A project report on

“Uber-3”

submitted in fulfillment of requirements for

CS6360-002 Database Design, Fall 2019

By-

Name	Net ID
Gaurang Ratnakar	GJR180000
Tanuj Deria	TSD190001
Varsha Teckchandani	VXT180011

Erik Jonsson School of Engineering and Computer Science

The University of Texas at Dallas

800 W Campbell Rd, Richardson, TX 75080.

Table of Contents

Database Requirement

We are designing a system similar to Uber which is an online cab booking system. The system should have users. The details of the users such as the first name, last name, date of birth, email address, password, address should be stored in the system. Users must login into the system with an email id and password. There must be two types of users in the systems one who takes rides and one who offers a ride. The users should have a unique email address for registering on the system. The driver must have a driver license, a flag to indicate whether the driver is still serving for the company, bank account details for sending the earnings that he has earned, license number and expiry date. The bank account details include the bank account number and the routing number. Additionally, the system must also store the earnings the driver has earned which hasn't been sent out to the driver.

The drivers should offer ride to the riders. A driver can have a single car in the system and also that car cannot be driven by some other driver. The driver also has ratings which are given by the rider. The system should also store information about the rides such as the total amount payable for the ride, pickup and drop off address, start time and the end time for the trip. Each ride should also have a ride id to uniquely identify the rides. The GPS co-ordinates of the pickup address and the drop-off location should be stored in each ride. The GPS co-ordinates are used to show the markers for the pickup address and the drop off location on the map. Each ride will be having some status associated with it to give status updates to the user. The status can be 'Upcoming', 'Completed', 'Ongoing', 'Cancelled'. The status should also store the time when the status of the particular ride gets changed. The system should also store the feedbacks given by the riders for the ride. The feedback must store the rating for a particular ride that the rider has taken, some comment for the ride and also it should have a feedback id which should be unique.

The rider must pay for the rides using a card which is the default and the only method for payment in this system. The rider must be able to store multiple cards on the system. The card details such as the card number, CVV, expiry date and the billing address must be stored on the system. The rider should also have the facility to store multiple addresses on the system. Additionally, the user should also be able to categorize the address as 'Home', 'Office', 'Billing', 'Mailing' or some custom tag provided by the user.

The system must offer many types of cars for rides. The cars can be a Sedan, Convertible, SUV, Sports car etc. Each car will be uniquely identified by the Vehicle identification number. The system must also store details about the car such as VIN, color, model of the car, model year, model make, license plate number, model id and model type id.

Structure of Uber Database

User

User is the primary entity in the database. There can be two types of user in the system designed. A user can either be rider or a driver but can't be both. The user has many attributes like email id, password, cell no, first name, last name, date of birth, middle name. Here email id is the primary key for the system.

Rider

A rider is registered on the system by his email id. A rider takes a ride offered by a driver. He can have multiple addresses like the billing address, mailing address, office address etc. Similarly the rider can save multiple cards on the system.

Driver

A driver is also registered on the system by his email id. A driver offers a ride to the rider. A driver has multiple attributes like rating, active, account number, driver license, total earnings, license expiry date. The active attribute determines whether the driver is still active on the system or not. The rating attribute specifies the average rating of the driver given by the riders for the ride.

Ride

A rider is offered by the driver to the rider. Each ride has a unique ride id which is assigned by the system. Each ride has many attributes such as the start time, end time, amount payable, co-ordinates of the pickup address and the drop off location.

Ride_Status

Each ride has a status associated with it and each ride_status has a unique identifier associated with it. The ride has attributes such as status and status time. Status can take values such as 'Ongoing', 'Completed, 'Cancelled', 'Scheduled'. The status time specifies the time at which the status of the ride was changed.

Feedback

Each ride has a feedback associated with it and each feedback has a unique identifier associated with it. The feedback has attributes such as rating and comment.

Rider_Address

Each rider has multiple address associated, so we store addresses in a separate entity where each address has a unique identifier. The address has attributes like street, city, state, zip code, country and address tag. The address tag can be ‘Home’, ’Office’, ’Other’ or some personal tag given by the user.

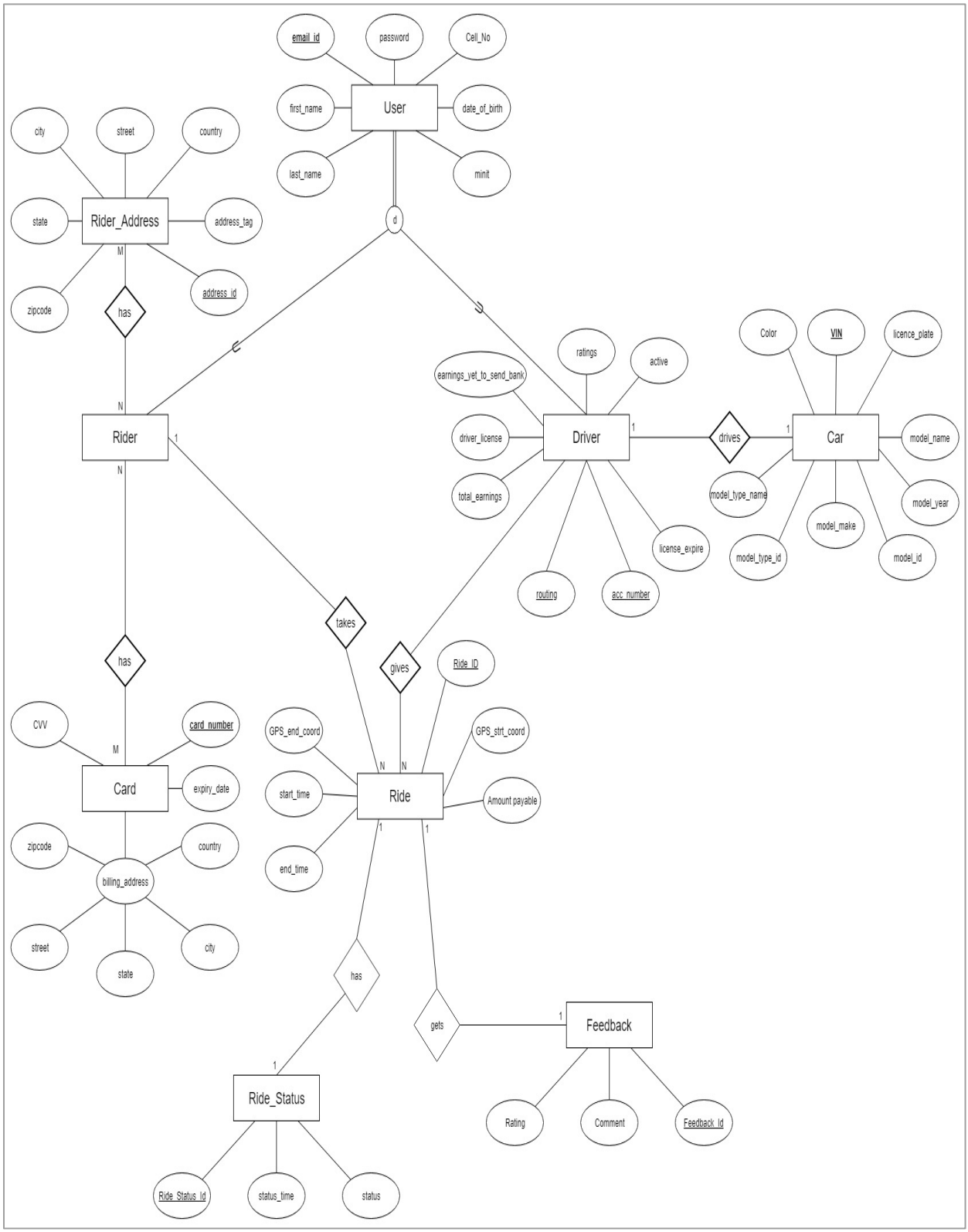
Card

Each rider has multiple cards, so we have created a table for cards in which each record will be uniquely identified by the card number. As per our assumption the rider can pay for the ride only using card. The card has attributes such as expiry date, cvv and billing address.

Car

As per our assumption, a driver can have only one car and he only can drive that car. A car will be uniquely identified by the Vehicle Identification Number (VIN). A car has many attributes such as color, license plate number, model name, model year, model id, model make, model type id, model name.

Modeling of Requirements as ER- Diagram :



The requirements can be summarized/ derived from ERD as below –

1. As per our assumption, the driver will drive only one car and that car can only be driven by a single driver. So the mapping between car and driver is 1:1.

2. A driver can offer multiple rides but the same ride cannot be offered by multiple drivers. So the mapping between Driver and Ride is 1:N.

3. A feedback can be assigned to one ride at a time and the same ride cannot get multiple feedbacks. So the mapping between Ride and Feedback is 1:1.

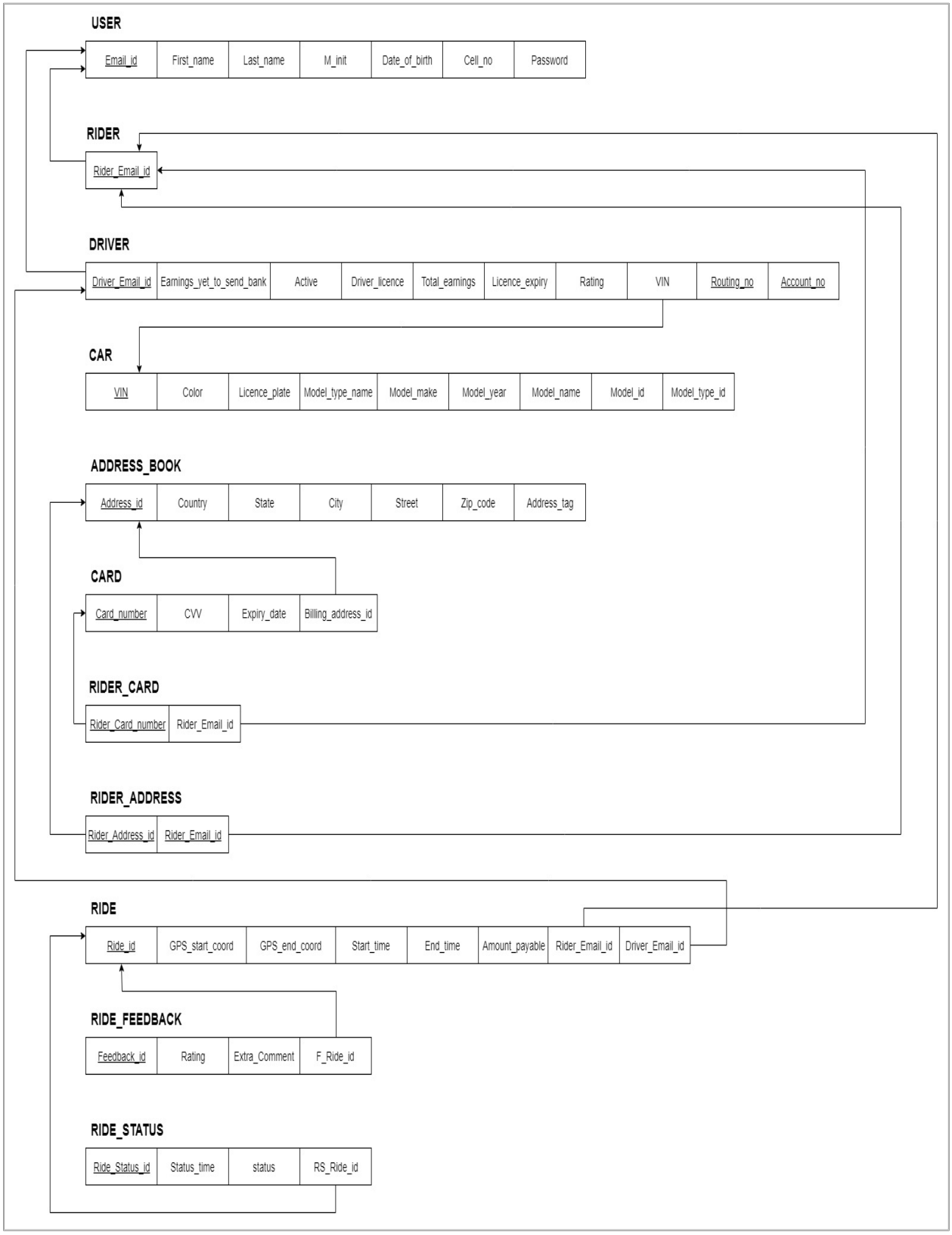
4. A ride can have only one status which gets changed over time and a single status can be applied to a ride. So the mapping between Ride and Ride_Status is 1:1.

5. As per our assumption a rider can take multiple rides the same ride cannot be taken by multiple riders. So the mapping between Rider and Ride is 1:N.

6. A rider can have multiple address stored in the database and the same address can also be shared by multiple riders. So the mapping between Rider and Rider_Address is M:N.

7. A rider can have multiple cards and the card can also be shared with multiple users. So the mapping between Rider and Card is M:N.

Relational Schema Before Normalization



Functional Dependencies

For the Car table, following are the Functional Dependencies:

FD1: VIN → License_plate, color, Model_id, Model_type_id, Model_name, Model_year, Model_make, Model_type_name

FD2: Model_id → Model_type_id, Model_name, Model_year, Model_make

FD3: Model_type_id → Model_type_name

From the following Functional Dependencies, we can see that the relation is in 2nd Normal form but not in 3rd Normal form because of 2 Transitive Dependencies which are –

1] VIN → Model_id and Model_id → Model_type_id, Model_name, Model_year, Model_make

2] Model_id → Model_type_id and Model_type_id → Model_type_name

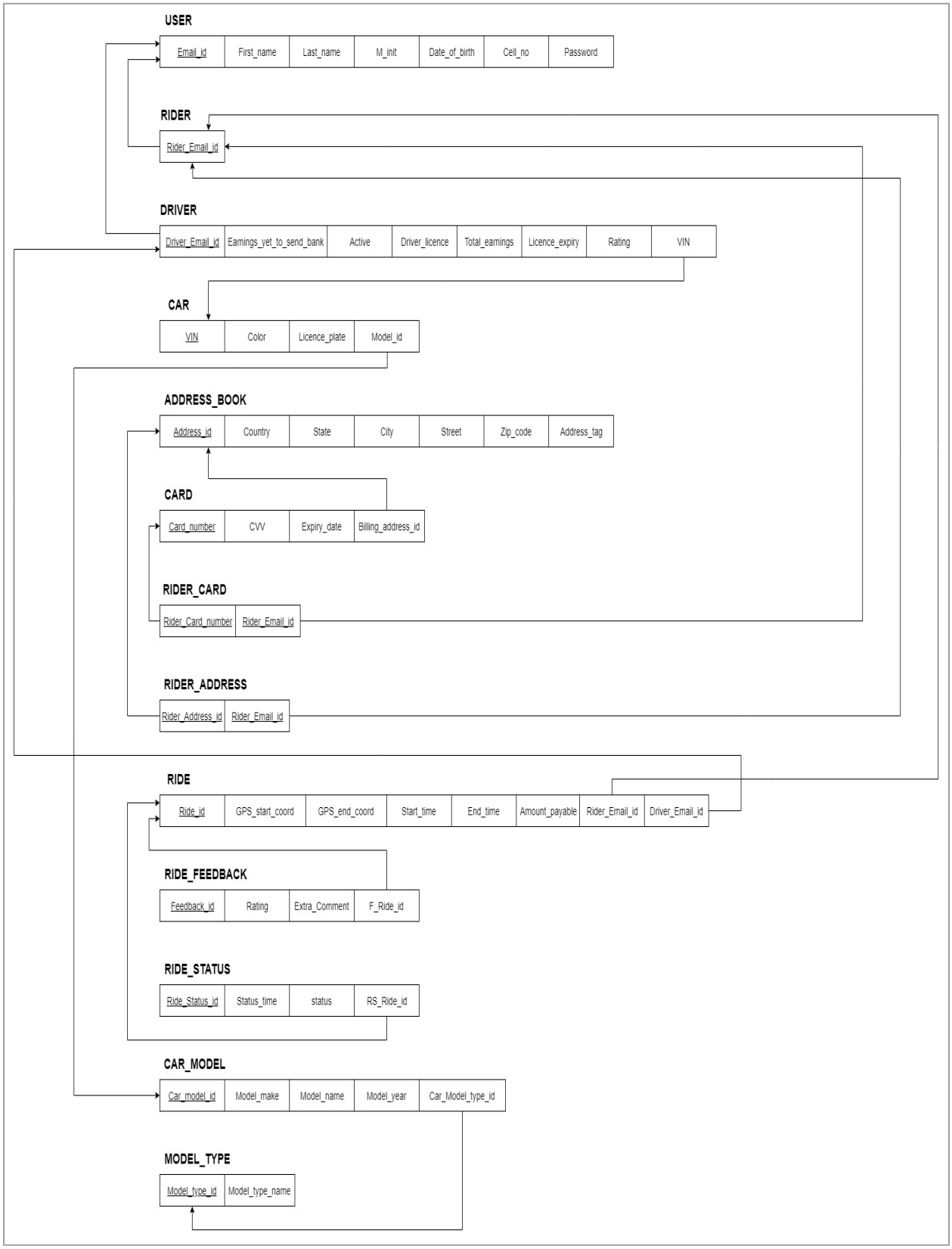
So, in order to increase the efficiency and reduce the redundancy we need to normalize the relation into 3rd Normal form by removing the Transitive Dependencies and create new tables as following –

CAR: (**VIN**, License_plate, color, Model_id)

CAR_MODEL: (**Car_model_id**, Model_type_id, Model_name, Model_year, Model_make)

MODEL_TYPE: (**Model_type_id**, Model_type_name)

Relational Schema After Normalization



Entities for Uber database after normalization

- **User**
- **Driver**
- **Rider**
- **Card**
- **Ride**
- **Ride_Status**
- **Feedback**
- **Car**
- **Rider Address**
- **Rider_Card**
- **Car_Model**
- **Model_Type**
- **Address_Book**

SQL statements to create relations in database and add constraints:

CREATE TABLE QUERIES: (TO BE EXECUTED FIRST)

```
DROP TABLE uuser;
CREATE TABLE uuser
  (
     email_id      VARCHAR(32) NOT NULL,
     password      VARCHAR(32) NOT NULL,
     cell_no       VARCHAR(16) NOT NULL,
     first_name    VARCHAR(32) NOT NULL,
     minit         VARCHAR(1),
     date_of_birth DATE,
     PRIMARY KEY (email_id)
  );

DROP TABLE rider;
CREATE TABLE rider
  (
     rider_email_id VARCHAR(32) NOT NULL,
     PRIMARY KEY (rider_email_id)
  );

DROP TABLE driver;
CREATE TABLE driver
  (
     driver_email_id           VARCHAR(32) NOT NULL,
     earnings_yet_to_send_bank INTEGER,
     active                    VARCHAR(1),
     driver_licence            VARCHAR(16) NOT NULL,
     total_earnings            INTEGER,
     licence_expiry            DATE NOT NULL,
     rating                    INTEGER,
     vin                       VARCHAR(32),
     PRIMARY KEY (driver_email_id)
  );

DROP TABLE car;
CREATE TABLE car
  (
     vin           VARCHAR(32) NOT NULL,
     color         VARCHAR(16) NOT NULL,
     licence_plate VARCHAR(6) NOT NULL,-
- Licence plates are 6 characters always
     model_id      INTEGER NOT NULL,
```

```
    PRIMARY KEY (vin)
  );

DROP TABLE car_model;
CREATE TABLE car_model
  (
    car_model_id      INTEGER NOT NULL,
    model_make        VARCHAR(32) NOT NULL,
    model_name        VARCHAR(32) NOT NULL,
    model_year        VARCHAR(4),
    car_model_type_id INTEGER NOT NULL,
    PRIMARY KEY (car_model_id)
  );

DROP TABLE model_type;
CREATE TABLE model_type
  (
    model_type_id   INTEGER NOT NULL,
    model_type_name VARCHAR(32) NOT NULL,
    PRIMARY KEY (model_type_id)
  );

DROP TABLE address_book;
CREATE TABLE address_book
  (
    address_id  INTEGER,
    country     VARCHAR(32) NOT NULL,
    state       VARCHAR(32) NOT NULL,
    city        VARCHAR(32) NOT NULL,
    street      VARCHAR(32) NOT NULL,
    zip_code    VARCHAR(5) NOT NULL,-
- Zip codes are max 5 characters
    address_tag VARCHAR(16) NOT NULL,
    PRIMARY KEY (address_id)
  );

DROP TABLE card;
CREATE TABLE card
  (
    card_number        VARCHAR(16) NOT NULL,
    cvv                VARCHAR(3) NOT NULL,-
- CVV is always 3 characters
    expiry_date        DATE NOT NULL,
    billing_address_id INTEGER NOT NULL,
    PRIMARY KEY (card_number)
```

```
  );

DROP TABLE rider_card;
CREATE TABLE rider_card
  (
     rider_card_number VARCHAR(16) NOT NULL,
     rider_email_id    VARCHAR(32) NOT NULL,
     PRIMARY KEY (rider_card_number, rider_email_id)
  );

DROP TABLE rider_address;
CREATE TABLE rider_address
  (
     rider_address_id INTEGER NOT NULL,
     rider_email_id   VARCHAR(32) NOT NULL,
     PRIMARY KEY (rider_address_id, rider_email_id)
  );

DROP TABLE ride;
CREATE TABLE ride
  (
     ride_id          INTEGER NOT NULL,
     gps_start_coord VARCHAR(16) NOT NULL,
     gps_end_coord   VARCHAR(16) NOT NULL,
     start_time      TIMESTAMP,
     end_time        TIMESTAMP,
     amount_payable  INTEGER,
     rider_email_id  VARCHAR(32) NOT NULL,
     driver_email_id VARCHAR(32) NOT NULL,
     PRIMARY KEY (ride_id)
  );

DROP TABLE ride_feedback;
CREATE TABLE ride_feedback
  (
     feedback_id   INTEGER NOT NULL,
     rating        INTEGER NOT NULL,
     extra_comment VARCHAR(32),
     f_ride_id     INTEGER NOT NULL,
     PRIMARY KEY (feedback_id)
  );

DROP TABLE ride_status;
CREATE TABLE ride_status
  (
```

```
    ride_status_id INTEGER NOT NULL,
    status_time    TIMESTAMP NOT NULL,
    status         VARCHAR(1),
    -
- I for incomplete, C for cancelled , O for ongoing, D for Completed
    rs_ride_id     INTEGER,
    PRIMARY KEY (ride_status_id)
);
```

ALTER TABLE QUERIES: (RUN AFTER EXECUTING CREATE QUERIES)

```
ALTER TABLE rider
ADD CONSTRAINT rider_user FOREIGN KEY(rider_email_id) REFERENCES uuser
(email_id) ON DELETE CASCADE;

ALTER TABLE driver
ADD CONSTRAINT driver_user FOREIGN KEY(driver_email_id) REFERENCES uus
er(email_id) ON DELETE CASCADE;

ALTER TABLE driver
ADD CONSTRAINT driver_car FOREIGN KEY(vin) REFERENCES car(vin) ON DELE
TE CASCADE;

ALTER TABLE car
ADD CONSTRAINT car_model FOREIGN KEY(model_id) REFERENCES car_model(
car_model_id) ON DELETE CASCADE;

ALTER TABLE car_model
ADD CONSTRAINT model_model_type FOREIGN KEY(car_model_type_id) REFEREN
CES model_type(model_type_id) ON DELETE CASCADE;

ALTER TABLE card
ADD CONSTRAINT card_billing_address FOREIGN KEY(billing_address_id) RE
FERENCES address_book(address_id) ON DELETE CASCADE;

ALTER TABLE rider_card
ADD CONSTRAINT rider_rider_card FOREIGN KEY(rider_email_id) REFERENCES
rider(rider_email_id) ON DELETE CASCADE;

ALTER TABLE rider_card
ADD CONSTRAINT card_rider_card FOREIGN KEY(rider_card_number) REFERENC
ES card(card_number) ON DELETE CASCADE;
```

```
ALTER TABLE rider_address
ADD CONSTRAINT rider_rider_address FOREIGN KEY(rider_email_id) REFEREN
CES rider(rider_email_id) ON DELETE CASCADE;

ALTER TABLE rider_address
ADD CONSTRAINT card_rider_address FOREIGN KEY(rider_address_id) REFERE
NCES address_book(address_id) ON DELETE CASCADE;

ALTER TABLE ride
ADD CONSTRAINT ride_rider FOREIGN KEY(rider_email_id) REFERENCES rider
(rider_email_id) ON DELETE CASCADE;

ALTER TABLE ride
ADD CONSTRAINT ride_driver FOREIGN KEY(driver_email_id) REFERENCES dri
ver(driver_email_id) ON DELETE CASCADE;

ALTER TABLE ride_feedback
ADD CONSTRAINT ride_feedback_ride FOREIGN KEY(f_ride_id) REFERENCES ri
de(ride_id) ON DELETE CASCADE;

ALTER TABLE ride_status
ADD CONSTRAINT ride_status_ride FOREIGN KEY(rs_ride_id) REFERENCES rid
e(ride_id) ON DELETE CASCADE;
```

PL/SQL Procedures

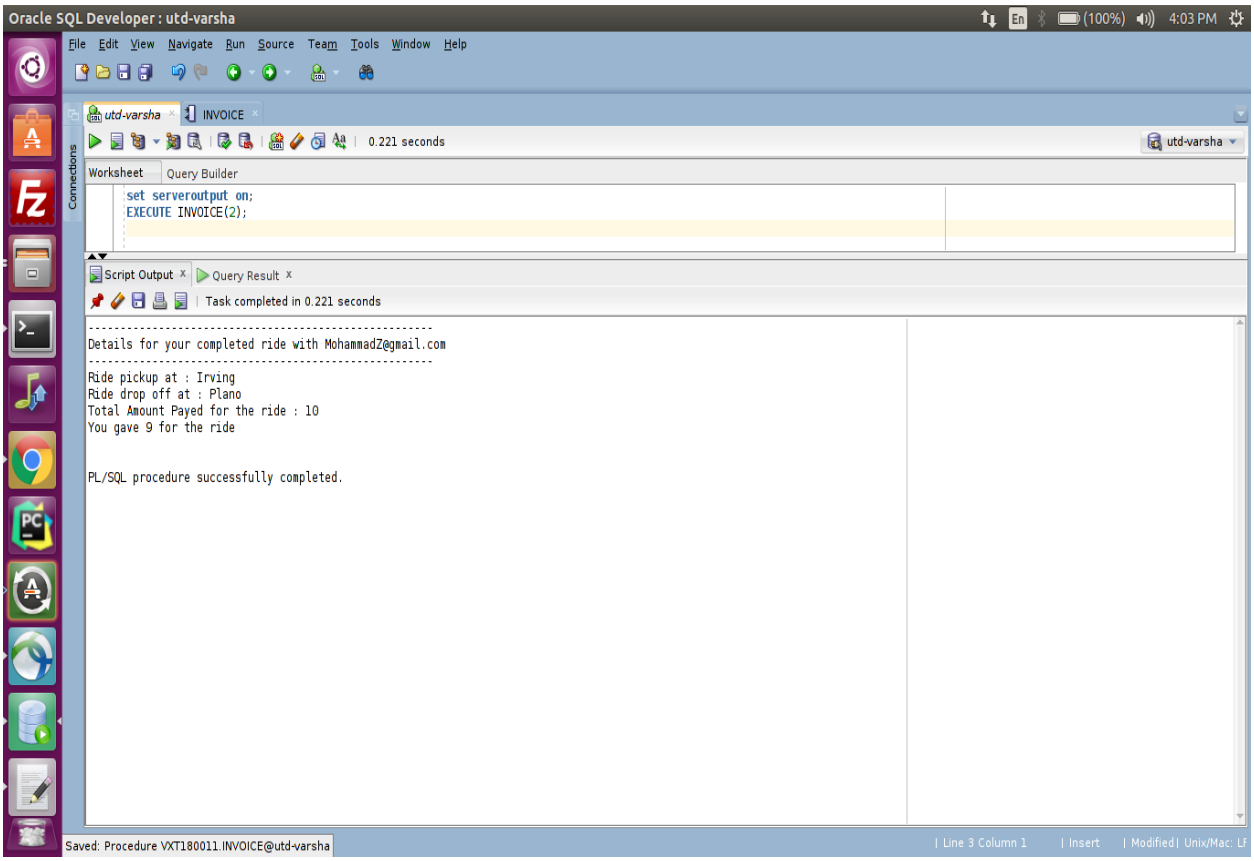
PROCEDURE – 1 (RETRIEVING THE INVOICE OF USER WHO PLACED RIDE WITH THE DRIVER)

It is a procedure which retrieves the invoice of the user for the ride completed and displays ratings to the driver based on the user feedback.

```
CREATE OR replace PROCEDURE Invoice(ride_id IN INTEGER)
AS
  amount           ride.amount_payable%TYPE;
  pickup_location  ride.gps_start_coord%TYPE;
  dropoff_location ride.gps_end_coord%TYPE;
  rider_email      ride.rider_email_id%TYPE;
  driver_email     ride.driver_email_id%TYPE;
  rating           ride_feedback.rating%TYPE;
  CURSOR rideforid IS
    SELECT R.amount_payable,
           R.gps_start_coord,
           R.gps_end_coord,
           R.rider_email_id,
           R.driver_email_id
    FROM   ride R,
           ride_status RS
    WHERE  R.ride_id = RS.rs_ride_id
           AND R.ride_id = ride_id
           AND RS.status = 'D';
  -- Getting only completed ride status ride details
  CURSOR ridefeedbackforrideid IS
    SELECT RF.rating
    FROM   ride_feedback RF
    WHERE  RF.f_ride_id = ride_id;
BEGIN
    OPEN rideforid;

    OPEN ridefeedbackforrideid;

    LOOP
        FETCH rideforid INTO amount, pickup_location, dropoff_location
,
        rider_email,
        driver_email;
```

```
        EXIT WHEN ( rideforid%NOTFOUND );

        FETCH ridefeedbackforrideid INTO rating;

dbms_output.Put_line('-----------------------------------------');

dbms_output.Put_line('Details for your completed ride with '
                     || driver_email);

dbms_output.Put_line('-----------------------------------------');

dbms_output.Put_line('Ride pickup at : ' || pickup_location);

dbms_output.Put_line('Ride drop off at : ' || dropoff_location);

dbms_output.Put_line('Total Amount Payed for the ride : ' || amount);

IF ridefeedbackforrideid%FOUND THEN
  dbms_output.Put_line('You gave ' || rating || ' for the ride');
ELSE
  dbms_output.Put_line('You have not rated this ride yet. Rate now !')
;
END IF;
END LOOP;

CLOSE rideforid;

CLOSE ridefeedbackforrideid;
END invoice;
```

OUTPUT 1:


Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
INVOICE
0.221 seconds
utd-varsha
Worksheet
Query Builder
set serveroutput on;
EXECUTE INVOICE(2);
Script Output
Query Result
Task completed in 0.221 seconds
Details for your completed ride with MohammadZ@gmail.com
Ride pickup at : Irving
Ride drop off at : Plano
Total Amount Payed for the ride : 10
You gave 9 for the ride
PL/SQL procedure successfully completed.
Saved: Procedure VXT180011.INVOICE@utd-varsha
Line 3 Column 1
Insert
Modified
Unix/Mac: LF

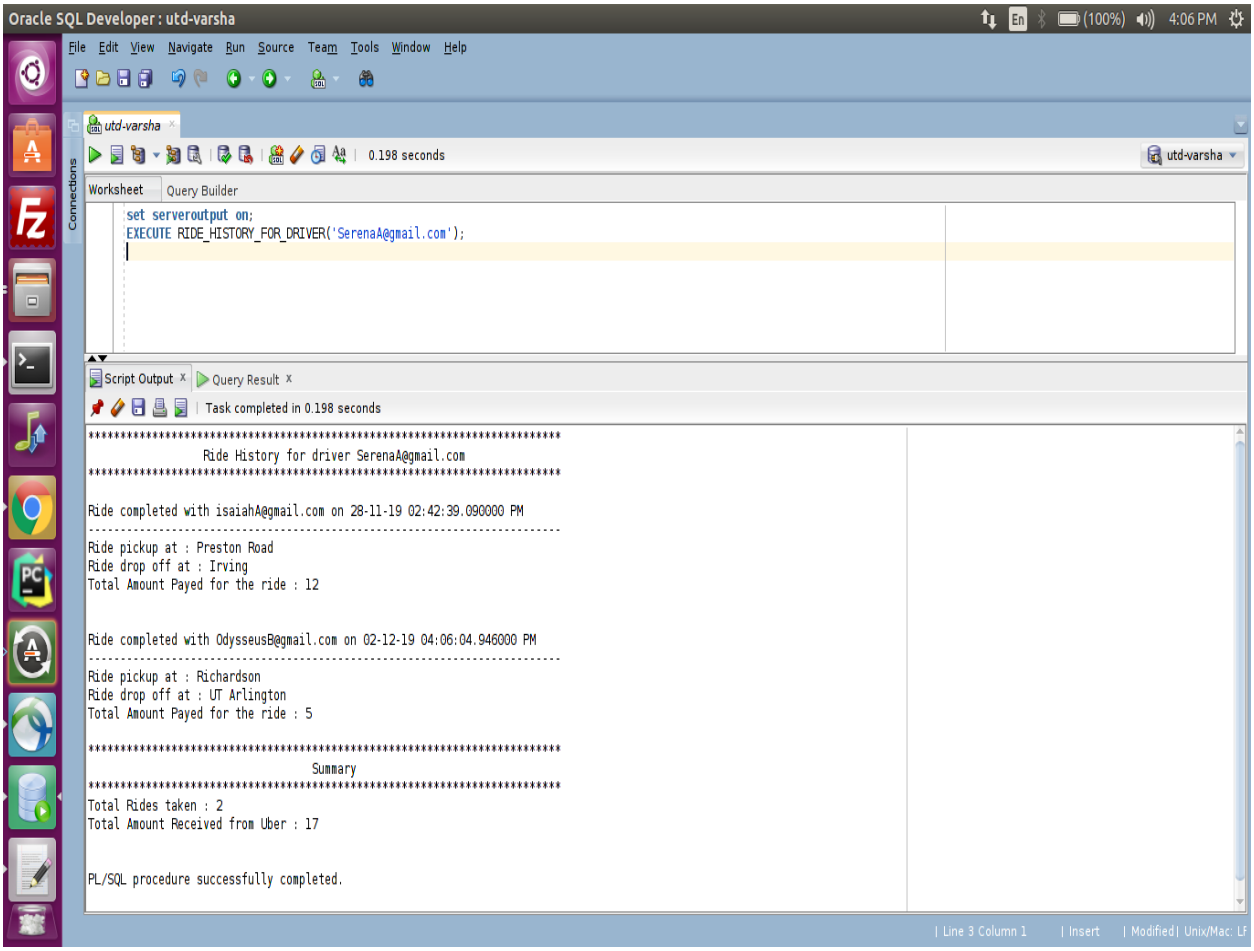
PROCEDURE – 2 (Generating ride history for Rider)

This procedure is about generating the ride history data from procedure and providing a summary of all the ride history to the rider which is user itself.

```
CREATE OR replace PROCEDURE Ride_history_for_rider(rider_email IN VARC
HAR)
AS
  ride_history       ride%ROWTYPE;
  total_amount_payed INTEGER;
  total_rides_taken  INTEGER;
  CURSOR ridesforrider IS
    SELECT *
    FROM   ride R
    WHERE  R.rider_email_id = rider_email;
-- Getting all the rides for a given rider
BEGIN
    total_rides_taken := 0;

    total_amount_payed := 0;

    OPEN ridesforrider;

dbms_output.Put_line('*********************************************');

dbms_output.Put_line('                    Ride History for '
                     || rider_email);

dbms_output.Put_line('*********************************************');

LOOP
    FETCH ridesforrider INTO ride_history;

    EXIT WHEN ( ridesforrider%NOTFOUND );

    total_rides_taken := total_rides_taken + 1;

    total_amount_payed := total_amount_payed
                          + ride_history.amount_payable;

    dbms_output.Put_line('');

    dbms_output.Put_line('Ride completed with '
                         || ride_history.driver_email_id
```

```
                        || ' on '
                        || ride_history.end_time);

    dbms_output.Put_line('----------------------------------------');

    dbms_output.Put_line('Ride pickup at : '
                        || ride_history.gps_start_coord);

    dbms_output.Put_line('Ride drop off at : '
                        || ride_history.gps_end_coord);

    dbms_output.Put_line('Total Amount Payed for the ride : '
                        || ride_history.amount_payable);

    dbms_output.Put_line('');
END LOOP;

CLOSE ridesforrider;

dbms_output.Put_line('********************************************');

dbms_output.Put_line('Summary');

dbms_output.Put_line('********************************************');

dbms_output.Put_line('Total Rides taken : '
                    || total_rides_taken);

dbms_output.Put_line('Total Amount Payed to Uber : '
                    || total_amount_payed);
END ride_history_for_rider;
```

OUTPUT 2:


Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
0.198 seconds
utd-varsha
Worksheet Query Builder
set serveroutput on;
EXECUTE RIDE_HISTORY_FOR_DRIVER('SerenaA@gmail.com');
Script Output
Query Result
Task completed in 0.198 seconds
Ride History for driver SerenaA@gmail.com
Ride completed with isaiahA@gmail.com on 28-11-19 02:42:39.090000 PM
Ride pickup at : Preston Road
Ride drop off at : Irving
Total Amount Payed for the ride : 12
Ride completed with OdysseusB@gmail.com on 02-12-19 04:06:04.946000 PM
Ride pickup at : Richardson
Ride drop off at : UT Arlington
Total Amount Payed for the ride : 5
Summary
Total Rides taken : 2
Total Amount Received from Uber : 17
PL/SQL procedure successfully completed.

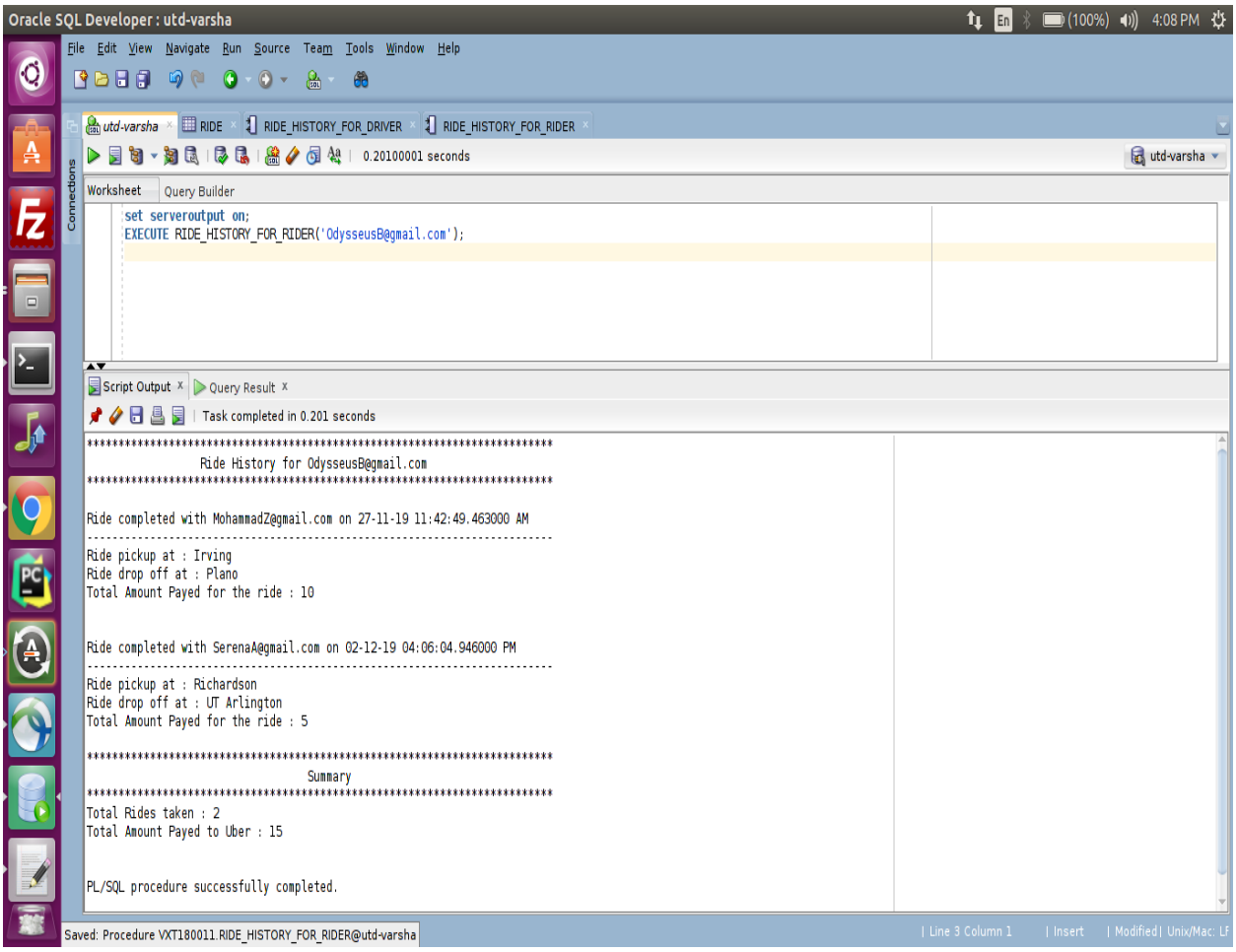
PROCEDURE - 3 (Generating ride history for Driver)

This procedure is about generating the ride history data from procedure and providing a summary of all the ride history to the driver.

```
CREATE OR replace PROCEDURE Ride_history_for_driver(driver_email
 IN VARCHAR)
AS
  ride_history          ride%ROWTYPE;
  total_amount_received INTEGER;
  total_rides_taken     INTEGER;
  CURSOR ridesfordriver IS
    SELECT *
    FROM   ride R
    WHERE  R.driver_email_id = driver_email;
-- Getting all the rides for a given rider
BEGIN
    total_rides_taken := 0;

    total_amount_received := 0;

    OPEN ridesfordriver;

dbms_output.Put_line('**********************************************
*******************************');

dbms_output.Put_line('                    Ride History for driver
'
                     || driver_email);

dbms_output.Put_line('**********************************************
*******************************');

LOOP
    FETCH ridesfordriver INTO ride_history;

    EXIT WHEN ( ridesfordriver%NOTFOUND );

    total_rides_taken := total_rides_taken + 1;

    total_amount_received := total_amount_received
                             + ride_history.amount_payable;
```

```
    dbms_output.Put_line('');

    dbms_output.Put_line('Ride completed with '
                         || ride_history.rider_email_id
                         || ' on '
                         || ride_history.end_time);

dbms_output.Put_line('-----------------------------------------
--------------------------------');

dbms_output.Put_line('Ride pickup at : '
                     || ride_history.gps_start_coord);

dbms_output.Put_line('Ride drop off at : '
                     || ride_history.gps_end_coord);

dbms_output.Put_line('Total Amount Payed for the ride : '
                     || ride_history.amount_payable);

dbms_output.Put_line('');
END LOOP;

CLOSE ridesfordriver;

dbms_output.Put_line('*****************************************
*******************************');

dbms_output.Put_line('                                        Summary
');

dbms_output.Put_line('*****************************************
*******************************');

dbms_output.Put_line('Total Rides taken : '
                     || total_rides_taken);

dbms_output.Put_line('Total Amount Received from Uber : '
                     || total_amount_received);
END ride_history_for_driver;
```

OUTPUT 3:


Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
RIDE
RIDE_HISTORY_FOR_DRIVER
RIDE_HISTORY_FOR_RIDER
0.20100001 seconds
Worksheet
Query Builder
set serveroutput on;
EXECUTE RIDE_HISTORY_FOR_RIDER('OdysseusB@gmail.com');
Script Output
Query Result
Task completed in 0.201 seconds
Ride History for OdysseusB@gmail.com
Ride completed with MohammadZ@gmail.com on 27-11-19 11:42:49.463000 AM
Ride pickup at : Irving
Ride drop off at : Plano
Total Amount Payed for the ride : 10
Ride completed with SerenaA@gmail.com on 02-12-19 04:06:04.946000 PM
Ride pickup at : Richardson
Ride drop off at : UT Arlington
Total Amount Payed for the ride : 5
Summary
Total Rides taken : 2
Total Amount Payed to Uber : 15
PL/SQL procedure successfully completed.
Saved: Procedure VXT180011.RIDE_HISTORY_FOR_RIDER@utd-varsha

PL/SQL TRIGGERS

The following triggers are used to implement different requirements –

Trigger – 1 (Insert/Update Rating)

Each time when the user takes the ride then the insert trigger will execute. It displays the new ratings given by the user to the driver along with the average ratings for the driver.

Trigger for Insert

```
CREATE OR replace TRIGGER driver_rating_feedback_insert
FOR INSERT on RIDE_FEEDBACK
COMPOUND TRIGGER
      driver_email          ride.driver_email_id%type;
      TYPE rating_t         IS TABLE OF ride_feedback.rating%type;
      SUM_OF_RATINGS        rating_t;
      TYPE integer_table    IS TABLE OF integer;
      NUMBER_OF_RATINGS     integer_table;
      TYPE driver_email_t IS TABLE OF ride.driver_email_id%type;
      DRIVER_EMAIL_IDS      driver_email_t;


      TYPE driver_ride_ratings_t

      IS TABLE OF ride_feedback.rating%type INDEX BY ride.driver_
      email_id%type;
      DRIVER_RIDE_SUM_OF_RATINGS     driver_ride_ratings_t;
      TYPE driver_count_ride_t
      IS TABLE OF integer INDEX BY ride.driver_email_id%type;
      DRIVER_RIDE_COUNT          driver_count_ride_t;
      AVERAGE_RATING                  ride_feedback.rating%type;
      BEFORE statement IS
      BEGIN
              SELECT       Nvl(Sum(rf.rating),0),
                           Nvl(Count(rf.feedback_id),0),
                     r.driver_email_id
```

```
            bulk collect
            INTO            sum_of_ratings,
                            number_of_ratings,
                            driver_email_ids
            FROM            ride r
            LEFT OUTER JOIN ride_feedback rf
            ON              r.ride_id=rf.f_ride_id
            GROUP BY        r.driver_email_id;FOR j IN 1..dri
            ver_email_ids.count()

      loop driver_ride_sum_of_ratings(driver_email_ids(j)) :
            = sum_of_ratings(j);

            DRIVER_RIDE_COUNT(driver_email_ids(j)) := number_
            of_ratings(j);
      END loop;
END
before statement;
AFTER each row IS
      BEGIN
        dbms_output.put_line('IN INSERT' || :new.rating || :
      NEW.f_ride_id);
       SELECT driver_email_id
            INTO   driver_email
            FROM   ride
            WHERE  ride_id = :NEW.f_ride_id;
      dbms_output.put_line(driver_ride_count(driver_email));
      AVERAGE_RATING := (driver_ride_sum_of_ratings(driver_e
      mail) + :NEW.rating)/ (driver_ride_count(driver_email)
       + 1);
      dbms_output.put_line('Updating rating for driver ' ||
      driver_email || '...' || average_rating);
```

```
        UPDATE driver
          SET    rating = average_rating
          WHERE  driver_email_id = driver_email;
           dbms_output.put_line('Rating updated for ' || driver
        _email || 'to be ' || average_rating);
      END after each row;
    END;
```

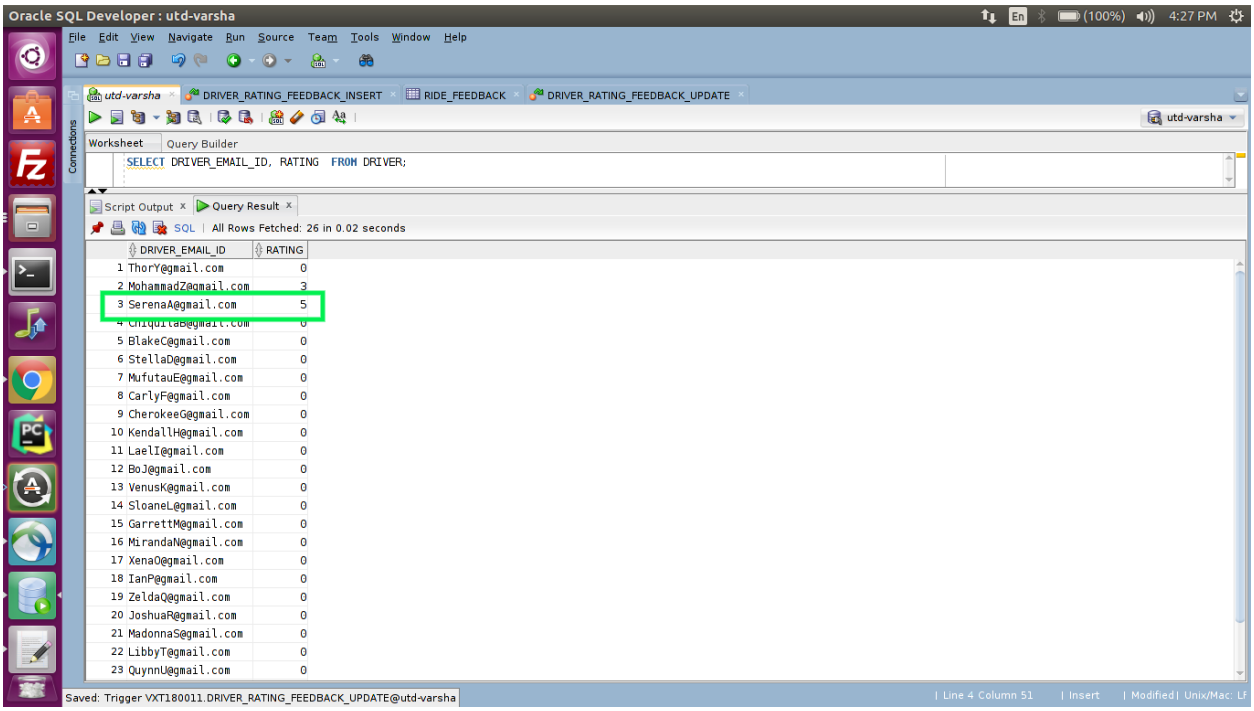
Trigger for Update

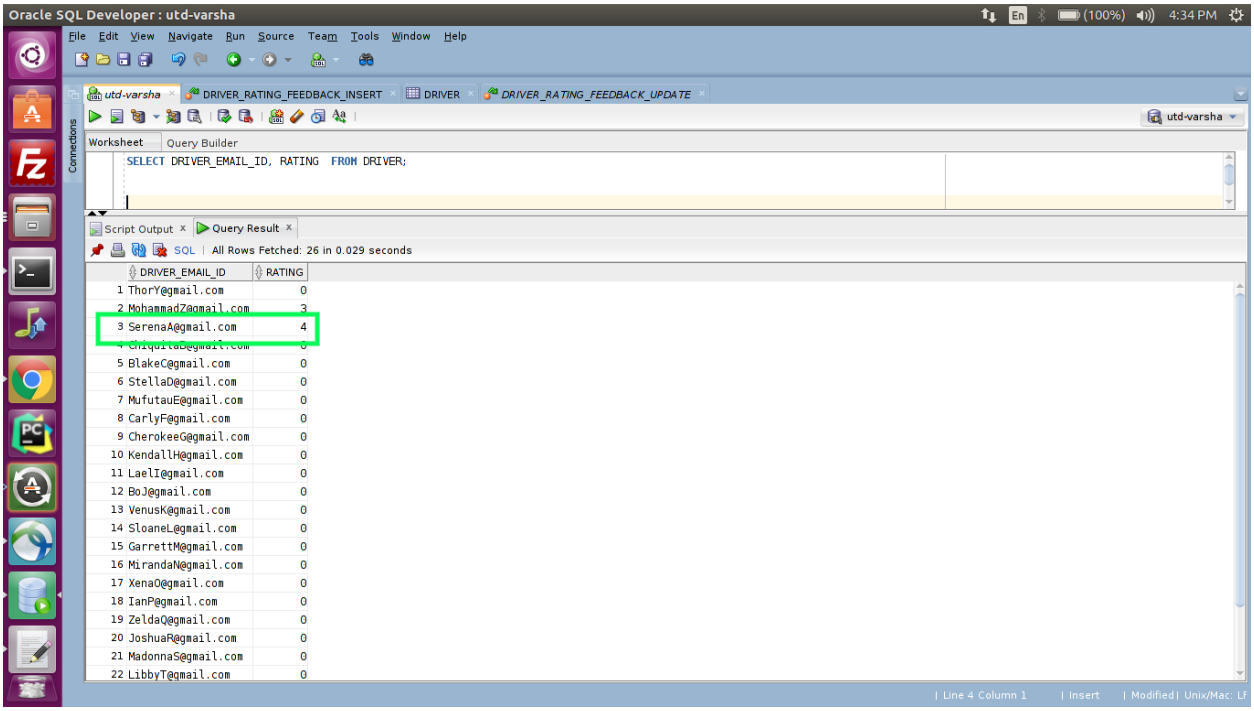
```
CREATE OR replace TRIGGER driver_rating_feedback_update
FOR UPDATE OF rating ON ride_feedback
COMPOUND TRIGGER
    driver_email        ride.driver_email_id%type;
    TYPE rating_t       IS TABLE OF ride_feedback.rating%type;
    SUM_OF_RATINGS      rating_t;
    TYPE integer_table  IS TABLE OF integer;
    NUMBER_OF_RATINGS   integer_table;

    TYPE driver_email_t         IS TABLE OF ride.driver_email_id
    %type;
    DRIVER_EMAIL_IDS            driver_email_t;
    TYPE driver_ride_ratings_t  IS TABLE OF ride_feedback.ratin
g%type INDEX BY ride.driver_email_id%type;
    DRIVER_RIDE_SUM_OF_RATINGS  driver_ride_ratings_t;
    TYPE driver_count_ride_t     IS TABLE OF integer INDEX BY ri
de.driver_email_id%type;
    DRIVER_RIDE_COUNT            driver_count_ride_t;
    AVERAGE_RATING               ride_feedback.rating%type;
    BEFORE statement IS
        BEGIN
```

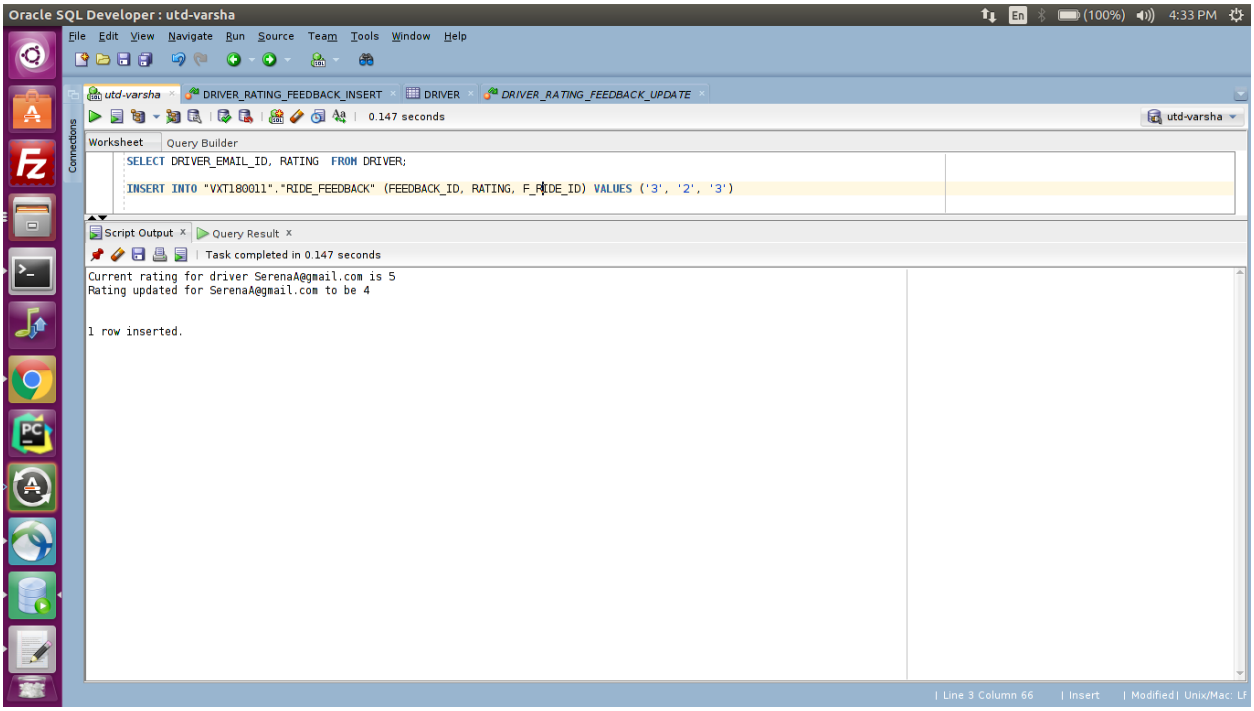
```
SELECT    Sum(rf.rating),
          Count(r.driver_email_id),
          r.driver_email_id

           bulk collect INTO sum_of_ratings,
       number_of_ratings,driver_email_ids
       FROM     ride r, ride_feedback rf
       WHERE    r.ride_id=rf.f_ride_id
       GROUP BY r.driver_email_id;
     FOR j IN 1..driver_email_ids.count() loop

               driver_ride_sum_of_ratings(driver_email_ids(
           j)) := sum_of_ratings(j);

           DRIVER_RIDE_COUNT(driver_email_ids(j)) := number_
           of_ratings(j);
     END loop;
END before statement;
AFTER each row IS
      BEGIN
           SELECT driver_email_id
                INTO   driver_email
                FROM   ride
                WHERE  ride.ride_id = :NEW.f_ride_id;

                AVERAGE_RATING := (driver_ride_sum_of_rating
                s(driver_email) -
                :OLD.rating + :NEW.rating)/ (driver_ride_cou
                nt(driver_email));
                dbms_output.put_line('Current rating for dri
                ver ' || driver_email || ' is ' || driver_ri
```

```
                    de_sum_of_ratings(driver_email));UPDATE driv
                    er
                      SET     rating = average_rating
                      WHERE   driver_email_id = driver_email;dbms
                    _output.put_line('Rating updated for ' || dr
                    iver_email || ' to be ' || average_rating);
        END after each row;
END;
```

OUTPUT 1:




Oracle SQL Developer : utd-varsha
4:34 PM
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
DRIVER_RATING_FEEDBACK_INSERT
DRIVER
DRIVER_RATING_FEEDBACK_UPDATE
Worksheet
Query Builder
SELECT DRIVER_EMAIL_ID, RATING FROM DRIVER;
Script Output
Query Result
SQL | All Rows Fetched: 26 in 0.029 seconds
DRIVER_EMAIL_ID
RATING
1 ThorY@gmail.com 0
2 MohammadZ@gmail.com 3
3 SerenaA@gmail.com 4
5 BlakeC@gmail.com 0
6 StellaD@gmail.com 0
7 MufutauE@gmail.com 0
8 CarlyF@gmail.com 0
9 CherokeeG@gmail.com 0
10 KendallH@gmail.com 0
11 LaelI@gmail.com 0
12 BoJ@gmail.com 0
13 VenusK@gmail.com 0
14 SloaneL@gmail.com 0
15 GarrettM@gmail.com 0
16 MirandaN@gmail.com 0
17 XenaO@gmail.com 0
18 IanP@gmail.com 0
19 ZeldaQ@gmail.com 0
20 JoshuaR@gmail.com 0
21 MadonnaS@gmail.com 0
22 LibbyT@gmail.com 0
Line 4 Column 1
Insert
Modified
Unix/Mac: LF

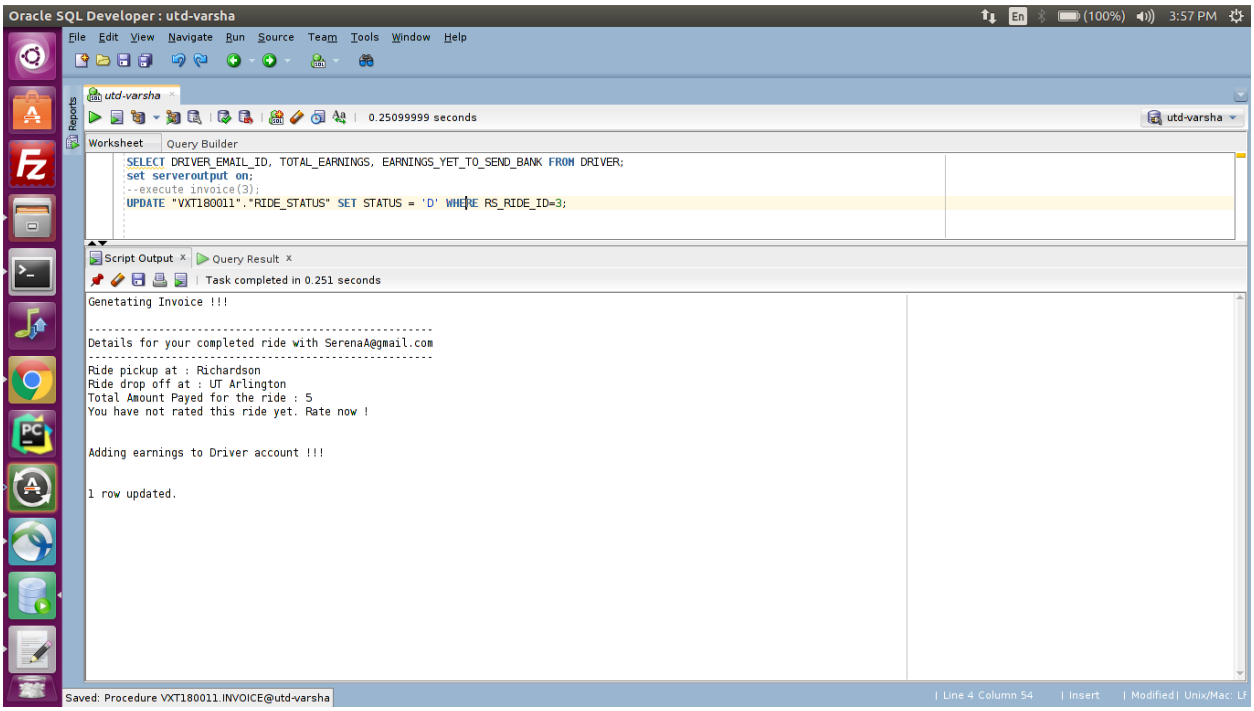

Oracle SQL Developer : utd-varsha
4:33 PM
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
DRIVER_RATING_FEEDBACK_INSERT
DRIVER
DRIVER_RATING_FEEDBACK_UPDATE
0.147 seconds
Worksheet
Query Builder
SELECT DRIVER_EMAIL_ID, RATING FROM DRIVER;
INSERT INTO "VXT180011"."RIDE_FEEDBACK" (FEEDBACK_ID, RATING, F_RIDE_ID) VALUES ('3', '2', '3')
Script Output
Query Result
Task completed in 0.147 seconds
Current rating for driver SerenaA@gmail.com is 5
Rating updated for SerenaA@gmail.com to be 4
1 row inserted.
Line 3 Column 66
Insert
Modified
Unix/Mac: LF

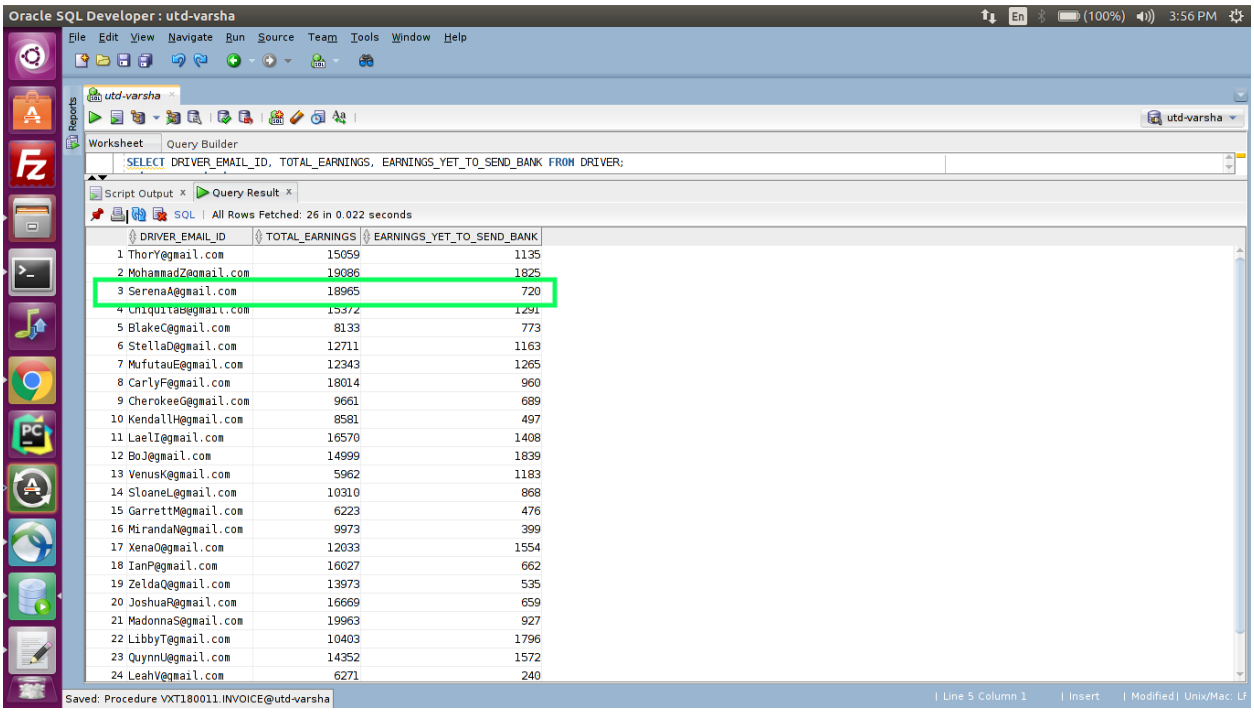
Trigger – 2 (On Ride Complete)

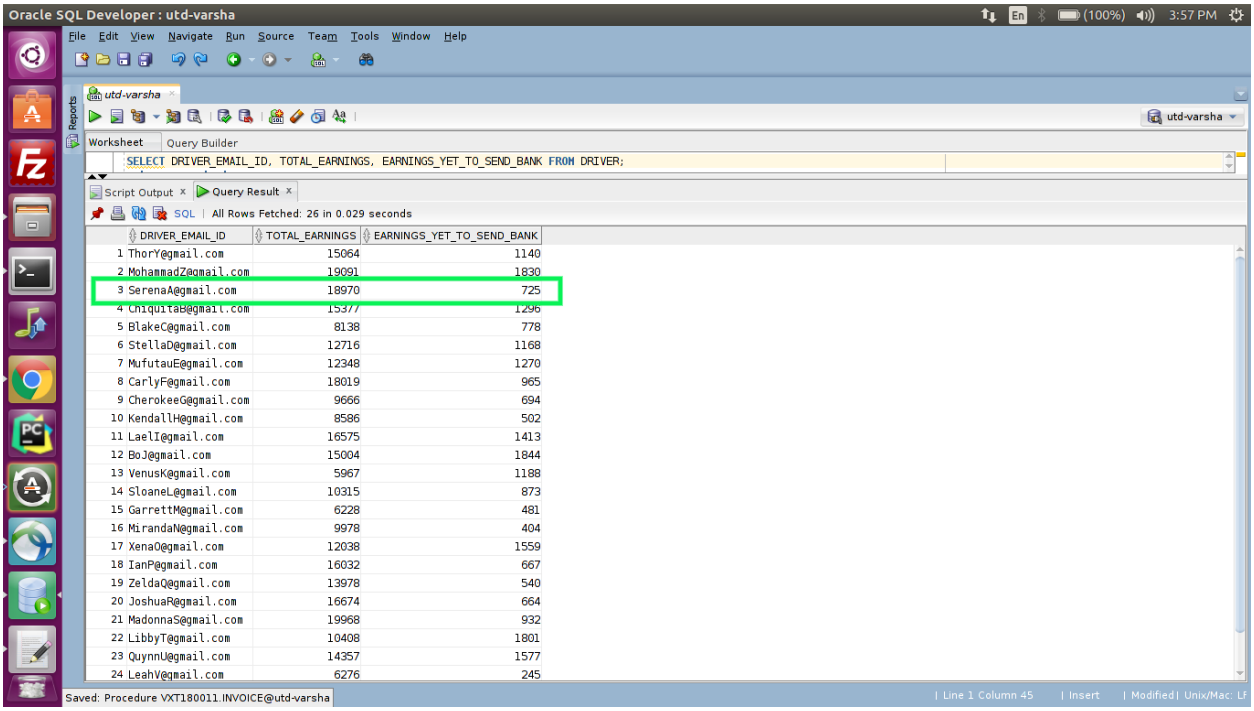
This trigger updates the driver's earnings to their respective accounts.

```
CREATE OR replace TRIGGER on_ride_complete
  AFTER UPDATE OF status ON ride_status
  FOR EACH ROW
DECLARE
    ride_amount ride.amount_payable%TYPE;
BEGIN
    IF :NEW.status != :OLD.status
       AND :NEW.status = 'D' THEN
      dbms_output.Put_line('Genetating Invoice !!!');

      dbms_output.Put_line('');

      Invoice(:NEW.rs_ride_id);

      dbms_output.Put_line('');

      dbms_output.Put_line('');

      dbms_output.Put_line('Adding earnings to Driver account !!! ');

      -- Get ride details in program variable
      SELECT amount_payable
      INTO   ride_amount
      FROM   ride
      WHERE  ride_id = :NEW.rs_ride_id;

      -
- Update earnings to send to bank and total earnings of the driver til
l date for current ride
      UPDATE driver
      SET    earnings_yet_to_send_bank = earnings_yet_to_send_bank +
ride_amount;

      UPDATE driver
      SET    total_earnings = total_earnings + ride_amount;
    END IF;
END;
```

OUTPUT 2:


Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
0.25099999 seconds
Worksheet Query Builder
SELECT DRIVER_EMAIL_ID, TOTAL_EARNINGS, EARNINGS_YET_TO_SEND_BANK FROM DRIVER;
set serveroutput on;
--execute invoice(3);
UPDATE "VXT180011"."RIDE_STATUS" SET STATUS = 'D' WHERE RS_RIDE_ID=3;
Script Output
Query Result
Task completed in 0.251 seconds
Genetating Invoice !!!
Details for your completed ride with SerenaA@gmail.com
Ride pickup at : Richardson
Ride drop off at : UT Arlington
Total Amount Payed for the ride : 5
You have not rated this ride yet. Rate now !
Adding earnings to Driver account !!!
1 row updated.
Saved: Procedure VXT180011.INVOICE@utd-varsha
Line 4 Column 54
Insert
Modified
Unix/Mac: LF


Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
Worksheet Query Builder
SELECT DRIVER_EMAIL_ID, TOTAL_EARNINGS, EARNINGS_YET_TO_SEND_BANK FROM DRIVER;
Script Output
Query Result
SQL
All Rows Fetched: 26 in 0.022 seconds
DRIVER_EMAIL_ID TOTAL_EARNINGS EARNINGS_YET_TO_SEND_BANK
1 ThorY@gmail.com 15059 1135
2 MohammadZ@gmail.com 19086 1825
3 SerenaA@gmail.com 18965 720
4 ChiquitaB@gmail.com 15372 1291
5 BlakeC@gmail.com 8133 773
6 StellaD@gmail.com 12711 1163
7 MufutauE@gmail.com 12343 1265
8 CarlyF@gmail.com 18014 960
9 CherokeeG@gmail.com 9661 689
10 KendallH@gmail.com 8581 497
11 LaelI@gmail.com 16570 1408
12 BoJ@gmail.com 14999 1839
13 VenusK@gmail.com 5962 1183
14 SloaneL@gmail.com 10310 868
15 GarrettM@gmail.com 6223 476
16 MirandaN@gmail.com 9973 399
17 XenaO@gmail.com 12033 1554
18 IanP@gmail.com 16027 662
19 ZeldaQ@gmail.com 13973 535
20 JoshuaR@gmail.com 16669 659
21 MadonnaS@gmail.com 19963 927
22 LibbyT@gmail.com 10403 1796
23 QuynnU@gmail.com 14352 1572
24 LeahV@gmail.com 6271 240
Saved: Procedure VXT180011.INVOICE@utd-varsha
Line 5 Column 1
Insert
Modified
Unix/Mac: LF

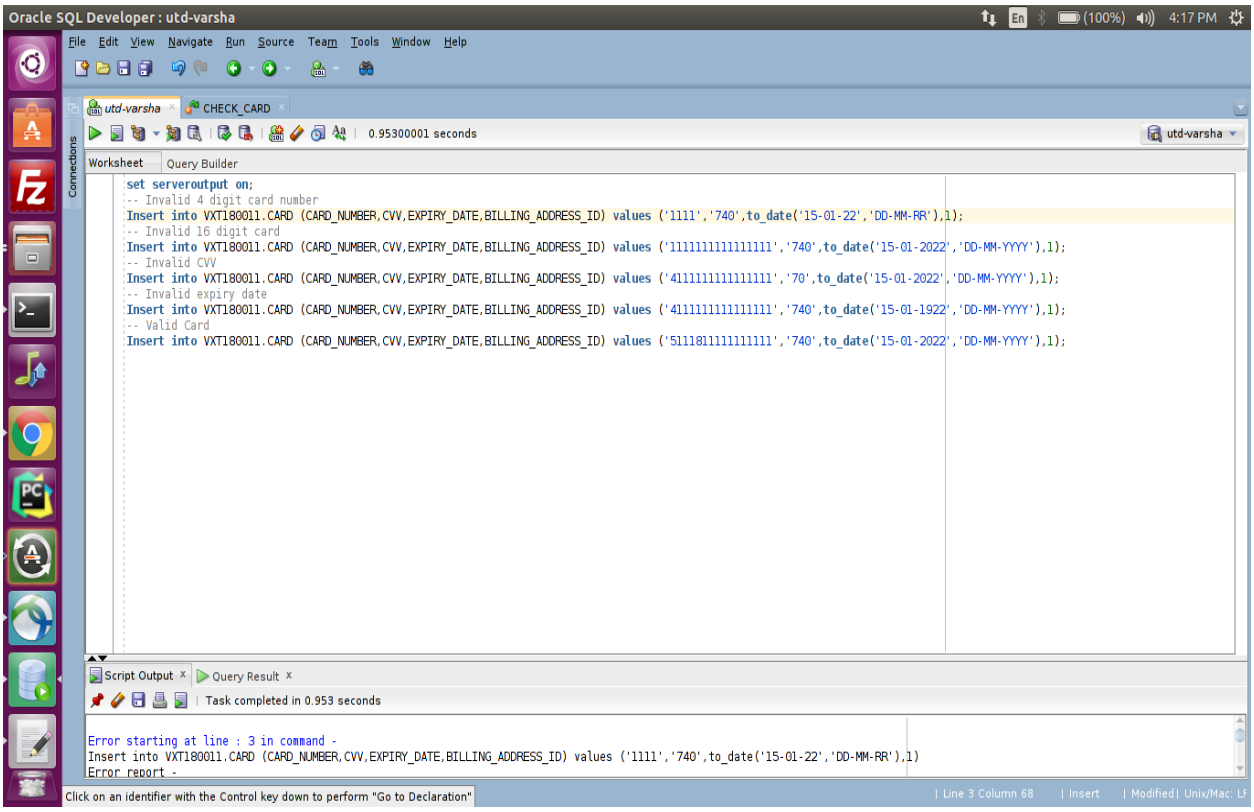

Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
Worksheet Query Builder
SELECT DRIVER_EMAIL_ID, TOTAL_EARNINGS, EARNINGS_YET_TO_SEND_BANK FROM DRIVER;
Script Output
Query Result
SQL | All Rows Fetched: 26 in 0.029 seconds
DRIVER_EMAIL_ID TOTAL_EARNINGS EARNINGS_YET_TO_SEND_BANK
1 ThorY@gmail.com 15064 1140
2 MohammadZ@gmail.com 19091 1830
3 SerenaA@gmail.com 18970 725
4 ChiquitaB@gmail.com 15377 1296
5 BlakeC@gmail.com 8138 778
6 StellaD@gmail.com 12716 1168
7 MufutauE@gmail.com 12348 1270
8 CarlyF@gmail.com 18019 965
9 CherokeeG@gmail.com 9666 694
10 KendallH@gmail.com 8586 502
11 LaelI@gmail.com 16575 1413
12 BoJ@gmail.com 15004 1844
13 VenusK@gmail.com 5967 1188
14 SloaneL@gmail.com 10315 873
15 GarrettM@gmail.com 6228 481
16 MirandaN@gmail.com 9978 404
17 XenaO@gmail.com 12038 1559
18 IanP@gmail.com 16032 667
19 ZeldaQ@gmail.com 13978 540
20 JoshuaR@gmail.com 16674 664
21 MadonnaS@gmail.com 19968 932
22 LibbyT@gmail.com 10408 1801
23 QuynnU@gmail.com 14357 1577
24 LeahV@gmail.com 6276 245
Saved: Procedure VXT180011.INVOICE@utd-varsha
Line 1 Column 45 Insert Modified Unix/Mac: LF

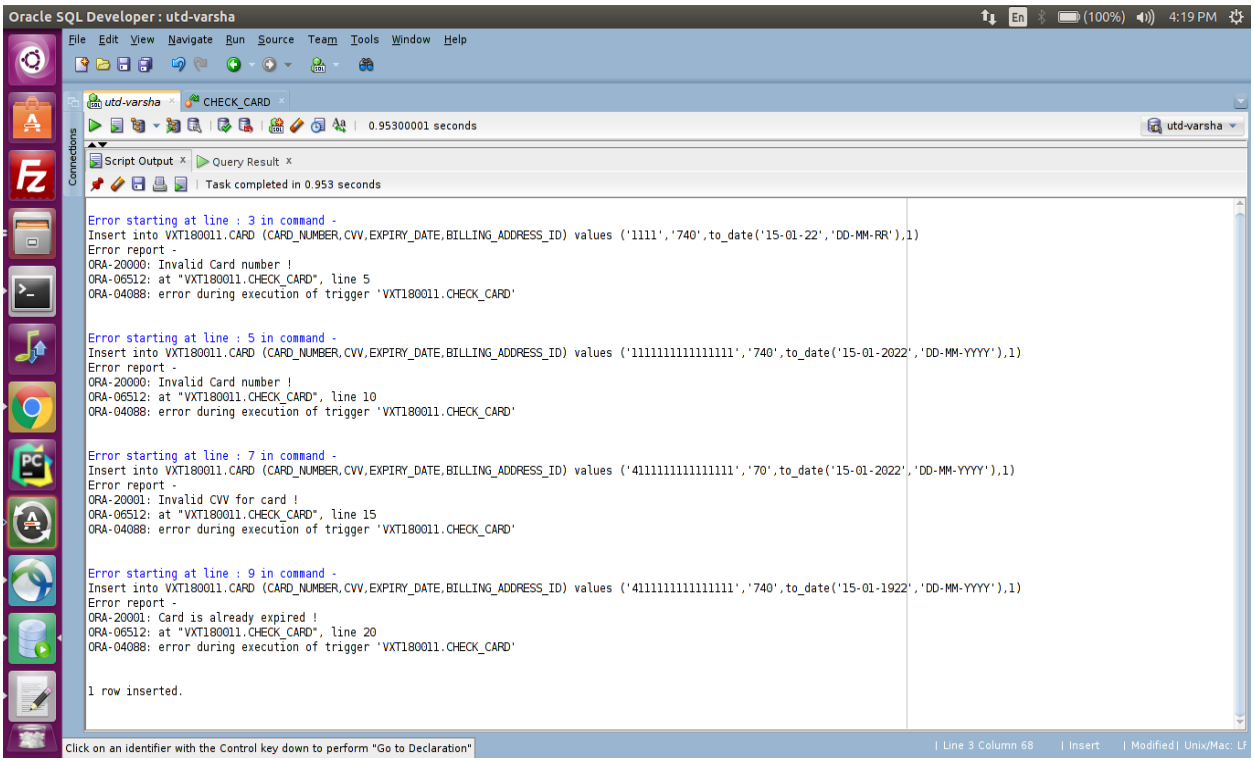
Trigger – 3 (Check Card Details)

This trigger checks for the authenticity of the card used by the user(rider) for the payment of the ride.

```
CREATE OR replace TRIGGER check_card
  BEFORE INSERT OR UPDATE OF card_number, cvv, expiry_date ON ca
rd
  FOR EACH ROW
BEGIN
    -
- Check card number to be between 13 and 16 characters as per no
rms
    IF Length(:NEW.card_number) > 16
        OR Length(:NEW.card_number) < 13 THEN
      Raise_application_error(-20000, 'Invalid Card number !');
    END IF;

    -
- Check card number to start with 4, 5, 6 or 37 for Visa, Master
 Card, Discover and American Express Card
    IF Substr(:NEW.card_number, 0, 1) NOT IN ( '4', '5', '6' )
       AND Substr(:NEW.card_number, 0, 2) != '37' THEN
      Raise_application_error(-20000, 'Invalid Card number !');
    END IF;

    -- Check CVV to be of length 3
    IF Length(:NEW.cvv) != 3 THEN
      Raise_application_error(-
20001, 'Invalid CVV for card !');
    END IF;

    -- Check expiry date to be not less than today's date
    IF :NEW.expiry_date < Trunc(SYSDATE) THEN
      Raise_application_error(-
20001, 'Card is already expired !');
    END IF;
END;
```

OUTPUT 3:


Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
CHECK_CARD
0.95300001 seconds
Worksheet
Query Builder
set serveroutput on;
-- Invalid 4 digit card number
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('1111','740',to_date('15-01-22','DD-MM-RR'),1);
-- Invalid 16 digit card
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('1111111111111111','740',to_date('15-01-2022','DD-MM-YYYY'),1);
-- Invalid CVV
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('4111111111111111','70',to_date('15-01-2022','DD-MM-YYYY'),1);
-- Invalid expiry date
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('4111111111111111','740',to_date('15-01-1922','DD-MM-YYYY'),1);
-- Valid Card
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('5111811111111111','740',to_date('15-01-2022','DD-MM-YYYY'),1);
Script Output
Query Result
Task completed in 0.953 seconds
Error starting at line : 3 in command -
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('1111','740',to_date('15-01-22','DD-MM-RR'),1)
Error report -
Click on an identifier with the Control key down to perform "Go to Declaration"
Line 3 Column 68
Insert
Modified
Unix/Mac: LF


Oracle SQL Developer : utd-varsha
File Edit View Navigate Run Source Team Tools Window Help
utd-varsha
CHECK_CARD
0.95300001 seconds
Script Output
Query Result
Task completed in 0.953 seconds
Error starting at line : 3 in command -
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('1111','740',to_date('15-01-22','DD-MM-RR'),1)
Error report -
ORA-20000: Invalid Card number !
ORA-06512: at "VXT180011.CHECK_CARD", line 5
ORA-04088: error during execution of trigger 'VXT180011.CHECK_CARD'
Error starting at line : 5 in command -
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('1111111111111111','740',to_date('15-01-2022','DD-MM-YYYY'),1)
Error report -
ORA-20000: Invalid Card number !
ORA-06512: at "VXT180011.CHECK_CARD", line 10
ORA-04088: error during execution of trigger 'VXT180011.CHECK_CARD'
Error starting at line : 7 in command -
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('4111111111111111','70',to_date('15-01-2022','DD-MM-YYYY'),1)
Error report -
ORA-20001: Invalid CVV for card !
ORA-06512: at "VXT180011.CHECK_CARD", line 15
ORA-04088: error during execution of trigger 'VXT180011.CHECK_CARD'
Error starting at line : 9 in command -
Insert into VXT180011.CARD (CARD_NUMBER,CVV,EXPIRY_DATE,BILLING_ADDRESS_ID) values ('4111111111111111','740',to_date('15-01-1922','DD-MM-YYYY'),1)
Error report -
ORA-20001: Card is already expired !
ORA-06512: at "VXT180011.CHECK_CARD", line 20
ORA-04088: error during execution of trigger 'VXT180011.CHECK_CARD'
1 row inserted.
Click on an identifier with the Control key down to perform "Go to Declaration"
Line 3 Column 68
Insert
Modified
Unix/Mac: LF