PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

Department of Computer Science and Engineering
Jan – May 2020

UE18CS252
Database Management Systems

Project Report

CAR_RENTAL DATABASE

SUDARSHAN R | PES1201801475
4th Sem Sec. A | Roll 38

1)Problem statement:

A Car_rental company which rents cars on hourly/daily basis .The company has many rental locations and it stores location_id and zip code etc.

The company stores its customer/user details like name ,address,phone_no and etc.. The company has a wide range of car collections based on the car type like for example it can be suv,xuv,sedan or minivan and each car has a unique vehicle identification number(VIN) and all of the car details describing the car are stored in the database.

And the user in advance has to pay a minimum insurance cost in case of accidents which might occur(mandatory),the user can reserve the required car (if available) in advance by booking in an online portal mentioning the date and time of requirement

There is an option to get an additional driver if required by the user but it will be cost accordingly and even additional accessories like GPS system or baby seat can also be provided.

And the company also provides a few discounts for the customers by giving some coupons which they can use during their next booking.

And the most important part is the payment option,it could be done by cash/card,if the user likes to pay by card then his card details like card_no ,cvv will also be stored in the company database.

PROJECT SUMMARY

2)Introduction

A collected information which is in an organized form for easier access, management, and various updating is known as a **database**.

Miniworld Description:
There are around 14 entities and several attributes describing them and description of the same is given here.

The Car_rental company which rents cars on hourly/daily basis .The company has many rental locations and it stores location_id and zip code etc.

The company stores its customer/user details like name ,address,phone_no and etc.. The company has a wide range of car collections based on the car type like for example it can be suv,xuv,sedan or minivan and each car has a unique vehicle identification number(VIN) and all of the car details describing the car are stored in the database.

And the user in advance has to pay a minimum insurance cost in case of accidents which might occur(mandatory),the user can reserve the required car (if available) in advance by booking in an online portal mentioning the date and time of requirement

There is an option to get an additional driver if required by the user but it will be cost accordingly and even additional accessories like GPS system or baby seat can also be provided.

And the company also provides a few discounts for the customers by giving some coupons which they can use during their next booking.

And the most important part is the payment option,it could be done by cash/card,if the user likes to pay by card then his card details like card_no ,cvv will also be stored in the company database.

Click to view the ER Diagram

Click to view the SCHEMA

3)Data Model:

Both Entity-Relational model and Relational model have been done for the database with appropriate relationships connecting the entities and unique primary key to identify each table.

Candidate keys:

To find the candidate key, the attribute closure of all the subsets of the relation is looked at. From this, the **super key** is determined. Super keys are those where the attribute closure contains all the attributes of the relation. Candidate keys are then identified by checking if it is unique with respect to each row of a table.A **minimal super-key gives the Primary Key.**

By following the above-mentioned procedure, we get, the keys as follows,

Relation name	**Primary Key/Candidate key**	**Data Type**
Rental_Location	Rental_Location_ID	integer
Car_Insurance	Car_type,Insurancetype	character,character
Car_type	Car_type	character
Insurance	Insurance no	character
Car	VIN,Reg_no	character
Car_User	License_no	character
User_Credentials	Login_id	character
Card_Details	Card_no	character
Reservation	Reservation_id	integer
Payment	Payment_id	integer
Offer_details	Promo_code	character
Additional_Driver	Name(partial key)	character
Accessories	Accessory_id	Integer
Accessory_reserved	Accessoryid,Reservation Id	integer,integer

4)FD and Normalization

i)FUNCTIONAL DEPENDENCIES(Based on Semantic Constraints):

A functional dependency is a constraint between two sets of attributes from the database .
A functional dependency, denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].

Relation name	X-->Y
CAR	{VIN,Regno}-->{Seat Capacity,Car type,Model ,color}
Car_User	{Phone,Licence_no}-->{Email,Fname,Lname}
Rental_Location	{Zip Code,Phone}-->StreetName
Card_details	{Card_no}-->Name-on-card
Reservation	reservation_id→{Lic_no,VIN,drop location}
Payment	payment_id→{Card_no,Name-on-card}
Offer_details	promo code-->Description,Discount_amt
Accessory	accessory_id-->Type

ii)Normalization:

The normalization process, as first proposed by Codd (1972), takes a relation schema through a series of tests to certify whether it satisfies a certain **normal form**

Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of
(1) Minimizing redundancy and
(2) Minimizing the insertion, deletion, and update anomalies .

1-Normal Form : It was defined to **disallow multivalued attributes, composite attributes, and their combinations.** It states that the domain of an attribute must include **only atomic** (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
In other words, **1NF disallows relations within relations or relations as attribute values within tuples**. The only attribute values permitted by 1NF are single atomic (or indivisible) values.
Example.

Rental_location

Location_id	phone	email	Address(State,Street,Zip Code)

Here Address attribute is a Composite attribute,therefore the relation is not in Normal Form,thus we decompose the relation as,

A

Location_id	phone	email	

B

Location id	Address(State,Street,Zip Code)

2-Normal Form: Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold anymore; that is, for any attribute A ε X, (X − {A}) does not functionally determine Y.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the **primary key contains a single attribute, the test need not be applied at all.**

Example
Card_details

Login id	Card no	name_on_ card	Expiry date	CVV	Billing Address

Here,{login_id,Card_no}-->Billing address but

{card_no}-->name_on_card,CVV,Expiry_date
thus it is partially dependent and not in 2NF

A

Login id	Card no	Billing Address

B

Card no	name_on_card	Expiry date	CVV

Note:2NF can be violated only for those relation having more than 1 candidate key by **inserting a column which does depend on only one of the candidate key**,thus partial dependency violating 2NF

3-Normal-form: Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency X → Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X → Z and Z → Y hold.

Example:Looking at the previous example of Card_detail

B

Card no	name_on_card	Expiry date	CVV

card_no-->CVV and CVV-->Expiry_date
Here card_no is PK and determines non-prime value but CVV is non-prime determining another non-prime value,thus violating 3NF

Decomposition:
B1

Card no	CVV	name_on_card

B2

CVV	Expiry_date date

5)TEST FOR LOSSLESS JOIN PROPERTY:

A relation R on decomposition into two relations R1, R2 and the natural join of R1 and R2 can get back relation R again. This implies that there has been no loss of data/reductant rows.

Additional tuples that were not in R are called **spurious tuples** because they represent spurious information that is not valid. The spurious tuples are marked by asterisks (*)

Example
Card_details

Login id	Card no	name_on_ card	Expiry date	CVV	Billing Address

AFTER DECOMPOSITION:

A

Login id	Card no	Billing Address

B

Card no	name_on_card	Expiry date	CVV

A and B on **Natural Join** on Card_no give the original relation ,thus do not **generate spurious tuples**

```
4
5 CREATE TABLE public.a
5 (
7     login_id character varying(15) COLLATE pg_catalog."default" NOT NULL,
8     card_no character(16) COLLATE pg_catalog."default" NOT NULL,
9     billing_address character varying(50) COLLATE pg_catalog."default" NOT NULL,
0     CONSTRAINT card_details_pkey2 PRIMARY KEY (login_id, card_no),
1     CONSTRAINT usrcardfk2 FOREIGN KEY (login_id)
2         REFERENCES public.user_credentials (login_id) MATCH SIMPLE
3         ON UPDATE NO ACTION
4         ON DELETE CASCADE
5 )
6
```

```
4
5 CREATE TABLE public.b
6 (
7     name_on_card character varying(50) COLLATE pg_catalog."default" NOT NULL,
8     card_no character(16) COLLATE pg_catalog."default" NOT NULL,
9     expiry_date date NOT NULL,
10    cvv character(3) COLLATE pg_catalog."default" NOT NULL,
11    CONSTRAINT card_details_pkey3 PRIMARY KEY (card_no)
12 )
```

After some values are inserted into both A and B tables we do the natural join to check the **non-additive join property,**

And the output matches the original table i,e.Card_details.Thus the decomposed tables satisfy lossless join property,which is shown below.

database project carrental-master/postgres@PostgreSQL 12

Query Editor | Query History

```
1 select *
2 from a natural join b;
```

Data Output | Explain | Messages | Notifications

	card_no character (16)	login_id character varying (15)	billing_address character varying (50)	name_on_card character varying (50)	expiry_date date	cvv character (3)
1	4735111122223333	mali	1530 S.Campbell Rd, Dallas, ...	Mehmet Ali Onde	2018-01-15	287
2	4233908110921001	sena	101 Meritline drive	Sena Bulut	2018-01-15	419
3	823990811009209	gozde	43 Greenville Road	Gozde Sismanuglu	2020-01-15	419
4	1200000210921909	hazal	43 Greenville Road	Hazal Fýrat	2019-05-15	419
5	4533777190721001	cem	43 Greenville Road	Cem Berke Çebi	2018-01-15	419

6)DDL

Few Data definition Statements of creating a table and inserting the values into it are shown here.

CAR table creation:

```
CREATE TABLE public.car
(
    vin character(17) COLLATE pg_catalog."default" NOT NULL,
    location_id integer NOT NULL,
    reg_no character varying(15) COLLATE pg_catalog."default",
    status character varying(15) COLLATE pg_catalog."default" NOT NULL,
    seating_capacity integer NOT NULL,
    disability_friendly character(1) COLLATE pg_catalog."default",
    car_type character varying(15) COLLATE pg_catalog."default" NOT NULL,
    model character varying(20) COLLATE pg_catalog."default",
    year character(4) COLLATE pg_catalog."default",
    color character varying(10) COLLATE pg_catalog."default",
    CONSTRAINT car_pkey PRIMARY KEY (vin),
    CONSTRAINT car_reg_no_key UNIQUE (reg_no),
    CONSTRAINT carvinrentalfk FOREIGN KEY (location_id)
        REFERENCES public.rental_location (location_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT carvintypefk FOREIGN KEY (car_type)
        REFERENCES public.car_type (car_type) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)
```

Inserting values into it:

```
INSERT
INTO CAR
(VIN,Location_ID,Reg_No,Status,Seating_Capacity,Disability_Friendly,Car_Type,Model,Year,Color)
VALUES
('F152206785240289',101,'TXF101','Available',5,'N','Economy','Mazda3','2007','Gold'),

('T201534710589051',101,'KYQ101','Available',5,'Y','Standard','Toyota Camry','2012','Grey'),

('E902103289341098',102,'XYZ671','Available',5,NULL,'Premium','BMW','2015','Black'),

('R908891209418173',103,'DOP391','Unavailable',7,NULL,'SUV','Mercedes','2014','White'),

('N892993994858292',104,'RAC829','Available',15,NULL,'MiniVan','Volvo','2013','Black');
```

2)PAYMENT table creation:

```

CREATE TABLE public.payment
(
    payment_id integer NOT NULL,
    amount_paid money NOT NULL,
    card_no character(16) COLLATE pg_catalog."default",
    expiry_date date,
    name_on_card character varying(50) COLLATE pg_catalog."default",
    cvv character(3) COLLATE pg_catalog."default",
    billing_address character varying(50) COLLATE pg_catalog."default",
    reservation_id integer NOT NULL,
    login_id character varying(15) COLLATE pg_catalog."default",
    saved_card_no character(16) COLLATE pg_catalog."default",
    paid_by_cash character(1) COLLATE pg_catalog."default",
    CONSTRAINT payment_pkey PRIMARY KEY (payment_id),
    CONSTRAINT paymentreservationfk FOREIGN KEY (reservation_id)
        REFERENCES public.reservation (reservation_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

```

Inserting values into payment table:

```
INSERT
INTO PAYMENT
(Payment_ID,Amount_Paid,Card_NO,Expiry_Date,Name_On_Card,CVV,Billing_Address,Reservation_ID,Login_ID,Saved_Card_No,Paid_By_Ca
VALUES
(1001,129.65,'4735111122223333',('2018/01/15'),'Mehmet Ali Onde','100','1530 S.Campbell Rd, Dallas, TX 75251',1,NULL,NULL,NUL

(1002,300.00,NULL,NULL,NULL,NULL,NULL,5,NULL,NULL,'Y'),

(1003,98.90,NULL,NULL,NULL,NULL,NULL,5,NULL,NULL,'Y'),

(1004,689.35,NULL,NULL,NULL,NULL,NULL,3,'cem','4735111122223333',NULL),

(1005,114.91,NULL,NULL,NULL,NULL,NULL,4,NULL,NULL,'Y');
```

7)Triggers

i)when the user tries to **delete licence_no** from the car_user table, this trigger comes into picture and raises an exception saying that '**you are not allowed to delete licence_no'.**

```
4
5  CREATE FUNCTION public.trg_nopermissonforlicensedeletion()
6      RETURNS trigger
7      LANGUAGE 'plpgsql'
8      COST 100
9      VOLATILE NOT LEAKPROOF
10 AS $BODY$begin
11    if(exists(select * from CAR_USER where license_no is not null )) then
12          raise EXCEPTION 'You are not allowed to delete license No:';
13     end if;
14 end;
15 $BODY$;
```

```
-- DROP TRIGGER trg_nopermissonforlicensedeletion ON public.car_user;

CREATE TRIGGER trg_nopermissonforlicensedeletion
    AFTER DELETE
    ON public.car_user
    FOR EACH ROW
    EXECUTE PROCEDURE public.trg_nopermissonforlicensedeletion();
```

ii)when the user tries to **update the VIN of CAR table**,this trigger comes into effect and raises an exception saying that **'VIN can't be changed'.**

```
CREATE FUNCTION public.trg_updatevin()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
begin
if(exists(select * from new,old where new.Reg_No=old.Reg_No and new.VIN!=old.VIN)) then
  RAISE EXCEPTION'VIN can not be change.';
 end if;

end;
$BODY$;
```

```
-- DROP TRIGGER trg_updatevin ON public.car;

CREATE TRIGGER trg_updatevin
    AFTER UPDATE
    ON public.car
    FOR EACH ROW
    EXECUTE PROCEDURE public.trg_updatevin();
```

8)VIEWS

Two views are created to show only some part of the relation and hide the remaining as a part of **securing** the data from being used .
Views are not materialized i,e.they are not physically stored.

i)gets_total-amount

```
CREATE OR REPLACE VIEW public.gettotalamount
 AS
 SELECT car_user.license_no,
    car_user.fname,
    car_user.lname,
    reservation.rental_amount,
    reservation.insurance_amount,
    reservation.tot_amount
   FROM reservation
     JOIN car_user ON car_user.license_no::text = reservation.license_no::text;
```

ii)gets_car_info.

```
CREATE OR REPLACE VIEW public.rentalcarinformation
 AS
 SELECT car.car_type,
    car.seating_capacity,
    car.model,
    car.year,
    car.color,
    car_type.price_per_day
   FROM car_type
     JOIN car ON car.car_type::text = car_type.car_type::text;
```

9)SQL QUERIES:

i)Queries using join:

1)Retrieve the promo code used by the user when reserving a car

2)Join Reservation and Car_user table on Licence_no to get fname ,lname ,rental amount etc..

Query Editor | Query History

```
select CAR_USER.License_No,Fname,Lname,Rental_Amount,Insurance_Amount,Tot_Amount
from RESERVATION inner join CAR_USER
on CAR_USER.License_No=RESERVATION.License_No
```

Data Output | Explain | Messages | Notifications

	license_no character varying (15)	fname character varying (15)	lname character varying (15)	rental_amount money	insurance_amount money	tot_amount money
1	E12905109	Mehmet	Onde	? 119.70	? 9.95	? 129.65
2	G30921561	Sena	Bulut	? 119.80	? 9.95	? 129.75
3	M12098127	Cem	Cebi	? 659.40	? 29.95	? 689.35
4	M12098187	Gözde	Sismanoglu	? 89.95	? 24.95	? 114.90
5	R12098127	Hazal	Fýrat	? 299.00	? 99.90	? 398.90

ii) **Queries on Aggregate function**

database project carrental-master/postgres@PostgreSQL

Query Editor | Query History

```
SELECT COUNT(VIN)
FROM CAR
WHERE SEATING_CAPACITY>4
```

Data Output | Explain | Messages | Notifications

	count bigint
1	5

database project carrental-master/postgres@PostgreSQL 12

Query Editor | Query History

```
SELECT COUNT(AMOUNT_PAID)
FROM PAYMENT
WHERE PAID_BY_CASH='Y'
```

Data Output | Explain | Messages | Notifications

	count bigint
1	3

3) **To get the number of people who stay in ‘Greenville’.**

database project carrental-master/postgres@PostgreSQL 12

Query Editor | Query History

```
select count(license_no)
from car_user
where address Like '%Greenville%';
```

Data Output | Explain | Messages | Notifications

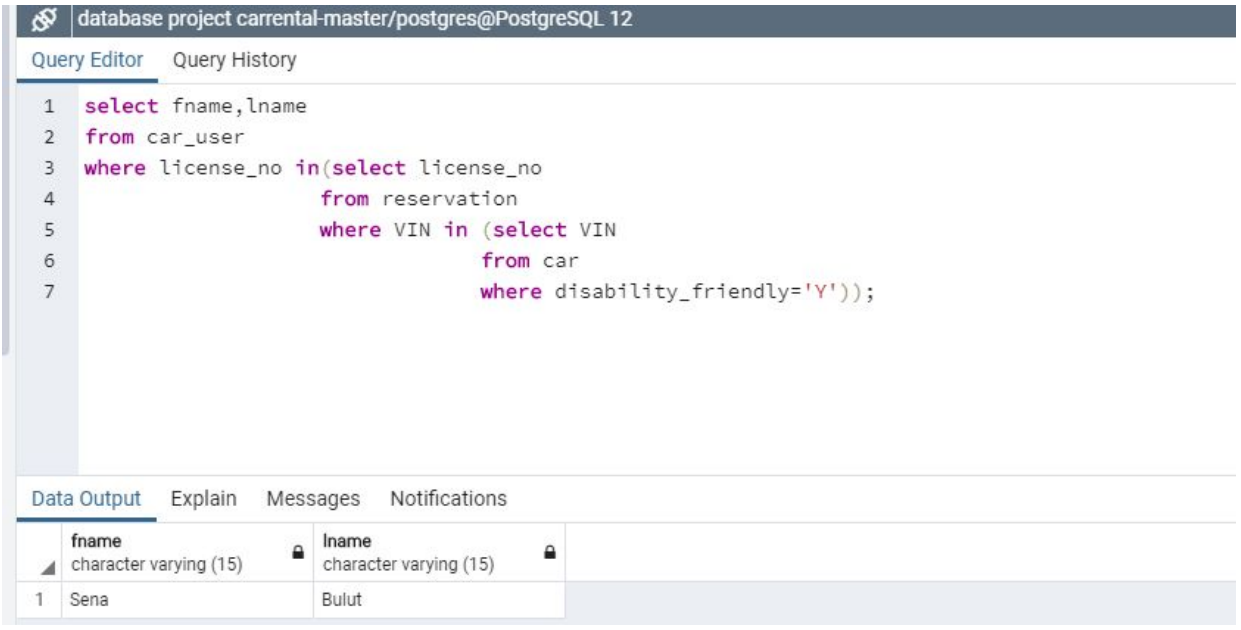
	count bigint
1	3

iii)**NESTED QUERIES**

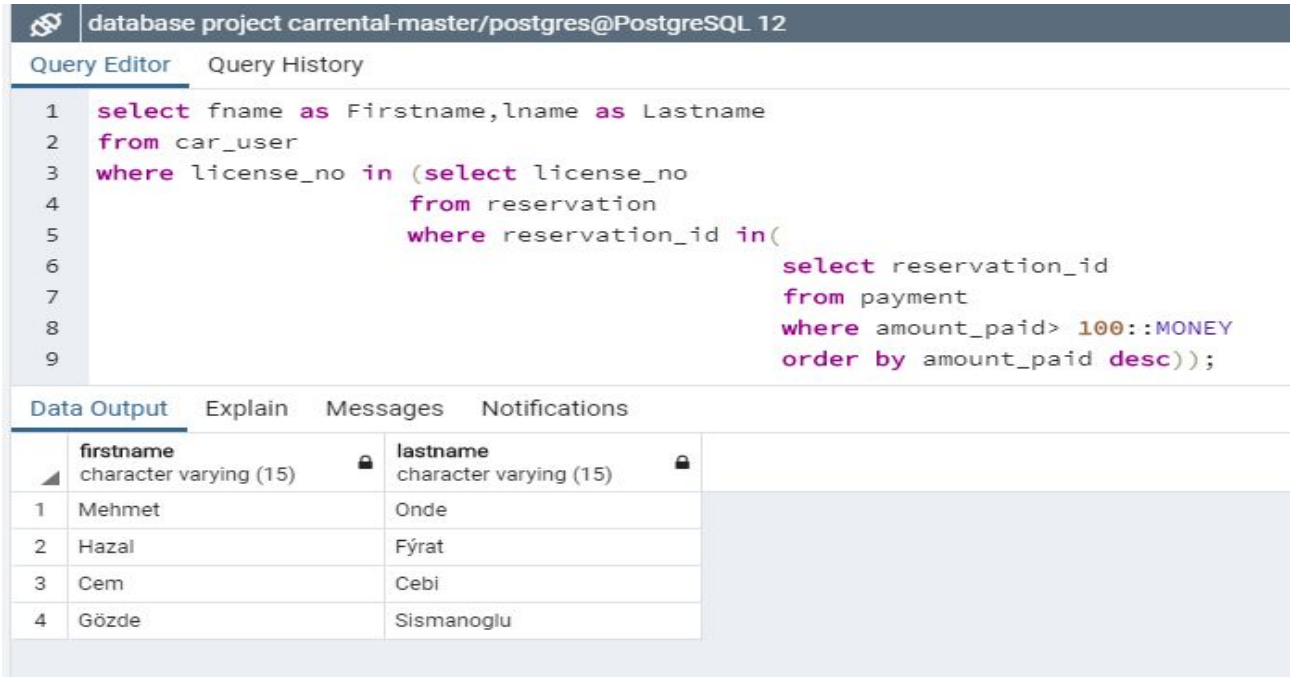
A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

1)To get the user name who has rented disability_friendly car.



2)To get the name of the user whose cost for the ride is greater than $100 arranged in descending order



3)To get the name of the user who has hired an additional driver.

database project carrental-master/postgres@PostgreSQL 12

Query Editor | Query History

```
1  select fname as Firstname,lname as Lastname
2  from car_user
3  where exists (select *
4                      from reservation
5                      where exists (
6                                          select reservation_id
7                                          from additional_driver
8                                          ));
```

Data Output | Explain | Messages | Notifications

	firstname character varying (15)	lastname character varying (15)
1	Mehmet	Onde
2	Sena	Bulut
3	Hazal	Fýrat
4	Cem	Cebi
5	Gözde	Sismanoglu

4)To get the reservation_id of the user who reserved his car

Query Editor | Query History

```
1  select reservation_id
2  from RESERVATION
3  where drop_location_id in(
4                      select location_id
5                      from rental_location
6                      where zip_code='76512');
7
```

Data Output | Explain | Messages | Notifications

	reservation_id [PK] integer
1	3

5)To get the Amt_paid by the user on reservation by tracking down his license_no.

Query Editor | Query History

```
select amount_paid
from payment
where reservation_id in(select reservation_id
                        from reservation
                        where license_no in (select license_no
                                             from car_user));
```

Data Output | Explain | Messages | Notifications

	amount_paid money
1	? 129.65
2	? 300.00
3	? 98.90
4	? 689.35
5	? 114.91

6)To get the mail_id of the user who has used promo_code and which can be used for more than one booking.

database project carrental-master/postgres@PostgreSQL 12

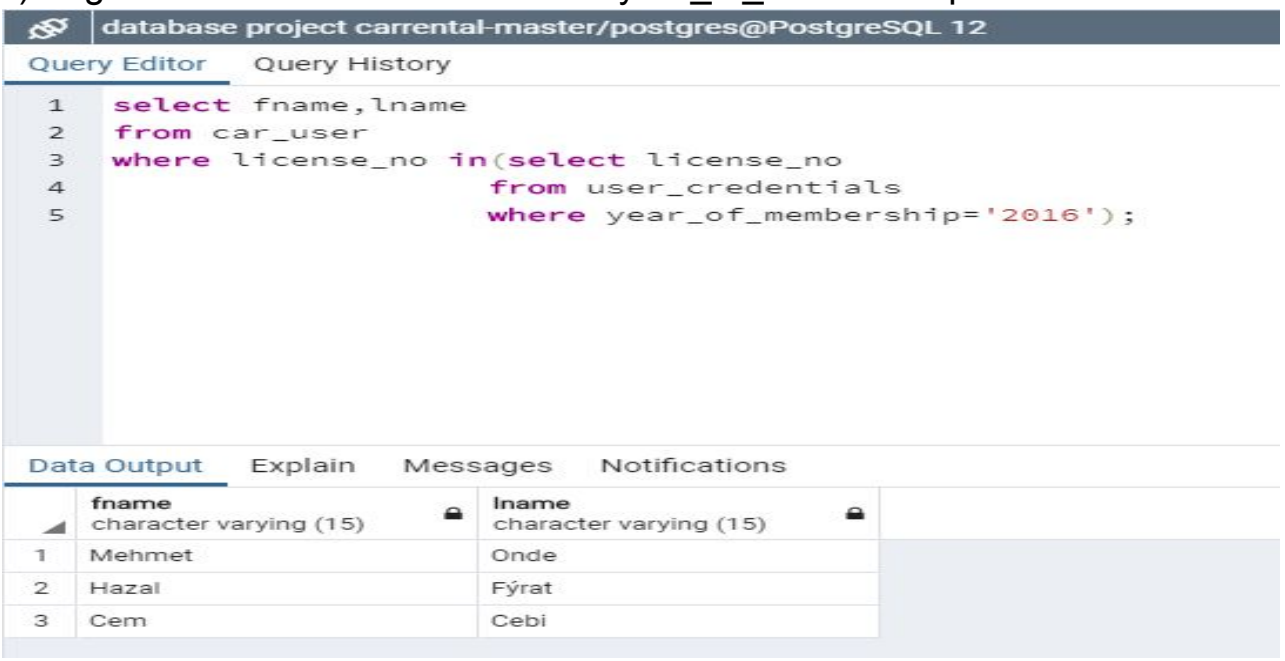
Query Editor | Query History

```
select  fname,lname,email
from car_user
where license_no in(select license_no
                    from reservation
                    where promo_code in (select promo_code
                                         from offer_details
                                         where is_one_time='N'));
```

Data Output | Explain | Messages | Notifications

	fname character varying (15)	lname character varying (15)	email character varying (25)
1	Cem	Cebi	cemberkecebi@gmail.com

7)To get the name of the user whose year_of_membership is 2016



10)Conclusion:

To conclude,the DBMS manages three important things :
i)The data,
ii)The database engine that allows the database to be accessed ,locked and modified and ,
iii)The database schema which defines the database logical structure.
These three foundational elements help provide concurrency,security,data integrity and uniform administration procedure

Capabilities:

1) This database is capable of providing all the related information from user's info to cars' and also the information about additional driver(if there).

2) This database can also handle a Reservation system, where multiple users can book for the same car but **Concurrency** is maintained.

3) This database also maintains the payment record of the users which can be useful for further booking

Limitations:

1)The total cost of the ride is calculated **not by the miles travelled but instead** it is done by calculating the time period for which the car was hired.

2)The more complex operations like **refund on a cancellation** are not done in this project.

3)It does not mention anything about the car's condition whether or not it should be serviced before handing it over to a customer .

Future enhancements:

1) To store the track of the car along the user's ride indicating its current location

2) Include all kinds of automobiles in the rental system.

3)Application of GUI and front-end software.

4) Web check in system could be emulated.