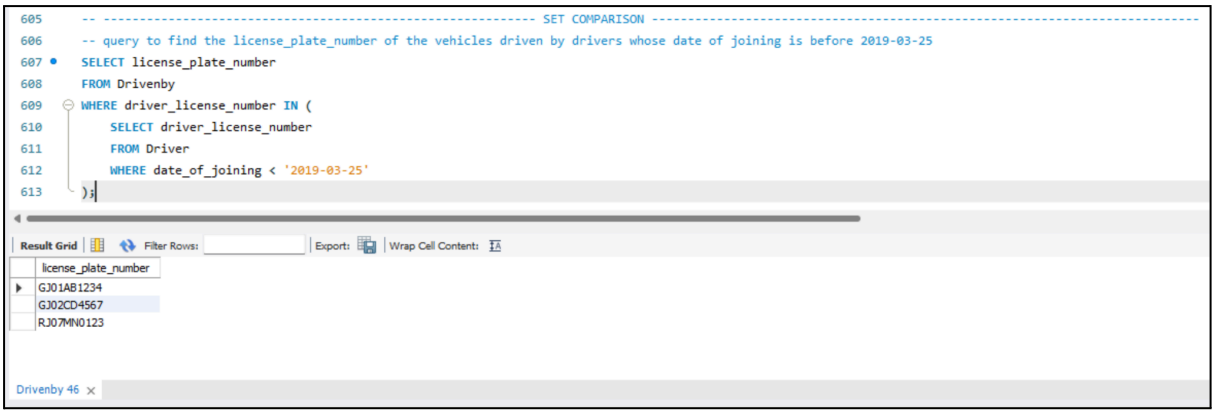
And
'{"first_name":"...","last_name":"...","driver_license_number":"..."}' in case of driver.

- Driver Table:
  - bank_details JSON: This would store the account details of drivers directly in a single column instead of having three separate columns for account_number, ifsc_code, branch_name.
    - General structure is '{"account_number":"..","ifsc_code":"..","branch_name":".."}'

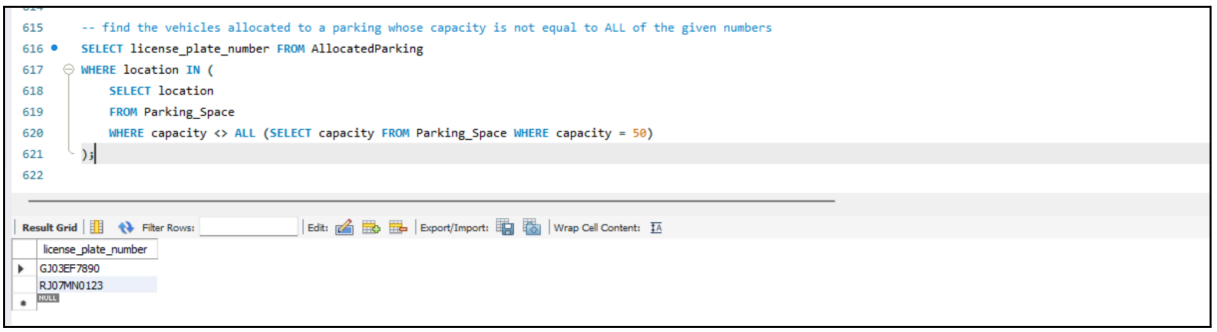
- We have also used ENUMS to represent values for an attribute whose domains aren't atomic but are finite. For example, in the Vehicle table, the type of the vehicle can be any one from ('Bus', 'Car', 'Two-wheeler', 'Auto', 'Van'). Also in the transportation log, we used enums to store the state of a log in the relation, that is, whether it was an entry record, or an exit record. These are however not exactly user defined, but are a very good substitute for UDDTs in MySQL.

SET Comparison:

- Query to find the license_plate_number of the vehicles driven by drivers whose date of joining is before 2019-03-25 (Used IN)



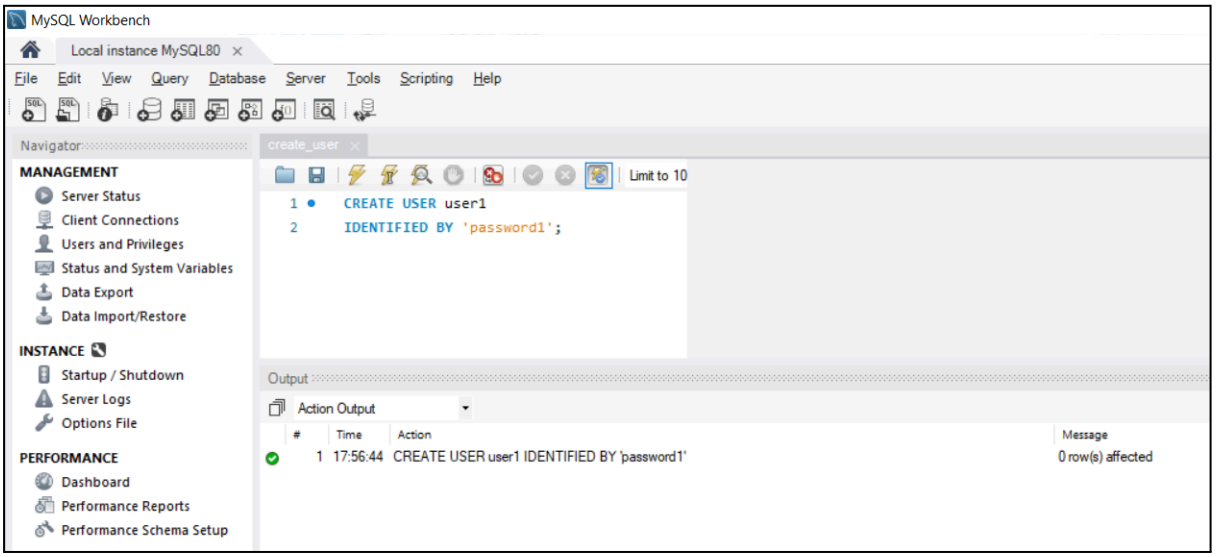
- Find the vehicles allocated to a parking whose capacity is not equal to ALL of the given numbers (Used ALL)



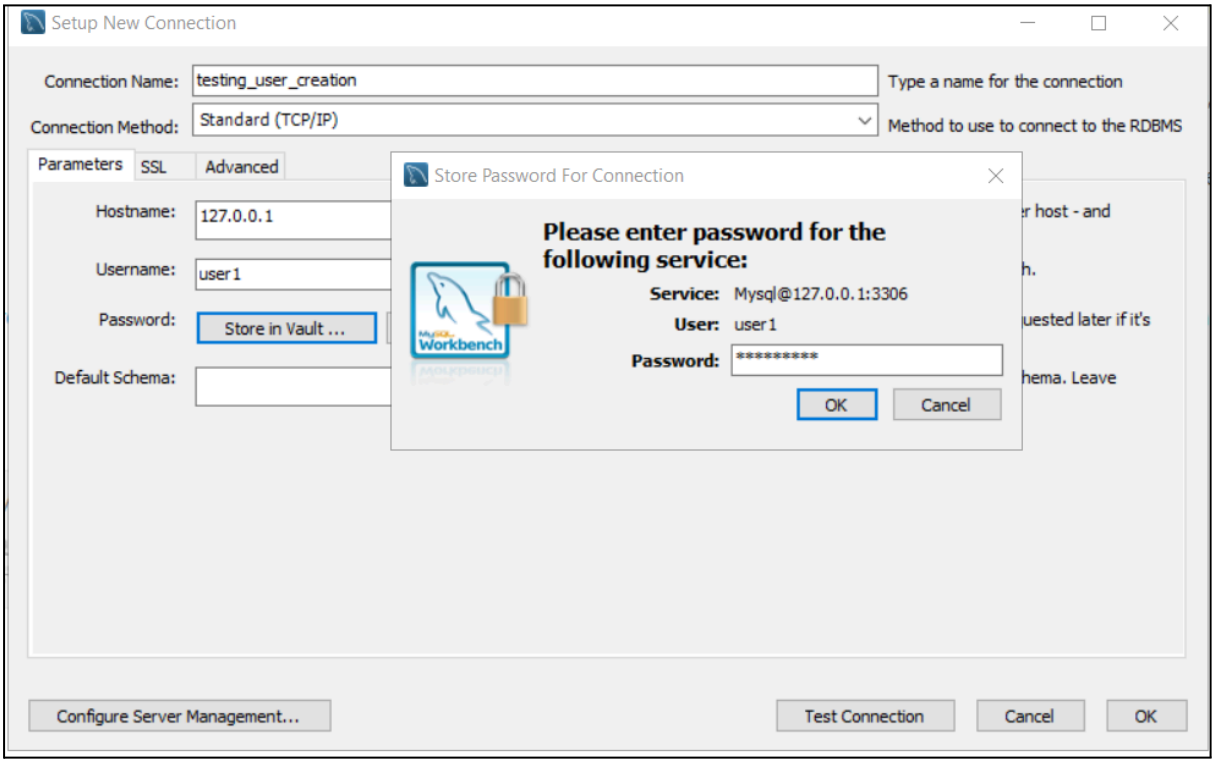
2. Responsibility of Group 2

Question 1:

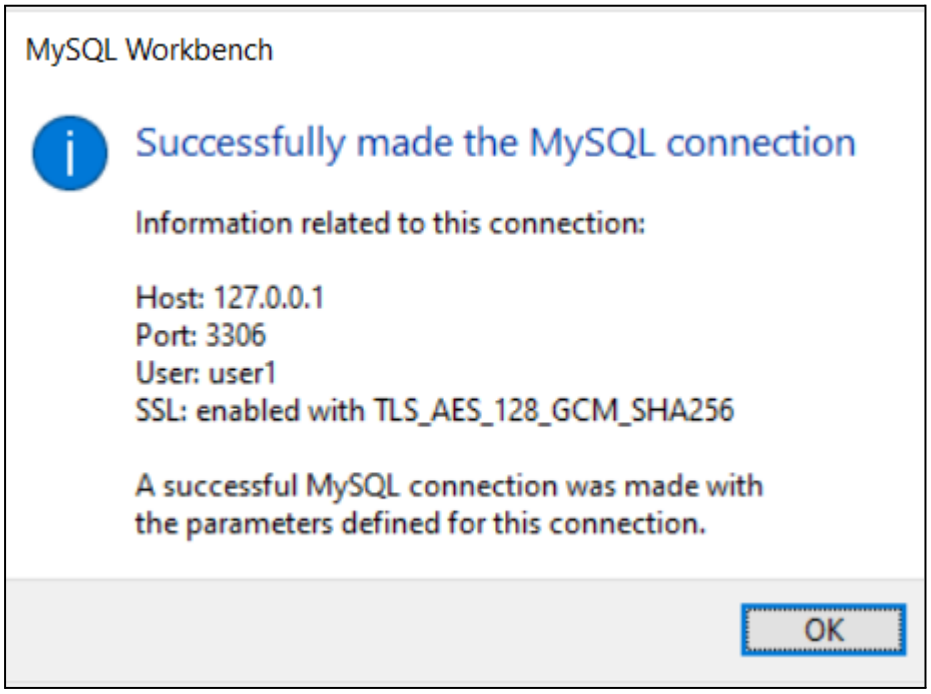
A1.



Creating a user named user1 and setting its password as ‘password1’



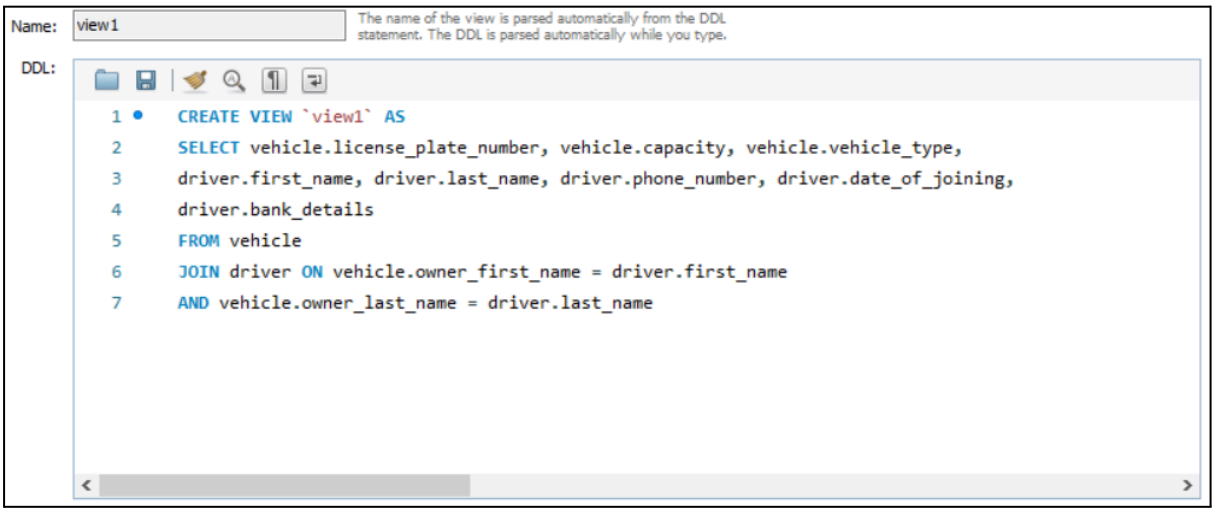
Testing the connection



Connection successful.

A2.

Creating view1:



Name: view1 — The name of the view is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `view1` AS
    SELECT
        `vehicle`.`license_plate_number` AS `license_plate_number`,
        `vehicle`.`capacity` AS `capacity`,
        `vehicle`.`vehicle_type` AS `vehicle_type`,
        `driver`.`first_name` AS `first_name`,
        `driver`.`last_name` AS `last_name`,
        `driver`.`phone_number` AS `phone_number`,
        `driver`.`date_of_joining` AS `date_of_joining`,
        `driver`.`bank_details` AS `bank_details`
    FROM
        (`vehicle`
        JOIN `driver` ON (((`vehicle`.`owner_first_name` = `driver`.`first_name`)
            AND (`vehicle`.`owner_last_name` = `driver`.`last_name`))))
```

This view is created on the tables: vehicle and driver.

It contains the columns:
licence_plate_number, capacity from vehicle table,

and the columns:
first_name, last_name, phone_number, date_of_joining from driver table.

It also contains two user-defined data types: vehicle_type and bank_details
using two different methods for creating user-defined data types, ENUM and JSON.

vehicle_type is a user-defined data type from the vehicle table created using ENUM
bank_details is a user-defined data type from the driver table created using JSON arrays.

Printing view1:

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

license_plate_number	capacity	vehicle_type	first_name	last_name	phone_number	date_of_joining	bank_details
GJ01AB1234	4	Car	Kora	Massey	6543210987	2020-04-10	{"ifsc_code": "4567890125487890", "branch_n...
GJ04GH0123	6	Van	Bryson	Greene	4321098765	2022-06-30	{"ifsc_code": "6789012345487890", "branch_n...
GJ09QR6789	4	Bus	Briana	Kirk	3210987654	2018-02-15	{"ifsc_code": "7890123455487890", "branch_n...
RJ07MN0245	35	Bus	Max	Verstappen	7654321098	2019-03-25	{"ifsc_code": "3456789015487890", "branch_n...

Creating view2:

Name: new_view

The name of the view is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
CREATE VIEW `view2` AS
SELECT driver.first_name, driver.last_name, driver.phone_number, driver.date_of_joining,
driver.bank_details, transportationlog.license_plate_number, transportationlog.starting_station,
transportationlog.ending_station, transportationlog.time_, transportationlog.num_of_passengers,
transportationlog.entry_exit
FROM driver
JOIN transportationlog ON driver.first_name = transportationlog.driver_first_name
AND driver.last_name = transportationlog.driver_last_name
```

Name: view2

The name of the view is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `view2` AS
    SELECT
        `driver`.`first_name` AS `first_name`,
        `driver`.`last_name` AS `last_name`,
        `driver`.`phone_number` AS `phone_number`,
        `driver`.`date_of_joining` AS `date_of_joining`,
        `driver`.`bank_details` AS `bank_details`,
        `transportationlog`.`license_plate_number` AS `license_plate_number`,
        `transportationlog`.`starting_station` AS `starting_station`,
        `transportationlog`.`ending_station` AS `ending_station`,
        `transportationlog`.`time_` AS `time_`,
        `transportationlog`.`num_of_passengers` AS `num_of_passengers`,
        `transportationlog`.`entry_exit` AS `entry_exit`
    FROM
        (`driver`
        JOIN `transportationlog` ON (((`driver`.`first_name` = `transportationlog`.`driver_first_na
            AND (`driver`.`last_name` = `transportationlog`.`driver_last_name`))))
```

This view is created on the tables: driver and transportationlog.

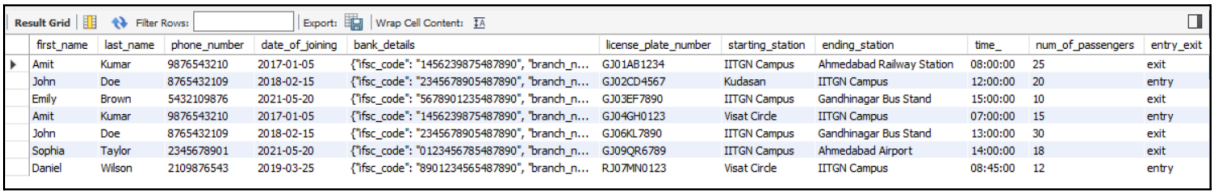
It contains the columns:
first_name, last_name, phone_number, date_of_joining from driver table,

and the columns:
license_plate_number, starting_station, ending_station, time_, num_of_passengers from transportationlog table

It also contains two user-defined data types: entry_exit and bank_details
using two different methods for creating user-defined data types, ENUM and JSON.

entry_exit is a user-defined data type from the transportationlog table created using ENUM
bank_details is a user-defined data type from the driver table created using JSON arrays.

Printing view2:



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

first_name	last_name	phone_number	date_of_joining	bank_details	license_plate_number	starting_station	ending_station	time_	num_of_passengers	entry_exit
Amit	Kumar	9876543210	2017-01-05	{"ifsc_code": "1456239875487890", "branch_n...	GJ01AB1234	IITGN Campus	Ahmedabad Railway Station	08:00:00	25	exit
John	Doe	8765432109	2018-02-15	{"ifsc_code": "2345678905487890", "branch_n...	GJ02CD4567	Kudasan	IITGN Campus	12:00:00	20	entry
Emily	Brown	5432109876	2021-05-20	{"ifsc_code": "5678901235487890", "branch_n...	GJ03EF7890	IITGN Campus	Gandhinagar Bus Stand	15:00:00	10	exit
Amit	Kumar	9876543210	2017-01-05	{"ifsc_code": "1456239875487890", "branch_n...	GJ04GH0123	Visat Circle	IITGN Campus	07:00:00	15	entry
John	Doe	8765432109	2018-02-15	{"ifsc_code": "2345678905487890", "branch_n...	GJ06KL7890	IITGN Campus	Gandhinagar Bus Stand	13:00:00	30	exit
Sophia	Taylor	2345678901	2021-05-20	{"ifsc_code": "0123456785487890", "branch_n...	GJ09QR6789	IITGN Campus	Ahmedabad Airport	14:00:00	18	exit
Daniel	Wilson	2109876543	2019-03-25	{"ifsc_code": "8901234565487890", "branch_n...	RJ07MN0123	Visat Circle	IITGN Campus	08:45:00	12	entry

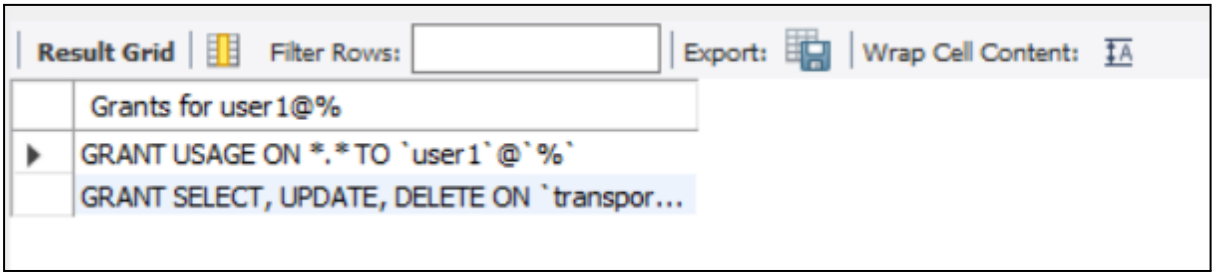
A3:

To grant user1 the SELECT, UPDATE, and DELETE permissions on the table driver:

```
1 •  USE transportmanagement;
2
3 •  GRANT SELECT, UPDATE, DELETE ON driver TO user1;
4
5 •  SHOW GRANTS FOR user1;
```

192	23:16:29	USE transportmanagement	0 row(s) affected		0.000 sec
193	23:16:29	GRANT SELECT, UPDATE, DELETE ON driver TO user1	0 row(s) affected		0.047 sec
194	23:16:29	SHOW GRANTS FOR user1	2 row(s) returned		0.000 sec / 0.000 sec

Output displaying the permissions granted to user1:



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

Grants for user1@%
GRANT USAGE ON *.* TO `user1`@`%`
GRANT SELECT, UPDATE, DELETE ON `transpor...

We can see that user1 has been granted the SELECT, UPDATE, and DELETE permissions on the driver table.

A4:

To grant user1 the SELECT permission on view1:

```
1 •  USE transportmanagement;
2
3 •  GRANT SELECT ON view1 TO user1;
4
5 •  SHOW GRANTS FOR user1;
```

```
195  23:24:42  USE transportmanagement                    0 row(s) affected    0.000 sec
196  23:24:42  GRANT SELECT ON view1 TO user1            0 row(s) affected    0.016 sec
197  23:24:42  SHOW GRANTS FOR user1                     3 row(s) returned    0.000 sec / 0.000 sec
```

Output displaying the permissions granted to user1:

Result Grid | Filter Rows: | Export:

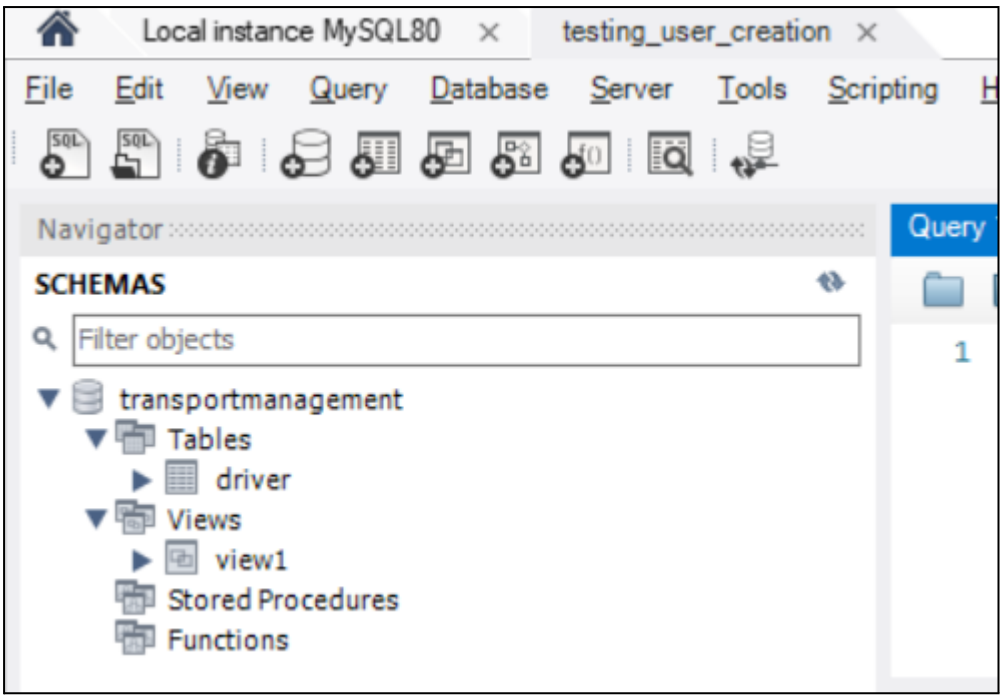
Grants for user1@%
GRANT USAGE ON *.* TO `user1`@`%`
GRANT SELECT, UPDATE, DELETE ON `transpor...
GRANT SELECT ON `transportmanagement`.`v...

We can see that user1 has now also been granted the SELECT permission on view1.

A5:

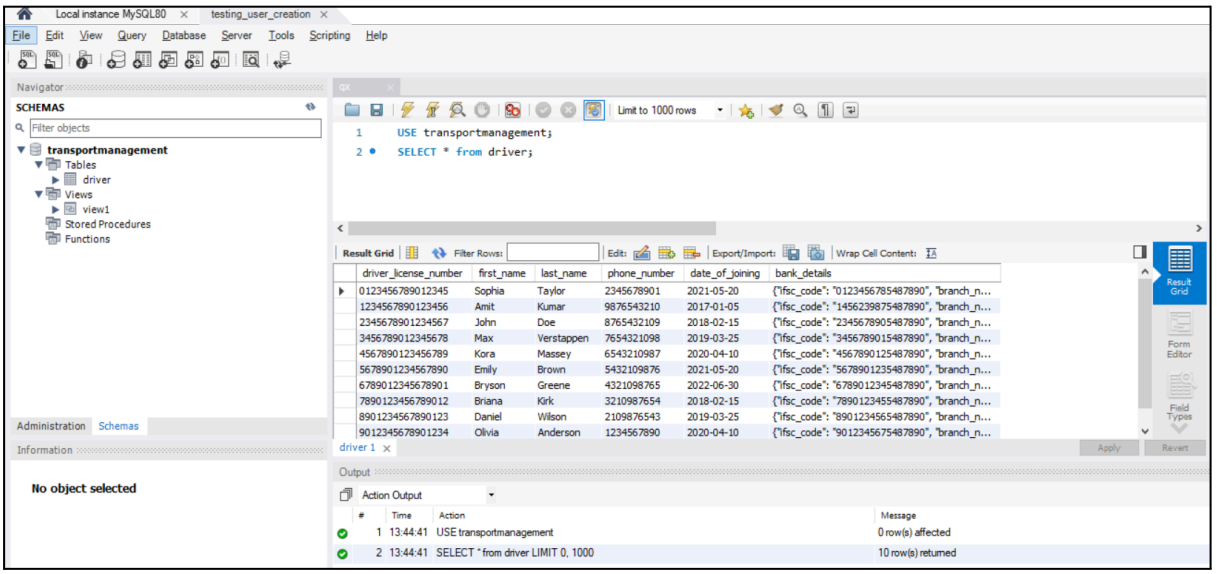
For this part, we go back to that connection, where we created the user1 and thus we will be able to see what all privileges are available for user1.

FOR TABLE1:



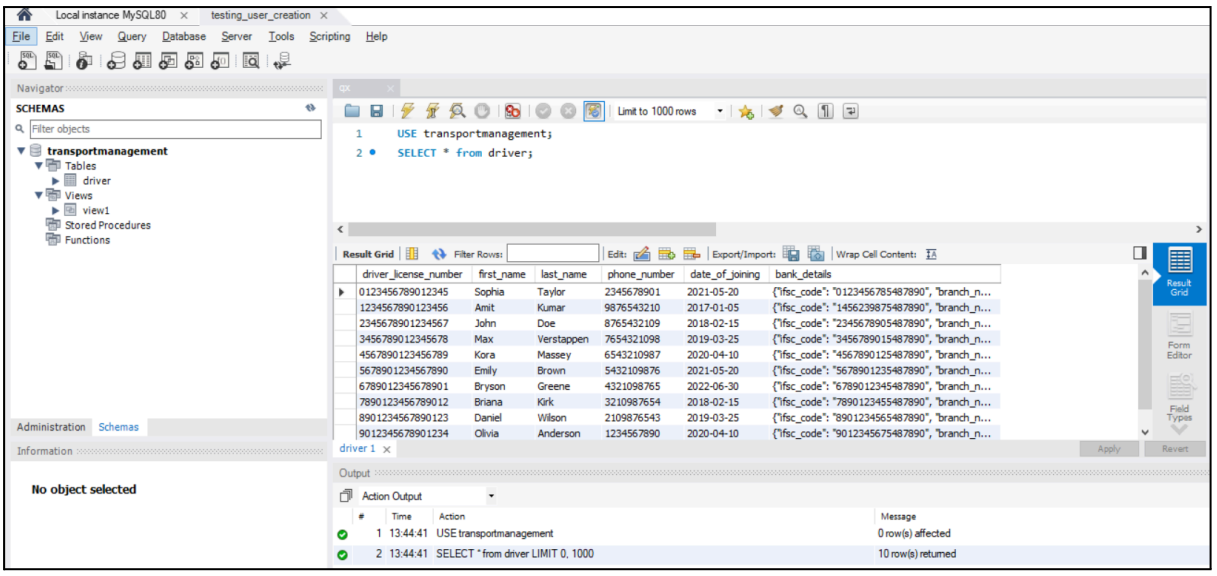
We can see here that user1 only has access to the driver table, and the view1. Rest all of the tables and views are not accessible by user1.

→ Performing <u>SELECT</u> operation on driver as user1:

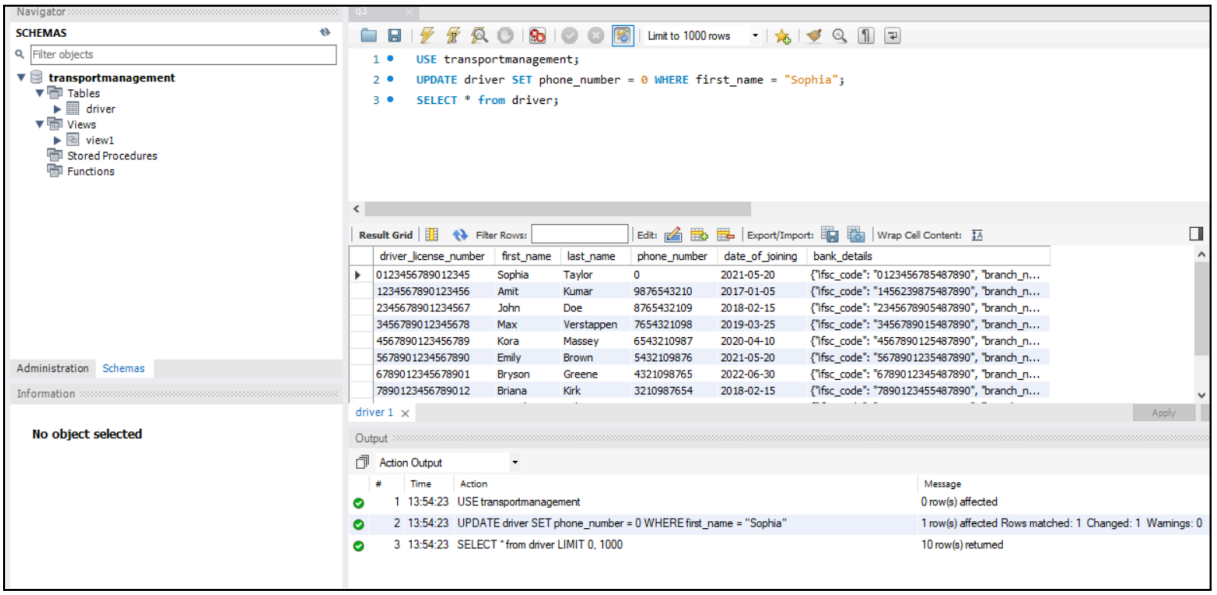


→ Performing <u>UPDATE</u> operation on driver as user1:

Before performing UPDATE, the table looks like:

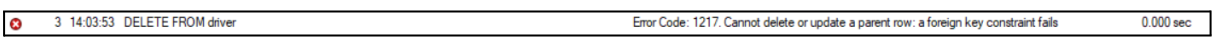


After performing UPDATE, the table looks like:



We can see that the phone_number of the driver having first_name = “Sophia” has been successfully updated to 0.

→ Performing <u>DELETE</u> operation as user1:



Now, in the driver table, the DELETE operation leads to an error stating the foreign key constraint is failing.

Thus to successfully show the working of the DELETE operation, I granted user1 the permissions to access the “users” table:

Before performing DELETE, the “users” table looks like:

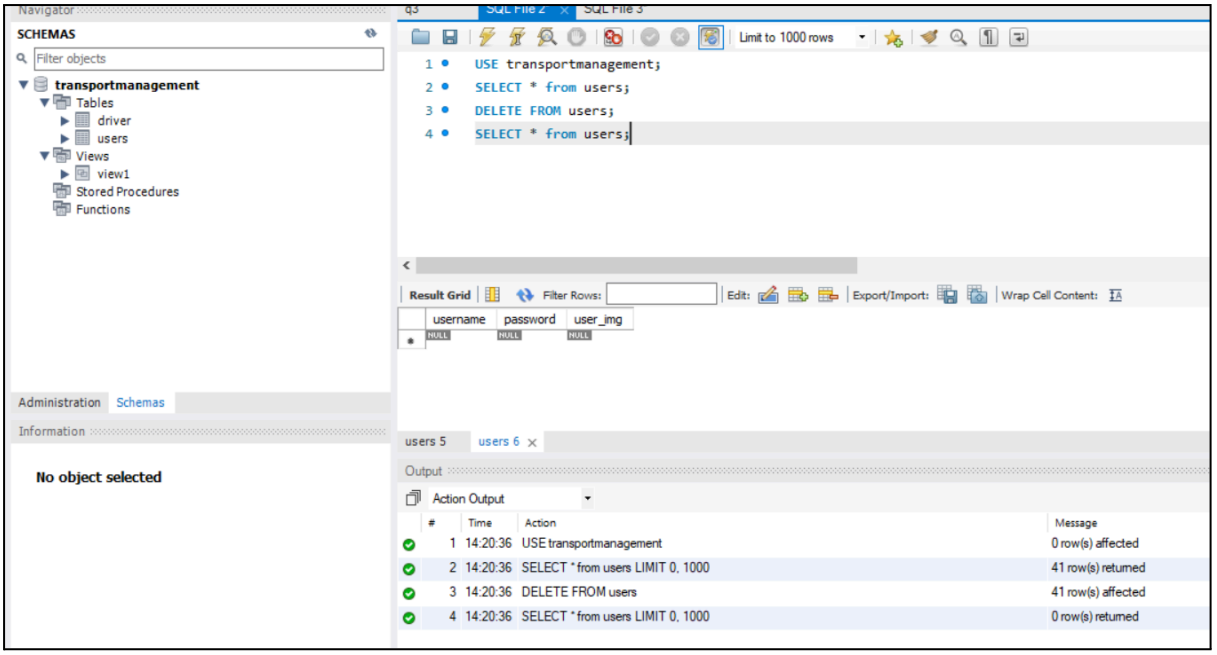
```
1 •  USE transportmanagement;
2    SELECT * from users;
```

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

username	password	user_img
Alice	f5f75389bdbeddbbd70ac31c0245e42e	NULL
Amit	f039eb446cc0bd7c5ad12b7a0e2a1dae	NULL
Daniel	e7f84ff145ced31368756c41af7fef9c	NULL
David	889211b122daa7f9f917d3d3b3475514	NULL
Emily	f6244d2fc24ec05bedfbf7a84a42ddd4	NULL
First_A	367ec700665670e8b07b6d10292ae521	NULL
First_B	01e54c813479b1ee86c20875258307f8	NULL
First_C	c8091c1f0ba01ee8f40bff623e97cf23	NULL
First_D	3af97df0f613f6cf48e680e21a301710	NULL
First_E	545269de19e5b868282731db913b3983	NULL
First_F	25eb68f5bcc2a69681938078e949fe67	NULL
First_G	0b97c298a0294bb79f699d214c6ac60f	NULL
First_H	6a7872e18226d4e68ecada00e8016504	NULL
First_I	12f55726019b4032d8f30333d3d2905b	NULL
First_J	8cf527de4f1df1d846c21306c4940505	NULL
Jessica	3bbd1873056226d83e6371a24d340340	NULL
John	6d40e095b7f43848dc76ec017592da29	NULL
Michael	fdec10f5e0fc93e0dadc8c53b9802623	NULL
Olivia	6fe76ed19c9386c1389512a53aaaa6c6	NULL

Note: Here the user_img attribute has NULL values as we hadn’t yet inserted images into the table at time of performing this activity, and the Caption attribute we added later.

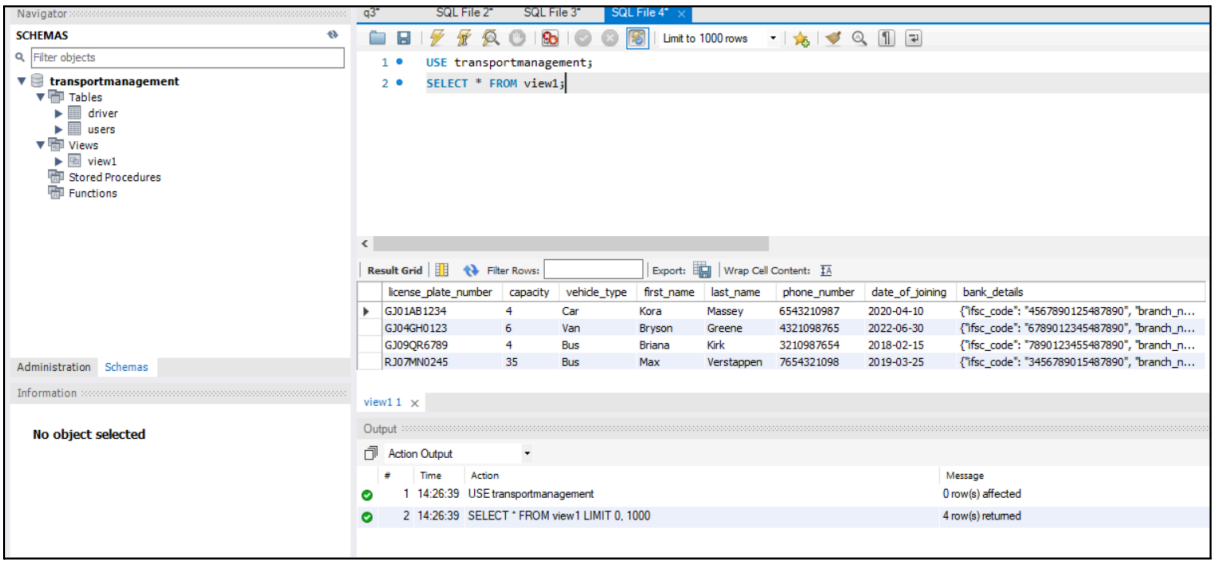
After performing DELETE, the “users” table looks like:



As we can see, the DELETE operation worked successfully.

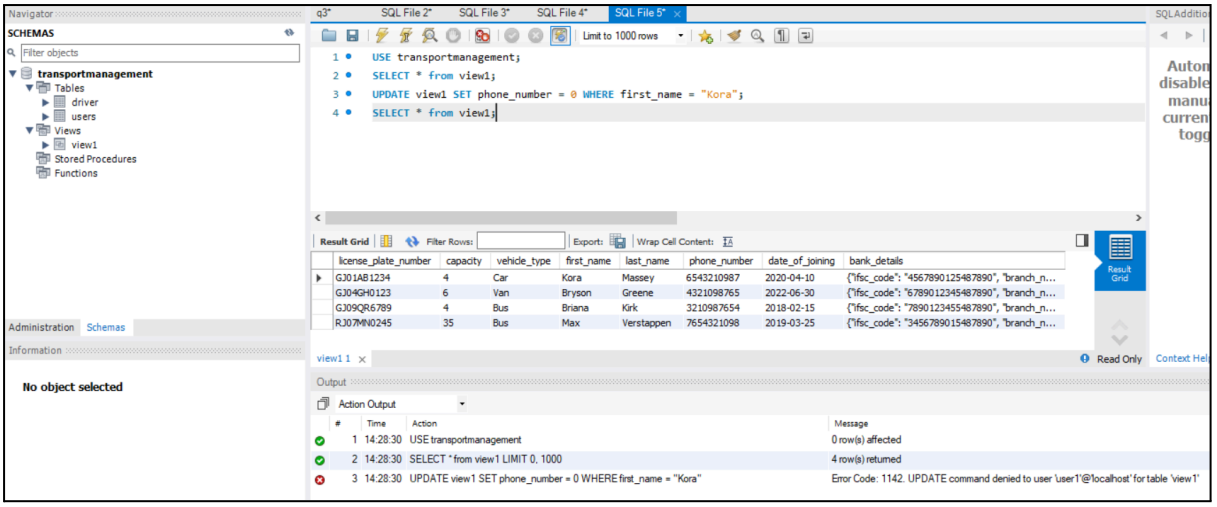
FOR VIEW1:

→ Performing SELECT operation on view1 as user1:



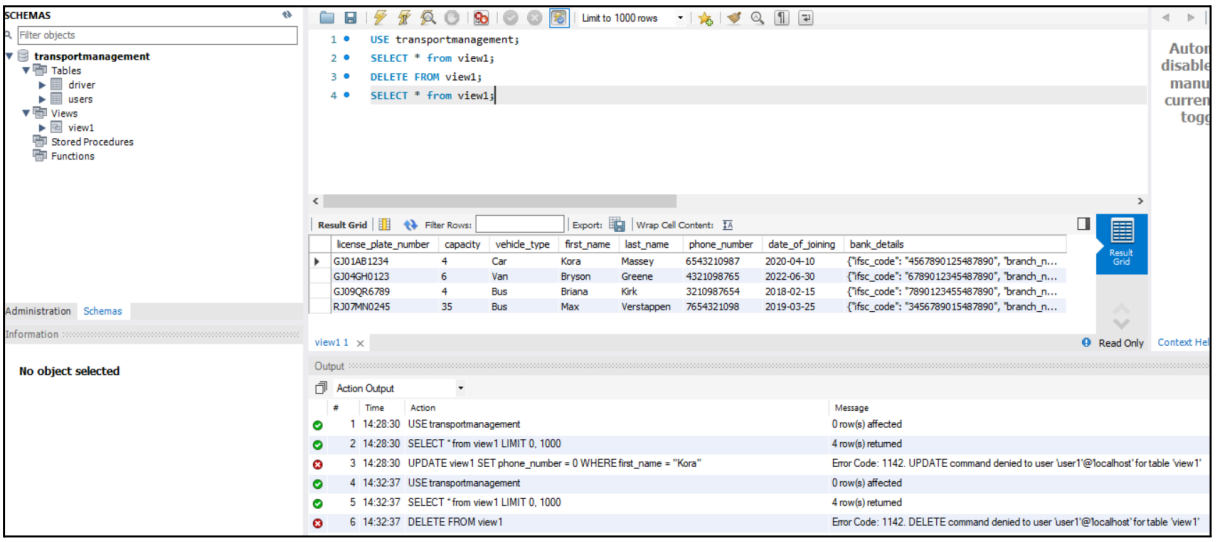
Thus, user1 is able to perform the SELECT operation on view1.

→ Performing UPDATE operation on view1 as user1:



Here, we can see that we get an error saying UPDATE command is denied to user1 on view1 while performing the UPDATE operation on view1 as user1. And this should indeed be the case, as we have not given the UPDATE permission to user1 on view1. Thus, the permissions we have given are working appropriately.

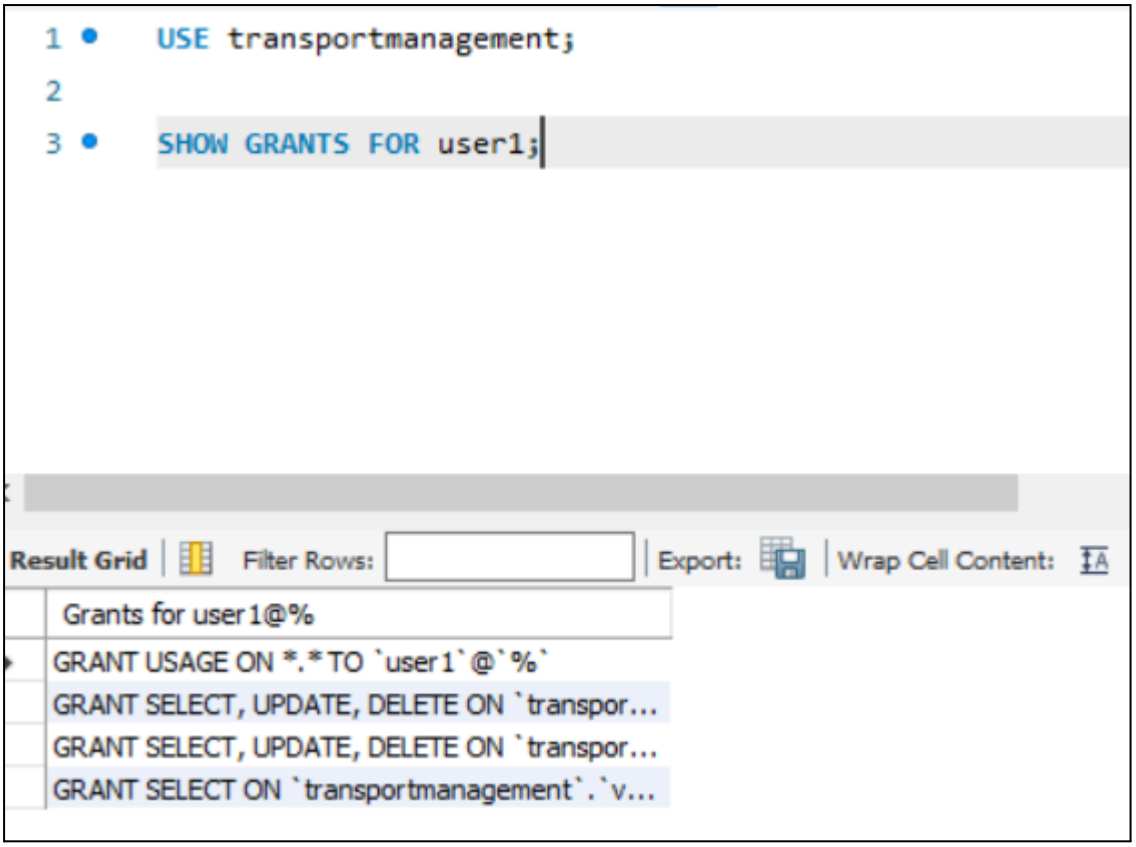
→ Performing <u>DELETE</u> operation on view1 as user1:



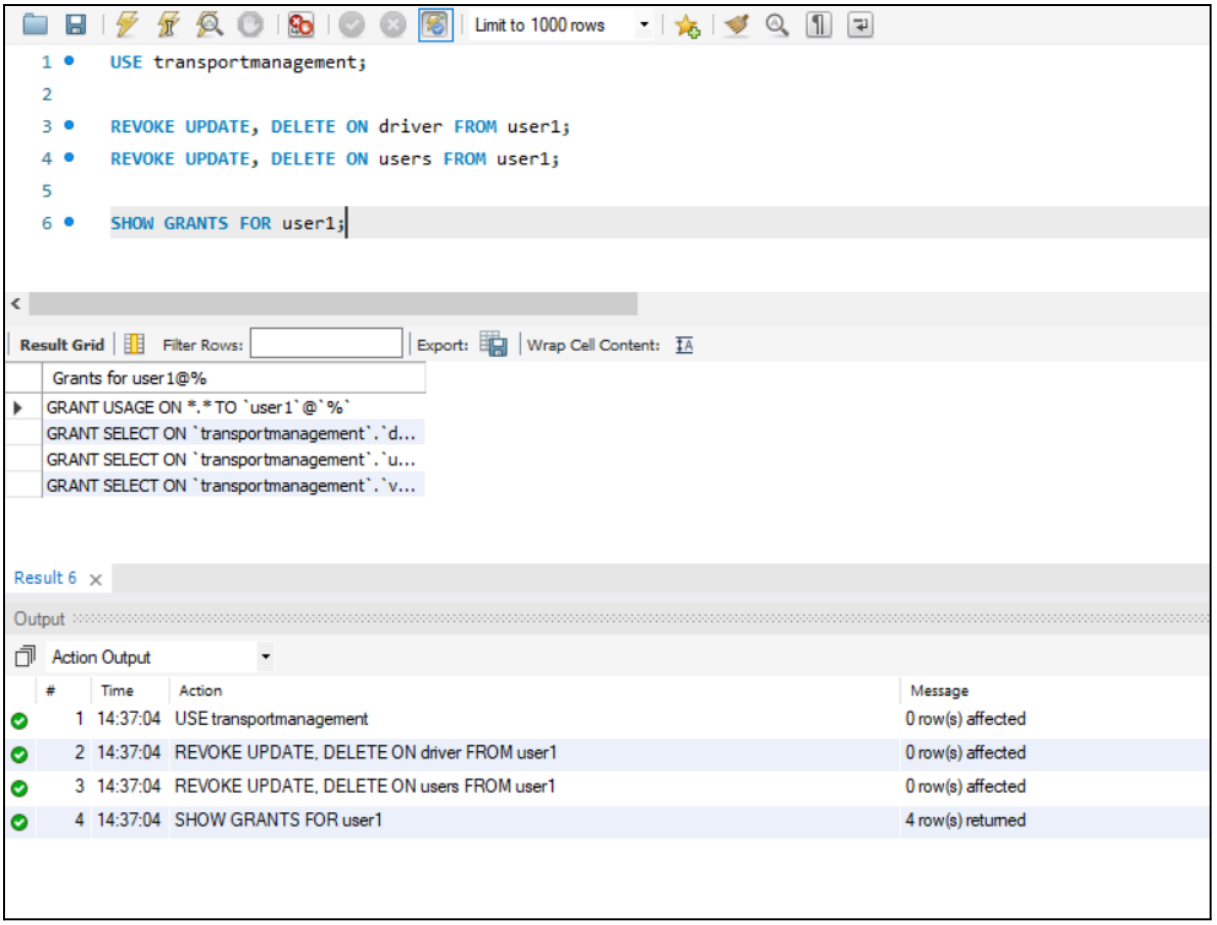
Here as well, as expected, we get an error saying DELETE command is denied to user1 on view1. As we have not given the DELETE permission to user1 on view1. Thus, the permissions we have given are working appropriately.

A6:

Before revoking the UPDATE and DELETE permissions on the two tables “driver” and “users” for user1, the permissions it has are:



After revoking the UPDATE and DELETE permissions on the two tables driver and users for user1:

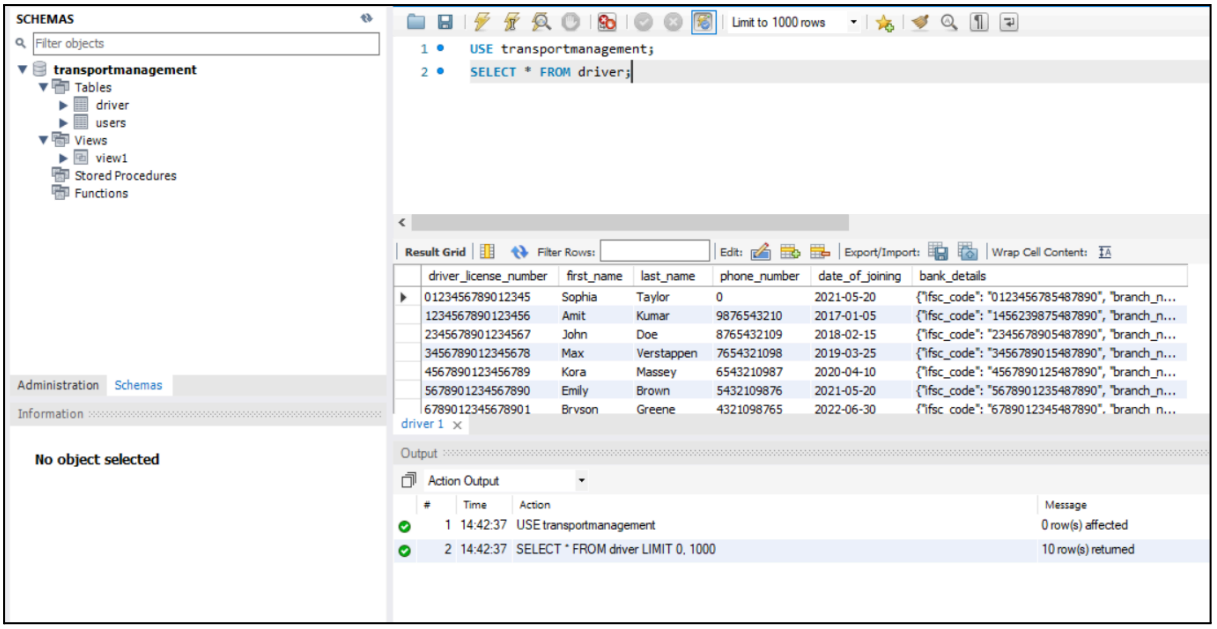


As we can see now user1 only has the SELECT permissions on the tables driver, users, and on the view view1. The UPDATE and DELETE permissions on the two tables driver and users for user1 have been successfully revoked.

A7:

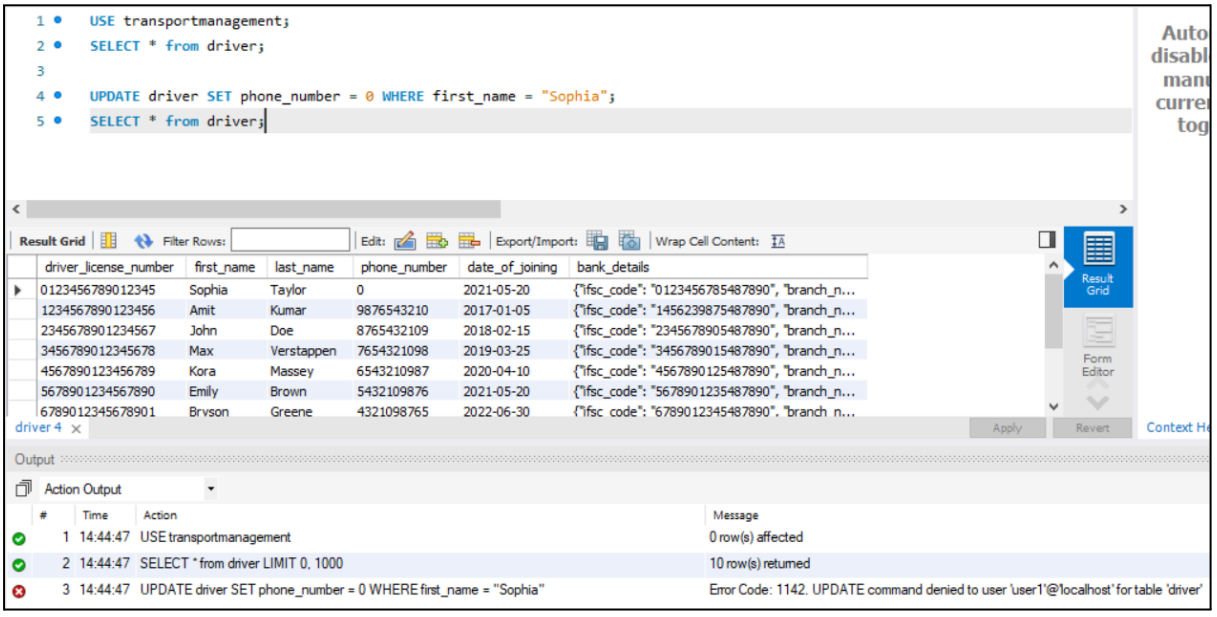
FOR TABLE1:

→ Performing SELECT operation on driver as user1:



As we can see, the SELECT operation works, as we have not revoked the SELECT permissions from user1.

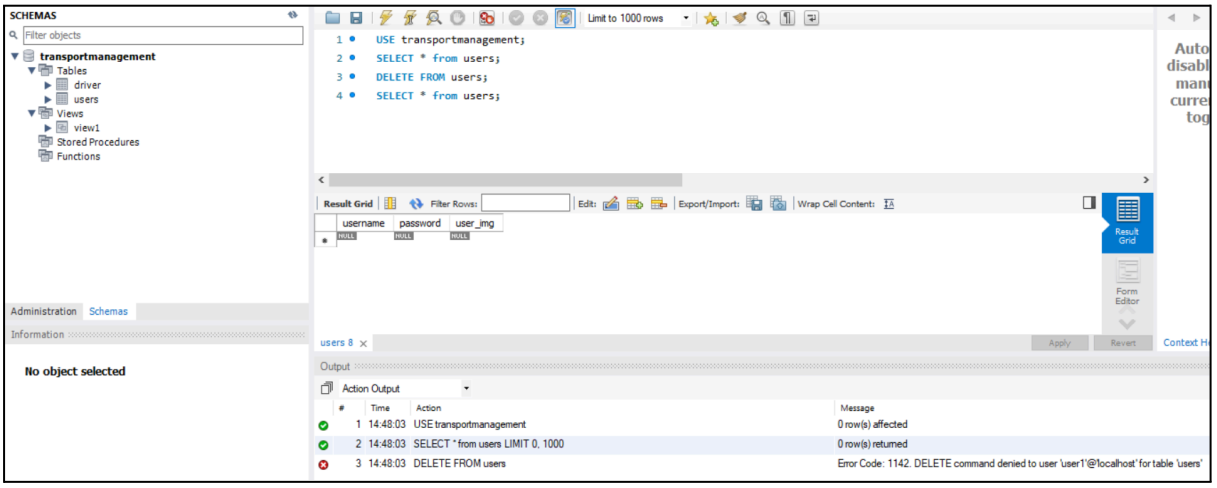
→ Performing <u>UPDATE</u> operation on driver as user1:



Now here we can see that user1 is not able to perform the UPDATE operation on the driver table, as we have revoked its UPDATE privileges from the driver table.

→ Performing <u>DELETE</u> operation on “users” table as user1:

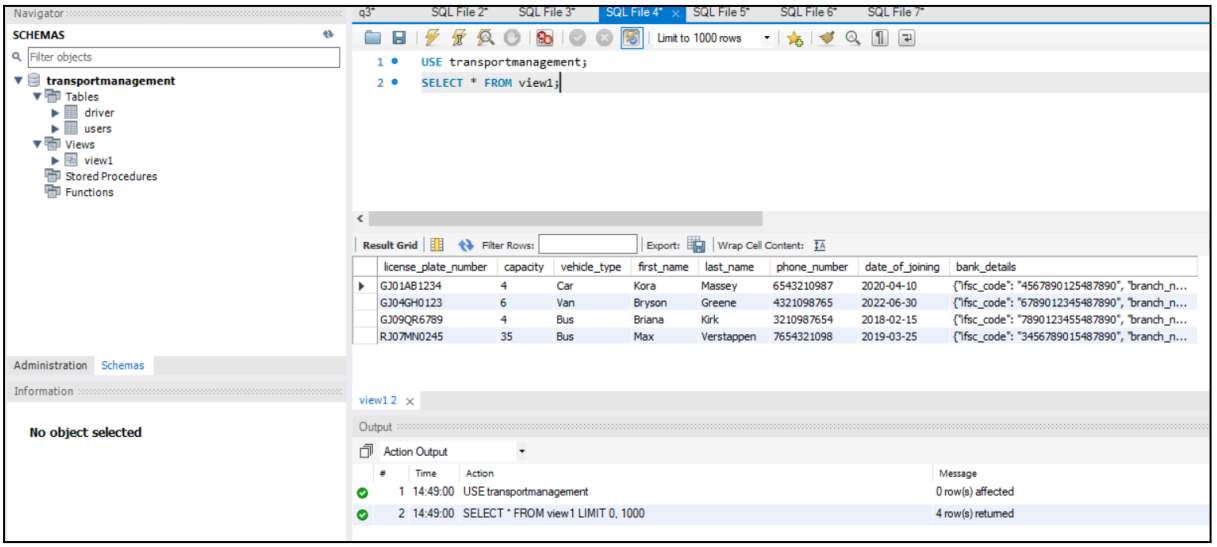
As stated above as well, as the DELETE operation on the driver table leads to the foreign key constraint failing error, I have given user1 access to the "users" table, and am performing the DELETE operation on the "users" table:



Here as well, as we expected, user1 cannot perform the DELETE operation on the "users" table, as we have revoked its permission to do so.

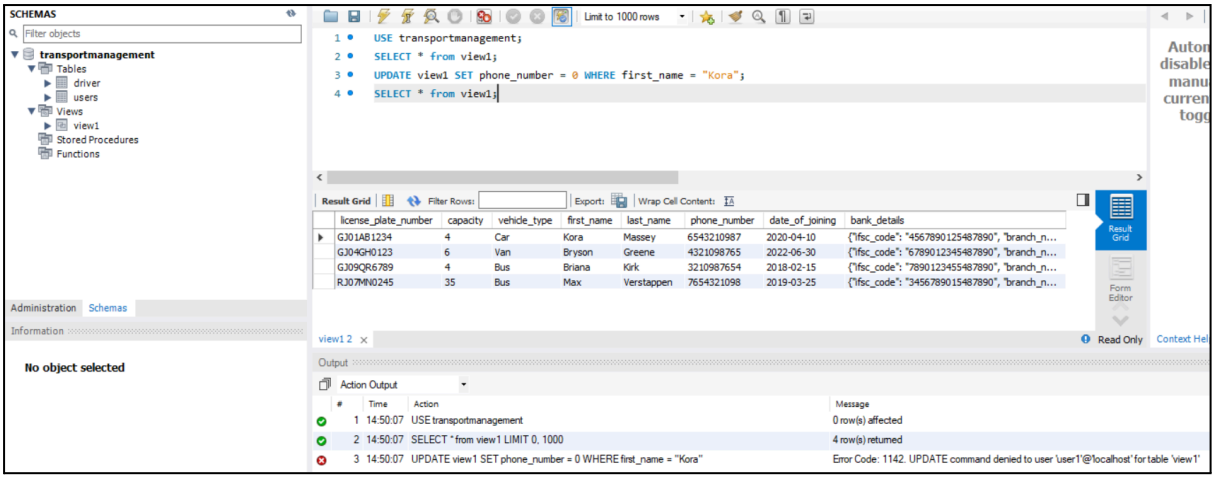
FOR VIEW1:

→ Performing SELECT operation on view1 as user1:



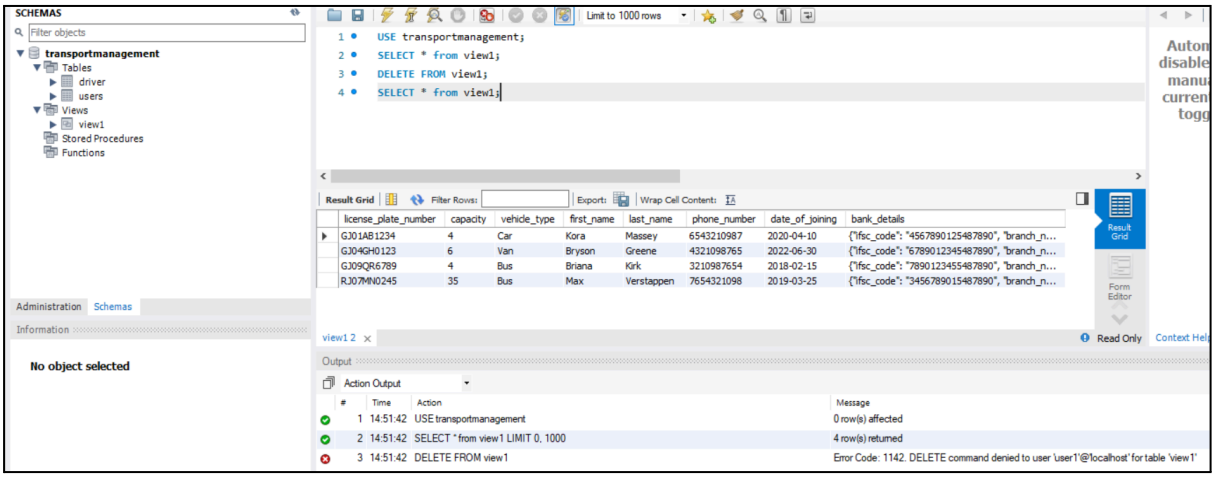
user1 is able to successfully perform the SELECT operation on view1, which is as we would expect, as we have not revoked the SELECT permission on view1 from user1.

→ Performing UPDATE operation on view1 as user1:



Here, user1 is obviously not able to perform the UPDATE operation on view1, as it had not been given the permission to do so.

→ Performing <u>DELETE</u> operation on view1 as user1:



Here as well, as expected, user1 is not able to perform the DELETE operation on view1, as it had not been given the permission to do so.

Question 2:

Answer:

CAUSES:

Referential integrity violations in a database can occur in a variety of scenarios:

1. Inserting invalid foreign keys :

Inserting records can violate referential integrity if the inserted values do not exist in the referenced tables. This occurs when you attempt to insert a value into a column that should relate to the primary key of another table, but the value does not exist in that table.

Examples in our database:

i) Attempting to add a log in TransportLog for a vehicle with a license plate number that is not present in the vehicle table.
ii) If we try to insert a record into the DrivenBy table with a driver_license_number or license_plate_number that does not exist in the Driver or Vehicle table, respectively, it would violate referential integrity.

2. Incorrectly updating, modifying or deleting referred rows :

If we update, modify or remove a row that is referenced by other rows in different tables, the relationships between them may be broken.

Examples in our database:

i) Deleting an entry in the TransportationLog table without changing the vehicle records results in an inconsistency.
ii) If we delete a record from the Driver table that is referenced by records in the DrivenBy table, it would create an orphaned record in the DrivenBy table, causing a referential integrity violation.

3. Data import or migration :

If data is imported or migrated from external sources and does not match the current relationships defined in the database schema, referential integrity issues may occur.

Various other miscellaneous examples of referential integrity errors in our database:

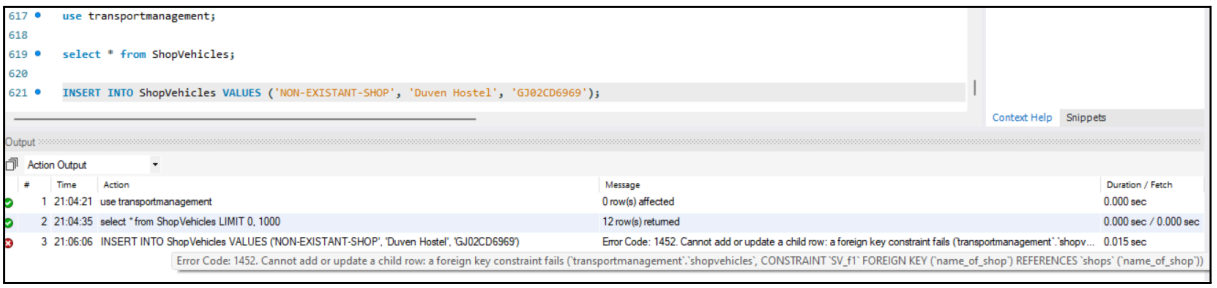
1. Inserting a record in the Goods table with name_of_shop or location that doesn't exist in the Shops table.

2. Inserting a record in the Insurance table with a license_plate_number that doesn't exist in the Vehicle table.
3. Inserting a record in the TransportationLog table with license_plate_number, starting_station, or ending_station that doesn't exist in the Vehicle or Route table, respectively.
4. Inserting a record in the Booking table with license_plate_number or email_id that doesn't exist in the Vehicle or Students table, respectively.
5. Inserting a record in the ShopVehicles table with name_of_shop, location, or license_plate_number that doesn't exist in the Shops or Vehicle table, respectively.
6. Inserting a record in the PrivateOwnership table with a license_plate_number that doesn't exist in the Vehicle table.
7. Inserting a record in the AllocatedParking table with liscence_plate_number that doesn't exist in the Vehicle table.
8. Inserting a record in the GoodsTransported table with license_plate_number that doesn't exist in the Vehicle table.
9. Inserting a record in the VehicleInsurance table with license_plate_number or insurance_id that doesn't exist in the Vehicle or Insurance table, respectively.
10. Deleting a Vehicle: Similarly, if you delete a record from the Vehicle table, and that vehicle has related records in the DrivenBy table (i.e., records where the license_plate_number matches the deleted vehicle's license plate number), it would also result in referential integrity violations.
11. Deleting a Route: Similarly, deleting a record from the Route table, if it is referenced by records in the Trip table, could lead to referential integrity issues.

Thus, to summarize, referential integrity errors will occur whenever we perform insert, delete, or update operations involving any table that has a foreign key referring to another table, or that table is the one whose primary key is the foreign key of another table. All of these scenarios are related to the foreign key constraints being violated, and thus to solve such problems we will need to focus on adding stricter referential integrity constraints on foreign keys in our database.

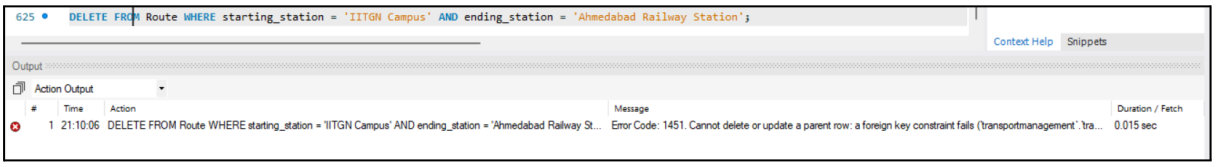
Now in the following part, we show some examples wherein we explain referential integrity violations through code and the respective errors thrown for them -

1. Inserting a Shop vehicle with to a Non-Existing Shop:



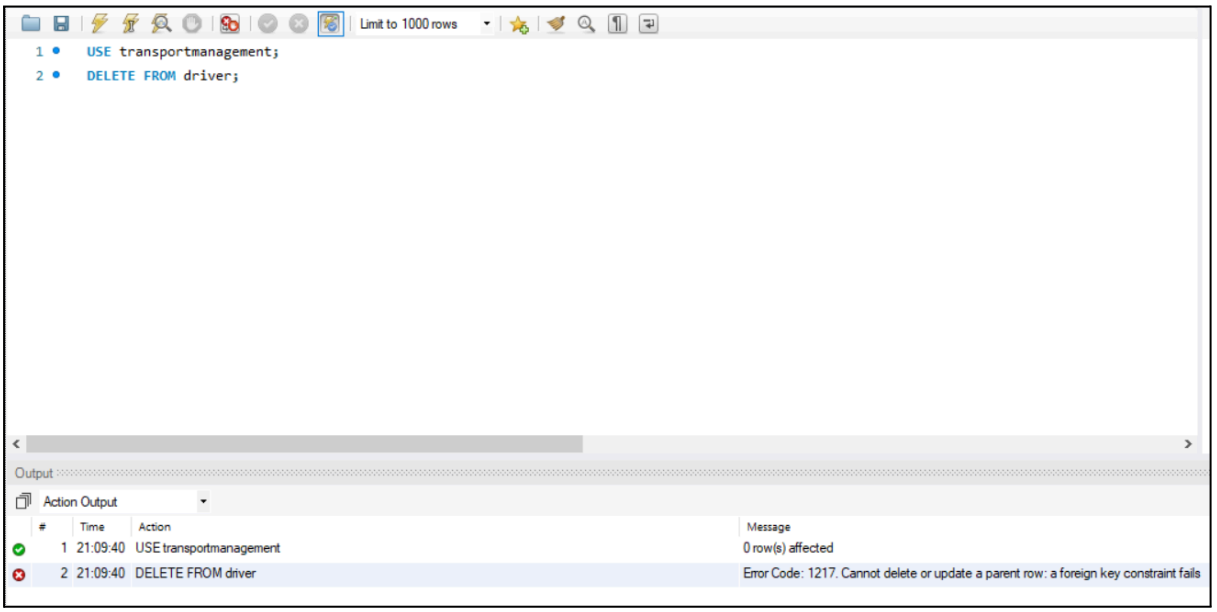
In the above screenshot, the statement attempts to insert a Shop vehicle to a shop (‘NON-EXISTENT-SHOP’) that does not exist.

2. Deleting a record from the Route table which is referenced by the Transportation Log table:



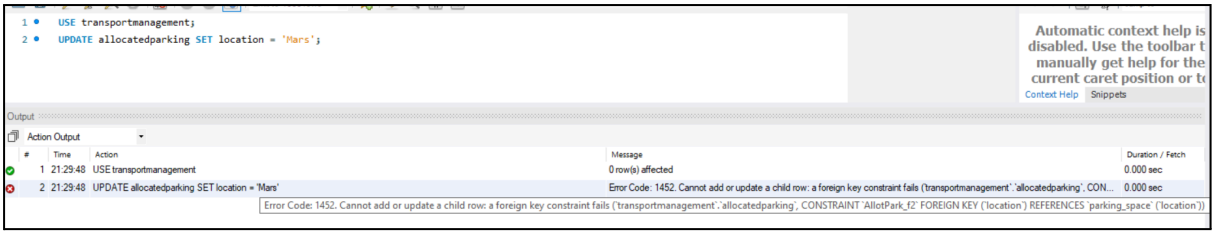
In the above screenshot, this statement deletes records from the ‘Route’ table (referenced by the Transportation Log table) where the starting station is ‘IITGN Campus’ and the ending station is ‘Ahmedabad Railway Station’.

3. Deleting the entire driver table which is associated to a lot of tables like the vehicles table:



In this case, we can see that we are trying to delete the entire driver table, but are not able to do so, as the driver table is associated with many other tables like vehicles, etc through foreign keys and thus deleting this table will lead to foreign key constraint violations.

4. Updating the allocatedparking table and setting the location attribute of all the records to ‘Mars’:



Here as well, we can see in the error that updating the location attribute in the allocatedparking table leads to referential integrity violation as location is a foreign key that references the primary key of the parking_space table, and thus updating it will lead to inconsistencies.

SOLUTIONS:

We can identify and eliminate referential integrity violations in the following way:

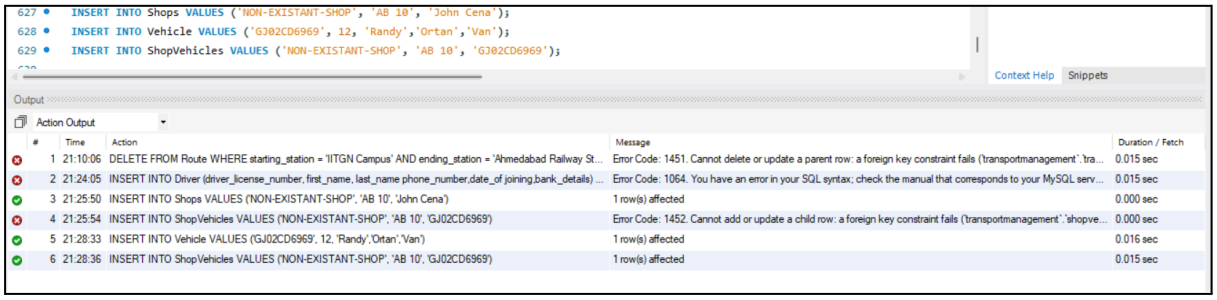
By adding foreign key constraints in the database schema. These restrictions will implement rules that prohibit the introduction of invalid references in our database. When an effort is made to break these restrictions, the database system will generate an error. Thus whilst sometimes being unable to do some desired but invalid operation, it will keep the database safe from referential integrity failures.

This can be done by adding ON DELETE CASCADE and ON UPDATE CASCADE to foreign key constraints. This enables the database to automatically update or remove associated rows whenever a referred row is modified or destroyed, ensuring referential integrity.

The following text summarizes the major solutions through code:

1. **Inserting**

If we want to insert a new Shop vehicle in the shop vehicles table we will have to manually update the shops and vehicles table so as to abide by referential integrity.



2. **Deleting**

ON DELETE CASCADE: If we want to ensure that when a driver is deleted, all records of that driver being associated with a car are also deleted, you can set up a foreign key constraint with ON DELETE CASCADE. This means that when a driver is deleted, all associated records in the "driven by" table will also be automatically deleted.

3. Updating

ON UPDATE CASCADE: When a referenced column in the parent table is updated, the corresponding columns in the child tables are also updated automatically to reflect the change.

Thus in this way, we can solve the referential integrity violations in our database, ensuring that our data is consistent and reliable.

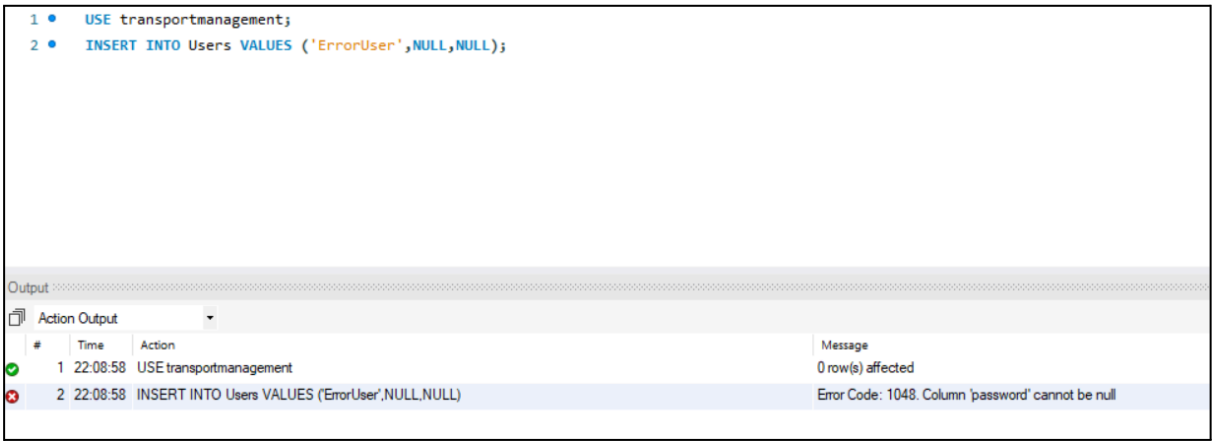
3. Responsibility of both Group 1 and Group 2

1.

a) Operation 1:

Query:

INSERT INTO Users VALUES ('ErrorUser',NULL,NULL);
Error Code: 1048. Column 'password' cannot be null.



The function of this operation is to insert the values (‘ErrorUser’, NULL, NULL) into the “users” table. But it generates an error as the password attribute has been set to NULL, which is not allowed.

INSERT operations do not have relational algebra representations.

Operation 1 satisfies specification 1.

b) Operation 2:

Query:

INSERT INTO Staff (email_id, first_name, last_name, phone_number) VALUES ('staffJ@iitgn.ac.in', 'First_JZ','Last_JZ','9998885554');
Error Code: 1062. Duplicate entry 'staffJ@iitgn.ac.in' for key 'staff.PRIMARY'

```
1 •  USE transportmanagement;
2
3 •  INSERT INTO Staff (email_id, first_name, last_name, phone_number) VALUES ('staffJ@iitgn.ac.in', 'First_JZ','Last_JZ','9998885554');
```

Output

Action Output

#	Time	Action	Message
1	22:09:48	USE transportmanagement	0 row(s) affected
2	22:09:48	INSERT INTO Staff (email_id, first_name, last_name, phone_number) VALUES ('staffJ@iitgn.ac.in', 'First_JZ','Last...	Error Code: 1062. Duplicate entry 'staffJ@iitgn.ac.in' for key 'staff.PRIMARY'

The function of this operation is to insert the values (‘staff@iitgn.ac.in’, ‘First_JZ’, ‘Last_JZ’, ‘999885554’) into the columns (email_id, first_name, last_name, phone_number) of the “users” table. But it generates an error as it is creating a duplicate entry of ‘staff@iitgn.ac.in’ which is the value for the primary key email_id of the table, which is not allowed.

INSERT operations do not have relational algebra representations.

Operation 2 satisfies specification 1.

2.

Operation 3:

Table containing the BLOB type to store images.

```
82      ---------------------------------- User Table ---------------------------------------------
83
84 •    CREATE table if not exists Users
85      (
86        username  varchar(20)  NOT NULL,
87        password  varchar(50)  NOT NULL UNIQUE,
88        user_img blob,
89        primary key (username)
90      );
91
```

Entries in the user table with the path of the image given to the load_file function:

```
548     -- Insert entries from Faculty table
549 •   INSERT INTO Users (username, password,user_img)
550     VALUES ('First_A', MD5('First_A@123'), load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
551            ('First_B', MD5('First_B@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
552            ('First_C', MD5('First_C@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
553            ('First_D', MD5('First_D@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
554            ('First_E', MD5('First_E@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
555            ('First_F', MD5('First_F@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
556            ('First_G', MD5('First_G@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
557            ('First_H', MD5('First_H@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
558            ('First_I', MD5('First_I@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png")),
559            ('First_J', MD5('First_J@123'),load_file("D:\Books & Assignments 3rdYear\Sem6\DBMS\Assignment2\icons8-male-user-material-rounded\icons8-male-user-96.png"));
```

We also added the Caption column later.

```
590
591 •   use transportmanagement;
592 •   ALTER TABLE Users ADD COLUMN Caption varchar(20);
593 •   select * from Users;
594
595 •   Update Users Set Caption = "User Image";
```

Similarly in this operation we have put all the users from the students, staff and other necessary tables. Load_file has been given the local path where the image has been stored. Please download the image from the drive link and place it in the desired path. Place the path in the load_file("Place here") and then run the query.

Operation 3 satisfies specification 2.

3.

Operation 4:

Natural Join Operation:

```
1 •  SELECT v.license_plate_number, v.capacity, v.owner_first_name, v.owner_last_name, v.vehicle_type,
2    i.insurance_id, i.start_date, i.end_date
3    FROM vehicle v
4    JOIN insurance i ON v.license_plate_number = i.license_plate_number;
```

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

license_plate_number	capacity	owner_first_name	owner_last_name	vehicle_type	insurance_id	start_date	end_date
GJ01AB1234	4	Kora	Massey	Car	INS123456789	2024-01-01	2029-01-01
GJ02CD4567	7	Donald	Davila	Two-wheeler	INS234567890	2024-02-01	2027-02-01
GJ03EF7890	5	Rayne	Romero	Car	INS345678901	2024-03-01	2029-03-01
GJ04GH0123	6	Bryson	Greene	Van	INS456789012	2024-04-01	2027-04-01
GJ05IJ4567	4	Selena	Wood	Car	INS567890123	2024-05-01	2027-05-01
GJ06KL7890	8	Carson	Conner	Van	INS678901234	2024-06-01	2029-06-01
GJ09QR6789	4	Briana	Kirk	Bus	INS901234567	2024-09-01	2029-09-01
GJ09QT6969	35	Lewis	Hamilton	Bus	INS234567890	2024-12-01	2027-12-01
GJ10ST0123	6	Alessandro	Carso	Van	INS012345678	2024-10-01	2027-10-01
RJ07MN0123	5	Alondra	Bell	Car	INS789012345	2024-07-01	2029-07-01
RJ07MN0245	35	Max	Verstappen	Bus	INS123456789	2024-11-01	2029-11-01
RJ08OP3456	7	Emmett	Hood	Two-wheeler	INS890123456	2024-08-01	2027-08-01

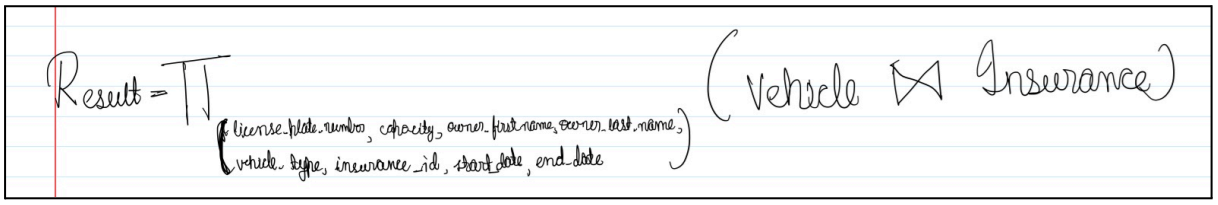
Result 1

Output

Action Output

#	Time	Action	Message
1	17:18:17	SELECT v.license_plate_number, v.capacity, v.owner_first_name, v.owner_last_name, v.vehicle...	12 row(s) returned

Relations Algebra for this operation;



This function performs a natural join operation between the tables vehicle and insurance.

Operation 4 satisfies specification 3.

→

Operation 5:

OUTER JOIN:

```
1 •  SELECT *
2    FROM vehicle
3    LEFT OUTER JOIN allocatedparking ON vehicle.license_plate_number = allocatedparking.license_plate_number;
```

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

license_plate_number	capacity	owner_first_name	owner_last_name	vehicle_type	license_plate_number	location
GJ01AB1234	4	Kora	Massey	Car	GJ01AB1234	Housing Block Parking Area
GJ02CD4567	7	Donald	Davila	Two-wheeler	GJ02CD4567	Hostel Parking Area
GJ03EF7890	5	Rayne	Romero	Car	GJ03EF7890	Behind Academic Block
GJ04GH0123	6	Bryson	Greene	Van	GJ04GH0123	Academic Block Parking lot
GJ05IJ4567	4	Selena	Wood	Car	GJ05IJ4567	Housing Block Parking Area
GJ06KL7890	8	Carson	Conner	Van	GJ06KL7890	Hostel Parking Area
GJ09QR6789	4	Briana	Kirk	Bus	GJ09QR6789	Housing Block Parking Area
GJ09QT6969	35	Lewis	Hamilton	Bus	NULL	NULL
GJ10ST0123	6	Alessandro	Carso	Van	GJ10ST0123	Hostel Parking Area
RJ07MN0123	5	Alondra	Bell	Car	RJ07MN0123	Behind Academic Block
RJ07MN0245	35	Max	Verstappen	Bus	NULL	NULL
RJ08OP3456	7	Emmett	Hood	Two-wheeler	RJ08OP3456	Academic Block Parking lot

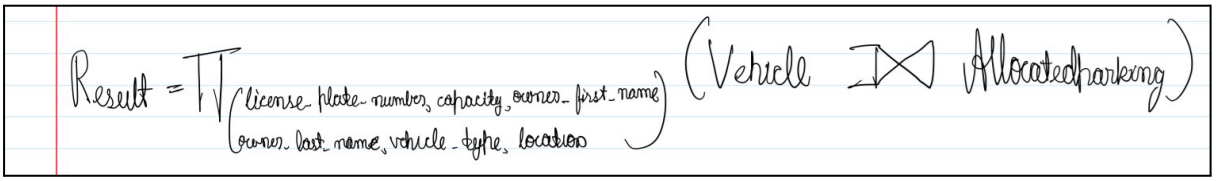
Result 1

Output

Action Output

#	Time	Action	Message
1	17:43:23	SELECT * FROM vehicle LEFT OUTER JOIN allocatedparking ON vehicle.license_plate_number...	12 row(s) returned

Relations Algebra for this operation;



This function retrieves all records from the "vehicle" table along with matching records from the "allocatedparking" table based on the common column "license_plate_number" using LEFT OUTER JOIN. If there is no match, columns from the "allocatedparking" table will contain NULL values.

Operation 5 satisfies specification 3.

→

Operation 6:

CASE Operation:

```
1 •  SELECT license_plate_number, capacity, owner_first_name, owner_last_name,
2    CASE WHEN vehicle_type = "Bus" THEN 'InstituteBus' ELSE 'Other' END AS vehicle_type
3    FROM vehicle;
```

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

license_plate_number	capacity	owner_first_name	owner_last_name	vehicle_type
GJ01AB1234	4	Kora	Massey	Other
GJ02CD4567	7	Donald	Davila	Other
GJ03EF7890	5	Rayne	Romero	Other
GJ04GH0123	6	Bryson	Greene	Other
GJ05IJ4567	4	Selena	Wood	Other
GJ06KL7890	8	Carson	Conner	Other
GJ09QR6789	4	Briana	Kirk	InstituteBus
GJ09QT6969	35	Lewis	Hamilton	InstituteBus
GJ10ST0123	6	Alessandro	Carso	Other
RJ07MN0123	5	Alondra	Bell	Other
RJ07MN0245	35	Max	Verstappen	InstituteBus
RJ08OP3456	7	Emmett	Hood	Other

Result 1

Output

Action Output

#	Time	Action	Message
1	17:18:17	SELECT v.license_plate_number, v.capacity, v.owner_first_name, v.owner_last_name, v.vehicle...	12 row(s) returned
2	17:25:39	SELECT license_plate_number, capacity, owner_first_name, owner_last_name, CASE WHEN ve...	12 row(s) returned

This function selects the license plate number, capacity, owner names, and categorizes vehicles as 'InstituteBus' or 'Other' based on their type using CASE statement from the "vehicle" table.

Operation 6 satisfies specification 3.

→

Operation 7:

```
672  -- Total number of trips from IITGN Campus to Ahmedabad Airport
673  SELECT COUNT(*) AS TotalTrips
674  FROM TransportationLog
675  WHERE starting_station = 'IITGN Campus' AND ending_station = 'Ahmedabad Airport';
676
677
```

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

TotalTrips
4

Result 22 | Read Only

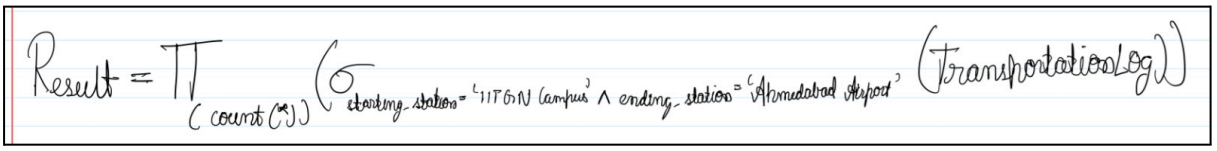
Output

Action Output

#	Time	Action	Message	Duration / Fetch
36	16:15:00	explain analyze SELECT username from Users where username="Sophia"	1 row(s) returned	0.000 sec / 0.000 sec
37	16:18:25	create index booking_index on booking(email_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.062 sec
38	16:19:07	create index translog_index on transportationlog(license_plate_number)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.062 sec
39	17:17:07	select count(*) username from users LIMIT 0, 1000	1 row(s) returned	0.031 sec / 0.000 sec
40	17:25:17	SELECT COUNT(*) AS TotalTrips FROM TransportationLog WHERE starting_station = 'IITGN Cam...	1 row(s) returned	0.000 sec / 0.000 sec

This operation counts the number of buses that go from IITGN campus to Ahmedabad Airport.

Relations Algebra for this operation;



Operation 7 satisfies specification 3.

→

Operation 8:

```
686
687  SELECT AVG(route_distance) AS AverageDistance
688  FROM Route;
```

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

AverageDistance
20.0000

Result 24 | Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	17:46:10	ALTER TABLE DrivenBy RENAME TO VehicleAssociations	0 row(s) affected	0.015 sec
2	17:47:48	SELECT AVG(route_distance) AS AverageDistance FROM Route LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

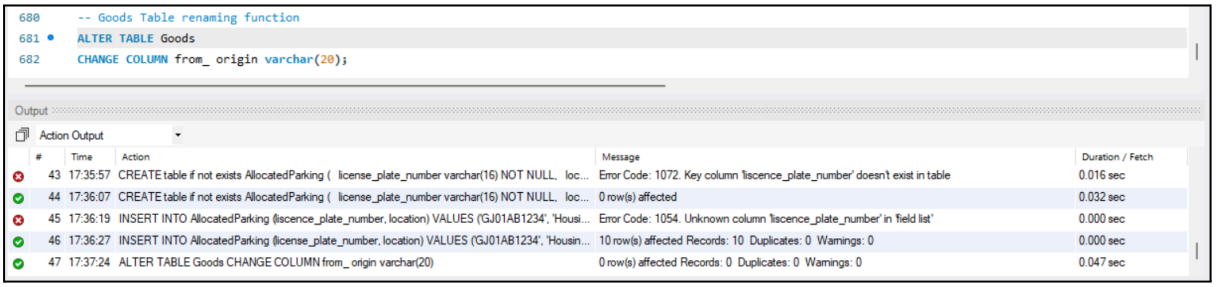
This operation uses the AVG() function to calculate the average route distance from the ROUTE table.

Relational Algebra for this operation:

Operation 8 satisfies specification 3.

→

Using the RENAME operation for the Goods table:

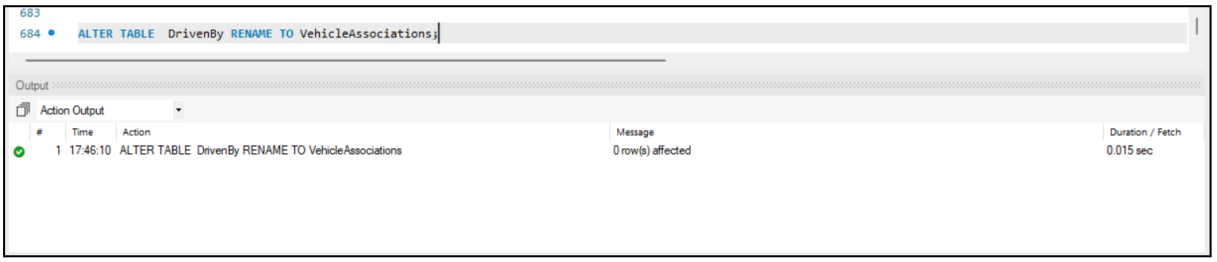


Operation 8 satisfies specification 3.

→

Operation 9:

Using the RENAME operation on Drivenby table to change its name to VehicleAssociations:



Operation 9 satisfies specification 3.

Note: We have provided the relational algebra for four operations.

4. Contributions

Name	Roll Number	Contribution
Ayush Modi (G2)	21110039	3.2, 3.3 1) Helped in overall documentation 2) Answered the questions from 3.2 to including 3.3 3) Helped in creating enums and json arrays for uddts 5) Created the views asked in 3.2 namely view1 and view2 4) Created the user1 and granted user1 the permissions on table1 and view1 5) Wrote the queries asked in 3.2 and provided screenshots and reasonings of the outputs received. 6) Helped in the specification 3 of the question 3.3
Mithil Pechimuthu (G1)	21110129	3.1, 3.3 1) Designed and implemented the relational schemas of the database and their instances, alongside Vedant Kumbhar and Shreesh Agrawal. 2) Created table extensions. 3) Helped complete questions in 3.3 alongside Vedant Kumbhar 4) Documented all the steps and answers.
Vedant Kumbhar(G1)	21110234	3.1, 3.3 1) Contributed making tables of entities and relationships. Also added constraints on the tables wherever required. 2) Helped populate the data into the tables. 3) Implemented a few functionalities such as aggregate functions, commit/rollback, creating indexes, inserting images using BLOB data type and user defined data types using the JSON arrays.
Shreesh Agrawal (G1)	21110198	3.1, 3.3 1) Helped in overall documentation. 2) Made tables of entities, relationships with constraints. 3) Searched for inconsistencies, errors and redundancy in G1 tables 4) Populated the tables with dummy data. 5) Checked the suitable places for user defined data types. 6) Implemented user defined data types in JSON dictionary format and updated the whole dummy data according to that. 7) Implemented Set Comparison queries. 8) Did error check and correction in various places like referential integrity and Users table (as mentioned in UDDTs).

Anushk Bhana (G2)	21110031	3.2 1) Identified causes of referential integrity violations in the database. 2) Provided examples of each cause within the database context, including scenarios such as inserting records with non-existent keys, updating rows without maintaining relationships, and importing data inconsistent with schema. 3) Offered solutions to address referential integrity violations. 4) Summarized major solutions through code. 5) Concluded by emphasizing the importance of maintaining referential integrity for data consistency and reliability in the database.
Abhishek Mandlik (G2)	21110122	3.2 Answered question based on referential integrity