Design of Car Rental System

By:

Milan Regmi (C0900873)

Roshan Shrestha (C0901142)

Srijesh Khanal (C0901118)

Table of Contents

In this project a database design for car rental system is shown. This is the system for car rental office, so the database design is made accordingly. This project is divided into three sections; 1. Logical Design, 2. Normalization/Constraints and 3. Physical Database Design. In this car rental system, customer can rent a car based on the model. Customer can choose the car from different location for pick up and drop. Customer can confirm this with the employees working in that office. The rented car can also have insurance which customer can buy based on their convenience. One of the assumption is that all the car belongs to rental office itself. The entities and relation associated with the entities are described in details in the sections below.

1. Logical Design

1.1. Entities

a. Customer

Customer will be the one who will be using car rental system to rent a car. Customer can be member or non-member. Attributes like firstName, lastName, memberId, address, drivingLicense, etc will be for customer.

b. Car

The next entity will be Car. It keeps the information of cars. Car types, model and any other information related to the Car will be the attributes of this entity. It can have one attribute named isAvailable which can indicate whether the car is available to rent.

c. Insurance

Insurance here represents car rental insurance. Customer can already have insurance or can buy one while signing the contract during renting process.

d. Location

Location represents the pickup and drop location while renting the car.

e. Office

Office can be where customer makes deal for renting the car.

f. Employee

Employee is the one with whom the customer is making deal

g. Payment

Depending on the nature of contract the payment is done by customer

h. Contract

The contract while renting the car. It is between office and customer. It also includes billing method.

1.2. Assumptions

- Car should be rented only by car rental office (Not individual)
- The driver license is required for customer in order to rent car
- Pick up and drop location should be same

1.3. Attributes

The possible attributes for above entities with the primary key is shown below.

a. Customer

Mandatory Attributes	Optional Attributes
Driver_License_Number	Member_ID
First_Name	Gender
Last_Name	
Street	
City	
Postal_Code	
Province	
Phone	
Email	

b. Car

Mandatory Attributes	Optional Attributes
Chassis_Number	Make
Model	Condition
Model_Number	
Is_Available	
Mileage	
No_Of_Person	
Price_Per_Day	
Late_Fee_Per_Hour	
No_Of_Luggage	

Insurance_Code	

c. Insurance

Mandatory Attributes	Optional Attributes
Insurance_Code	Name
Coverage_Type	
Cost_Per_Day	

d. Location

Mandatory Attributes	Optional Attributes
Location_ID	
Street	
City	
Postal_Code	
Province	

e. Office

Mandatory Attributes	Optional Attributes
Office_ID	Province
Name	
Address	
Postal_Code	

f. Employee

Mandatory Attributes	Optional Attributes
Employee_ID	Gender
First_Name	Age
Last_Name	
Address	
Office_ID	

g. Payment

Mandatory Attributes	Optional Attributes
Payment_ID	Advance_Amount
Payment_Type	Cancelation_Charge
Payment_Due_Date	Late_Fee
Total_Amount	
Tax_Amount	
Payment_Status	
Driver_License_Number	
Contract_ID	

h. Contract

Mandatory Attributes	Optional Attributes
Contract_ID	
Start_Date	
End_Date	
Contract_Status	
Return_Date	
Amount	
Chassis_Number	
Driver_License_Number	
Office_ID	
Location_ID	

1.3. Relations

a. Customer to Contract

Customer will rent a car as per the desire. Customer have to rent a car via contract, so customer will be related to car via contract. So the relation between Customer to Contract is named as “Choose”.

b. Car to Insurance

Car to be rented must have insurance. Car can have many type of insurance, like full coverage, partial, third party, etc. So the relation between Car and Insurance can be named as “Has”.

c. Contract to Location

Customer can rent a car from the desire location which will be defined in contract. They can mention pickup and drop location. So Contract “Mentions” Location.

d. Employee to Office

Employee “Works On” office which rents car to the customer.

e. Customer to Payment to Contract

Customer should make the payment on the basis of contract. If contract is cancelled before payment, payment is not done. The relation between Customer and Payment can be named as “Gives”. The relation between Payment and Contract can be named as “Based On”. If contract is canceled after advance payed certain charge will be deducted.

f. Office to Contract

Office makes the contract so the relation name is “Makes”

1.4. Cardinality

To determine the cardinality between above defined tables, we need to understand the relationship between them. Based on the attribute mentioned in above table description, the cardinality between the tables can be as follows.

- Customer:

  Customer must have 1 Driver_License_Number.

- Car:

  Car must have 1 Chassis_Number.

  Car can have 0 or 1 Insurance_Code

- Insurance:

  Insurance must have 1 Insurance_Code.

- Location:

  Location must have 1 Location_ID.

- Office:

  Office must have 1 Office_ID.

- Employee:

  Employee must have 1 Employee_ID.

  Employee must have 1 Office_ID.

- Payment:

  Payment must have 1 Payment_ID.

  Payment can have 0 or 1 Driving_License.

  Payment can have 0 or 1 Contract_ID.

- Contract:

  Contract must have 1 Contract_ID.

  Contract must have 1 Chassis_Number.

  Contract must have 1 Driving_License.

  Contract must have 1 Office_ID.

  This can be shown in table below.

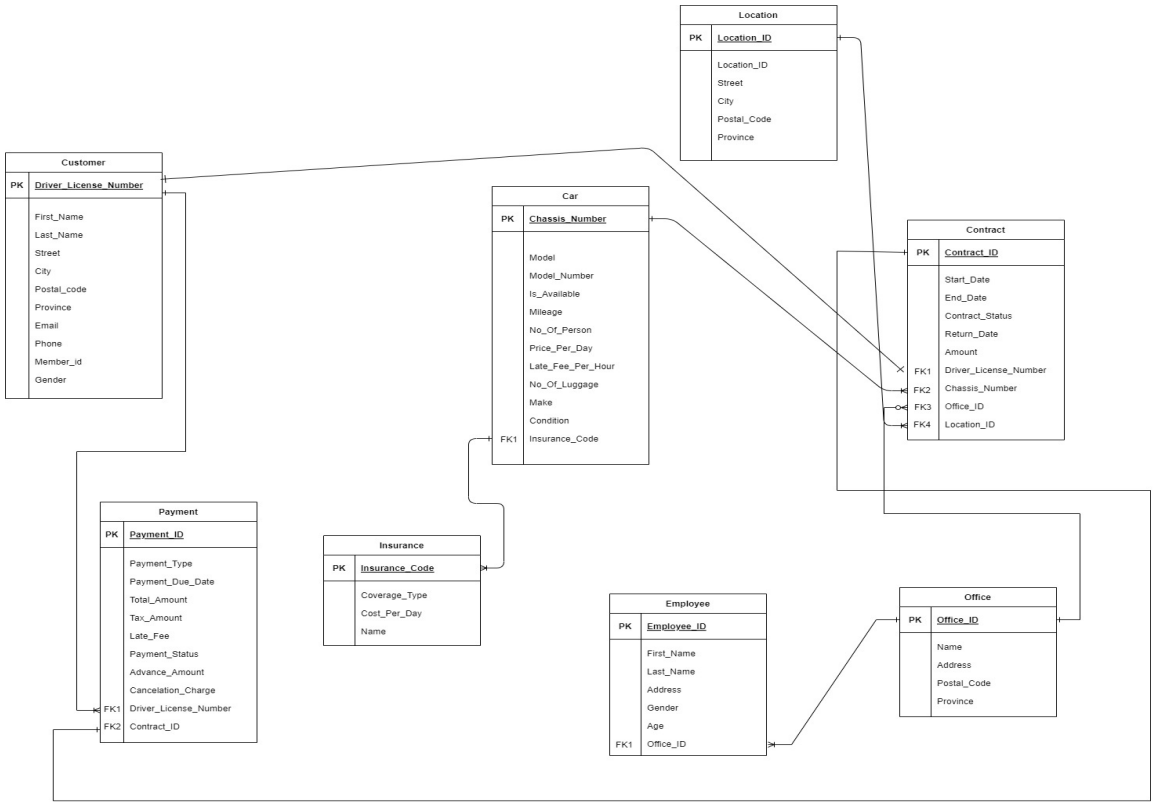
	Customer	Car	Insurance	Location	Office	Employee	Payment	Contract
Customer							N	N
Car			N					N
Insurance		1						
Location								N
Office						N		N
Employee					1			
Payment	1							1
Contract	1	1		1	1		1	

1.5. Relation Matrix

The following table shows the relationship matrix

	Insurance	Location	Office	Payment	Contract
Customer				Gives	Choose
Car	Has				
Location					
Office					Makes
Employee			Works On		
Payment					Based On
Contract		Mentions			

1.6. Entity Relationship Diagram (ERD)



2. Normalization and Constraints

2.1. Normalization

Normalization is the application of rules (normal forms) to table structures that results in a design that is free of data redundancy and other anomalies in the database. Normalization is a crucial step in achieving an efficient database design. Its main goals are to eliminate unnecessary repetitions of data and ensure that the data in each table are logically related. By undergoing normalization, databases experience reduced redundancies, fewer anomalies, and improved overall efficiency. The two primary purposes of normalization are as follows:

- Remove duplicate data, avoiding the storage of the same information in multiple tables.
- Establish strong relationships between the data within each table.

There are various types of normalization. The fundamental forms are 1NF, 2NF, and 3NF. The following are the criteria for 1NF, 2NF, and 3NF.

a. 1NF:

The table is said to be in first normal form (1NF) when:

- There are no repeated groups of data.
- There are no columns that store multiple values.
- The table has a primary key defined.
- All the columns in the table rely on the primary key for their values.

b. 2NF:

The second normal form (2NF) is relevant for tables with composite primary keys. A table is considered to be in 2NF when it meets the following conditions:

- It has already achieved first normal form (1NF).
- There are no partial dependencies, meaning that each non-key attribute is dependent on the entire composite primary key.

c. 3NF:

- A table in a database is considered to be in the third normal form (3NF) under the following conditions:
- It has already achieved the second normal form (2NF).

- The table is free from non-key dependencies, also known as transitive dependencies. A non-key attribute in the table should not determine the value of another non-key attribute. To meet the requirements of 3NF, each attribute should solely depend on the primary key.

Besides, 1NF, 2NF and 3NF there is also another normal form called Boyce-Code normal form (BCNF). A table in database is said to be in BCNF when every determinant is a candidate key. D determinant refers to any column in a row that determines the value of another column.

For this project we only focus on 1NF, 2NF and 3NF. The normalization of each tables is explained below.

Customer Table

Driver_License_Number	First_Name	Last_Name	Street	City	Postal_Code	Province	Phone	Email	Member_ID	Gender
OB1234	Milan	Regmi	Pressed Brick Dr	Brampton	L6V4L4	Ontario	4371237000	regmi@gmial.com	NULL	M
OM9878	Roshan	Shrestha	Derry Road W	Mississauga	L5N7L7	Ontario	4374568000	roshan@gmail.com	CMR2345	M
OB0525	Srijesh	Khanal	Bartley Bull PkWy	Brampton	L6W2L5	Ontario	4377899000	sk@gmail.com	CBS0021	M
OM0012	Mae	Parker	Square One Dr	Mississauga	L5B0E2	Ontario	4372002000	mae@gmail.com	NULL	F

The records in above Customer table is only the sample data to show the normalization. The above Customer table satisfies the condition of 1NF i.e. it does not have multivalued columns, there is no repeated groups and the primary key is identified and other column are dependent on primary key. It is also in 2NF because it satisfies the condition of 1NF and also there is no any partial key dependency (non-key attribute depends on entire on primary key). But the table is not in 3NF because it has transitive dependency, i.e. the table contains non-key dependency. Transitive dependency in Customer table is shown below:

Driver_License_Number => First_Name, Last_Name, Street, City, Postal_Code, Province, Phone, Email, Member_ID, Gender

Postal_Code => Province, City, Province

We can achieve 3NF by decomposing the Customer table further as shown below.

Personal_Info

Driver_License_Number	First_Name	Last_Name	Phone	Email	Member_ID	Gender	Postal_Code
OB1234	Milan	Regmi	4371237000	regmi@gmial.com	NULL	M	L6V4L4
OM9878	Roshan	Shrestha	4374568000	roshan@gmail.com	CMR2345	M	L5N7L7
OB0525	Srijesh	Khanal	4377899000	sk@gmail.com	CBS0021	M	L6W2L5
OM0012	Mae	Parker	4372002000	mae@gmail.com	NULL	F	L5B0E2

Address_Info

Postal_Code	Street	City	Province
L6V4L4	Pressed Brick Dr	Brampton	Ontario
L5N7L7	Derry Road W	Mississauga	Ontario
L6W2L5	Bartley Bull PkWy	Brampton	Ontario
L5B0E2	Square One Dr	Mississauga	Ontario

But for convenience of our database design we keep Customer table in 2NF instead of 3NF. As in 2NF Customer table do not have partial dependencies and also from the functional perspective, 2NF also guarantees the data integrity and reduce data redundancy and also if we normalize into 3NF it will lead to more table creation and the relation might become more complex as well as more query have to be written and takes more time which could be reduced if we keep it in 2NF.

Car Table

Chassis_Number	Model	Model_Number	Is_Available	Mileage	No_Of_Person	Price_Per_Day	Late_Fee_Per_Hour	No_Of_Luggage	Insurance_Code	Make	Condition
ABC123D45	Sedan	SD001	Yes	25000	5	50.00	5.00	2	INS1234	Toyota	Excellent
XYZ567F89	Truck	TR004	No	45000	3	120.00	12.00	5	INS5678	Chevrolet	Good
LMN092G92	SUV	SV023	Yes	17000	7	80.00	9.00	4	INS9012	Ford	Fair
PQR212U34	SUV	SV125	No	18000	7	82.00	9.00	4	INS8098	Ford	Good

The above Car table satisfies the condition of 1NF i.e. it does not have repeated groups, all attributes contains atomic value and the primary key is identified and other column are dependent on primary key. It is also in 2NF because it satisfies the condition of 1NF and also there is no any partial key dependency (non-key attribute depends on entire on primary key). But the table is not in 3NF because it has transitive dependency, i.e. the table contains non-key dependency. Transitive dependency in Car table is shown below:

Chassis_Number => Model, Model_Number, Is_Available, Mileage, No_Of_Person, Price_Per_day, Late_Fee_Per_Hour, No_Of_Luggage, Insurance_Code, Make, Condition

Model_Number => Model, Make

We can achieve 3NF by decomposing the Car table further as shown below.

Car_Info

Chassis_Number	Model_Number	Is_Available	Mileage	No_Of_Person	Price_Per_Day	Late_Fee_Per_Hour	No_Of_Luggage	Insurance_Code	Condition
ABC123D45	SD001	Yes	25000	5	50.00	5.00	2	INS1234	Excellent
XYZ567F89	TR004	No	45000	3	120.00	12.00	5	INS5678	Good
LMN092G92	SV023	Yes	17000	7	80.00	9.00	4	INS9012	Fair
PQR212U34	SV125	No	18000	7	82.00	9.00	4	INS8098	Good

Model_Info

Model_Number	Model	Make
SD001	Sedan	Toyota
TR004	Truck	Chevrolet
SV023	SUV	Ford
SV125	SUV	Ford

Again we can see that Model_Info table still have transitive dependency.

Model_Number => Model, Make

Model => Make

So we further decompose this table to make in 3NF as follows:

Model_Num_Model Table

Model_Number	Model
SD001	Sedan
TR004	Truck
SV023	SUV
SV125	SUV

Make_Info Table

Model	Make
Sedan	Toyota
Truck	Chevrolet
SUV	Ford
SUV	Ford

The final tables after normalizing to 3NF are Car_Info, Model_Num_Model and Make_Info tables.

But for convenience of our database design we keep Car table also in 2NF instead of 3NF. As in 2NF Car table do not have partial dependencies and also from the functional perspective, 2NF also guarantees the data integrity and reduce data redundancy and also if we normalize into 3NF it will lead to more table creation and the relation might become more complex as well as more query have to be written and takes more time which could be reduced if we keep it in 2NF.

Insurance Table

Insurance_Code	Coverage_Type	Cost_Per_Day	Name
INS1234	Basic	10.00	Standard Insurance
INS5678	Comprehensive	15.00	Premium Insurance
INS9012	Extended	20.00	Elite Insurance
INS8098	Basic	12.00	Superior Insurance
INS7543	Basic	11.00	NULL
INS2288	Extended	25.00	Superior Insurance

The Insurance table satisfy all the condition of 3NF based on following rules:

Satisfies 1NF because

There are no repeating groups, and all the values are atomic with identified primary key.

Satisfies 2NF because

Table's attributes are fully functionally dependent on the Insurance_Code (primary key), there are no partial dependencies in this case. Each non-key attribute (Coverage_Type, Cost_Per_Day, Name) depends on the entire primary key (Insurance_Code).

Satisfies 3NF because

There are not any transitive dependencies in the given data, as no non-key attribute is determining another non-key attribute.

Thus the table already satisfies the requirements of 3NF, there is no need to further decompose it into additional tables. The current structure is well-organized and normalized, ensuring data integrity and eliminating redundancy.

Location Table

Location_ID	Street	City	Postal_Code	Province
LOC0001	Bartley Bull PkWy	Brampton	L6W2J4	Ontario
LOC0010	Maple Lane	Toronto	M5V2T6	Ontario
LOC0021	Main Street	Brampton	L2V9Y3	Ontario

The above Location table satisfies the condition of 1NF i.e. it does not have repeated groups, all attributes contains atomic value and the primary key is identified and other column are dependent on primary key. It is also in 2NF because it satisfies the condition of 1NF and also there is no any partial key dependency (non-key attribute depends on entire on primary key). But the table is not in 3NF because it has transitive dependency, i.e. the table contains non-key dependency. Transitive dependency in Location table is shown below:

Location_ID => Street, City, Postal_Code, Province

Postal_Code => Street, City, Province

We can achieve 3NF by decomposing the Location table further as shown below

Location_Info

Location_ID	Postal_Code
LOC0001	L6W2J4
LOC0010	M5V2T6
LOC0021	L2V9Y3

Address_Info

Postal_Code	Street	City	Province
L6W2J4	Bartley Bull PkWy	Brampton	Ontario
M5V2T6	Maple Lane	Toronto	Ontario
L2V9Y3	Main Street	Brampton	Ontario

But for convenience of our database design we keep Location table in 2NF instead of 3NF. As in 2NF Location table do not have partial dependencies and also from the functional perspective, 2NF also guarantees the data integrity and reduce data redundancy and also if we normalize into 3NF it will lead to more table creation and the relation might become more complex as well as more query have to be written and takes more time which could be reduced if we keep it in 2NF.

Office Table

Office_ID	Name	Address	Postal_Code	Province
Off0001	Asian UHaul	378 Bartley Bull PkWy	W7H1O2	Ontario
Off0010	Maple Rental	7210Maple Lane	L5E0T2	Ontario
Off0021	GTA Cars	243 Main Street	L3W5Y2	Ontario

The above Office table satisfies the condition of 1NF i.e. it does not have repeated groups, all attributes contains atomic value and the primary key is identified and other column are dependent on primary key. It is also in 2NF because it satisfies the condition of 1NF and also there is no any partial key dependency (non-key attribute depends on entire on primary key). But the table is not in 3NF because it has transitive dependency, i.e. the table contains non-key dependency. Transitive dependency in Office table is shown below:

Office_ID => Name, Address, Postal_Code, Province

Postal_Code => Address, Province

We can achieve 3NF by decomposing the Location table further as shown below

Office_Info

Office_ID	Name	Postal_Code
Off0001	Asian UHaul	W7H1O2
Off0010	Maple Rental	L5E0T2
Off0021	GTA Cars	L3W5Y2

Address_Info

Postal_Code	Address	Province
W7H1O2	378 Bartley Bull PkWy	Ontario
L5E0T2	7210Maple Lane	Ontario
L3W5Y2	243 Main Street	Ontario

But for convenience of our database design we keep Office table in 2NF instead of 3NF. As in 2NF Office table do not have partial dependencies and also from the functional perspective, 2NF also guarantees the data integrity and reduce data redundancy and also if we normalize into 3NF it will lead to more table creation and the relation might become more complex as well as more query have to be written and takes more time which could be reduced if we keep it in 2NF.

Employee Table

Employee_ID	First_Name	Last_Name	Address	Gender	Age	Office_ID
EmpO001E001	John	Doe	234 Rambler Dr W Ontario	M	31	Off0001
EmpO001E012	Myra	Starc	310 Kennedy Ontario	F	27	Off0001
EmpO021E005	David	Bolt	243 Main Street Ontario	M	27	Off0021
EmpO010E003	Jenifer	Sharma	234 Rambler Dr W Ontario	F	28	Off0010

The Employee table satisfy all the condition of 3NF based on following rules:

Satisfies 1NF because

There are no repeating groups, and all the values are atomic with identified primary key.

Satisfies 2NF because

Table's attributes are fully functionally dependent on the Employee_ID (primary key), there are no partial dependencies in this case. Each non-key attribute

(First_Name, Last_Name, Address, Gender, Age, Office_ID) depends on the entire primary key (Employee_ID).

Satisfies 3NF because

There are not any transitive dependencies in the given data, as no non-key attribute is determining another non-key attribute.

Thus the table already satisfies the requirements of 3NF, there is no need to further decompose it into additional tables. The current structure is well-organized and normalized, ensuring data integrity and eliminating redundancy.

Payment Table

Mandatory Attributes
Payment_ID
Payment_Type
Payment_Due_Date
Total_Amount
Tax_Amount
Payment_Status
Advance_Amount
Cancelation_Charge
Late_Fee
Driver_License_Number
Contract_ID

The Payment table satisfy all the condition of 3NF based on following rules:

Satisfies 1NF because

There are no repeating groups, and all the values are atomic with identified primary key.

Satisfies 2NF because

Table's attributes are fully functionally dependent on the Payment_ID (primary key), there are no partial dependencies in this case.

Satisfies 3NF because

There are not any transitive dependencies in the given data, as no non-key attribute is determining another non-key attribute.

Contract Table

Mandatory Attributes
Contract_ID
Start_Date
End_Date
Contract_Status
Return_Date
Amount
Chassis_Number
Driver_License_Number
Office_ID
Location_ID

The Contract table satisfy all the condition of 3NF based on following rules:

Satisfies 1NF because

There are no repeating groups, and all the values are atomic with identified primary key.

Satisfies 2NF because

Table's attributes are fully functionally dependent on the Contract_ID (primary key), there are no partial dependencies in this case.

Satisfies 3NF because

There are not any transitive dependencies in the given data, as no non-key attribute is determining another non-key attribute.

Depending on our car rental system application requirements and potential future changes, some of the tables, i.e. Customer, Car, Location and Office will be in a denormalized (2NF) structure because it might offer more flexibility and adaptability.

2.2. Constraints

A constraint, often known as an integrity rule, is a rule or policy that guarantees the data in a database is correct and acceptable in accordance with business requirements. A primary key constraint, for example, ensures that each row in a table has a unique

identifier. When entering, updating, or removing entries, the database management system (DBMS) examines these limitations. When a constraint is violated, the system stops the operation and reports an error.

Three types of integrity rules, namely Data Integrity, Entity Integrity, and Referential Integrity allow for the definition of major constraints that the management system applies automatically during database modifications. These rules secure not only the stated values in columns, but also the identity and interrelationships of rows.

Some of the example of constraints are shown with the help of few tables of Car Rental System.

a. Primary Key Constraint

Customer Table was created using table level primary key constraint. The query is shown below.

```
CREATE TABLE CUSTOMER (
   Driver_License_Number VARCHAR(20),
   First_Name VARCHAR(50),
   Last_Name VARCHAR(50),
   Street VARCHAR(100),
   City VARCHAR(50),
   Postal_Code VARCHAR(10),
   Province VARCHAR(50),
   Phone VARCHAR(20),
   Email VARCHAR(100),
   Member_ID VARCHAR(20),
   Gender CHAR(2),
   CONSTRAINT cus_pk PRIMARY KEY (Driver_License_Number)
);
```

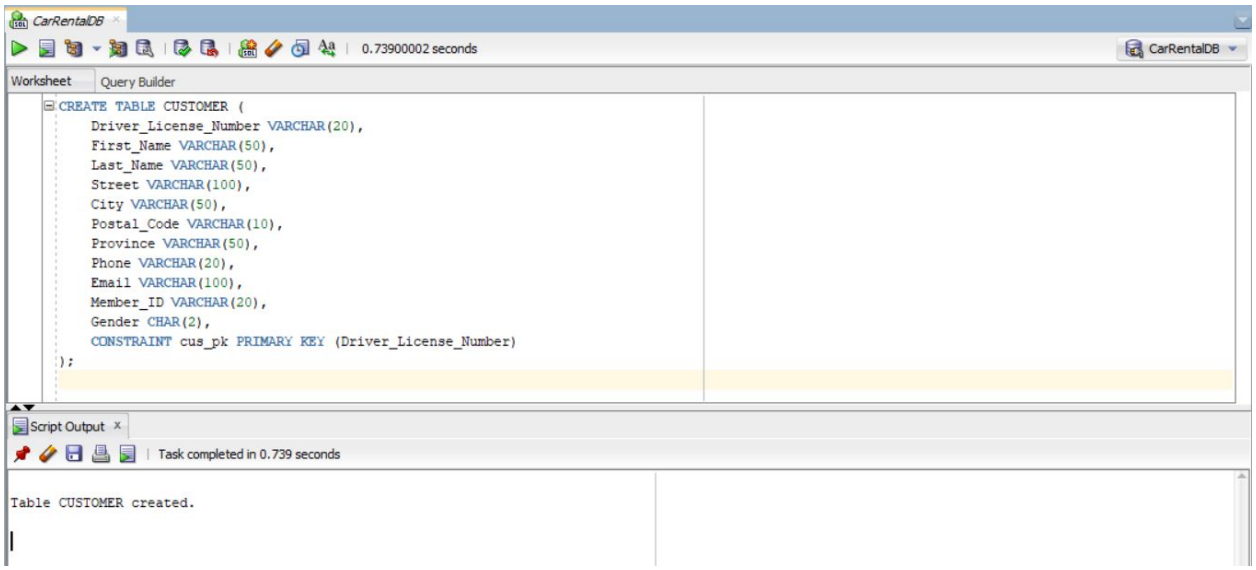


Figure 2: Creating CUSTOMER table using Primary Key Constraint (Table Level)

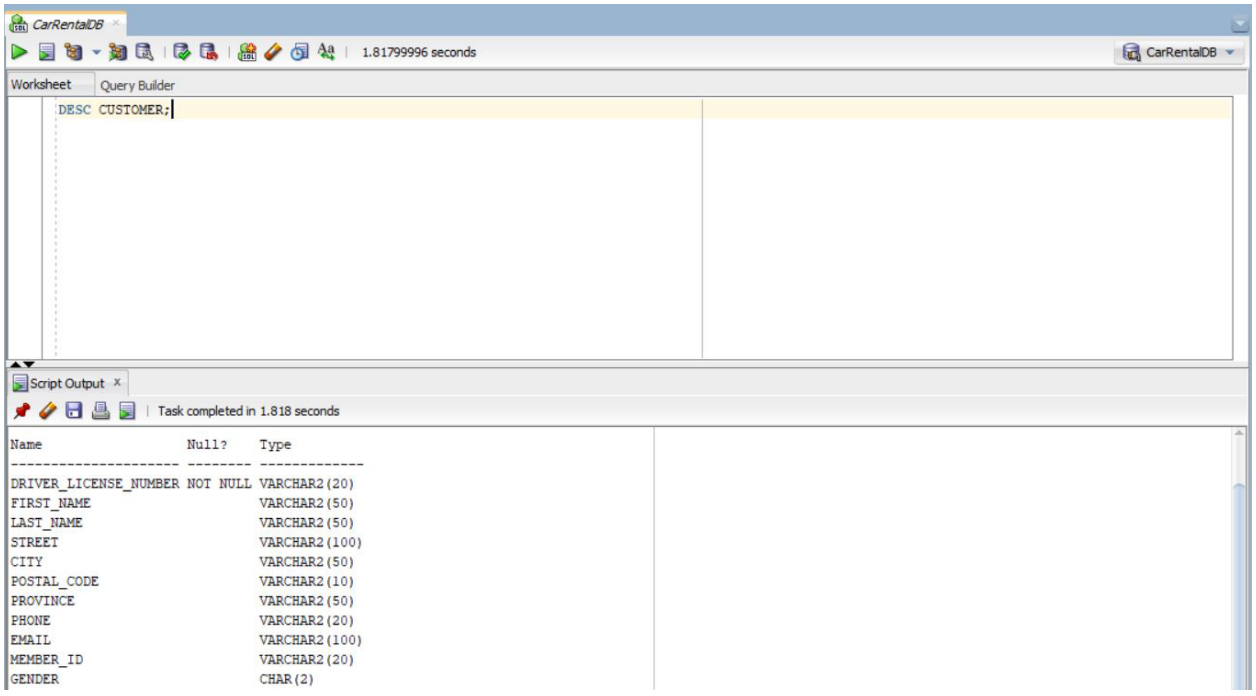


Figure 3: Describing Courses_C0901118 table after creating Primary Key Constraint

Insert data in CUSTOMER table.

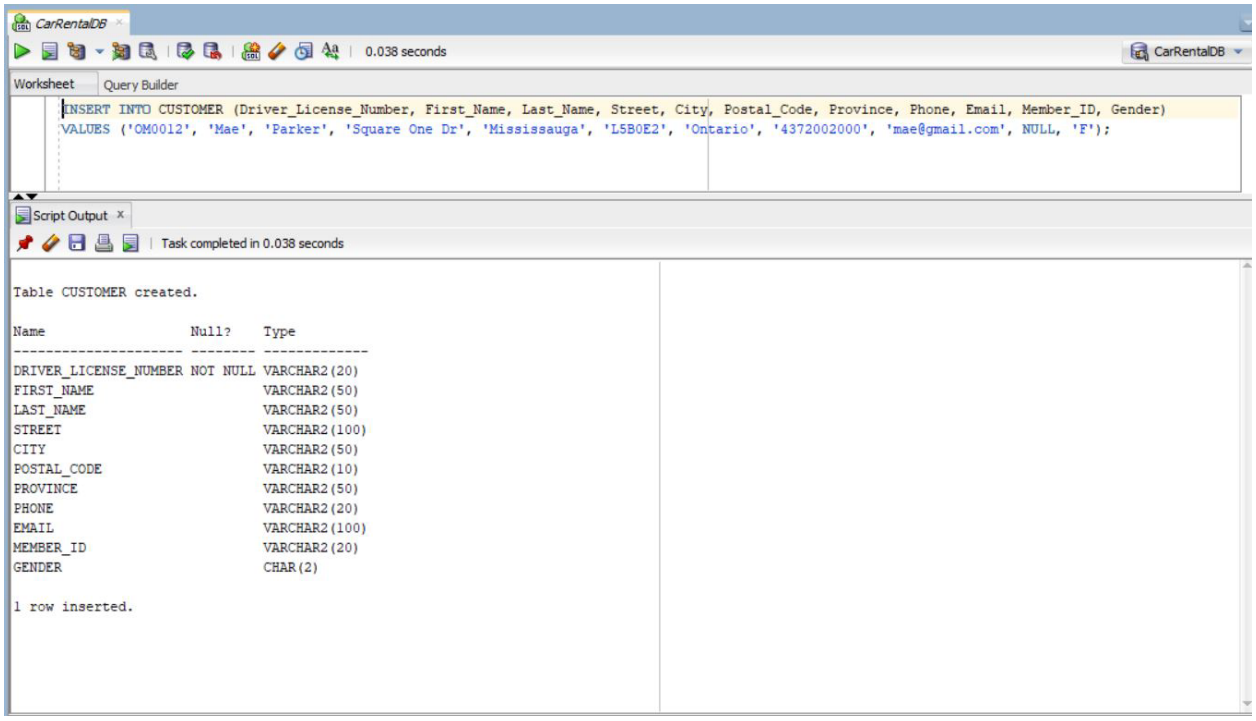


Figure 4: Insert data in CUSTOMER table

Now we try to enter data with same primary key.

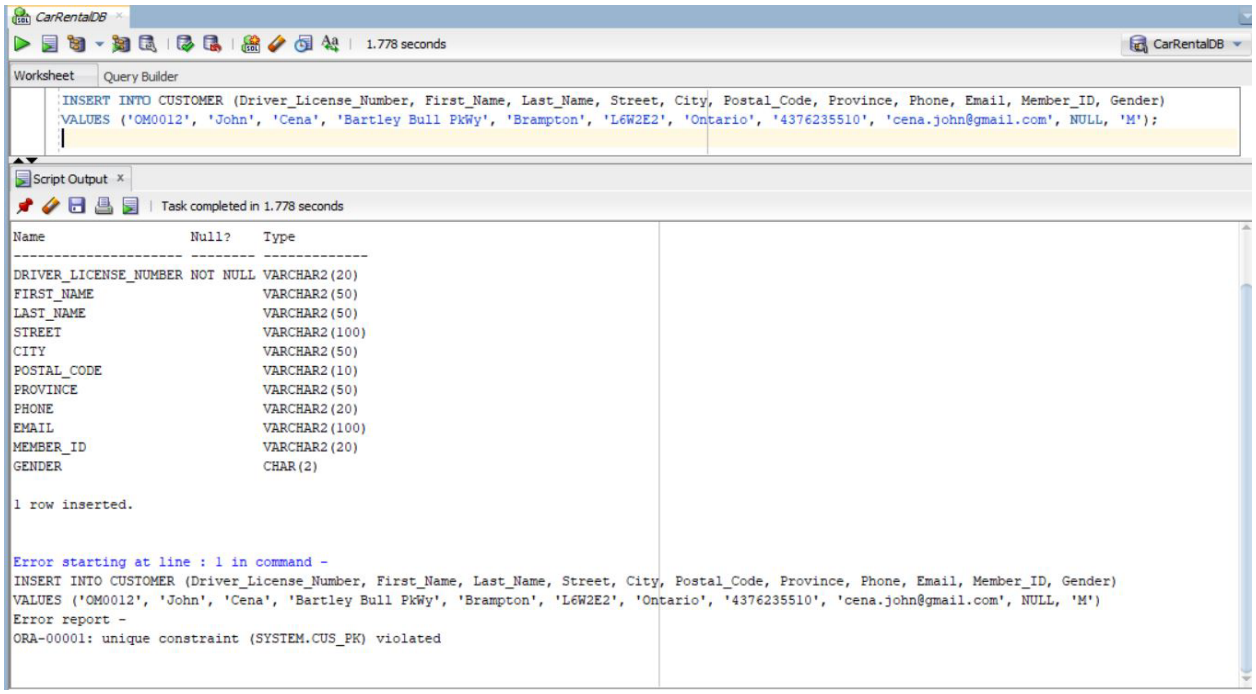


Figure 5: Example that shows Primary Key Constraint

After defining Primary Key constraint, if we try to values in CUSTOMER table with same value in Driver_License_Number, it is not allowed because it violates primary key constraints because Primark Key must be unique value.

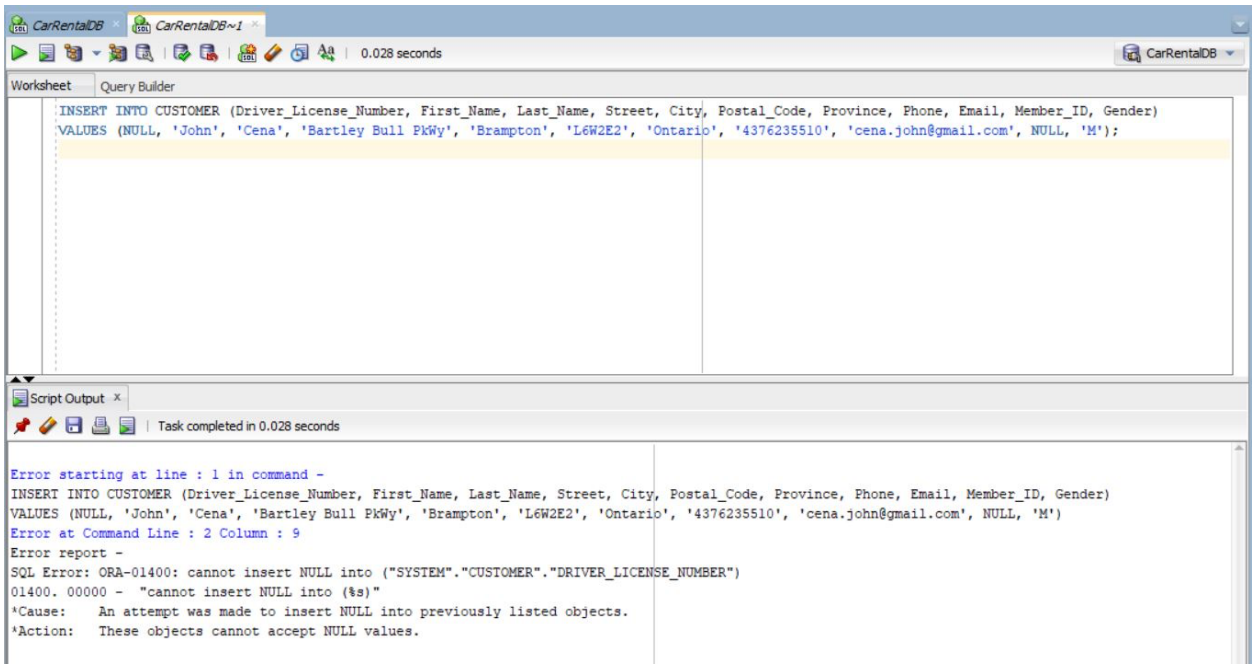


Figure 6: Cannot Insert NULL value in primary key field

Similarly, after creating primary key constraints we cannot enter NULL value in primary key field, i.e. Driver_License_Number. Doing so gives the error as shown in Figure 6.

b. Not Null Constraint

Before defining NOT NULL Constraint we can enter NULL values. For example, in CUSTOMER table not any fields are defined with NOT NULL constraint.

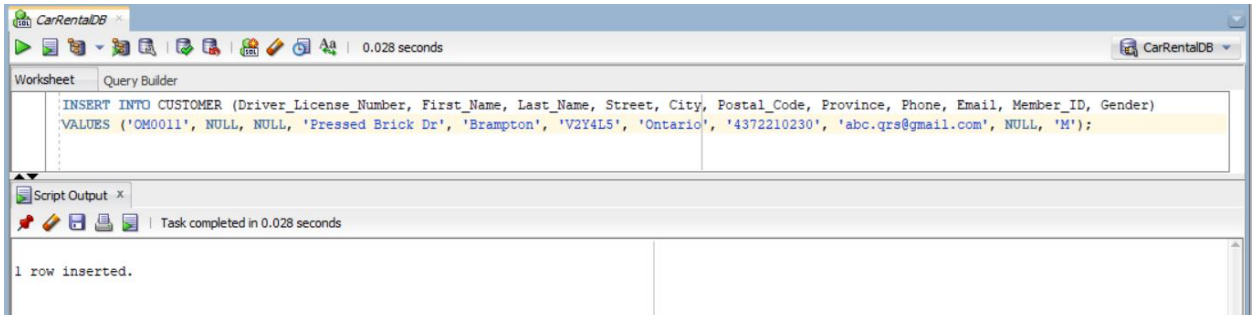


Figure 7: NULL value in First_Name and Last_Name are allowed to insert

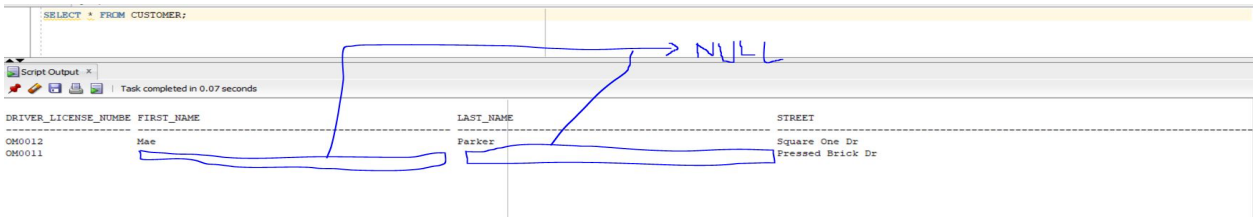


Figure 8: Showing NULL value inserted before defining NOT NULL constraint.

After applying the NOT NULL constraint to a field, that particular field cannot be left with a NULL value. If an attempt is made to do so, an error will occur. The NOT NULL constraint ensures that the field must always contain a non-NULL value, enforcing data integrity and preventing any NULL entries for that attribute. NOT NULL constraint is shown in the figure below. The constraint defined is column level. First_Name and Last_Name are defined as NOT_NULL.

```
ALTER TABLE CUSTOMER
MODIFY First_Name VARCHAR(50) NOT NULL,
MODIFY Last_Name VARCHAR(50) NOT NULL;
```

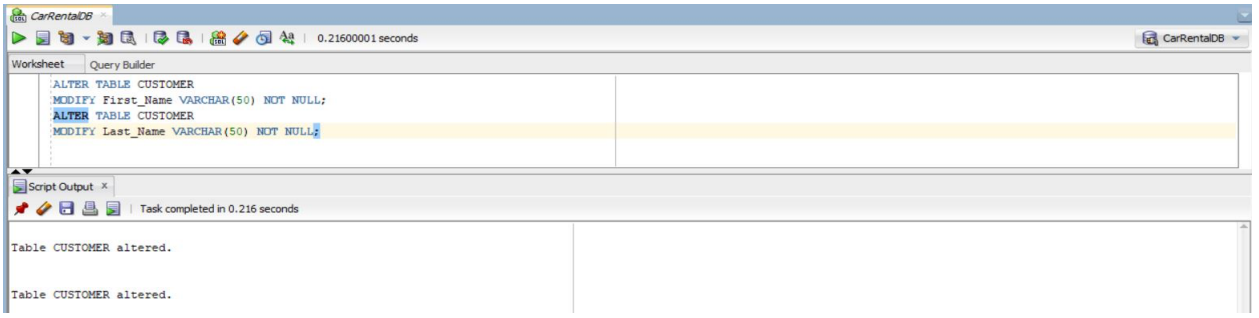
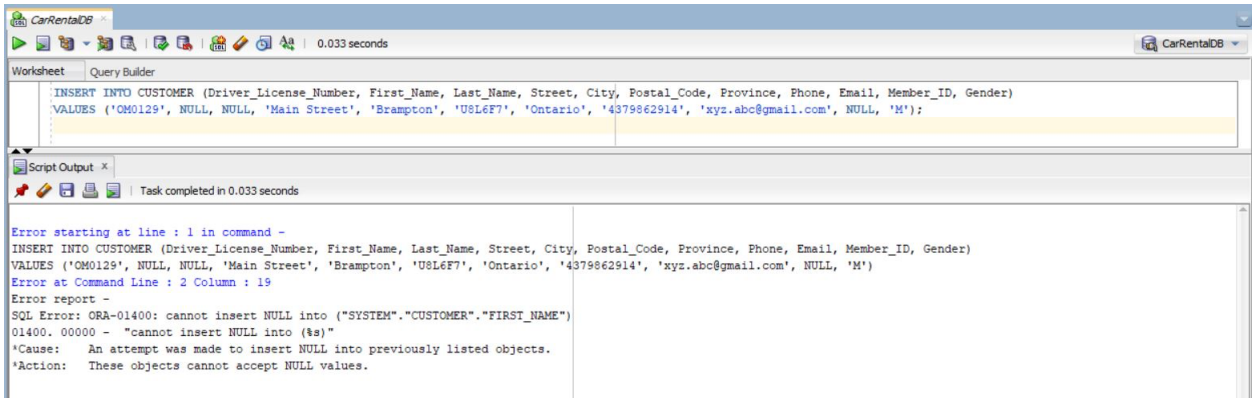


Figure 9: Defining NOT NULL Constraint, Column Level in First_Name and



Last_Name

Figure 10: Error while trying to insert NULL value in First_Name and Last_Name

c. Unique Key Constraint

Unique constraint in the column is used to make that column unique so that no duplicate value can be inserted. But Unique constraint allows NULL value to insert because NULL is considered to be unique. This helps to maintain data integrity by preventing the insertion of duplicate data into the table. Single-column unique constraints and multi-level (composite) unique constraints are the two basic categories of unique constraints.

Following is the example of single column unique constraints. In this example phone field of CUSTOMER table is made unique from column level.

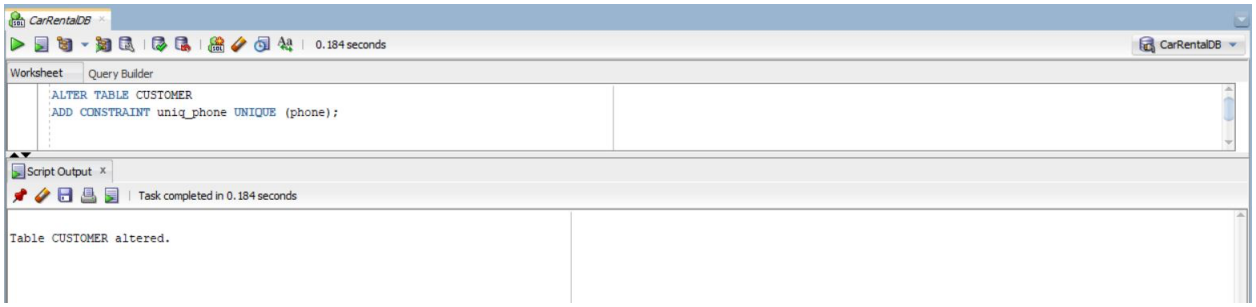


Figure 11: Creating Column Level Single-Column UNIQUE Constraint in Phone

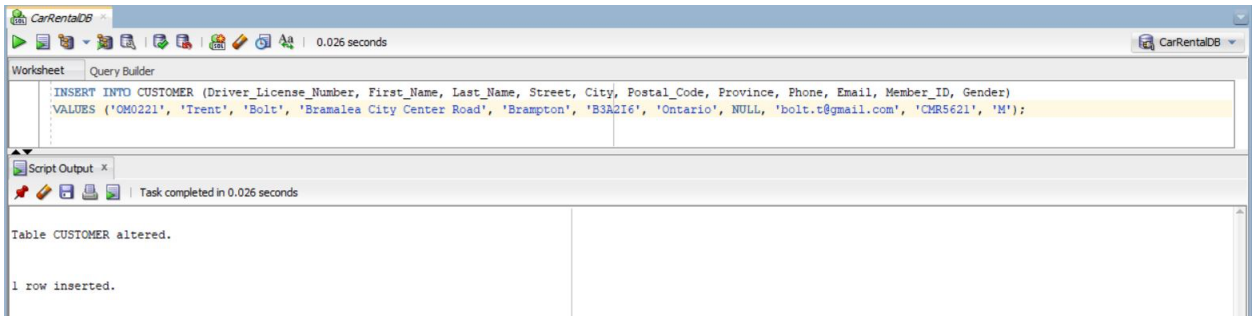


Figure 12: Allows to enter NULL as unique value

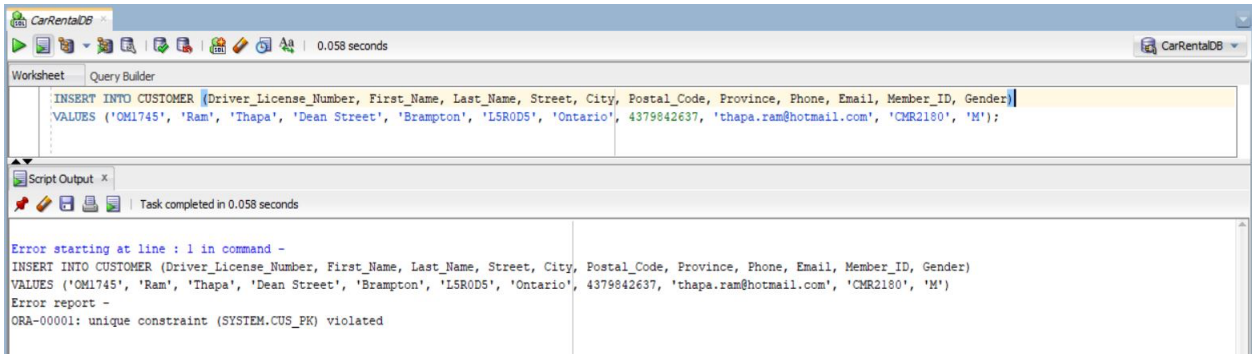


Figure 13: Showing UNIQUE Constraint

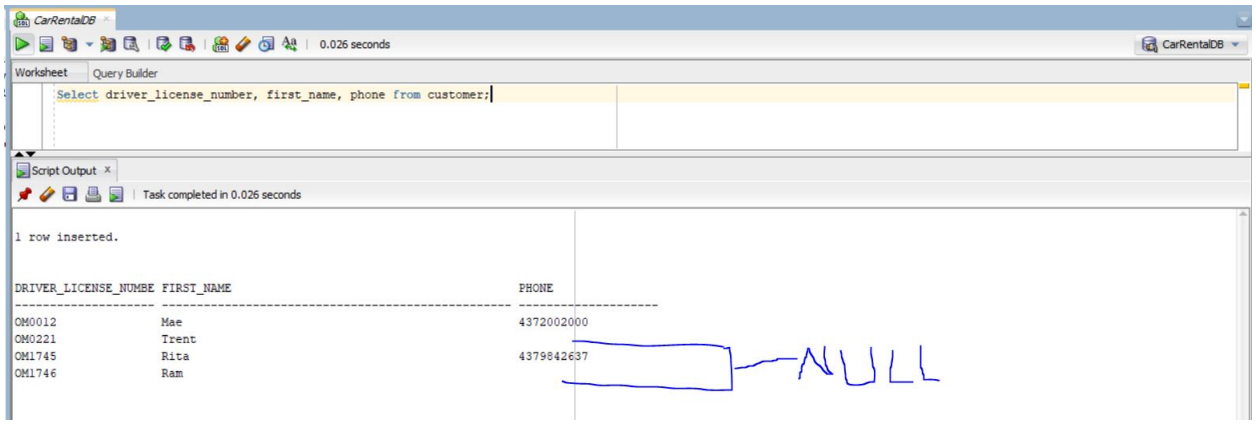


Figure 14: NULL can be unique

Creating multi-column unique constraint in table level for CAR table.

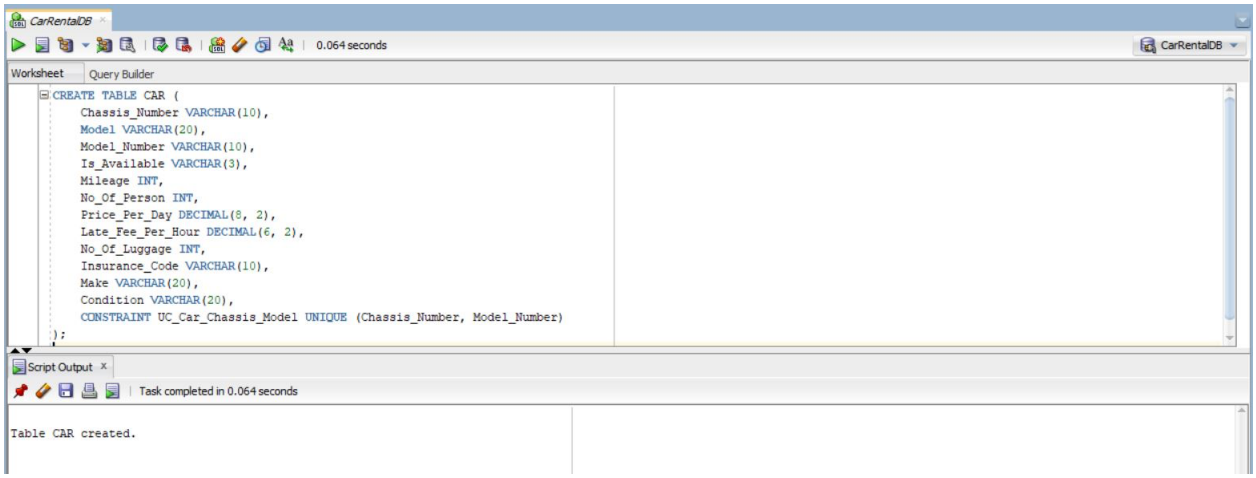


Figure 15: Multi-column constraint in table level for CAR table

The "Chassis_Number" and "Model_Number" columns are combined by the UC_Car_Chassis_Model constraint to guarantee that each combination in the CAR database is distinct. As a result, there cannot be two entries in the table with the identical "Chassis_Number" and "Model_Number" values.

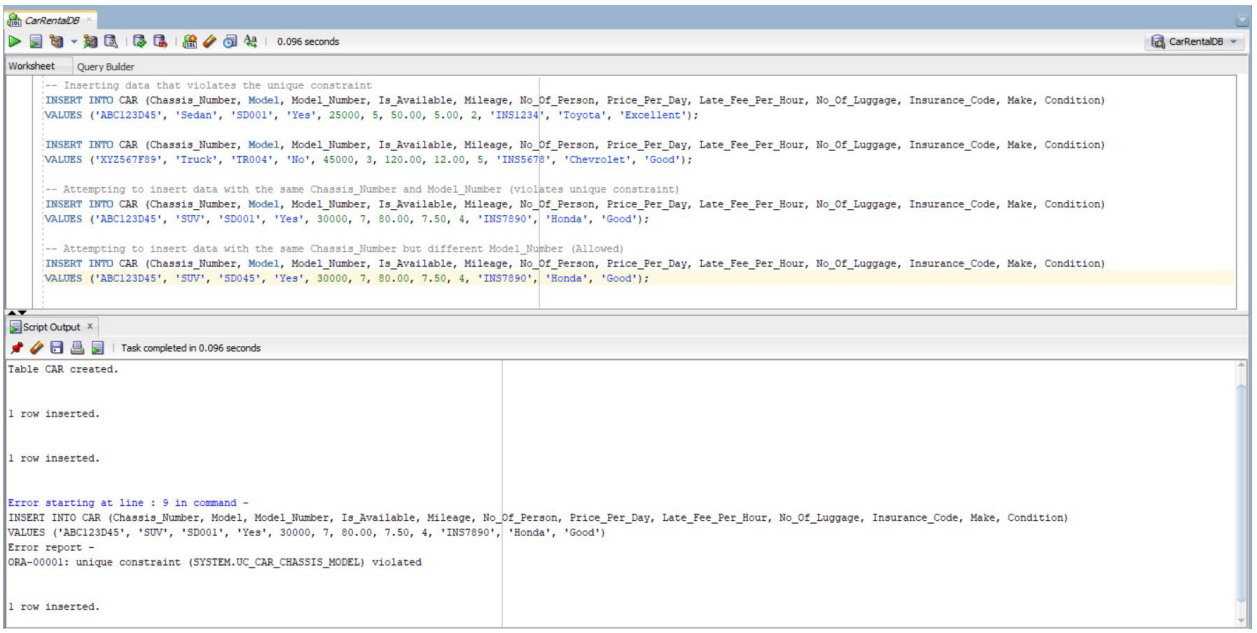


Figure 16: Showing Multi-Column Constraint

In above example of inserting data, in the third INSERT statement, we are attempting to insert data with the same "Chassis_Number" ('ABC123D45') and "Model_Number" ('SD001') as the first row. This would violate the unique constraint we defined on the combination of "Chassis_Number" and "Model_Number," and the database would raise an error, preventing the insertion of the duplicate data. But again in fourth INSERT statement when we insert data with same “Chassis_Number” ('ABC123D45') but different “Model_Number” (‘SUV045) the combination becomes unique and thus it allows to insert data.

We do not use multi-column unique constraint in our car rental project for CAR table this is to show an example. Thus we drop this table and create new CAR table without unique constraint.

d. Default Constraint

When an INSERT statement in SQL Server does not explicitly specify a value for a column, a default constraint is used to provide a default value for that column. It guarantees that the default value will be used if no value is entered during the insert.

If we want to create default constraint in existing table, we use the ALTER TABLE command in SQL. The following shows the table-level default constraint in Location table.

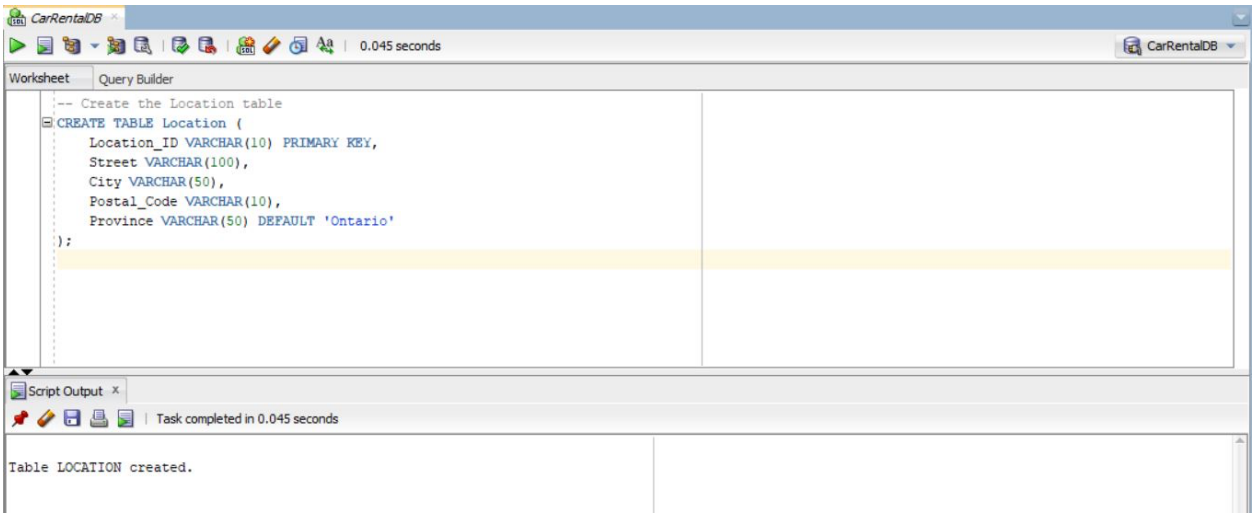


Figure 17: Table-Level Default Constraint in Location Table

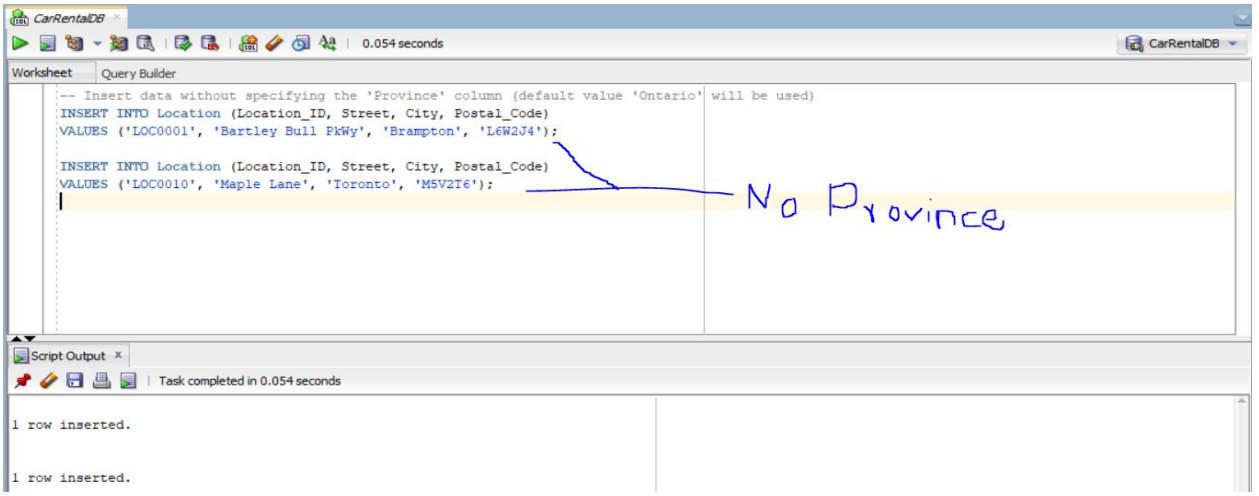


Figure 18: Inserting Data without value provided for Province

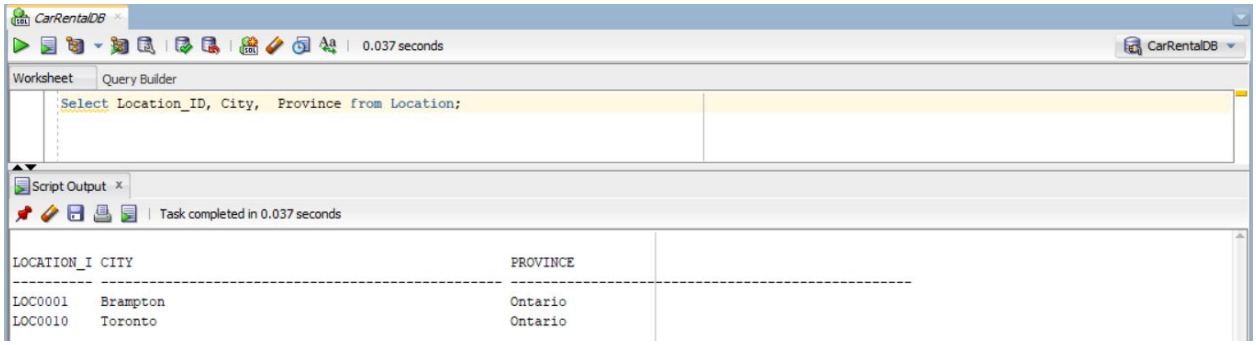


Figure 19: Showing Default Value for Province

e. Referential Integrity Constraint

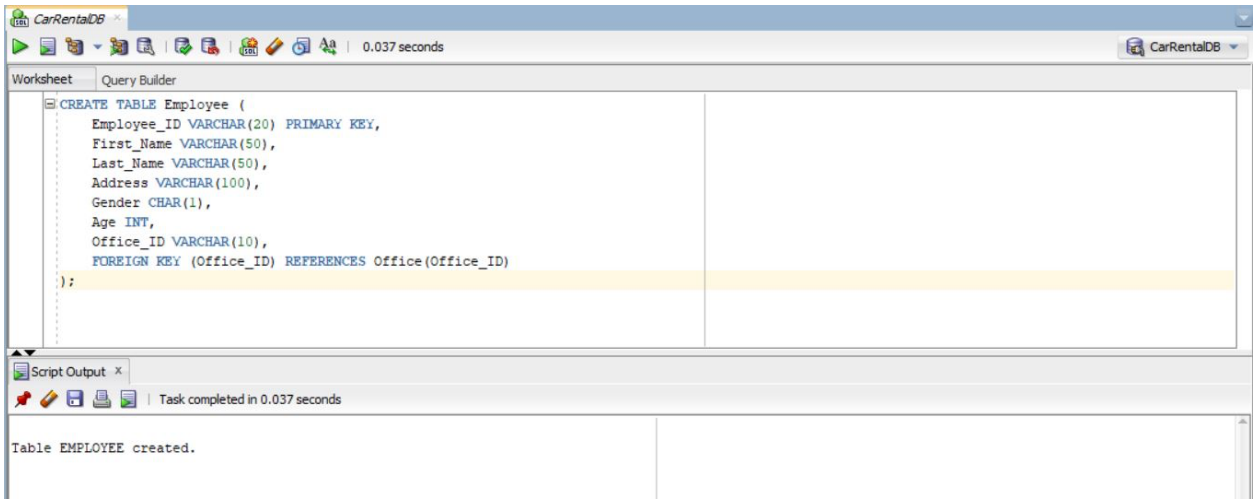


Figure 19: Referential-Integrity Constraint in Employee Table

We already have Office table with the specified columns and also data into it. Then, we create the Employee table with the specified columns and use a FOREIGN KEY constraint to link the "Office_ID" column in the Employee table to the "Office_ID" column in the Office table, establishing the referential integrity relationship.

The FOREIGN KEY constraint ensures that the values in the "Office_ID" column of the Employee table must exist in the "Office_ID" column of the Office table. This constraint maintains data integrity by preventing the insertion of invalid Office IDs in the Employee table.

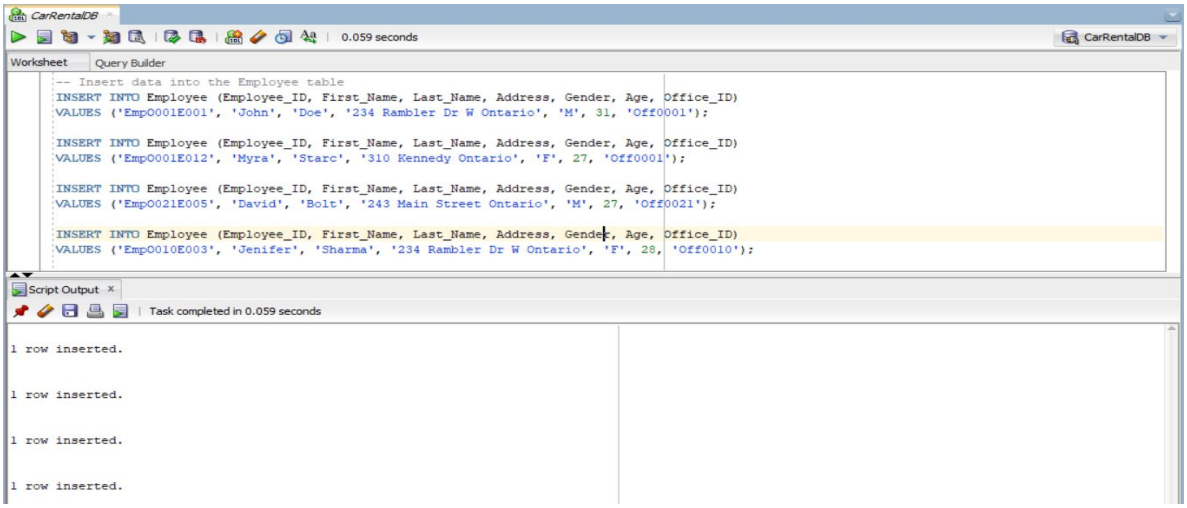


Figure 20: Inserting Value in Employee Table with existing Office_IDs

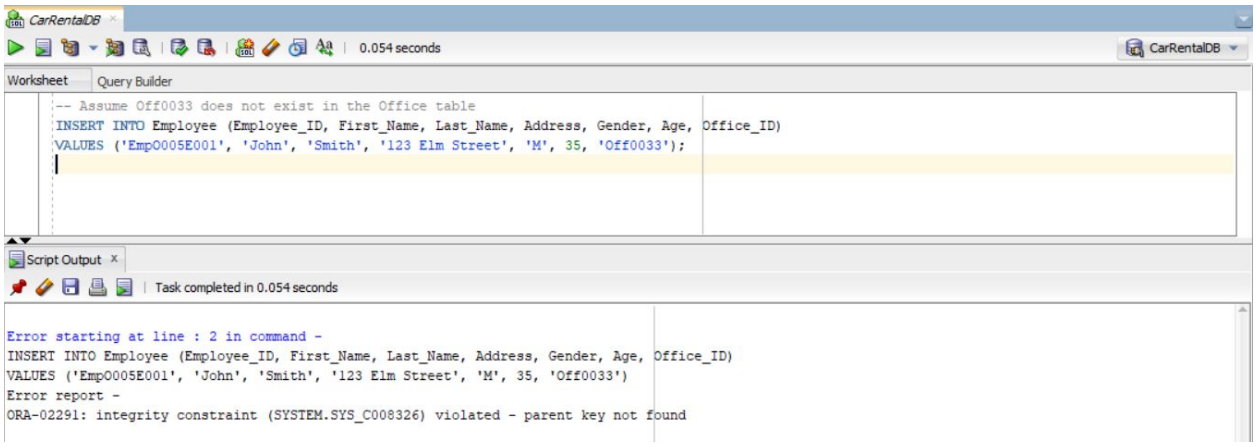


Figure 21: Violation of Referential Integrity Constraint

In Figure 20, we are trying to insert an employee record with "Office_ID" 'Off0033'. However, there is no corresponding 'Off0033' value in the Office table, so the referential integrity constraint will be violated, and the insertion will fail. The database will raise an error shown in the Figure 20 output.

f. CASCADE DELETE Constraint

A cascade delete constraint is a referential integrity constraint that automatically deletes related rows in a child table when the corresponding row in the parent table is deleted. This ensures that data consistency is maintained across the tables.

In this project we do not use cascade delete operation but we show the example using alias tables of Employee and Office.

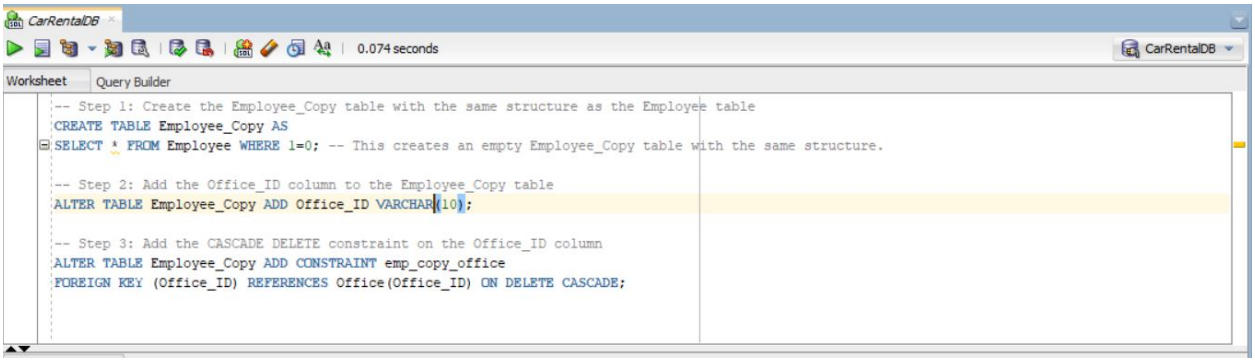


Figure 22: Creating Copy Employee Table to Perform Cascade Delete

Figure 23: Employee_Copy before deleting Office_ID = Off0001

With the cascade delete constraint in place, if you delete an office from the "Office" table, all employees associated with that office will be automatically deleted from the "Employee_Copy" table:

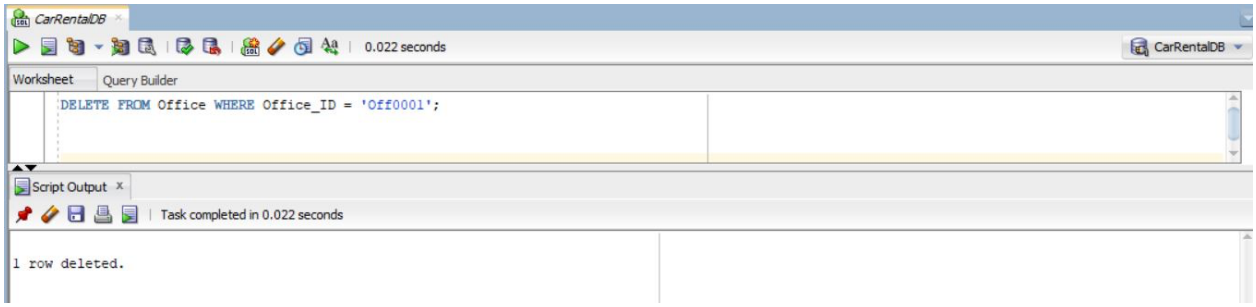


Figure 24: Deleting Office_ID = Off0001

After executing the delete statement, the office with "Office_ID" 'Off0001' ('Asian UHaul') will be deleted from the "Office" table, and all employees associated with that office (John Doe and Myra Starc) will be automatically deleted from the "Employee_Copy" table due to the cascade delete constraint. The "Employee_Copy" table will no longer have any entries related to the deleted office.

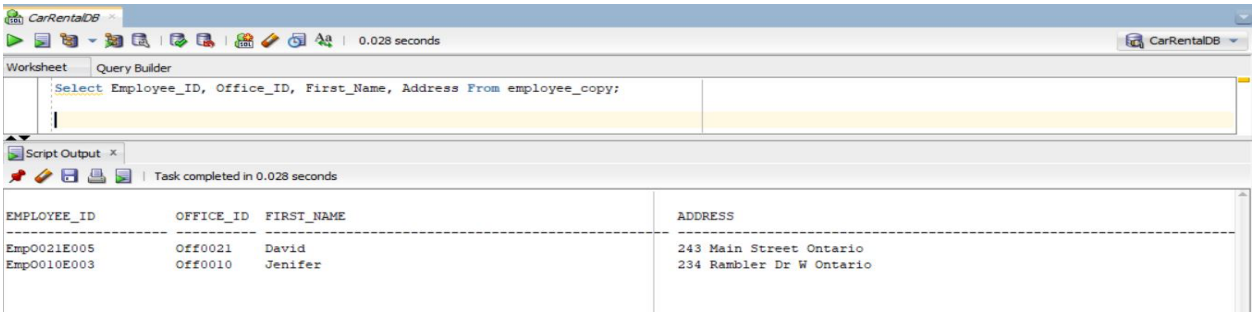


Figure 25: Cascade Delete Operation

3. Physical Database Design

a. Creating a table Office_Copy From Office

```
CREATE TABLE Office_Copy AS
SELECT * FROM Office;
```

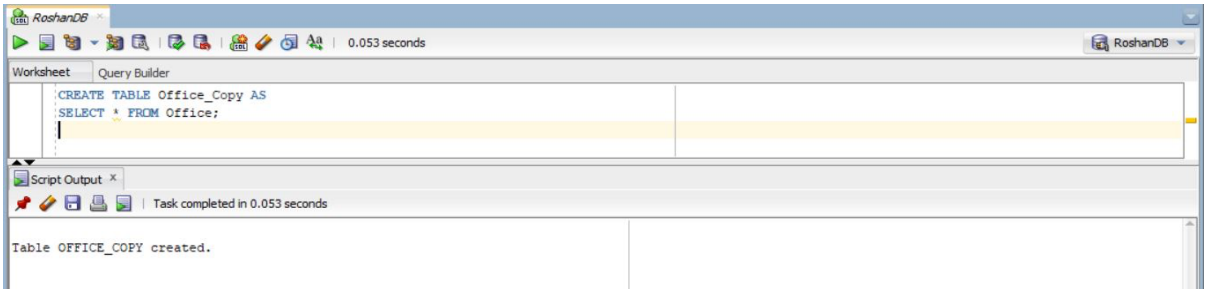


Figure 26: Created Office_Copy Table from Insurance Table

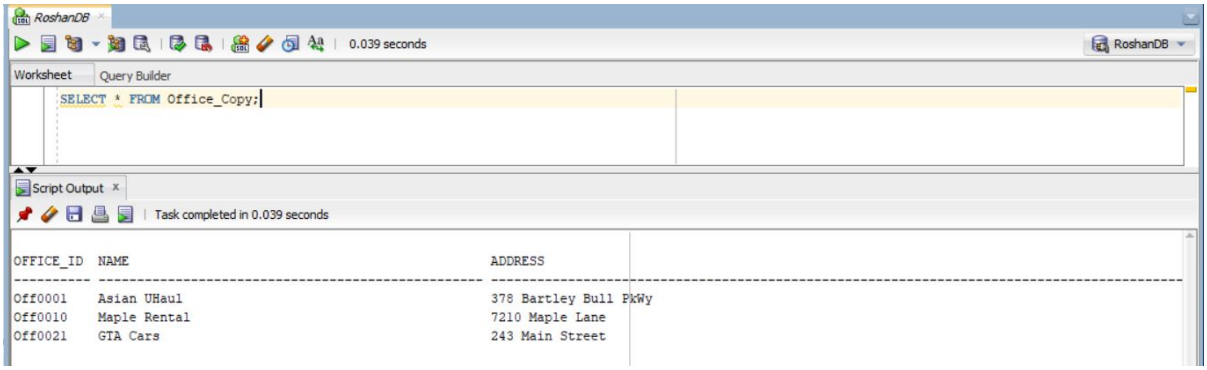


Figure 27: Office_Copy created successfully with same data as Insurance

b. Inserting 5 records to the table using SQL Script

```
INSERT INTO Office_Copy (Office_ID, Name, Address, Postal_Code, Province)
VALUES ('Off0032', 'City Car Rentals', '135 Downtown Ave', 'M4B2T6',
'Ontario');
INSERT INTO Office_Copy (Office_ID, Name, Address, Postal_Code, Province)
VALUES ('Off0067', 'Highway Auto Rentals', '789 Highway Rd', 'L6A3C9',
'Ontario');
INSERT INTO Office_Copy (Office_ID, Name, Address, Postal_Code, Province)
VALUES ('Off0099', 'Urban Wheels', '65 Urban Street', 'N2L4G5', 'Ontario');
INSERT INTO Office_Copy (Office_ID, Name, Address, Postal_Code, Province)
VALUES ('Off0145', 'EasyDrive Rent-A-Car', '1001 Easy Dr', 'K3J2R7',
'Ontario');
INSERT INTO Office_Copy (Office_ID, Name, Address, Postal_Code, Province)
VALUES ('Off0202', 'Fast Lane Rentals', '11 Speedy Blvd', 'H8N3W4', 'Ontario');
```

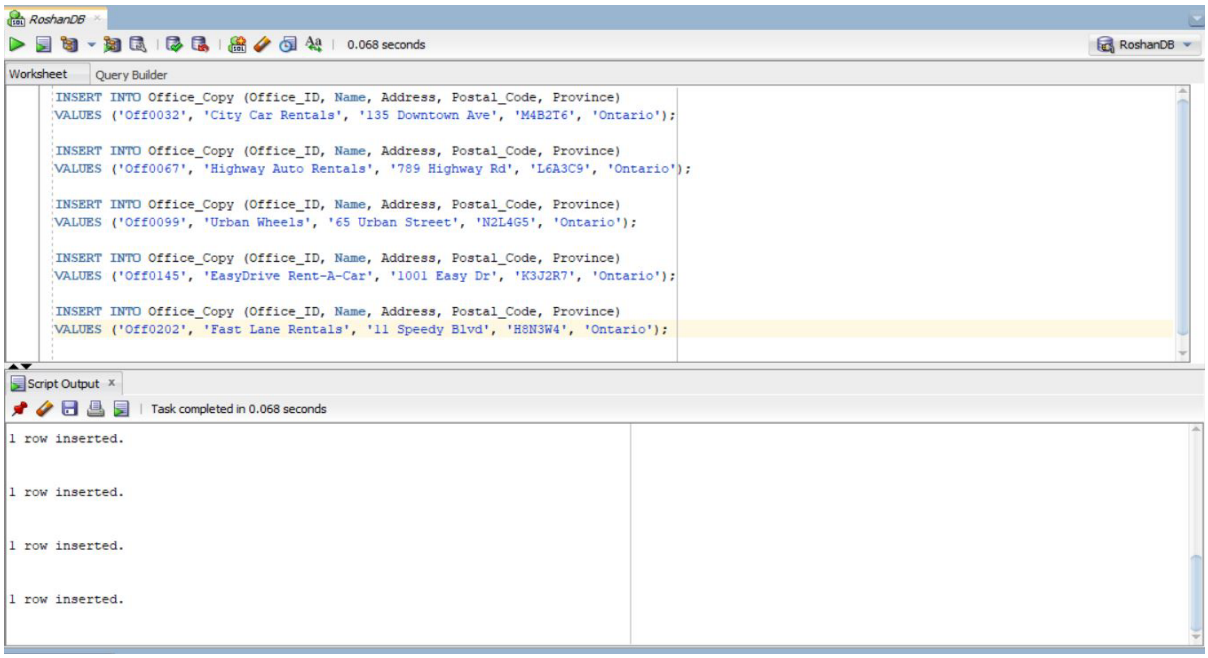


Figure 28: Insert Statement to Insert Data in Office_Copy

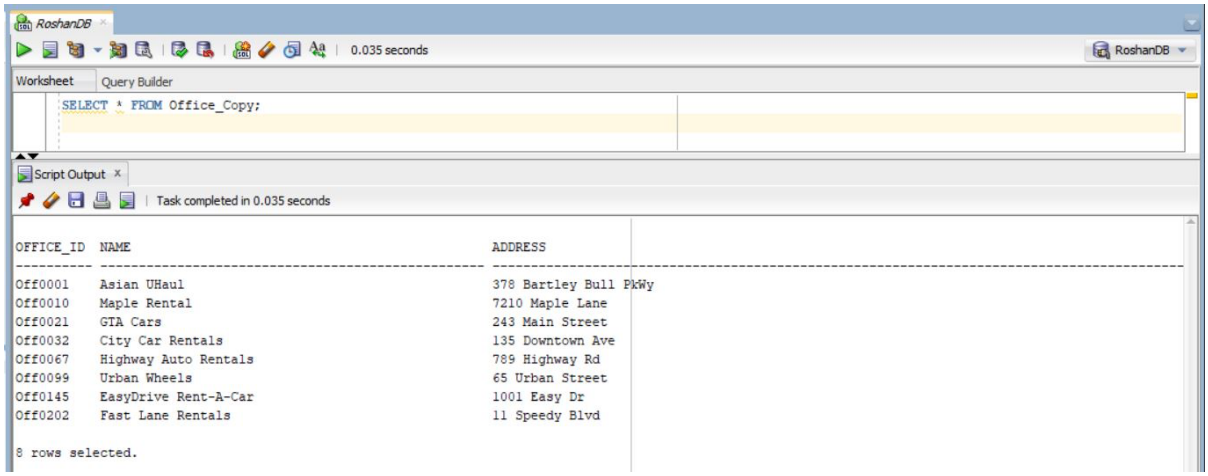


Figure 29: After Insertion Data inserted successfully in Office_Copy Table

c. Updating two records (Using Script)

```
UPDATE Office_Copy
SET Name = 'FastCars Rental', Postal_Code = 'M5V1R3'
WHERE Office_ID = 'Off0032';

UPDATE Office_Copy
SET Name = 'Urban Auto', Postal_Code = 'L3R7K8'
WHERE Office_ID = 'Off0202';
```

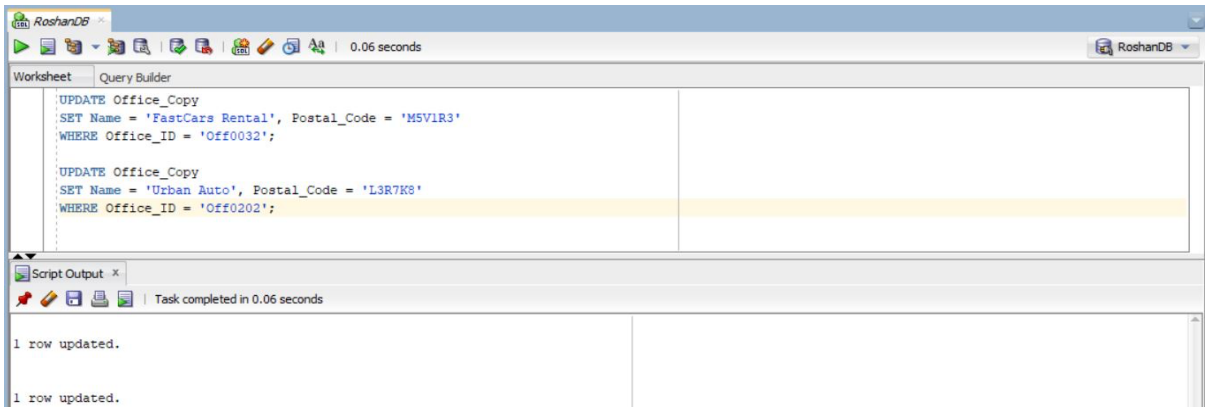


Figure 30: Updated Two data of Office_Copy

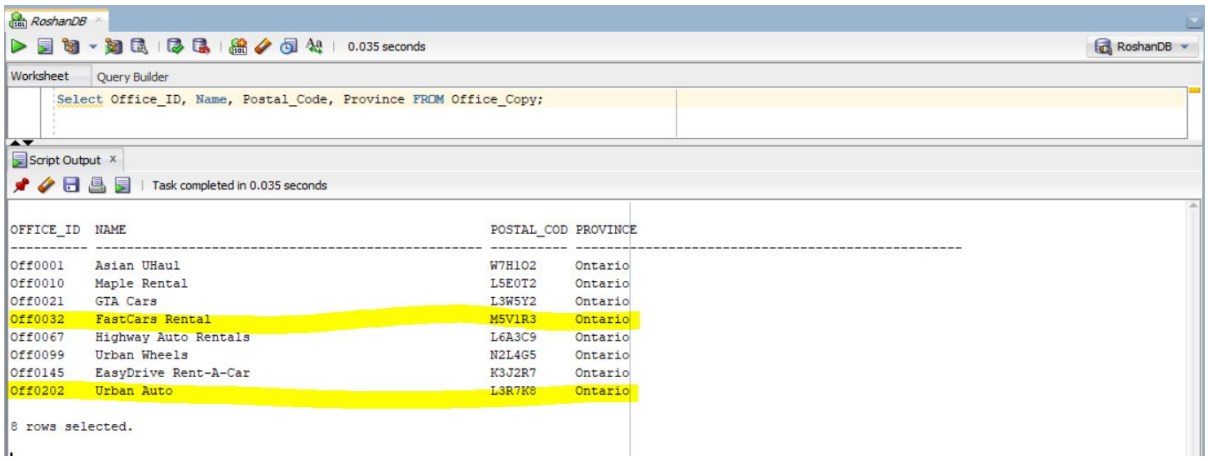


Figure 31: Result of Successfully Update

d. Deleting two records (Using Script)

```
DELETE FROM Office_Copy
WHERE Office_ID = 'Off0067';

DELETE FROM Office_Copy
WHERE Office_ID = 'Off0145';
```

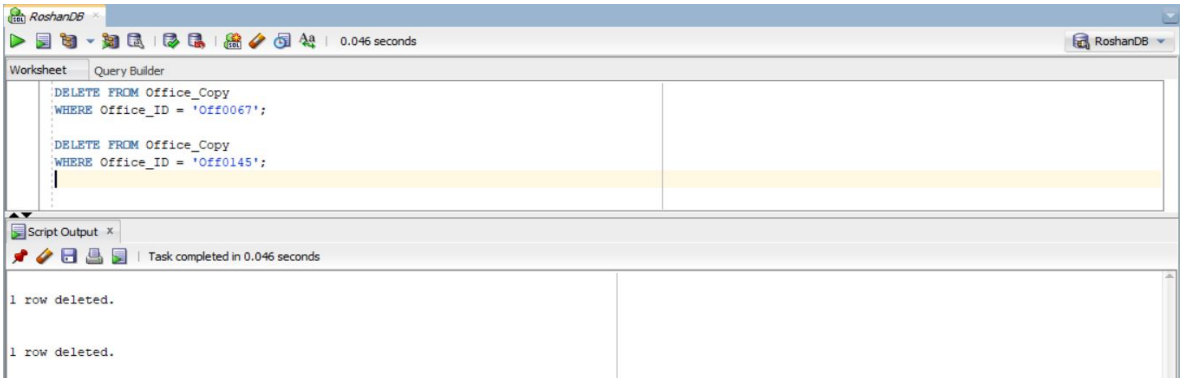


Figure 32: Delete Two Records from Office_Copy Table

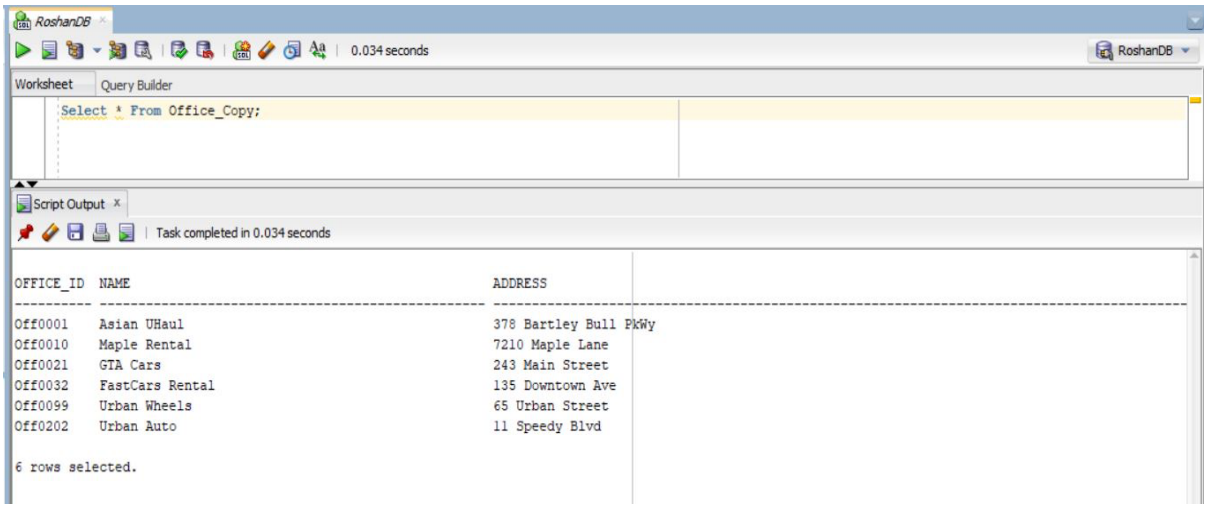


Figure 33: Result After Delete

e. DROP/TRUNCATE the Office_Copy Table

TRUNCATE

TRUNCATE TABLE Office_Copy;

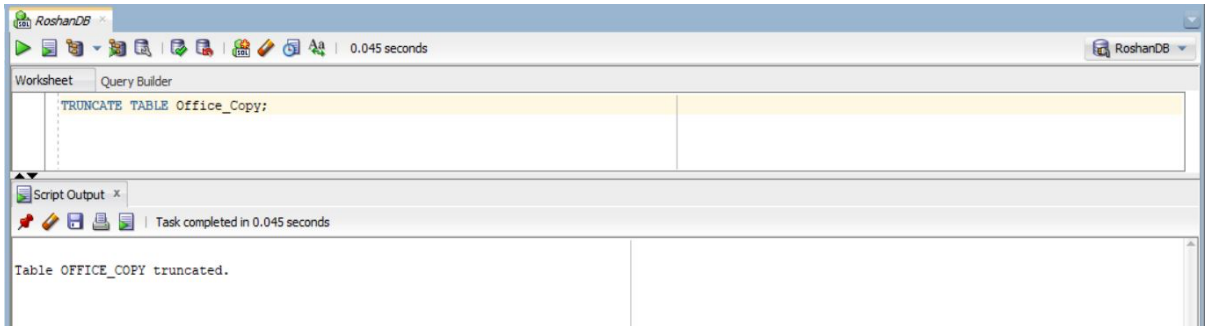


Figure 34: Truncate Table Office_Copy

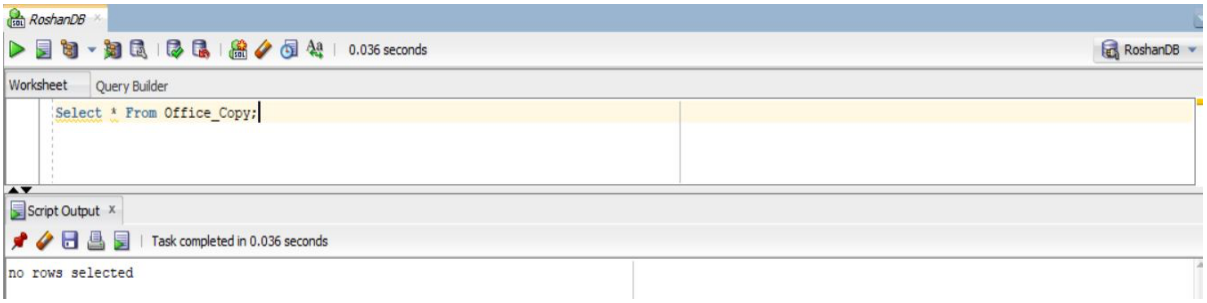


Figure 35: Successfully Truncate Table Office_Copy

DROP

DROP TABLE Office_Copy;

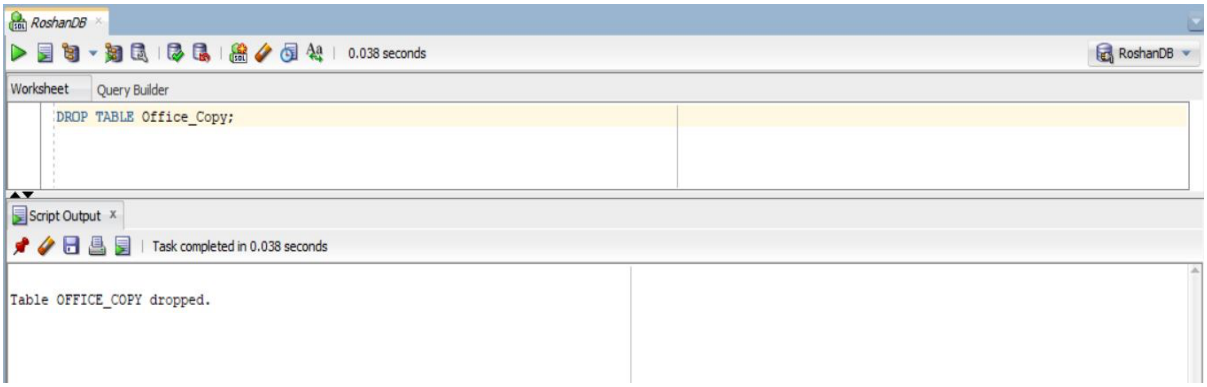


Figure 36: Statement to drop table Office_Copy

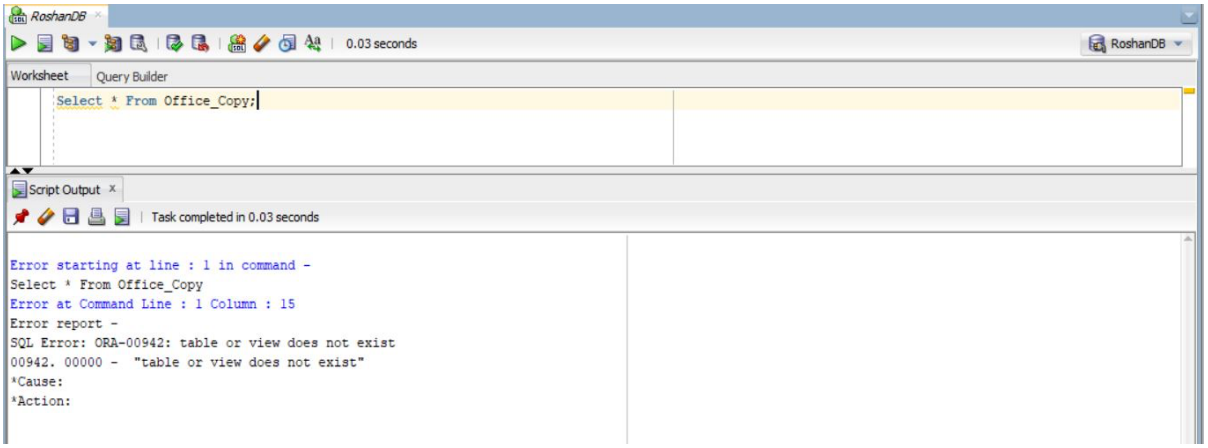


Figure 37: Table Dropped Successfully