DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES1201801580 K. Sreesh Reddy

Write a summary of the example chosen.

The real-world example implemented here is a simple Car Rental System. An admin exists whose responsibility is to update the registrations and update the rents and returns. In this rental a Customer who has been registered in the database can rent a car for any no. of days they want for a certain amount of fee. Cars are also registered in the database by the admin based on their make, model, and availability. Whenever a car has been rented by a customer, its availability status changes from “Yes” to “No” and a due date (date on which car will supposedly be returned) is provided. Customer details include name, address, customer id, phone no. and age. Should the person fail to return the car before or on the due date, they will be charged with a fine. For each day the customer is unable to return the car, they will be fined Rs. 100. A log has also been created for the car registration so that whatever changes are made to the car registration, they are shown in the logs and it also shows the date at which the changes were made so as to make it more secure. The car id for the car and the customer id for the customer are unique so that they can be recognized easily.

Index

Introduction

Database Management is important in any organization. We try to understand here the concepts and schematics of a DBMS to optimize data storage. Here, we use 4 tables. One to store the details of the cars registered, one to store the details of the customers registered, one to store the information of the cars rented. The relational database management system used is MySQL.

Data Model

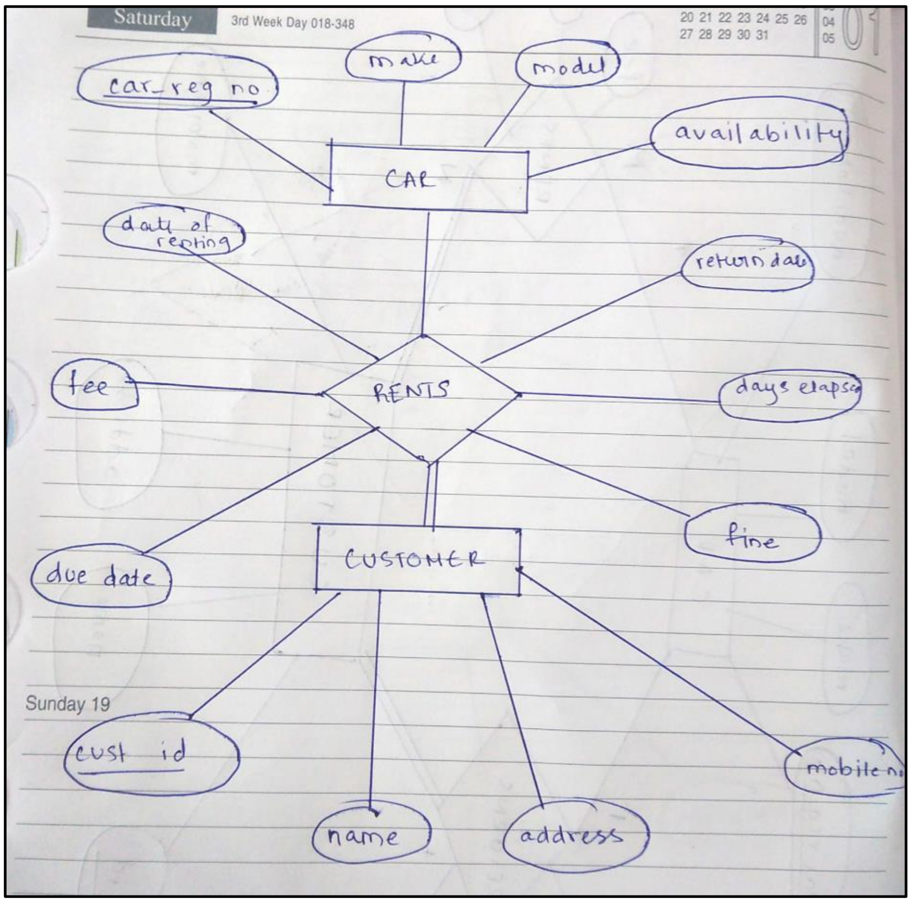


Fig: ER Diagram

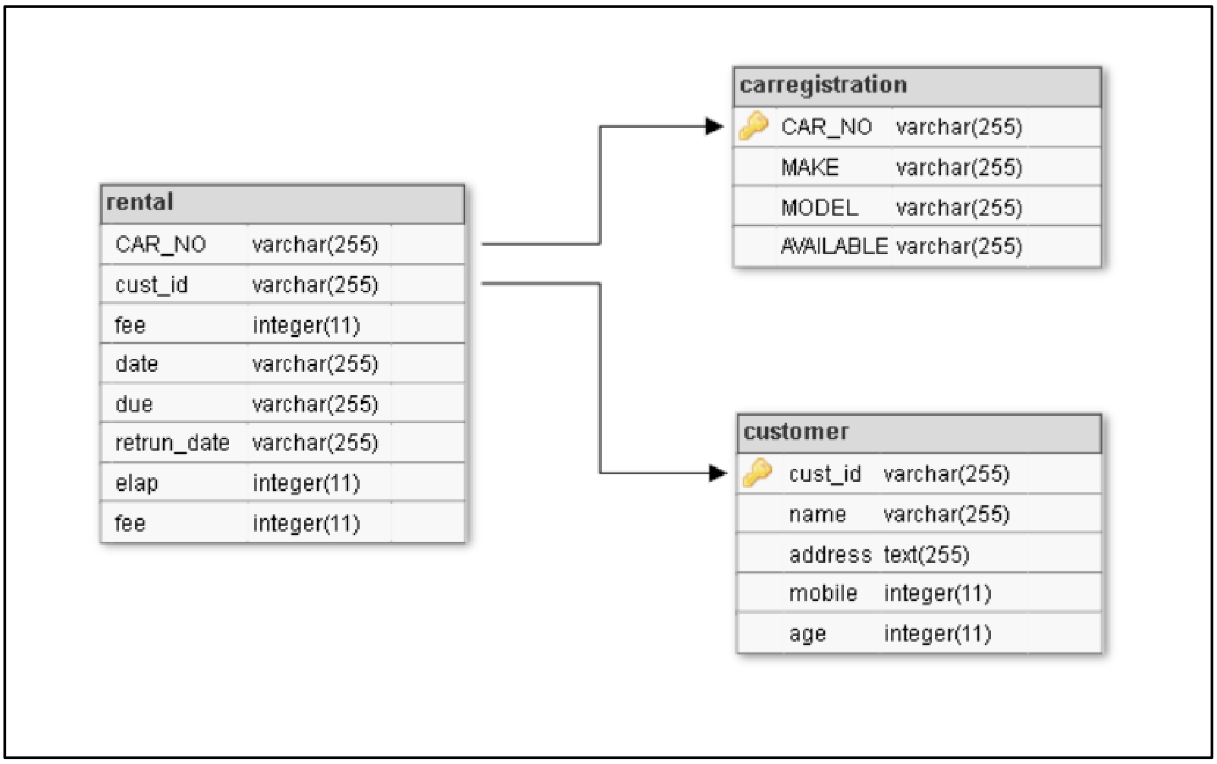


Fig: Schema representation of the ER Diagram

The CAR_NO in rental is a foreign key which references the primary key of the carregistration table which is CAR_NO.

The cust_id in rental is a foreign key which references the primary key of the customer table which is cust_id.

FD and Normalization

a) Functional Dependencies of carregistration:

- {CAR_NO} → {CAR_NO}
- {CAR_NO} → {MODEL}
- {CAR_NO} → {MAKE}
- {MODEL} → {MAKE}
- {CAR_NO} → {AVAILABLE}

b) Functional Dependencies of customer:

- {cust_id} → {age}
- {mobile} → {name}
- {cust_id} → {address, mobile}
- {cust_id} → {name}

c) Functional Dependencies of rental:

- {CAR_NO, cust_id, date} → {CAR_NO, cust_id, fee, date, due, return_date, elap, fine}
  PRIMARY KEY: {CAR_NO, cust_id, date}

⇨ Let us try to identify the key using the functional dependencies of a relation. We take, for example, the carregistration table:

The functional dependencies are:

1) {CAR_NO} → {CAR_NO}
2) {CAR_NO} → {MODEL}
3) {MODEL} → {MAKE}
4) {CAR_NO} → {AVAILABLE}

Using 2) and 3) we get, {CAR_NO} → {MAKE} (transitive dependency)

If we were to write the closure of the CAR_NO attribute, we obtain:

$\{CAR\_NO\}^{+}$ = {CAR_NO, MAKE, MODEL, AVAILABLE}

Which are all the attributes of the relation carregistration. Hence, we can conclude that <u>CAR_NO</u> is the primary key.

Normalization

First Normal Form (1NF): As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Second Normal Form (2NF): A table is said to be in 2NF if both the following conditions hold:
1. Table is in First Normal form.
2. No nonprime attribute is dependent on the proper subset of the candidate key of the table. That means, every nonprime attribute should be fully dependent on the key and not a part of it.

Third Normal Form (3NF): A table is said to be in 3NF if both the following conditions hold:
1. Table should be in Second Normal form.
2. Transitive functional dependency of non-prime attribute on any super key should be removed.

From functional dependencies, we can observe that all rows can have atomic or single values,
Therefore, the schema is in 1NF.

Since it is in first normal form and there is no partial dependency present in any of the relations we can say that the schema is in 2NF.

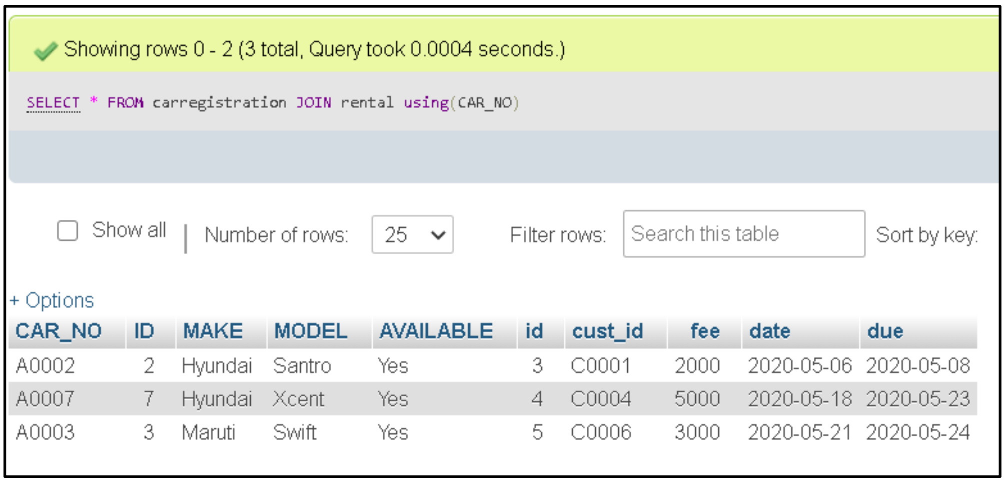
Also, the schema is in 2NF and there is no transitive dependency of non-prime attributes on primary key. They are directly dependent on the key. Hence, the schema is in 3NF.

Testing for lossless join property

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.
Decomposition is lossless if R1 ⋈ R2 = R

There are three conditions for testing lossless join decomposition:
Attributes of R ∪ Attributes of R2 = R
Attributes of R ∩ Attributes of R2 ≠ NULL
Attributes of R1 ∩ Attributes of R2 = key of either R1 or R2

We try testing the lossless joining of carregistration and rental tables:


Showing rows 0 - 2 (3 total, Query took 0.0004 seconds.)

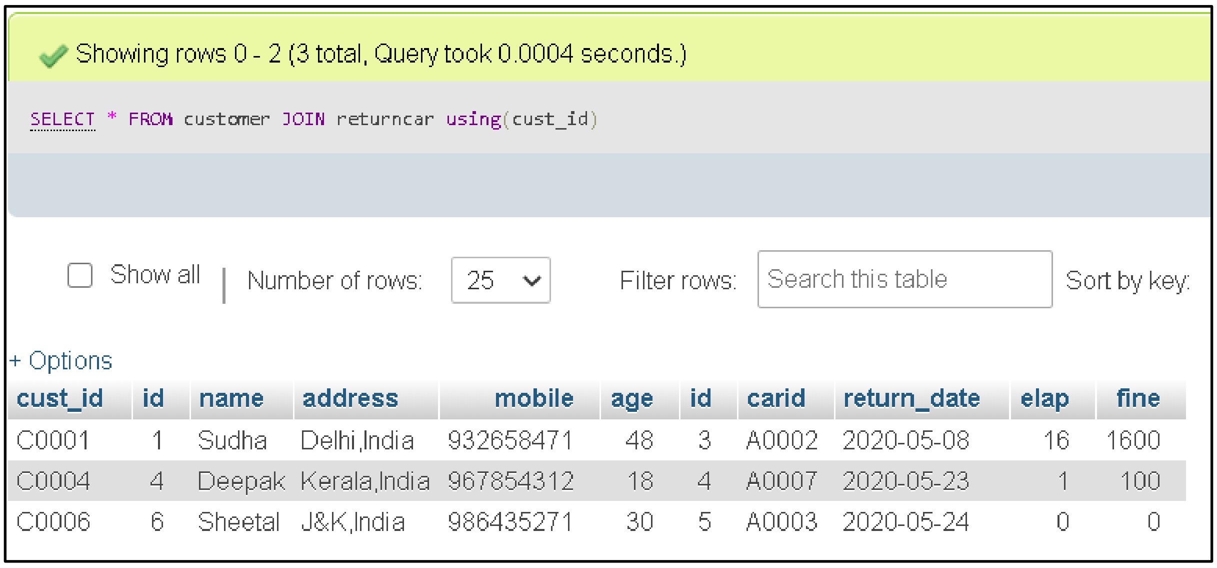
```
SELECT * FROM carregistration JOIN rental using(CAR_NO)
```

CAR_NO	ID	MAKE	MODEL	AVAILABLE	id	cust_id	fee	date	due
A0002	2	Hyundai	Santro	Yes	3	C0001	2000	2020-05-06	2020-05-08
A0007	7	Hyundai	Xcent	Yes	4	C0004	5000	2020-05-18	2020-05-23
A0003	3	Maruti	Swift	Yes	5	C0006	3000	2020-05-21	2020-05-24

We see that it returns all attributes which are present in their union, and their intersection is the CAR_NO which is the primary key of carregistration.

Another test using customer and returncar tables:

It can be observed that their NATURAL JOIN returns all the union of the attributes. The intersection of the attributes is cust_id which is the key of customer. Hence, we can observe that there is no loss in information.


Showing rows 0 - 2 (3 total, Query took 0.0004 seconds.)
Show all
Number of rows: 25
Filter rows: Search this table
Sort by key:
+ Options

```
SELECT * FROM customer JOIN returncar using(cust_id)
```

cust_id	id	name	address	mobile	age	id	carid	return_date	elap	fine
C0001	1	Sudha	Delhi,India	932658471	48	3	A0002	2020-05-08	16	1600
C0004	4	Deepak	Kerala,India	967854312	18	4	A0007	2020-05-23	1	100
C0006	6	Sheetal	J&K,India	986435271	30	5	A0003	2020-05-24	0	0

We use another method to test the lossless join. Using the algorithm

- **Input**: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a set F of functional dependencies.

Here, our D = {carregistration, customer, rental}

Step 1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.

Step 2. Set S(i,j) := b_{ij} for all matrix entries. (* each bij is a distinct symbol associated with indices (i,j) *).

```
Step 3. For each row i representing relation schema $R_i$
  {
        for each column j representing attribute Aj
        {
                if (relation $R_i$ includes attribute $A_j$)
                    then set S(i,j):= $a_j$;
        };
   };
```

- (* each a_j is a distinct symbol associated with index (j) *)

Therefore, according to the algorithm, we have 3 rows (corresponding to carregistration, customer and rental) respectively, and 15 columns (corresponding to CAR_NO, MAKE, MODEL, AVAILABLE, cust_id, name, address, mobile, age, date, due, fee, return_date, elap, fine) respectively.

The matrix is:

$$\begin{bmatrix} a_1 & a_1 & a_1 & a_1 & b_{1,5} & b_{1,6} & b_{1,7} & b_{1,8} & b_{1,9} & b_{1,1}\,0 & b_{1,11} & b_{1,12} & b_{1,13} & b_{1,14} & b_{1,15} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & a_2 & a_2 & a_2 & a_2 & a_2 & b_{2,10} & b_{2,11} & b_{2,12} & b_{2,13} & b_{2,14} & b_{2,15} \\ a_3 & b_{3,2} & b_{3,3} & b_{3,4} & a_3 & b_{3,6} & b_{3,7} & b_{3,8} & b_{3,9} & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 \end{bmatrix}$$

Step 4. Repeat the following loop until a complete loop execution results in no changes to S

```
    {
  for each functional dependency X →Y in F
      {
    for all rows in S which have the same symbols in the columns corresponding to attributes in X
          {
          make the symbols in each column that correspond to an attribute in Y be the same in all these
          rows as follows:
              If any of the rows has an "a" symbol for the column, set the other rows to that same "a" symbol
              in the column.
              If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of
              the rows for the attribute and set the other rows to that same "b" symbol in the column;
          };
      };
    };
```

By using FDs:

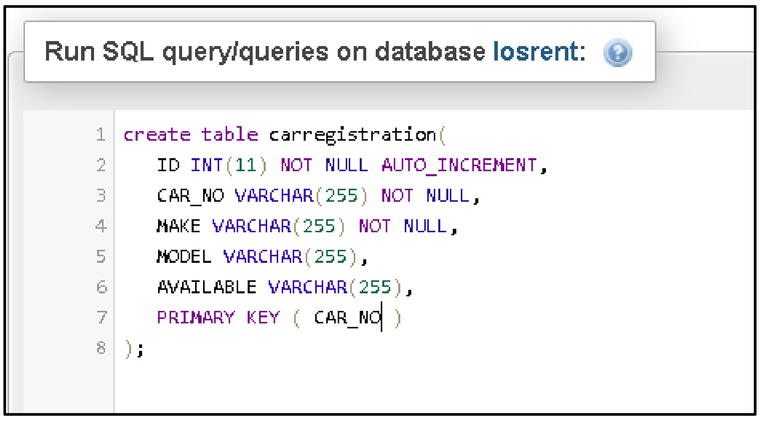
- ➔ {CAR_NO} → {CAR_NO, MAKE, MODEL, AVAILABLE} -------- (carregistration)
- ➔ {cust_id} → {cust_id, name, address, mobile, age} --------- (customer)
- ➔ {CAR_NO, cust_id, date} → {CAR_NO, cust_id, date, due, fee, return_date, elap, fine} --------- (rental)

We get:

$$\begin{bmatrix} a_1 & a_1 & a_1 & a_1 & a_1 & a_1 & a_1 & a_1 & a_1 & b_{1,1}\,0 & b_{1,11} & b_{1,12} & b_{1,13} & b_{1,14} & b_{1,15} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & a_2 & a_2 & a_2 & a_2 & a_2 & b_{2,10} & b_{2,11} & b_{2,12} & b_{2,13} & b_{2,14} & b_{2,15} \\ a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 & a_3 \end{bmatrix}$$

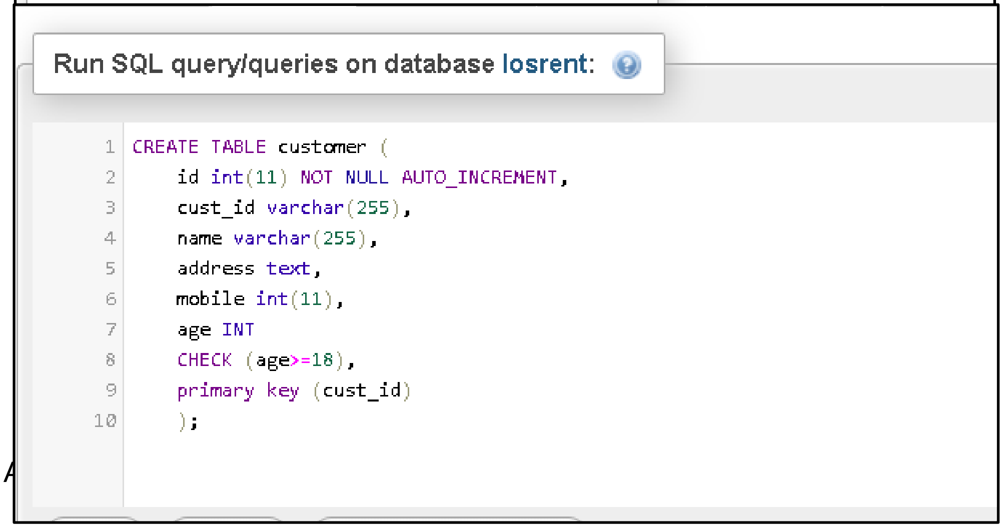
Since, we get the third row completely filled with as, we can conclude that the decomposition has lossless join property.

DDL



Run SQL query/queries on database losrent:

```
create table carregistration(
   ID INT(11) NOT NULL AUTO_INCREMENT,
   CAR_NO VARCHAR(255) NOT NULL,
   MAKE VARCHAR(255) NOT NULL,
   MODEL VARCHAR(255),
   AVAILABLE VARCHAR(255),
   PRIMARY KEY ( CAR_NO )
);
```

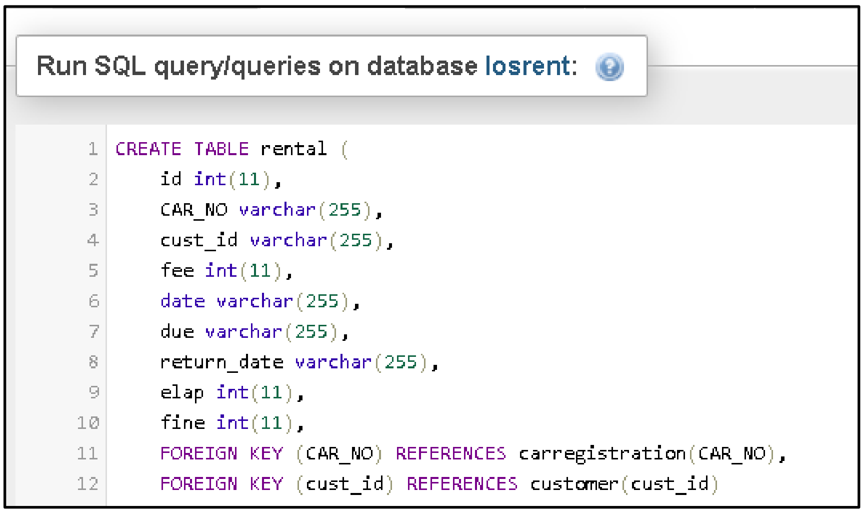


Run SQL query/queries on database losrent:

```
CREATE TABLE customer (
    id int(11) NOT NULL AUTO_INCREMENT,
    cust_id varchar(255),
    name varchar(255),
    address text,
    mobile int(11),
    age INT
    CHECK (age>=18),
    primary key (cust_id)
    );
```

A

2. Creating a table customer to store customer details with a check constraint


Run SQL query/queries on database losrent:

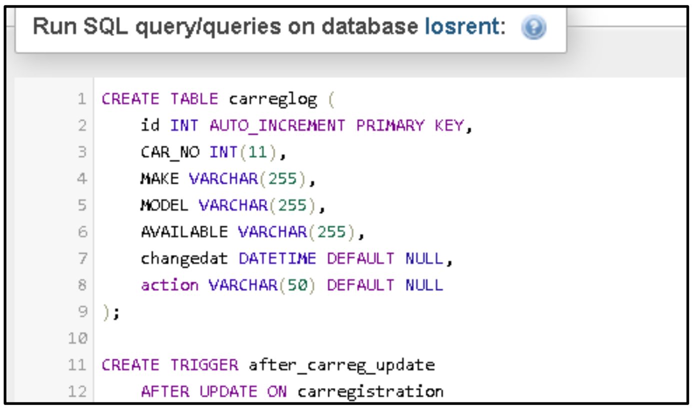
```
CREATE TABLE rental (
    id int(11),
    CAR_NO varchar(255),
    cust_id varchar(255),
    fee int(11),
    date varchar(255),
    due varchar(255),
    return_date varchar(255),
    elap int(11),
    fine int(11),
    FOREIGN KEY (CAR_NO) REFERENCES carregistration(CAR_NO),
    FOREIGN KEY (cust_id) REFERENCES customer(cust_id)
```

3. Creating a table rental with referential integrity constraints

Triggers

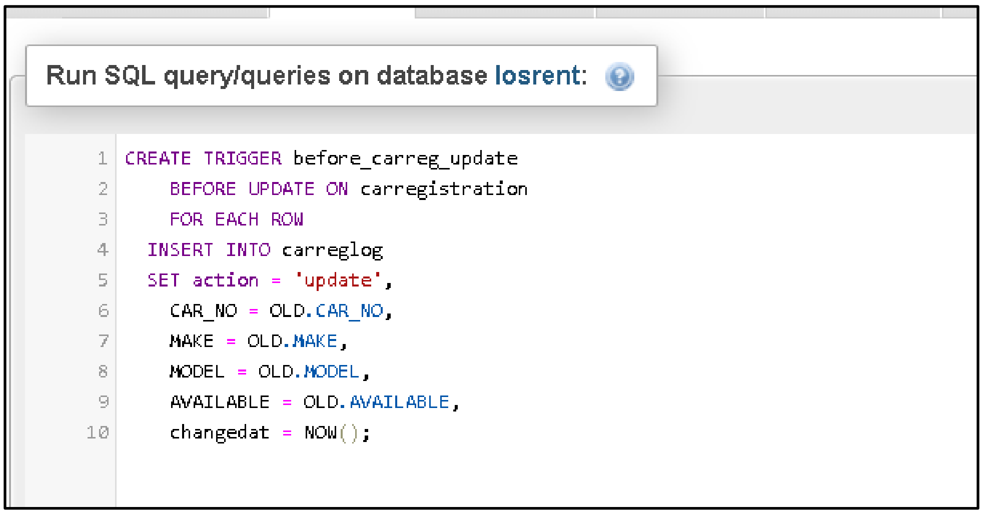
We create a trigger in MySQL to log the changes done in the cars table.

First, we create a table carreglog to keep the changes of the carregistration table.


Run SQL query/queries on database losrent:

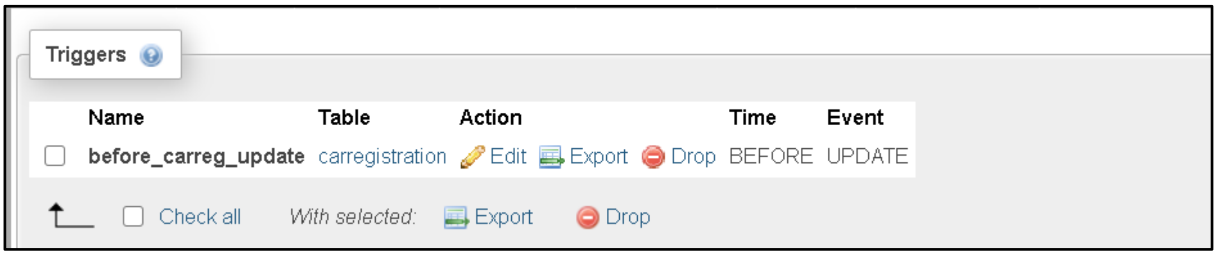
```
CREATE TABLE carreglog (
    id INT AUTO_INCREMENT PRIMARY KEY,
    CAR_NO INT(11),
    MAKE VARCHAR(255),
    MODEL VARCHAR(255),
    AVAILABLE VARCHAR(255),
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);

CREATE TRIGGER after_carreg_update
    AFTER UPDATE ON carregistration
```

Next, create a BEFORE UPDATE trigger that is invoked before a change is made to the carregistration table. Inside the body of the trigger, we used the OLD keyword to access values of the columns CAR_NO, MAKE, MODEL, AVAILABLE of the row affected by the trigger. Trigger is used for security so that all the changes and time of change are being recorded.



```
CREATE TRIGGER before_carreg_update
    BEFORE UPDATE ON carregistration
    FOR EACH ROW
  INSERT INTO carreglog
  SET action = 'update',
    CAR_NO = OLD.CAR_NO,
    MAKE = OLD.MAKE,
    MODEL = OLD.MODEL,
    AVAILABLE = OLD.AVAILABLE,
    changedat = NOW();
```

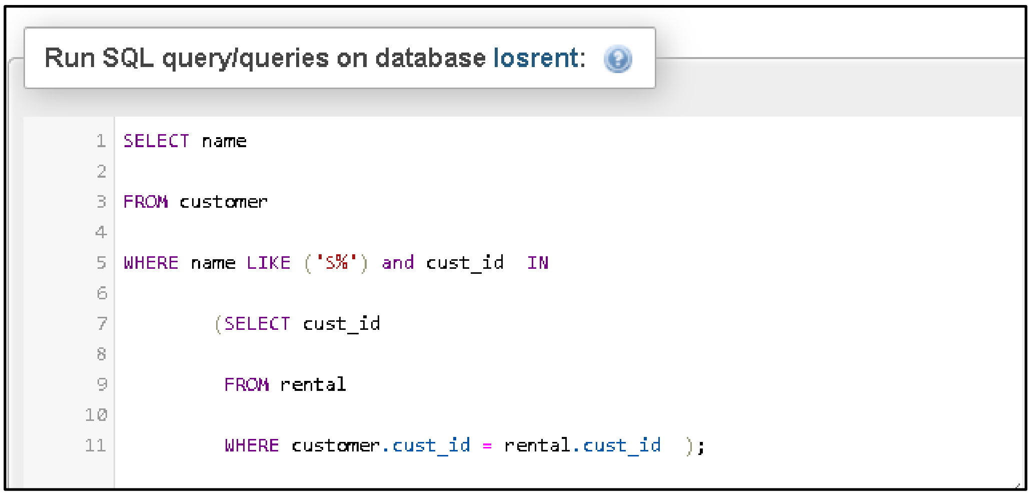
Then, show all triggers in the current database by using the SHOW TRIGGERS statement.



Name	Table	Action	Time	Event
before_carreg_update	carregistration	Edit Export Drop	BEFORE	UPDATE

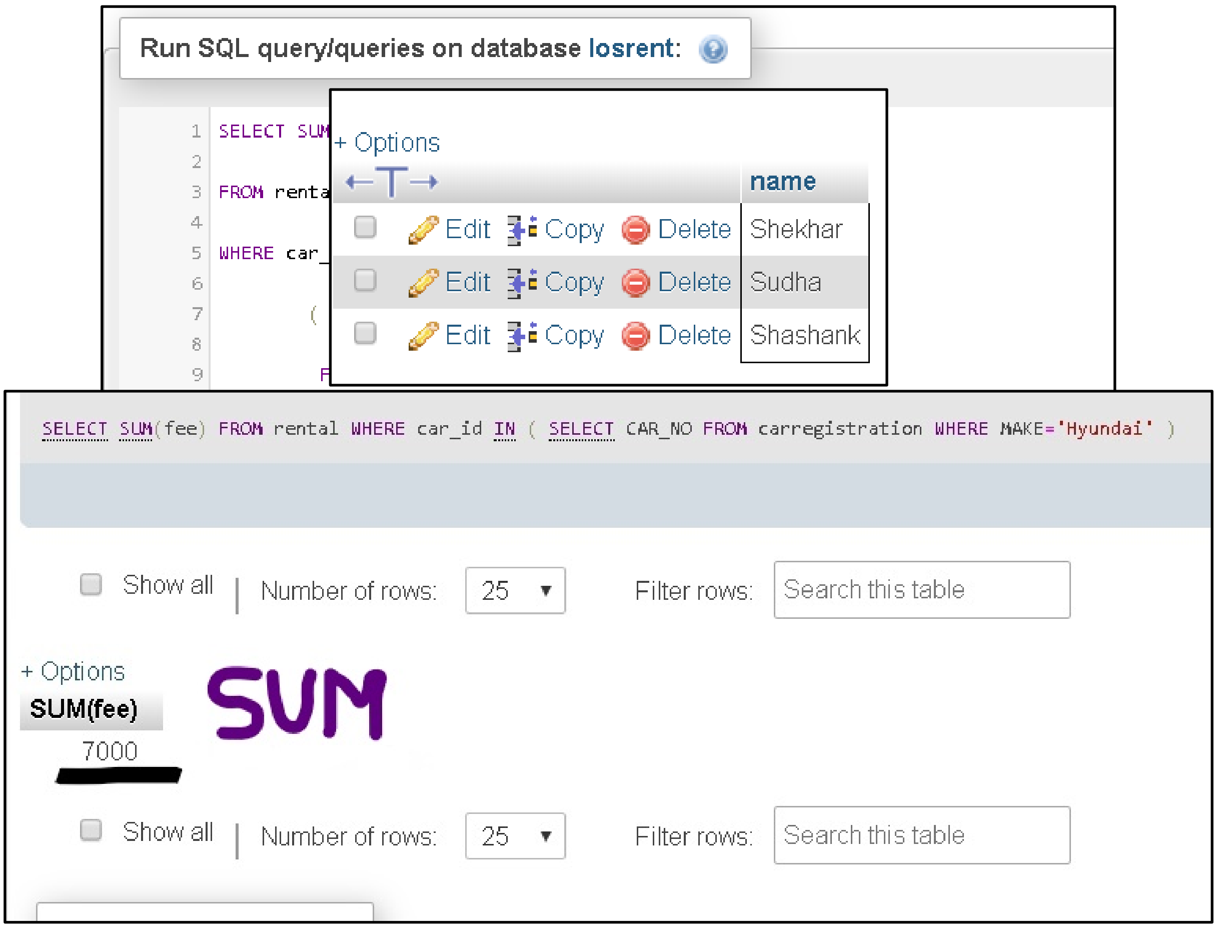
SQL Queries

1. In this query, the names of the customer who have returned their cars and have names starting with S. We use the customer id present in rental and compare those ids with the ids present in the customer table and then obtain the names from those.



```
SELECT name

FROM customer

WHERE name LIKE ('S%') and cust_id  IN

        (SELECT cust_id

         FROM rental

         WHERE customer.cust_id = rental.cust_id  );
```

2. In this query, we find out how much total revenue is made by Hyundai only on the basis of fee specified by the rental company. We use the car id from the car table to identify which ones are manufactured by Hyundai and then using these car ids we get the corresponding fee in the rental table. We finally use the aggregate function SUM () to find out the total.



Run SQL query/queries on database losrent:

```
SELECT SUM

FROM renta

WHERE car_

        (

         F
```

+ Options

	name
☐ Edit Copy Delete	Shekhar
☐ Edit Copy Delete	Sudha
☐ Edit Copy Delete	Shashank

```
SELECT SUM(fee) FROM rental WHERE car_id IN ( SELECT CAR_NO FROM carregistration WHERE MAKE='Hyundai' )
```

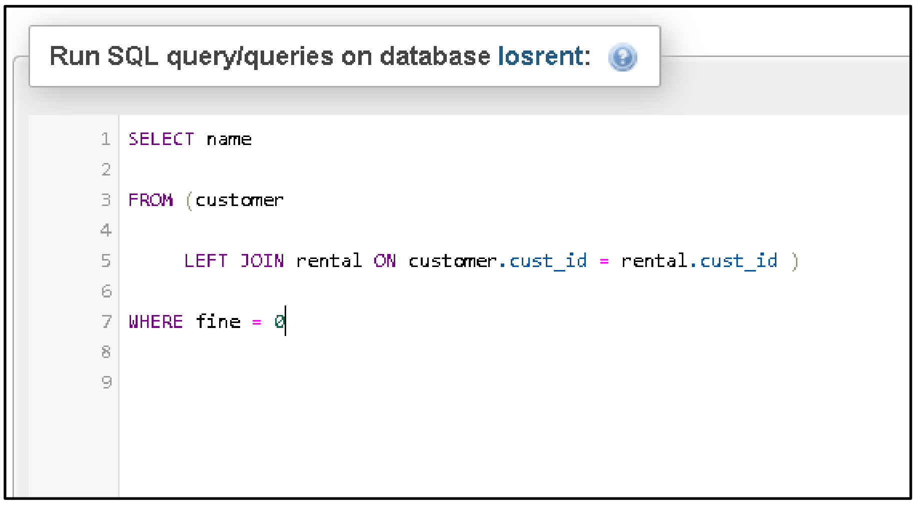
☐ Show all | Number of rows: 25 ▾ Filter rows: Search this table

+ Options

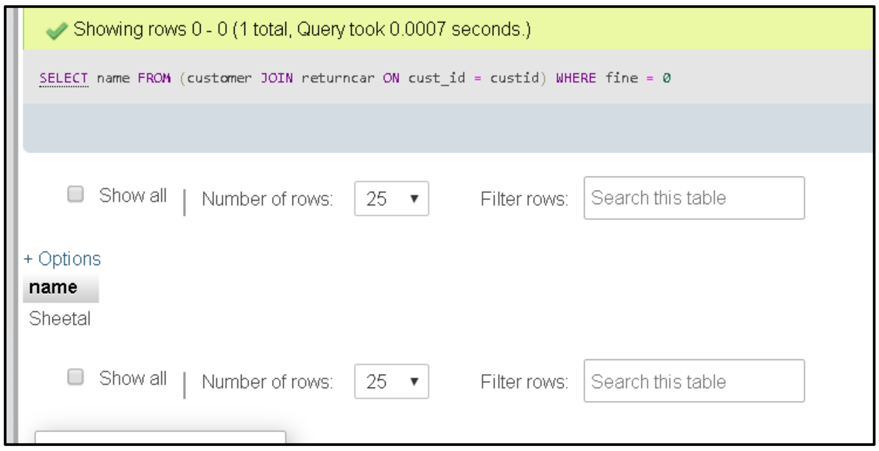
SUM(fee)
7000

☐ Show all | Number of rows: 25 ▾ Filter rows: Search this table

3. In this query, we obtain the names of the customers who haven’t had to pay the fine, i.e., they have returned the car before or on the due date. We use the JOIN function here to join the two tables customer and rental, so that we can use all attributes of both of them. Then we compare the customer id in the customer table to the customer id in the rental table and check where the fine is equal to 0 and return the name.


Run SQL query/queries on database losrent:

```
SELECT name

FROM (customer

     LEFT JOIN rental ON customer.cust_id = rental.cust_id )

WHERE fine = 0
```


Showing rows 0 - 0 (1 total, Query took 0.0007 seconds.)

```
SELECT name FROM (customer JOIN returncar ON cust_id = custid) WHERE fine = 0
```

name
Sheetal

Conclusion

In this project, we dealt with the implementation of a real-world example onto a database management system. We also learnt how to create a database from the ground up and how to normalize it. We encountered the usage of a trigger and writing complex SQL queries to retrieve information. Although it represents a basic structure and functioning of a car rental system, it is very primitive and trivial. The information collected is certainly not equivalent to the vast amount of information a real-world rental system would contain.