DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

|  |  |  |
| --- | --- | --- |
| **SRN :**  **PES1201800152** | **Name** :  **ROHIT VISHWAKARMA** | **Evaluation date and time:**  **Date : 23 - 05 - 2020**  **Time - 10 : 00 am** |

**Functional Dependencies**

**2**

**Identifying Keys based on FDs 2**

[**Normalization**](#_wc0v2611q54c) **& testing for lossless join property 2+2**

[**DDL**](#_pn441yprtk2f)**: Table creation with all constraints 2+2**

[**Triggers**](#_a9bswny9ffj) **2**

[**SQL Queries**](#_exvna2br67rz) **2**

**Viva / modifications (Unit III/ IV concepts) 2+2**

**Taxi Service Database**

|  |
| --- |
| **Problem statement:**  Suppose that you want to start a taxi service in your city or nearby regions . Design a database for the taxi service where you as a owner of the taxi service can decide what  variety of cabs you want to keep (Ex . Mini, micro, sedan, etc) and also you can set their  cost per Km in (₹) . You should be able to allot cabs of the above mentioned types in the  database along with the number of cabs of each type . You should store the recruited drivers in the database along with the salary and cab\_id which they ride . Your database  should give the option to delete the cab type(Ex. Mini , micro, sedan etc) and also the associated cabs of that type . There should be an option to convert or upgrade a cab type  ie.(You could change all the mini cabs to micro etc.). Coming to the relations the db must have a Relation for the drivers with details like (phone, cab number they drive , cab\_type,salary ,experience ) , client relation with (client\_id, fname,lname , e-wallet id , phone number city) and a wallet relation with (wallet id , balance ), ride relation with  client id, driver\_id and cab , pickup and drop location , ride status (D’ for done completed , ‘C’ for confirmed to arrived‘) ride fare (only e-wallet so every client must maintain a wallet). This db should be have the following constraints that no drivers have the difference between age and experience (age – experience > 20 yrs ), no two  clients have same phone number and no two drivers can have same phone numbers.  There should be a trigger that automatically check the e-wallet balance and let the transaction happen if the ride fare is less or equal to current balance otherwise should return None. There must be views in the database which display the ride duration of each driver from the joining . There facilities for the client to add money in the e -wallet , check there past rides . This db should have relation for the revenue from each of the cab type  and also the overall revenue .  1. Have a console based front end  2. Backend Postgres 11 |

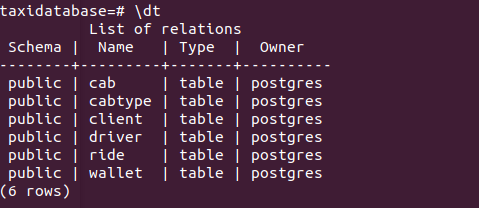
**click on the below link for source code**

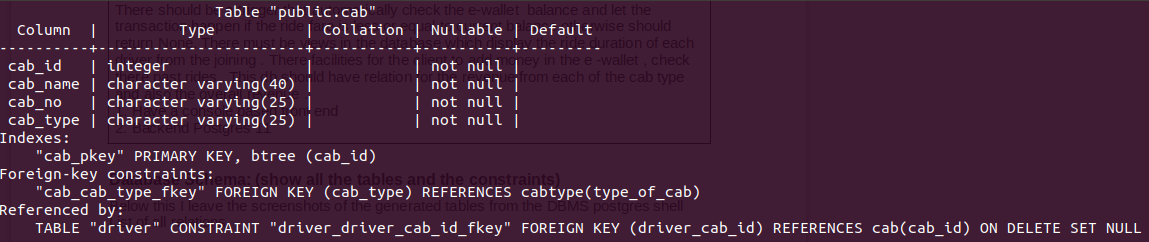
[**https://github.com/ROHIT25082000/Destination-Travels**](https://github.com/ROHIT25082000/Destination-Travels)

# **Database Schema: (show all the tables and the constraints)**

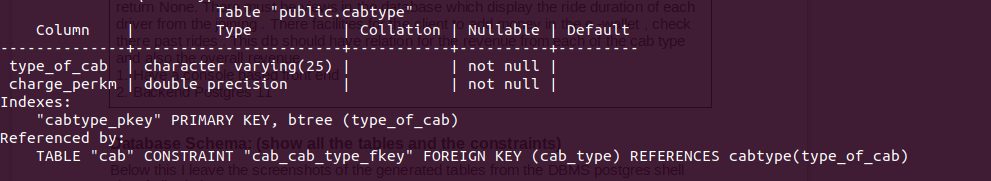
Below this I leave the screenshots of the generated tables from the DBMS postgres shell

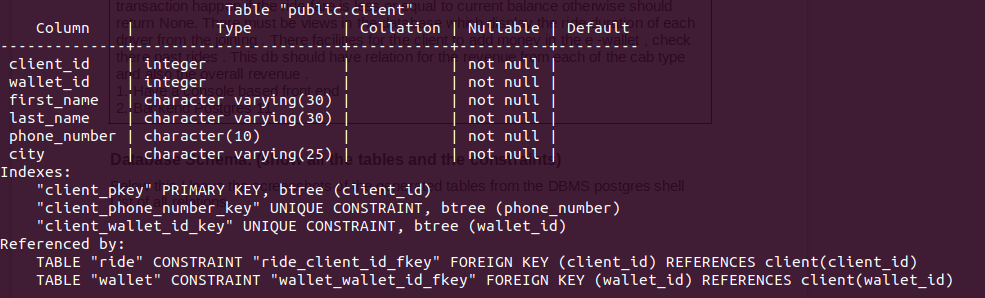
**1.) List of all relations in the Db**



 **2.) cab(cab\_id , cab\_name, cab\_no, cab\_type)**

**3.) cabtype(type\_of\_cab , charge\_perkm)**



 **4.)client(client\_id ,wallet\_id ,first\_name,last\_name,phone\_number)**

# **5.) driver(driver\_id, first\_name,last\_name,salary,experience , age,phone\_number,driver\_cab\_id)**

# 

# **6.) ride( client\_id , driver\_id , ride\_start\_time, ride\_end\_time , source , destination , ride\_status , ride\_fare , ride\_otp )**

# 

# **7.) wallet (wallet\_id , balance)**

# 

# **Functional Dependencies: (List based on your application constraints)**

1.) *Functional Dependencies in the relation*

*client( Client\_id , first\_name , last\_name , wallet\_id, phone\_number, city)*

***(i) client\_id →first\_name , last\_name , wallet\_id , phone\_number , city***

**(ii) wallet\_id → client\_id , first\_name , last\_name , phone\_number , city**

2.) *Functional Dependencies in the relation*

*cab(cab\_id , cab\_name, cab\_no,cab\_type, charge\_perKm)*

***The above relation was normalized***

**(i) cab\_id → cab\_name , cab\_no, cab\_type , charge\_perKm**

**(ii) cab\_name, cab\_no → (cab\_type , cab\_id)**

**(iii) cab\_type → charge\_perKm**

*3.) Functional Dependencies in the relation*

*driver(driver\_id , first\_name , last\_name , salary , experience ,age,phone\_number , driver\_cab\_id)*

**(i) driver\_id → (driver\_id , first\_name , last\_name , salary , experience , age,phone\_number, driver\_cab\_id)**

*4.) Functional Dependencies in the relation*

wallet(wallet\_id , balance)

**(i) wallet\_id → balance**

5.) Functional Dependencies in the relation

*ride( client\_id , driver\_id , ride\_start\_time, ride\_end\_time , source , destination , ride\_status , ride\_fare , ride\_otp )*

**(i) client\_id , driver\_id , ride\_start\_time, ride\_end\_time → source , destination , ride\_status , ride\_fare , ride\_otp)**

# **Candidate keys: (Justify how did you get these as keys)**

**1.)  *client( Client\_id , first\_name , last\_name , wallet\_id, phone\_number, city)***

In this relation we have the functional dependencies

***(i) client\_id →first\_name , last\_name , wallet\_id , phone\_number , city***

**(ii) wallet\_id → client\_id , first\_name , last\_name , phone\_number , city**

**Candidate keys : client\_id , wallet\_id**

**Clearly Attribute closure of (client\_id)+ and (wallet\_id)+  = Total relation “client”**

**So it is a super key and also is the minimal hence they are the candidate keys.**

**I have choosen “client\_id” as the primary key of the available candidate keys.**

**2.)**

***driver(driver\_id ,first\_name ,last\_name ,salary ,experience ,age,phone\_number ,driver\_cab\_id)***

In this relation we have the functional dependencies

**(i) driver\_id → (driver\_id , first\_name , last\_name , salary , experience , age,phone\_number ,driver\_cab\_id)**

**Candidate key : driver\_id**

**Clearly Attribute closure of (driver\_id)+  = Total relation “driver”**

**So it is a super key and also is the minimal hence it is the candidate keys.**

**I have choosen “client\_id” as the primary key of the available candidate keys.**

**3.)**

***ride( client\_id , driver\_id , ride\_start\_time, ride\_end\_time , source , destination , ride\_status , ride\_fare , ride\_otp )***

In this relation we have the functional dependencies

***(i)(c*lient\_id , driver\_id , ride\_start\_time, ride\_end\_time) → source ,destination , ride\_status , ride\_fare , ride\_otp**

**Candidate key : (*c*lient\_id , driver\_id , ride\_start\_time, ride\_end\_time)**

**This is because in the ride table attribute closure of**

**(*c*lient\_id , driver\_id , ride\_start\_time, ride\_end\_time)+ = Relation (ride)**

**and dropping any of the attributes of the above makes it lose the closure of the**

**whole relation and then it is not possible to find a unique a tuple in the ride.**

***4.)* wallet(wallet\_id , balance)**

**(i) wallet\_id → balance**

**Candidate key : wallet\_id**

**As (wallet\_id)+ = relation wallet and it also minimal so candidate key is wallet\_id .**

**5.)***Functional Dependencies in the relation*

*cab(cab\_id , cab\_name, cab\_no,cab\_type, charge\_perKm)*

**(i) cab\_id → cab\_name , cab\_no, cab\_type , charge\_perKm**

**(ii) cab\_name, cab\_no → (cab\_type , cab\_id)**

**(iii) cab\_type → charge\_perKm**

***Candidate keys : (cab\_id) , (cab\_name, cab\_no)***

**This is because in the ride table attribute closure of**

**(*cab*\_id)+ , (cab\_name ,cab\_no) + = Relation (ride) and lossing any one is**

**makes it not key.**

# **Normalization and testing for lossless join property:**

# **According to the above discussion on functional dependencies and candidate keys for the relations**

1.) *Functional Dependencies in the relation*

*client( Client\_id , first\_name , last\_name , wallet\_id, phone\_number, city)*

***(i) client\_id →first\_name , last\_name , wallet\_id , phone\_number , city***

**(ii) wallet\_id → client\_id , first\_name , last\_name , phone\_number , city**

**Candidate keys are : client\_id, wallet\_id**

**Since all the functional dependencies f(A) → B in Client have A as candidate key which is in turn is a super key hence is it is in BCNF and hence in 3NF**

**also**

**Result : Client relation is normalized in BCNF, 3NF**

2.) *Functional Dependencies in the relation*

*driver(driver\_id , first\_name , last\_name , salary , experience ,age, driver\_cab\_id)*

**(i) driver\_id → (driver\_id , first\_name , last\_name , salary , experience , age,phone\_number, driver\_cab\_id)**

**Candidate key : driver\_id**

**Since all the functional dependencies in the relation f(A) → B have their**

**determinant as a SuperKey *(every candidate key is a super key)* so this**

**driver relation is also in BCNF , 3NF .**

**Result : Driver relation is normalized in BCNF, 3NF**

**3.)** *Functional Dependencies in the relation*

*wallet(wallet\_id , balance)*

**(i) wallet\_id → balance**

***Candidate key : wallet\_id***

**Since all the functional dependencies in the relation f(A) → B have their**

**determinant as a SuperKey *(every candidate key is a super key)* so this**

**driver relation is also in BCNF , 3NF .**

**Result : Wallet relation is normalized in BCNF, 3NF**

**4.)***Functional Dependencies in the relation*

*cab(cab\_id , cab\_name, cab\_no,cab\_type, charge\_perKm)*

**(i) cab\_id → cab\_name , cab\_no, cab\_type , charge\_perKm**

**(ii) cab\_name, cab\_no → (cab\_type , cab\_id)**

**(iii) cab\_type → charge\_perKm**

***Candidate keys : (cab\_id) , (cab\_name, cab\_no)***

***Therefore***

***prime attributes : cab\_id , cab\_name, cab\_no***

***non prime attributes : cab\_type, charge\_perKm***

**In the functional dependencies (iii) cab\_type is not a super key hence it is not in BCNF.**

**Lets see the 3 NF condition**

**We have in functional dependencies (i)and (ii) determinant ( A) as super key**

**but there is a functional dependency from (Non prime ) to (Non prime)**

**we call this is as transitive dependency and thus is not in 3NF**

**So this relation has to be normalized**

***cab(cab\_id , cab\_name, cab\_no,cab\_type, charge\_perKm)***

**Cab(cab\_id, cab\_name ,cab\_no,cabtype) Cabtype(cabtype,charge\_perKm)**

**So now both are in BCNF as Cab has only superkey as fd : (cab\_id)+  = relation Cab and even Cabtype is in BCNF as only superkey as fd : (cabtype)+= retation**

# **Now we can verify that**

**attr(Cab) U att(Cabtype) = attr(Original\_relation)**

**attr(Cab) ꓵ att(Cabtype) != attr(Original\_relation)**

**and attr (cabtype) is superkey in Cabtype relation so this**

**proves that this decomposition is lossless**

# **5.) *ride( client\_id , driver\_id , ride\_start\_time, ride\_end\_time , source , destination , ride\_status , ride\_fare , ride\_otp )***

In this relation we have the functional dependencies

***(i)(c*lient\_id , driver\_id , ride\_start\_time, ride\_end\_time) → source ,destination , ride\_status , ride\_fare , ride\_otp**

**Candidate key : (*c*lient\_id , driver\_id , ride\_start\_time, ride\_end\_time)**

**Only one functional dependencies is have superkey as the determinant so in**

**BCNF , 3NF**

**Result : ride is Normalized in BCNF , 3NF .**

# **DDL**

# **The below are the DDL for the database they run in a python file named schema.py using psycopg2 adapter .**

Sql = """

CREATE TABLE client(

client\_id int primary key,

wallet\_id int unique not null,

first\_name VARCHAR(30) not null,

last\_name VARCHAR(30) not null,

phone\_number CHAR(10) unique not null,

city VARCHAR(25) not null

)

"""

cursor.execute(sql)

sql = """

CREATE TABLE cabtype(

type\_of\_cab VARCHAR(25) primary key,

charge\_perKm FLOAT not null

)

"""

cursor.execute(sql)

sql = """

CREATE TABLE cab(

cab\_id int primary key,

cab\_name VARCHAR(40) not null,

cab\_no VARCHAR(25) not null,

cab\_type VARCHAR(25) not null references cabtype(type\_of\_cab)

)

"""

cursor.execute(sql)

sql = """

CREATE TABLE driver(

driver\_id int primary key,

first\_name VARCHAR(30) not null,

last\_name VARCHAR(30) not null,

salary FLOAT not null,

experience int not null,

age int not null check (age - experience >= 20),

phone\_number CHAR(10) unique not null,

driver\_cab\_id int references cab(cab\_id) on delete set null)"""

cursor.execute(sql)

sql = """

CREATE TABLE wallet(

wallet\_id int not null references client(wallet\_id),

balance float not null

); ”””

cursor.execute(sql)

sql = """

CREATE TABLE ride(

client\_id int not null references client(client\_id),

driver\_id int not null references driver(driver\_id),

ride\_start\_time timestamp not null,

ride\_end\_time timestamp not null,

source VARCHAR(300) not null,

destination VARCHAR(300) not null,

ride\_status char not null,

ride\_fare float not null,

ride\_otp int not null,

PRIMARY KEY(client\_id,driver\_id,ride\_start\_time,ride\_end\_time)

)

"""

cursor.execute(sql)

# **Triggers:**

In my database application when the client tries to update or a new row is inserted into the

***wallet***relation then a trigger will check **BEFORE inserting or updating** whether the

**new.balance < 0 if it is then the trigger returns Null else returns the new.balance**

**The following is the stored procedure and the trigger in the sql .**

sql\_trigger = """

create or replace function tell\_about\_wallet()

returns trigger language plpgsql as $$

begin

return case

when new.balance < 0 then null

else new

end;

end $$;

"""

cursor.execute(sql\_trigger)

sql = """

create trigger tell\_about\_wallet

before insert or update on wallet

for each row execute procedure tell\_about\_wallet();

"""

cursor.execute(sql)

# **Views:**

# According to the problem statement the cab service owner also needs a view to get the ride duration of all the drivers in the company called “ driver\_working\_hours ”

View Description

This view in the sql returns the driver details and the sum of the hours they have worked till now in the descending order

sql\_view = """

CREATE VIEW driver\_working\_hours as

select

driver\_id ,first\_name||' '||last\_name as Fullname ,age,phone\_number, cab\_type ,total\_ride\_time

from

cab join driver on cab.cab\_id = driver.driver\_cab\_id natural join

(select driver\_id ,SUM((cast(ride\_end\_time as date) - cast(ride\_start\_time as date))\*24) as total\_ride\_time

from ride natural join driver

group by driver\_id)

as t

order by total\_ride\_time

DESC;

"""

cursor.execute(sql\_view)

# **SQL Queries:**

# **Insertions into relation takes place by various python classes which have the**

# **Sql query in the bold .**

# **1.) Insertion Queries .**

**Insertion into the client relation**

class Client:

#fp = open("meta.dat","r")

#mystr = fp.read().split(',')

CLIENT\_ID = None#int(mystr[0])

WALLET\_ID = None#int(mystr[1])

mystr = None # List

#fp.close()

def \_\_init\_\_(self,first\_name,last\_name,phone,city,balance):

Client.set\_class\_var\_client() # set them

self.client\_id = Client.CLIENT\_ID

self.wallet\_id = Client.WALLET\_ID

self.first\_name = first\_name

self.last\_name = last\_name

self.phone = phone

self.city = city

**sql = """**

**INSERT INTO client(client\_id,wallet\_id,first\_name,last\_name,phone\_number,city)**

**VALUES ("""+str(self.client\_id)+","+str(self.wallet\_id)+","+"\'"+str(self.first\_name)+"\'"+","+"\'"+str(self.last\_name)+"\'"+","+"\'"+str(self.phone)+"\'"+","+"\'"+str(self.city)+"\'"+")"**

**cursor.execute(sql)**

obj = Wallet(Client.WALLET\_ID,balance)

Client.increment\_number\_of\_client()

Client.change()

**Insertion into the Cabtype relation**

class Cabtype:

cabtype\_num = 0

def \_\_init\_\_(self,type\_of\_cab,cost\_per\_km):

self.type\_of\_cab = type\_of\_cab

self.cost\_per\_km = cost\_per\_km

**sql = """**

**INSERT INTO cabtype(type\_of\_cab,charge\_perKm)**

**VALUES ("""+"\'"+str(self.type\_of\_cab)+"\'"+","+"\'"+str(self.cost\_per\_km)+"\'"+")"**

**cursor.execute(sql)**

Cabtype.increment\_number\_of\_cabtype()

**Insertion into Cab relation**

class Cab:

#fp = open("meta.dat","r")

mystr = None#fp.read().split(',')

CAB\_ID =None# int(mystr[2])

#fp.close()

def \_\_init\_\_(self,cab\_name,cab\_no,cab\_type):

Cab.set\_class\_var\_cab() #set them

self.cab\_id = Cab.CAB\_ID # auto generated

self.cab\_name = cab\_name

self.cab\_no = cab\_no

self.cab\_type = cab\_type

**sql = """**

**INSERT INTO cab(cab\_id,cab\_name,cab\_no,cab\_type)**

**VALUES ("""+"\'"+str(self.cab\_id)+"\'"+","+"\'"+str(self.cab\_name)+"\'"+","+"\'"+str(self.cab\_no)+"\'"+","+"\'"+self.cab\_type+"\'"+")"**

**cursor.execute(sql)**

Cab.increment\_number\_of\_cabs()

**Insertion into driver**

class Driver:

#fp = open("meta.dat","r")

mystr = None #fp.read().split(',')

DRIVER\_ID = None # int(mystr[3])

#fp.close()

def \_\_init\_\_(self,first\_name,last\_name,salary,experience,age,phone\_number,driver\_cab\_id):

Driver.set\_class\_var\_driver()

self.driver\_id = Driver.DRIVER\_ID

self.first\_name = first\_name

self.last\_name = last\_name

self.salary = salary

self.experience = experience

self.age = age

self.phone\_number = phone\_number

self.driver\_cab\_id = driver\_cab\_id

**sql = """**

**INSERT INTO driver(driver\_id,first\_name,last\_name,salary,experience,age,phone\_number,driver\_cab\_id)**

**VALUES ("""+"\'"+str(self.driver\_id)+"\'"+","+"\'"+self.first\_name+"\'"+","+"\'"+self.last\_name+"\'"+","+"\'"+str(self.salary)+"\'"+","+"\'"+str(self.experience)+"\'"+","+"\'"+str(self.age)+"\'"+","+"\'"+str(phone\_number)+"\'"+","+"\'"+str(self.driver\_cab\_id)+"\'"+")"**

**cursor.execute(sql)**

Driver.increment\_number\_of\_driver()

**Insertion in ride and wallet query**

class Ride:

number\_of\_rides = 0

def \_\_init\_\_(self, client\_id, driver\_id, ride\_start\_time, ride\_end\_time, source, destination, ride\_status, ride\_fare, ride\_otp):

self.client\_id = client\_id

self.driver\_id = driver\_id

self.ride\_start\_time = ride\_start\_time

self.ride\_end\_time = ride\_end\_time

self.source = source

self.destination = destination

self.ride\_status = ride\_status

self.ride\_fare = ride\_fare

self.ride\_otp = ride\_otp

**sql = """**

**INSERT INTO ride(client\_id, driver\_id, ride\_start\_time, ride\_end\_time, source, destination, ride\_status, ride\_fare, ride\_otp)**

**VALUES ("""+"\'"+str(self.client\_id)+"\'"+","+"\'"+str(self.driver\_id)+"\'"+","+"\'"+str(self.ride\_start\_time)+"\'"+","+"\'"+str(self.ride\_end\_time)+"\'"+","+"\'"+str(self.source)+"\'"+","+"\'"+str(self.destination)+"\'"+","+"\'"+str(self.ride\_status)+"\'"+","+"\'"+str(self.ride\_fare)+"\'"+","+"\'"+str(self.ride\_otp)+"\'"+")"**

**cursor.execute(sql)**

class Wallet:

number\_of\_wallets = 0

def \_\_init\_\_(self,wallet\_id,balance = 0):

self.wallet\_id = wallet\_id

self.balance = balance

**sql = """**

**INSERT INTO wallet(wallet\_id,balance)**

**VALUES ("""+"\'"+str(self.wallet\_id)+"\'"+","+"\'"+str(self.balance)+"\'"+")"**

**cursor.execute(sql)**

**Wallet.increase()**

**Some retrieval querires**

**1.) *A query to check the current wallet balance and returns the balance***

**sql = """**

**Select balance from client natural join**

**wallet where client\_id = """+str(current\_obj.client\_id)**

**cursor.execute(sql)**

**balance = cursor.fetchall()**

**2.) *A query which checks the vacancy for the drivers returns the number of vacant posts***

**sql = """**

**select count(\*) from (**

**select cab\_id**

**from cab**

**except**

**select driver\_cab\_id**

**from driver) as t**

**"""**

**cursor.execute(sql)**

**vacancy = cursor.fetchall()**

**3.) *A query which return the data by using a View defined earlier “driver\_working\_hours”***

**sql = "SELECT \* FROM driver\_working\_hours "**

**cursor.execute(sql)**

**rows = cursor.fetchall()**

***4.)A query to get the revenue per cabtype of the cabs who have completed the ride and have ride status = ‘D’ which means completed***

**sql = """**

**select cab\_type , sum(ride\_fare) as Total\_revenue\_generated**

**from ride natural join driver join cab on driver.driver\_cab\_id = cab.cab\_id**

**where ride\_status = 'D' group by cab\_type;**

**"""**

**cursor.execute(sql)**

**rows = cursor.fetchall()**

**5.) *A query to get all cabtypes with charge\_perKm in ascending order***

**sql = """**

**SELECT \* from cabtype order by charge\_perKm**

**"""**

**cursor.execute(sql)**

**rows = cursor.fetchall()**

**6.) A query to check whether a cabtype exists in the database**

**sql = """select exists(**

**select 1**

**from cabtype**

**where type\_of\_cab = """+"\'"+name\_type+"\'"+")"**

**cursor.execute(sql)**

**row = cursor.fetchall()**

**7.) *A query to fetch 1 driver\_id which are not booked yet and has a free cab***

**sql = """**

**select driver\_id from (**

**select \* from driver join cab on driver.driver\_cab\_id = cab.cab\_id where cab\_type ="""+"\'"+name\_type+"\'" +")"+" as t "+"""**

**except**

**select driver\_id from**

**ride where ride\_status = 'C'**

**LIMIT 1**

**"""**

**The above query returns nothing or [ ] if all drivers with cabs are booked**

***8.) A query to get total revenue from the completed rides***

**sql = """**

**select sum(ride\_fare) as Total\_revenue\_generated**

**from ride natural join driver join cab on driver.driver\_cab\_id = cab.cab\_id**

**where ride\_status = 'D';**

**"""**

**cursor.execute(sql)**

**rows = cursor.fetchall()**

**Some Update queries**

**1.)*A query to update a cab type***

**sql = """**

**Update cab**

**set cab\_type = """+"\'"+new\_type+"\'"+"**

**where cab\_type ="+"\'"+name\_type+"\'"**

***2.) A query to change the ride status to = ‘D’ if a ride is completed***

**sql = """**

**UPDATE ride**

**set ride\_status = 'D'**

**where client\_id = """+str(current\_obj.client\_id)+"and driver\_id ="+str(dd)+" and ride\_start\_time ="+"\'"+str(ride\_start\_time)+"\'"+" and ride\_end\_time ="+"\'"+str(ride\_end\_time)+"\'"**

**cursor.execute(sql)**

***3.) A query to update the ride status to = ‘X’ if the ride is cancelled***

***in the between after the booking the cab***

**sql = """**

**UPDATE ride**

**set ride\_status = 'X'**

**where client\_id = """+str(current\_obj.client\_id)+"and driver\_id ="+str(dd)+" and ride\_start\_time ="+"\'"+str(ride\_start\_time)+"\'"+" and ride\_end\_time ="+"\'"+str(ride\_end\_time)+"\'"**

**cursor.execute(sql)**

**4.) A query which updates the wallet balance based on the return value from the trigger defined earlier**

**sql = """**

**UPDATE wallet**

**set balance = """+str(balance\_new)+" where wallet\_id ="+str(current\_obj.wallet\_id)+**

**"returning balance;"**

**cursor.execute(sql)**

**5.) A query to update a driver\_cab\_id (or in other words allot a driver a cab )**

**sql = """**

**UPDATE driver**

**set driver\_cab\_id = """+str(cabid)+"where driver\_id ="+str(driverid)**

**cursor.execute(sql)**

**Here if a drivers\_cab\_id was set to null then the above will give the cab driver a new cab\_id**

**6.) The query to convert one cab\_type to other by update**

**sql = """**

**Update cab**

**set cab\_type = """+"\'"+new\_type+"\'"+"where cab\_type ="+"\'"+name\_type+"\'"**

**cursor.execute(sql)**

**Some Delete queries**

**1.) *Delete a cabtype from the database .***

**sql = """**

**DELETE from cabtype**

**where type\_of\_cab = """ + "\'"+name\_type+"\'"**

**cursor.execute(sql)**

**For accessing the source code like on the below link**

<https://github.com/ROHIT25082000/Destination-Travels>