**Final Project Document**

Project Deliverable 4 - IS 420

Professor Bandaru

Group 8

Members

Donald Duncan

Mykah Rather

Abou Keita

Jamal Cody

Gianni Laditi

**Drop Table Statements**

drop table schedule\_station;

drop table schedule;

drop table trip;

drop table transactions;

drop table line\_station;

drop table station;

drop table train\_line;

drop table metro\_card;

drop table discount;

drop table rider;

**Create Table Statements**

CREATE TABLE rider(

rider\_id NUMBER NOT NULL,

rider\_first\_name VARCHAR2(50 CHAR) NOT NULL,

rider\_last\_name VARCHAR2(50 CHAR) NOT NULL,

rider\_email VARCHAR2(50 CHAR) NOT NULL UNIQUE,

rider\_password VARCHAR2(50 CHAR) NOT NULL,

rider\_age NUMBER NOT NULL,

CONSTRAINT rider\_pk PRIMARY KEY(rider\_id)

);

CREATE TABLE discount(

discount\_id NUMBER NOT NULL,

rate NUMBER NOT NULL,

CONSTRAINT discount\_pk PRIMARY KEY (discount\_id)

);

CREATE TABLE metro\_card (

card\_id NUMBER NOT NULL,

card\_balance NUMBER NOT NULL,

rider\_id NUMBER NOT NULL,

discount\_id NUMBER NOT NULL,

CONSTRAINT metro\_card\_pk PRIMARY KEY(card\_id),

CONSTRAINT rider\_fk FOREIGN KEY (rider\_id) REFERENCES rider (rider\_id),

CONSTRAINT discount\_fk FOREIGN KEY (discount\_id) REFERENCES discount (discount\_id)

);

CREATE TABLE train\_line(

line\_id NUMBER NOT NULL,

line\_name VARCHAR2(50 CHAR) NOT NULL UNIQUE,

station\_total NUMBER NOT NULL,

CONSTRAINT line\_pk PRIMARY KEY (line\_id)

);

CREATE TABLE station(

station\_id NUMBER NOT NULL,

station\_name VARCHAR2(50 CHAR) NOT NULL UNIQUE,

station\_address VARCHAR2(50 CHAR) NOT NULL UNIQUE,

status NUMBER(1) DEFAULT (0),

CONSTRAINT station\_pk PRIMARY KEY (station\_id)

);

CREATE TABLE trip (

trip\_id NUMBER NOT NULL,

entrance\_station\_id NUMBER NOT NULL,

exit\_station\_id NUMBER NOT NULL,

entrance\_time INTERVAL DAY TO SECOND NOT NULL,

exit\_time INTERVAL DAY TO SECOND NOT NULL,

card\_id NUMBER NOT NULL,

cost DECIMAL NOT NULL,

CONSTRAINT trip\_pk PRIMARY KEY(trip\_id),

CONSTRAINT entrance\_station\_fk FOREIGN KEY (entrance\_station\_id) REFERENCES station (station\_id),

CONSTRAINT exit\_station\_fk FOREIGN KEY (exit\_station\_id) REFERENCES station (station\_id),

CONSTRAINT metro\_card\_fk FOREIGN KEY (card\_id) REFERENCES metro\_card (card\_id)

);

CREATE TABLE line\_station(

line\_id NUMBER NOT NULL,

station\_id NUMBER NOT NULL,

sequence\_num NUMBER NOT NULL,

CONSTRAINT line\_fk FOREIGN KEY (line\_id) REFERENCES train\_line (line\_id),

CONSTRAINT station\_fk FOREIGN KEY (station\_id) REFERENCES station (station\_id)

);

CREATE TABLE transactions(

transaction\_id NUMBER NOT NULL,

card\_id NUMBER NOT NULL,

addition\_date DATE NOT NULL,

addition\_time INTERVAL DAY TO SECOND NOT NULL,

addition\_amount NUMBER NOT NULL,

CONSTRAINT transaction\_pk PRIMARY KEY (transaction\_id),

CONSTRAINT metro\_card\_to\_transaction\_fk FOREIGN KEY (card\_id) REFERENCES metro\_card (card\_id)

);

CREATE TABLE schedule(

schedule\_id NUMBER NOT NULL,

line\_id NUMBER NOT NULL,

direction NUMBER NOT NULL,

CONSTRAINT schedule\_pk PRIMARY KEY (schedule\_id),

CONSTRAINT line\_to\_schedule\_fk FOREIGN KEY (line\_id) REFERENCES train\_line (line\_id)

);

CREATE TABLE schedule\_station(

arrival\_time INTERVAL DAY TO SECOND NOT NULL,

schedule\_station\_id NUMBER NOT NULL,

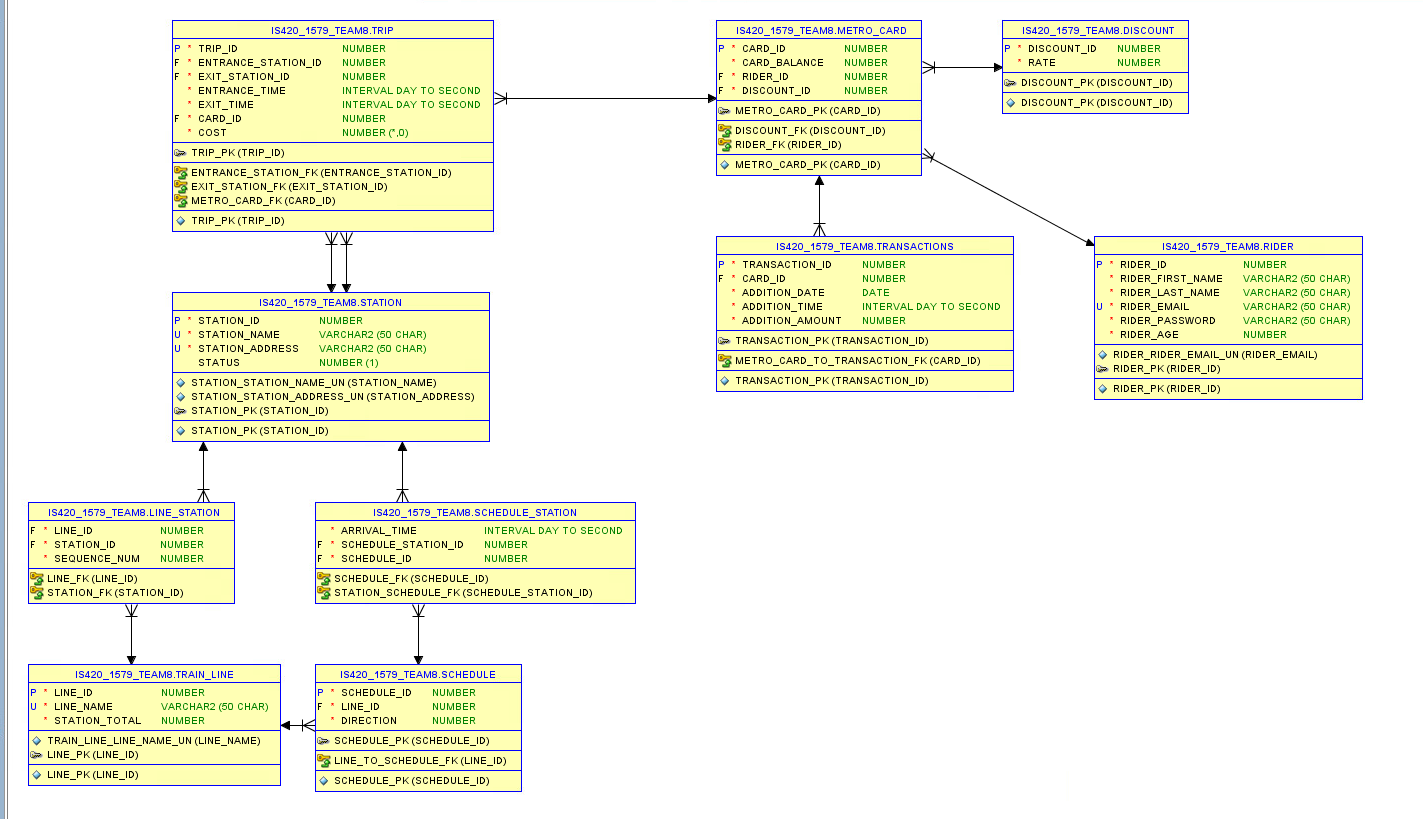
schedule\_id NUMBER NOT NULL,

CONSTRAINT station\_schedule\_fk FOREIGN KEY (schedule\_station\_id) REFERENCES station (station\_id),

CONSTRAINT schedule\_fk FOREIGN KEY (schedule\_id) REFERENCES schedule (schedule\_id)

);

**ERD**



**Task Specifications**

Task 1 (Donald Duncan) –

1. Passenger needs to create a rider account
   1. Input
      1. Must Include
         1. Email address
         2. Name
         3. Password
         4. Age
      2. Check for Valid email
         1. Input email
   2. Output
      1. Check for Valid email
         1. If valid print “Account Exists”
         2. If not create a new account
            1. New account id

Age

Print new account ID

* 1. Test
     1. Using a valid email
     2. Then using invalid email
        1. Forcing the program to create a new account

Task 2 (Donald Duncan) –

1. Login to an Account
   1. Input
      1. Email address
      2. Password
   2. Output
      1. If login information is correct
         1. print successful login
      2. If not email does not match
         1. print no such account
      3. If email is correct and password is not
         1. print wrong password
   3. Test
2. Use official account to receive successful login message
3. Use wrong email to print no such account
4. Use incorrect password to print wrong password

Task 3 (Abou Keita) –

1. INPUT
   1. account ID and initial payment
      1. Account\_id is verified against database to see if it exist
      2. If yes, then a new row to the metro card table is added with the given account id, balance and a new card id (using sequence).
      3. If no, print out a message saying “the account does not exist”
2. OUTPUT
   1. Verify the age of passenger [case statement]
      1. If the holder's age <= 12, set discount id to 2 (child).
      2. If the holder's age >= 65, set discount id to 3 (senior).
      3. Otherwise the discount rate should be 1 (regular).

Task 4 (Abou Keita) –

1. INPUT
   1. Passenger should be allowed to add money to an existing card using input of account id and amount
   2. Metro card with card ID is checked for existence or else print out “saying no such card ”
2. OUTPUT
   1. If card ID EXISTS
      1. Add input amount to balance of the card
      2. Insert a row to metro card transaction table with time as current time
      3. Print: new balance
   2. If card ID DOES NOT EXIST
      1. Print: no such card.

Task 5 (Mykah Rather) –

1. Input
   1. Passenger will be able to list all cards related to an account
      1. Card information listed in table
2. Output
   1. Print out card id
   2. Print out card balance
   3. Print out error message ‘invalid account id’ if account is null/not valid

Task 6 (Mykah Rather) –

1. Input
   1. Passenger will be able to look up all transactions of a card
      1. Transaction information of a card listed in table
2. Output
   1. Print out transactions of a card in given time frame
   2. Print out card id
   3. Print out start date
   4. Print out end date
   5. Print out error message ‘invalid card’
      1. Otherwise, print out transaction date and amount added for each transaction during the period.

Task 7 (Jamal Cody) –

1. Input information to trip table
   1. Input:
      1. card Id
      2. Entrance station id
      3. Exit station id
      4. Entrance time
      5. Exit time
2. Output
   1. If card id is not valid, print error message ‘invalid card’
   2. Else print updated trip id, computed cost, entrance station name and exit station name
   3. Update card balance

Task 8 ( Gianni Laditi) –

1. Create PL/SQL procedure
2. Input:
3. Time of arrival
4. Name of station
5. Output
6. Stations and their departure time
7. Station id
8. Departure time
9. Line name
10. Direction 1or 2
11. Else “Station not found”

Task 9 (Mykah Rather) –

1. Create PL/SQL function
   1. Input:
      1. Name of an origin station
      2. Name of a destination station
      3. ID of line station(s) are on
   2. Output:
      1. Direction (1 for increasing order and 2 for descending order)

Task 10 (Jamal Cody)

1. Input:
   1. Start time
   2. Time gap
   3. Names of origin and destination station
   4. Line name
2. Output
   1. If station name is invalid print error message ‘wrong station’
   2. If line name is invalid print error message ‘wrong line name’
   3. If all names are valid, print origin station and destination station
   4. Print out schedule ID
   5. Print scheduled arrival time at origin
   6. Print Destination station for each schedule

Task 11 (Gianni and Abou)

1. INPUT
   1. Passenger origin
   2. Two destination stations assumed to be on two different lines
   3. At least one station that the passenger can transfer between these two lines.
      1. First needs to check:
         1. Whether the two station names exist and
         2. Whether they are on different lines.
2. OUTPUT
   1. Not valid
      1. Print out an error message explaining the problem.
   2. Valid
      1. Prints out the transfer station name
      2. Line name
      3. Direction (1 or 2) from origin to transfer station
      4. Direction from transfer station to destination

Task 12 (Donald Duncan)

1. Create a Procedure
   1. Input
      1. Procedure
   2. Output
      1. Print
         1. Total number of accounts
         2. Total spending (cost of all trips)
         3. Average number of trips per account
         4. The station appears most often as the entrance station in trips
         5. The station appears most often as the exit station in trips

**Procedure/Function Statements**

**Task 1: (Donald)** allow a passenger to create a rider account with input including email address, name, password, and age. First check whether email exists. If so, print a message that the account exists. Otherwise, create a new rider account with new account id and age and print out the new account id.

CREATE OR REPLACE PROCEDURE createRiderAccount(first\_name VARCHAR2, last\_name VARCHAR2, email VARCHAR2, password VARCHAR2, age NUMBER)

AS

new\_rider\_id NUMBER;

account\_exists NUMBER := 0;

BEGIN

SELECT COUNT(\*) INTO account\_exists

FROM rider

WHERE rider\_email = email;

IF (account\_exists <> 0) THEN

DBMS\_OUTPUT.PUT\_LINE('An account already exists with the provided email: ' || email);

RETURN;

END IF;

INSERT INTO rider (rider\_first\_name, rider\_last\_name, rider\_email, rider\_password, rider\_age)

VALUES (first\_name, last\_name, email, password, age)

returning rider\_id INTO new\_rider\_id;

DBMS\_OUTPUT.PUT\_LINE('New account id: ' || new\_rider\_id);

END;

**Task 2:(Donald)** login to an account with email address and password. If both matches an existing account, print a message login is successful. If email does not match, print a message no such account. If email matches but password does not match, print wrong password.

CREATE OR REPLACE PROCEDURE loginRider(email VARCHAR2, password VARCHAR2)

AS

email\_exists NUMBER := 0;

rider\_account\_id NUMBER := 0;

BEGIN

SELECT COUNT(\*) INTO email\_exists

FROM rider

WHERE rider\_email = email;

IF (email\_exists = 0) THEN

DBMS\_OUTPUT.PUT\_LINE('No account exists with the provided email: ' || email);

RETURN;

END IF;

SELECT rider\_id INTO rider\_account\_id

FROM rider

WHERE rider\_email = email

AND rider\_password = password;

DBMS\_OUTPUT.PUT\_LINE('Login is successful!');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.put\_line('Wrong password!');

END;

**Task 3:(Abou)**

Create a procedure to allow a passenger to buy a metro card and add it to an existing account. Input includes account id, initial balance (payment).

* The procedure first checks if the input account id exists. If not, print out a message saying that the account does not exist.
* Otherwise the procedure inserts a new row to the metro card table with the given account id, balance and a new card id (using sequence).
* In addition, please look up the age of the account holder. If holder's age <= 12, set discount id to 2 (child). If holder's age >= 65, set discount id to 3 (senior). Otherwise the discount rate should be 1 (regular).

create or replace procedure BuyNewCard (acct\_id integer,InitialPayement integer)

As

checkID integer:= 0;

newPayement integer:= 0;

begin

SELECT acct\_id INTO checkid

FROM metro\_card

WHERE rider\_id = acct\_id;

IF (checkID = 0) THEN

DBMS\_OUTPUT.PUT\_LINE('No account exists with the account ID: ' || acct\_id);

RETURN;

END IF;

Update metro\_card set card\_balance = card\_balance + InitialPayement

WHERE rider\_id = acct\_id;

SELECT card\_balance into newPayement

From metro\_card

WHERE rider\_id = acct\_id;

DBMS\_OUTPUT.PUT\_LINE('Account ID is: ' || checkID);

DBMS\_OUTPUT.PUT\_LINE('New Balance is: ' || newpayement);

end;

**Look-up Age for discountID**

*SET SERVEROUTPUT ON*

*DECLARE*

*Discount\_ID INT;*

*Age INT;*

*BEGIN*

*SELECT AGE INTO Age*

*FROM RIDERS*

*WHERE account\_id = 3;*

*IF AGE <= 12 THEN*

*Discount\_ID := 2;*

*dbms\_output.put\_line( 'The discount ID for this rider is: ' || Discount\_ID );*

*ELSIF AGE >= 65 THEN*

*Discount\_ID := 3;*

*dbms\_output.put\_line( 'The discount ID for this rider is: ' || Discount\_ID );*

*ELSE*

*Discount\_ID := 1;*

*dbms\_output.put\_line( 'The discount ID for this rider is: ' || Discount\_ID );*

*END IF;*

*END;*

**Task 4:(Abou)**

**Allow a passenger to add money to an existing card, with input card id and amount. The procedure first checks if there is a metro card with given card id. If not, print a message saying no such card. Otherwise, add input amount to balance of the card and insert a row to metro card transaction table with time as current time. Finally print out new balance.**

create or replace procedure AddMoney(inputcard\_id in INT, amount in number, printAmount out number)

IS

begin

Update metro\_card set card\_balance = card\_balance + amount

WHERE card\_id = inputcard\_id;

SELECT card\_balance into printAmount

From metro\_card

WHERE card\_id = inputcard\_id;

exception

when no\_data\_found then

dbms\_output.put\_line('no such card id exist');

end;

**Task 5:(Mykah)** *Create a procedure to allow a passenger to list all cards related to an account, and print out card id and balance on each card. In case the account id is not valid, print out a message saying invalid account id.*

-- Task 5

SET serveroutput ON;

CREATE OR REPLACE PROCEDURE CardDetails (rider\_id in number) IS

/\*

Since each account can be uniquely identified

by the associated rider ID, the user is prompted

for said rider number

\*/

r\_id rider.rider\_id%TYPE := '&r\_id';

/\*

This task takes a given account id for a rider

and matches said id to all metro cards linked

to said account

\*/

c\_id metro\_card.card\_id%TYPE;

c\_balance metro\_card.card\_balance%TYPE;

BEGIN

IF r\_id = rider\_id THEN

SELECT card\_id, card\_balance

INTO c\_id, c\_balance

FROM metro\_card

WHERE rider\_id = r\_id;

dbms\_output.put\_line('Card ID: ' || c\_id);

dbms\_output.put\_line('Card Balance: ' || c\_balance);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

dbms\_output.put\_line('Account Not Found');

END;

/

**Task 6:(Mykah)** *Create a procedure to allow a passenger to look up all transactions of a card in a given time period. Input includes card id, start date and end date. In case the input card id is invalid, print a message saying invalid card. Otherwise print out the transaction date and amount added for each transaction during the period.*

-- Task 6

SET serveroutput ON;

CREATE OR REPLACE PROCEDURE CardTransaction (card\_id in number) IS

-- take input from user for card id, start date, and end date

card\_id\_input transactions.card\_id%TYPE := &card\_id\_input;

start\_date transactions.transaction\_date%TYPE := '&start\_date';

end\_date transactions.transaction\_date%TYPE := '&end\_date';

-- declare variables for requested output values

t\_time transactions.transaction\_date%TYPE;

time\_check transactions.addition\_time%TYPE;

balance\_check transactions.addition\_amount%TYPE;

BEGIN

IF card\_id\_input = card\_id THEN

SELECT transaction\_date, addition\_time, addition\_amount

INTO t\_time, time\_check, balance\_check

FROM transactions

WHERE card\_id = card\_id\_input;

dbms\_output.put\_line('Transaction Date: ' || t\_time);

dbms\_output.put\_line('Amount Deposited: ' || balance\_check);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

dbms\_output.put\_line('INVALID CARD');

END;

/

**Task 7:(Jamal)**

**Task 8:(Gianni)**

Create a procedure that given a time (just hours and minutes) and station name, list all schedules departing that station within X minutes after the given time. Here X is an input. The procedure first checks whether the input station name matches any existing station. If not it prints an error message saying wrong station name. Otherwise it prints out schedule ID, line name, direction (1 or 2), scheduled arrival time at the station.

SET serveroutput ON;

CREATE OR REPLACE PROCEDURE checkdeparture(station\_name, arrival\_time)

IS

-- take input from user for time, and station name

time\_input schedule\_station.arrival\_time%TYPE := '&time\_input';

station\_input station.station\_name%TYPE := '&station\_input';

-- declare variables for requested output values

schedule\_id\_output schedule.schedule\_id%TYPE;

line\_name\_output train\_line.line\_name%TYPE;

direction\_output schedule.direction%TYPE;

station\_name\_check station.station\_name%TYPE;

BEGIN

SELECT station.station\_name

INTO station\_name\_check

FROM station

WHERE station\_name = station\_input;

FOR result\_info IN (

SELECT schedule\_id, direction

INTO line\_name\_output, direction\_output

FROM schedule, train\_line

WHERE schedule.line\_id = train\_line.line\_id

)

LOOP

dbms\_output.put\_line('Line: ' || line\_name\_output ||

' Direction: ' || direction\_output);

END LOOP;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

dbms\_output.put\_line('Station Not Found');

END;

**Group Tasks**

**Task 9:**

CREATE OR REPLACE FUNCTION task9(origin\_station IN VARCHAR2, destination\_station IN VARCHAR2, line\_id IN NUMBER)

RETURN NUMBER

IS

origin\_seq\_num NUMBER;

destination\_seq\_num NUMBER;

BEGIN

-- get the sequence number of the origin station

SELECT sequence\_num INTO origin\_seq\_num

FROM station

JOIN line\_station ON station.station\_id = line\_station.station\_id

WHERE line\_id = line\_id

AND station\_name = origin\_station;

-- get the sequence number of the destination station

SELECT sequence\_num INTO destination\_seq\_num

FROM station

JOIN line\_station ON station.station\_id = line\_station.station\_id

WHERE line\_id = line\_id

AND station\_name = destination\_station;

-- check which station has a bigger sequence number to determine the direction

IF (origin\_seq\_num < destination\_seq\_num) THEN

RETURN 1;

ELSE

RETURN 2;

END IF;

END task9;

**Task 10:**

CREATE OR REPLACE PROCEDURE task10(start\_time INTERVAL DAY TO SECOND, time\_gap INTERVAL DAY TO SECOND, origin\_station VARCHAR2, destination\_station VARCHAR2, train\_line\_name VARCHAR2)

AS

TYPE array\_of\_varchar2 IS VARRAY(50) OF VARCHAR2(50);

TYPE array\_of\_interval IS VARRAY(50) OF INTERVAL DAY TO SECOND;

arrivals\_at\_origin array\_of\_interval;

arrivals\_at\_destination array\_of\_interval;

schedule\_ids array\_of\_varchar2;

schedule\_ids\_destination array\_of\_varchar2;

j NUMBER;

line\_direction NUMBER;

line\_id NUMBER;

valid\_origin NUMBER := 0;

valid\_destination NUMBER := 0;

is\_line\_name\_valid NUMBER := 0;

are\_stations\_on\_same\_line NUMBER := 0;

BEGIN

SELECT COUNT(\*) INTO valid\_origin

FROM station

WHERE station\_name = origin\_station;

--if there are 0 results, then the station name is invalid

IF (valid\_origin = 0) THEN

DBMS\_OUTPUT.put\_line('Invalid origin station name: ' || origin\_station);

RETURN;

END IF;

SELECT COUNT(\*) INTO valid\_destination

FROM station

WHERE station\_name = destination\_station;

--if there are 0 results, then the station name is invalid

IF (valid\_destination = 0) THEN

DBMS\_OUTPUT.put\_line('Invalid destination station name: ' || destination\_station);

RETURN;

END IF;

SELECT COUNT(\*) INTO is\_line\_name\_valid

FROM station

join line\_station

ON station.station\_id = line\_station.station\_id

join train\_line

ON train\_line.line\_id = line\_station.line\_id

WHERE train\_line.line\_name = train\_line\_name;

--if there are 0 results, then the line name is invalid

IF (is\_line\_name\_valid = 0) THEN

DBMS\_OUTPUT.put\_line('Invalid line name: ' || train\_line\_name);

RETURN;

END IF;

SELECT COUNT(\*) INTO are\_stations\_on\_same\_line

FROM station

join line\_station

ON station.station\_id = line\_station.station\_id

join train\_line

ON train\_line.line\_id = line\_station.line\_id

WHERE train\_line.line\_name = train\_line\_name

AND (station.station\_name = origin\_station

OR station.station\_name = destination\_station);

--if there aren't 2 results, then at least one of the stations isn't on the line given as parameter

IF (are\_stations\_on\_same\_line != 2) THEN

DBMS\_OUTPUT.put\_line('The stations aren''t both on train line: ' || train\_line\_name);

RETURN;

END IF;

--get line id

SELECT line\_id INTO line\_id

FROM train\_line

WHERE train\_line.line\_name = train\_line\_name;

--find out the correct direction using task9

line\_direction := task9(origin\_station, destination\_station, line\_id);

--apply all filters to select the viable schedule ids and the arrivals at the origin

SELECT schedule.schedule\_id, schedule\_station.arrival\_time BULK COLLECT INTO schedule\_ids, arrivals\_at\_origin

FROM schedule

join schedule\_station

ON schedule\_station.schedule\_id = schedule.schedule\_id

join station

ON schedule\_station.schedule\_station\_id = station.station\_id

WHERE line\_id = line\_id

AND station\_name = origin\_station

AND arrival\_time BETWEEN start\_time AND (start\_time + time\_gap)

AND direction = line\_direction

ORDER BY 2 ASC;

--get all arrival times in destination station from the start of the time window

SELECT schedule.schedule\_id, schedule\_station.arrival\_time BULK COLLECT INTO schedule\_ids\_destination, arrivals\_at\_destination

FROM schedule

join schedule\_station

ON schedule\_station.schedule\_id = schedule.schedule\_id

join station

ON schedule\_station.schedule\_station\_id = station.station\_id

WHERE line\_id = line\_id

AND station\_name = destination\_station

AND arrival\_time >= start\_time

AND direction = line\_direction

ORDER BY 1 ASC;

--loop over all viable trips and print the schedule

FOR i IN 1..schedule\_ids.COUNT LOOP

-- synchronize the station arrival arrays

j := i;

IF (schedule\_ids(i) != schedule\_ids\_destination(i)) THEN

j := i + 1;

END IF;

DBMS\_OUTPUT.put\_line('Schedule id: ' || schedule\_ids(i) || ' arrivat at ' || origin\_station || ': ' || arrivals\_at\_origin(i) || ' arrival at ' || destination\_station || ': ' || arrivals\_at\_destination(j));

END LOOP;

END;

**Task 11:**

CREATE OR REPLACE PROCEDURE task11(origin\_station VARCHAR2, destination\_station VARCHAR2)

AS

TYPE array\_of\_varchar2 IS VARRAY(50) OF VARCHAR2(50);

TYPE array\_of\_number IS VARRAY(50) OF NUMBER;

direction1 NUMBER;

direction2 NUMBER;

transfer\_station\_names array\_of\_varchar2;

--transfer\_station\_seq\_origin too long var name

ts\_seq\_origin array\_of\_number;

--transfer\_station\_seq\_destination too long var name

ts\_seq\_destination array\_of\_number;

origin\_station\_seq NUMBER;

destination\_station\_seq NUMBER;

origin\_line\_name VARCHAR(50);

origin\_line\_id NUMBER;

destination\_line\_name VARCHAR(50);

destination\_line\_id NUMBER;

is\_origin\_valid NUMBER := 0;

is\_destination\_valid NUMBER := 0;

num\_stations\_on\_same\_line NUMBER := 0;

multi\_station\_line VARCHAR(50);

BEGIN

SELECT COUNT(\*) INTO is\_origin\_valid

FROM station

WHERE station\_name = origin\_station;

--if there are 0 results, then the station name is invalid

IF (is\_origin\_valid = 0) THEN

DBMS\_OUTPUT.put\_line('Invalid origin station name: ' || origin\_station);

RETURN;

END IF;

SELECT COUNT(\*) INTO is\_destination\_valid

FROM station

WHERE station\_name = destination\_station;

--if there are 0 results, then the station name is invalid

IF (is\_destination\_valid = 0) THEN

DBMS\_OUTPUT.put\_line('Invalid destination station name: ' || destination\_station);

RETURN;

END IF;

-- select the first row, with the largest group count for line\_id

SELECT total\_stations, line\_name INTO num\_stations\_on\_same\_line, multi\_station\_line FROM (

-- select the line\_ids for the two station

-- group the rows by line\_name, and check the group count

-- if the stations are on separate lines, group counts will be 1

-- sort by descending

SELECT COUNT(\*) AS total\_stations, line\_name

FROM station

join line\_station

ON station.station\_id = line\_station.station\_id

join train\_line

ON train\_line.line\_id = line\_station.line\_id

WHERE station\_name = origin\_station

OR station\_name = destination\_station

GROUP BY line\_name

ORDER BY 1 DESC)

WHERE ROWNUM <= 1;

--if there are 0 results, then the station name is invalid

IF (num\_stations\_on\_same\_line = 2) THEN

DBMS\_OUTPUT.put\_line('The two stations are both on the ' || multi\_station\_line || ' line.');

RETURN;

END IF;

--select the origin line name for the output, and line\_id for future queries

SELECT train\_line.line\_name, train\_line.line\_id, line\_station.sequence\_num INTO origin\_line\_name, origin\_line\_id, origin\_station\_seq

FROM station

join line\_station

ON station.station\_id = line\_station.station\_id

join train\_line

ON train\_line.line\_id = line\_station.line\_id

WHERE station.station\_name = origin\_station;

--select the destination line name for the output, and line\_id for future queries

SELECT train\_line.line\_name, train\_line.line\_id, line\_station.sequence\_num INTO destination\_line\_name, destination\_line\_id, destination\_station\_seq

FROM station

join line\_station

ON station.station\_id = line\_station.station\_id

join train\_line

ON train\_line.line\_id = line\_station.line\_id

WHERE station.station\_name = destination\_station;

--store the names of all stations that are on both lines in an array

SELECT origin\_line.station\_name, origin\_line.sequence\_num, destination\_line.sequence\_num BULK COLLECT INTO transfer\_station\_names, ts\_seq\_origin, ts\_seq\_destination

FROM

(SELECT station.station\_id, station.station\_name, line\_station.sequence\_num

FROM station

join line\_station

ON station.station\_id = line\_station.station\_id

WHERE line\_station.line\_id = origin\_line\_id) origin\_line

join

(SELECT station.station\_id, station.station\_name, line\_station.sequence\_num

FROM station

join line\_station

ON station.station\_id = line\_station.station\_id

WHERE line\_station.line\_id = destination\_line\_id) destination\_line

ON origin\_line.station\_id = destination\_line.station\_id;

FOR i IN 1..transfer\_station\_names.COUNT LOOP

IF (origin\_station\_seq < ts\_seq\_origin(i)) THEN

direction1 := 1;

ELSE

direction1 := 2;

END IF;

IF (ts\_seq\_destination(i) < destination\_station\_seq) THEN

direction2 := 1;

ELSE

direction2 := 2;

END IF;

DBMS\_OUTPUT.put\_line('Take the ' || origin\_line\_name || ' in direction ' || direction1 || ' transfer at ' || transfer\_station\_names(i) || ' and then take ' || destination\_line\_name || ' in direction ' || direction2);

END LOOP;

END;

**Task 12:**

CREATE OR REPLACE PROCEDURE task12

AS

total\_accounts NUMBER := 0;

total\_cards NUMBER := 0;

total\_spending DECIMAL := 0;

total\_trips NUMBER := 0;

avg\_trips\_per\_account double precision := 0;

top\_entrance\_station VARCHAR2(50);

top\_exit\_station VARCHAR2(50);

BEGIN

SELECT COUNT(\*) INTO total\_accounts FROM rider;

SELECT COUNT(\*) INTO total\_cards FROM metro\_card;

SELECT COUNT(\*) INTO total\_trips FROM trip;

--get the discount rate for every trip

--calculate how much of the cost in percentage turn the rider paid (1 - discount\_rate)

--multiply the trip cost by portion of the cost paid by the rider to get the actual spending per trip

--sum all spending to get the total spending

SELECT SUM(cost \* (1 - rate)) INTO total\_spending

FROM trip

join metro\_card

ON trip.card\_id = metro\_card.card\_id

join discount

ON metro\_card.discount\_id = discount.discount\_id;

--calculate the average trips per rider

avg\_trips\_per\_account := total\_trips / total\_accounts;

--select only the name of the top entry station

SELECT station\_name INTO top\_entrance\_station FROM (

--group all trips by entry station name

--calculate the total count of trips per entry station

SELECT COUNT(\*) "total\_entry", station\_name

FROM trip

join station

ON station.station\_id = entrance\_station\_id

GROUP BY station\_name

ORDER BY 1 DESC)

WHERE ROWNUM <= 1;

--select only the name of the top exit station

SELECT station\_name INTO top\_exit\_station FROM (

--group all trips by exit station name

--calculate the total count of trips per exit station

SELECT COUNT(\*) "total\_exit", station\_name

FROM trip

join station

ON station.station\_id = exit\_station\_id

GROUP BY station\_name

ORDER BY 1 DESC)

WHERE ROWNUM <= 1;

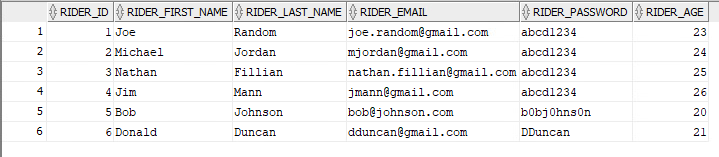
DBMS\_OUTPUT.put\_line('accounts: ' || total\_accounts || ', cards: ' || total\_cards || ', spending: ' || total\_spending || ', trips per account: ' || avg\_trips\_per\_account || ', top entrance station: ' || top\_entrance\_station || ', top exit station: ' || top\_exit\_station);

END;

**Demo Procedures**

**Task 1:**

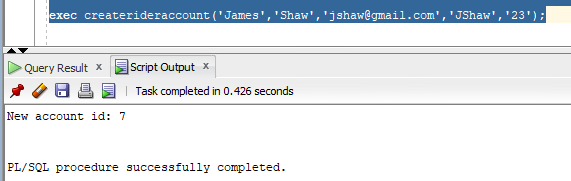
Select \* From Rider;



--Test 1 Create a new account

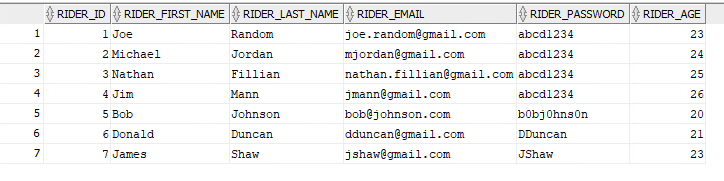
Set serveroutput on;

Exec createrideraccount('James', 'Shaw', 'jshaw@gmail.com', 'JShaw', 23);



--Show updated table

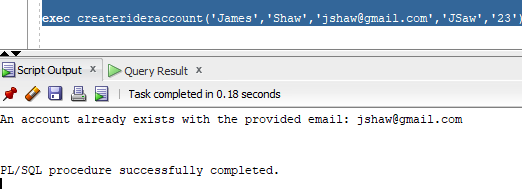
Select \* from rider;



-- Test 2 Account already exists

Set serveroutput on;

Exec createrideraccount('James', 'Shaw', 'jshaw@gmail.com', 'JShaw', 23);

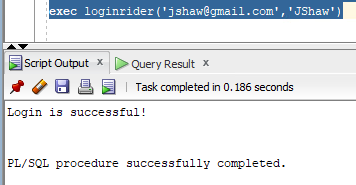


**Task 2:**

--Test 1 Successful login

Set serveroutput on;

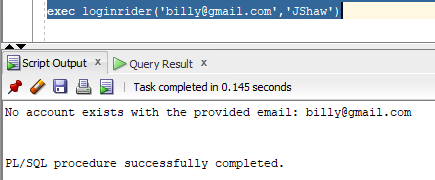
Exec loginrider(‘jshaw@gmail.com’,’JShaw’);



--Test 2 Invalid email

Set serveroutput on;

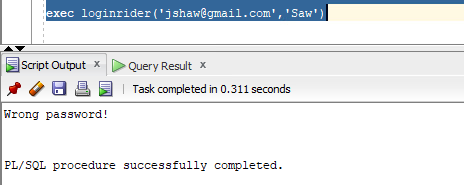
Exec loginrider(‘billy@gmail.com’,JShaw’);



--Test 3 Wrong password

Set serveroutput on;

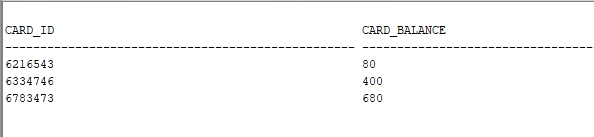
Exec loginrider(‘jshaw@gmail.com’, ‘Saw’);



**Task 3:**

Select \* From Metro card

---- card id for each account id starting with 1

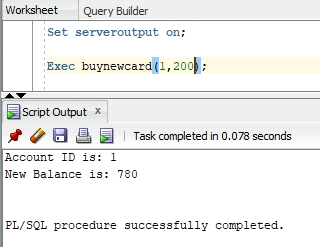


---- Create new Metro Card with valid account id

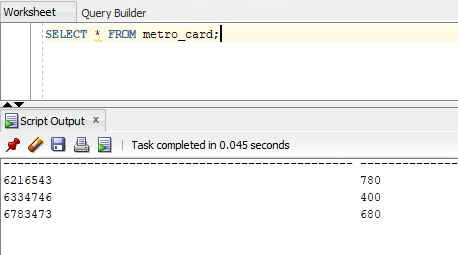
Valid Input

Set serveroutput on;

Exec buynewcard(1,200);

****

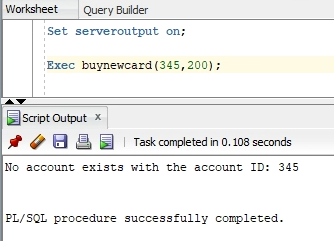
---- Metro card balance increased for account ID 1, tried multiple times thats why the new balance shown does not add up

****

Invalid Input

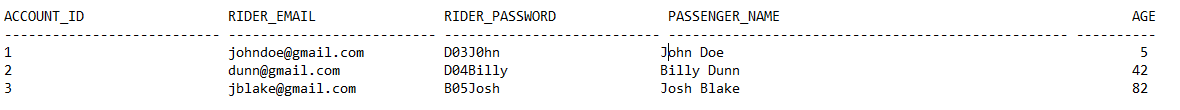
Set serveroutput on;

Exec buynewcard(345,200)

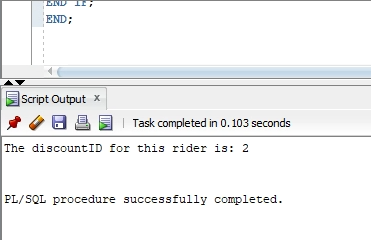
****

**----- All riders ages**

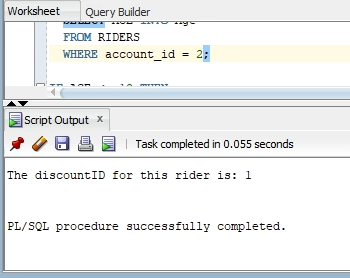
**Select \* from riders**



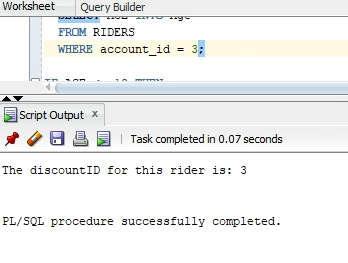
**----- Look up discount ID for account 1**

****

**----look up for discount ID for account 2**

****

**----look up for discount id for account 3**

****

**Task 4:**

Set serveroutput on;

declare

newBalance number;

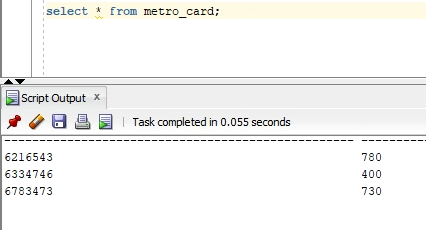
begin

Addmoney(6783473,50,newBalance);

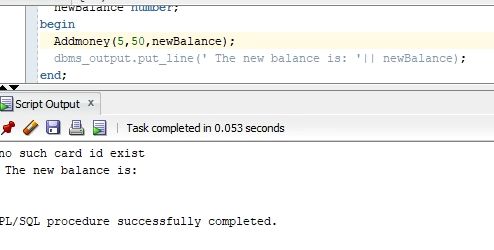
dbms\_output.put\_line(' The new balance is: '|| newBalance);

end;

Valid input

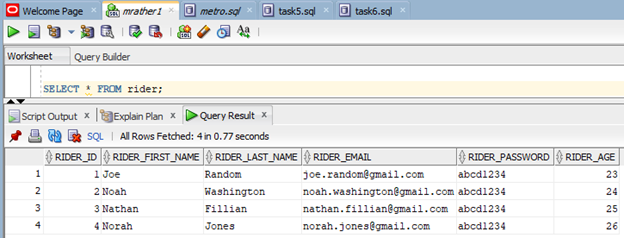
****

Invalid Input

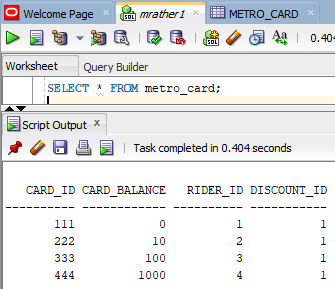
****

**Task 5:**

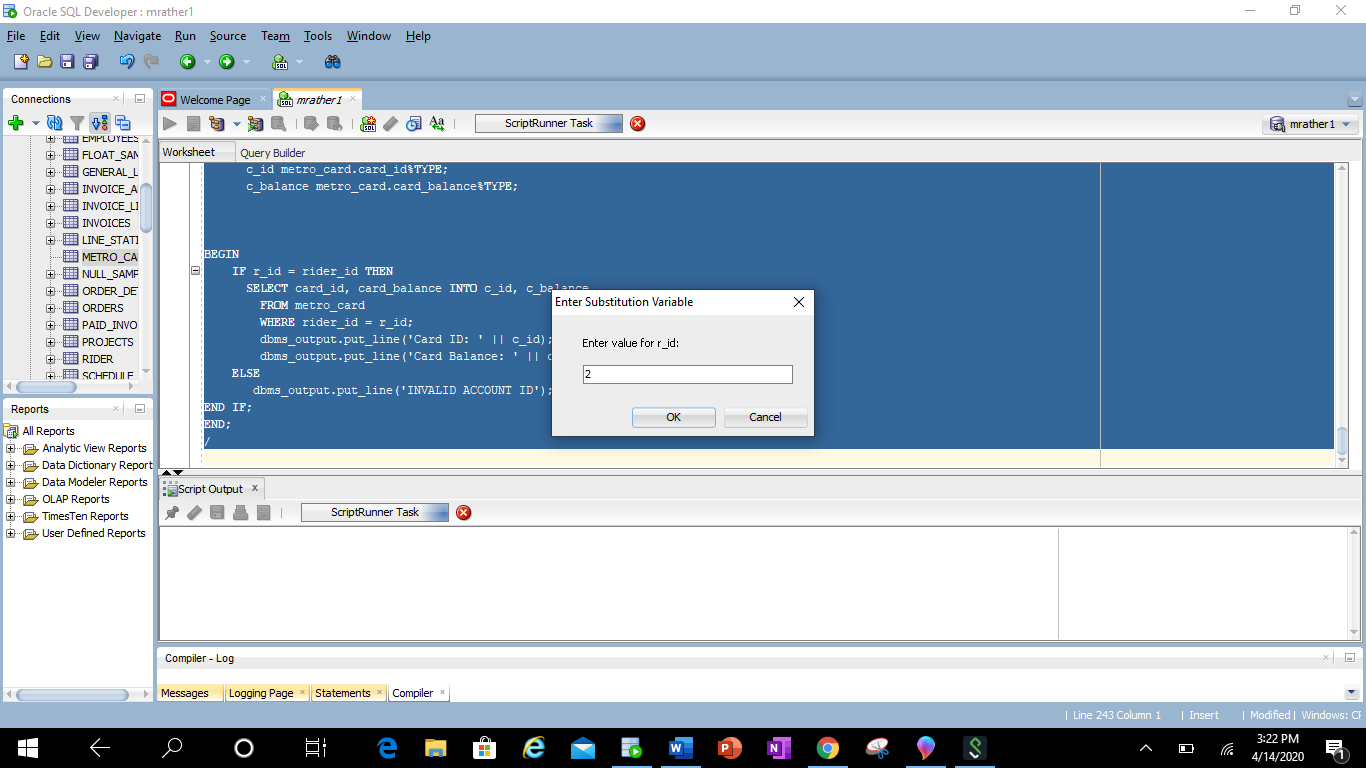
*SELECT \* FROM rider;*

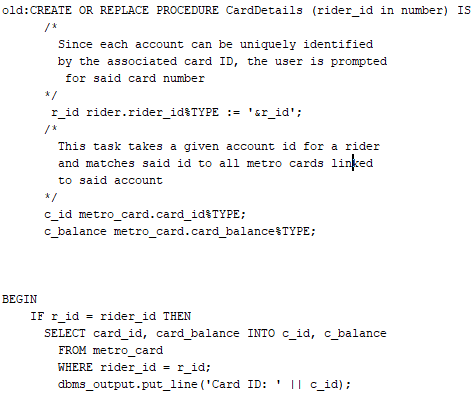


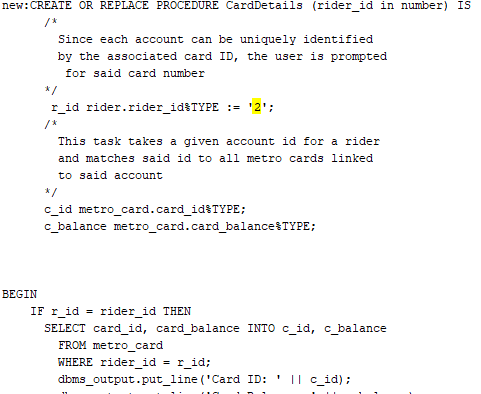
*SELECT \* FROM metro\_card;*

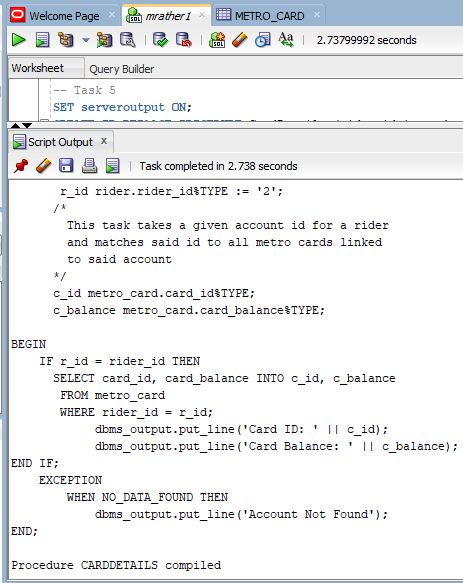


***RIDER 2: Valid Test Scenario***





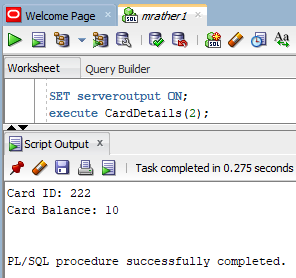




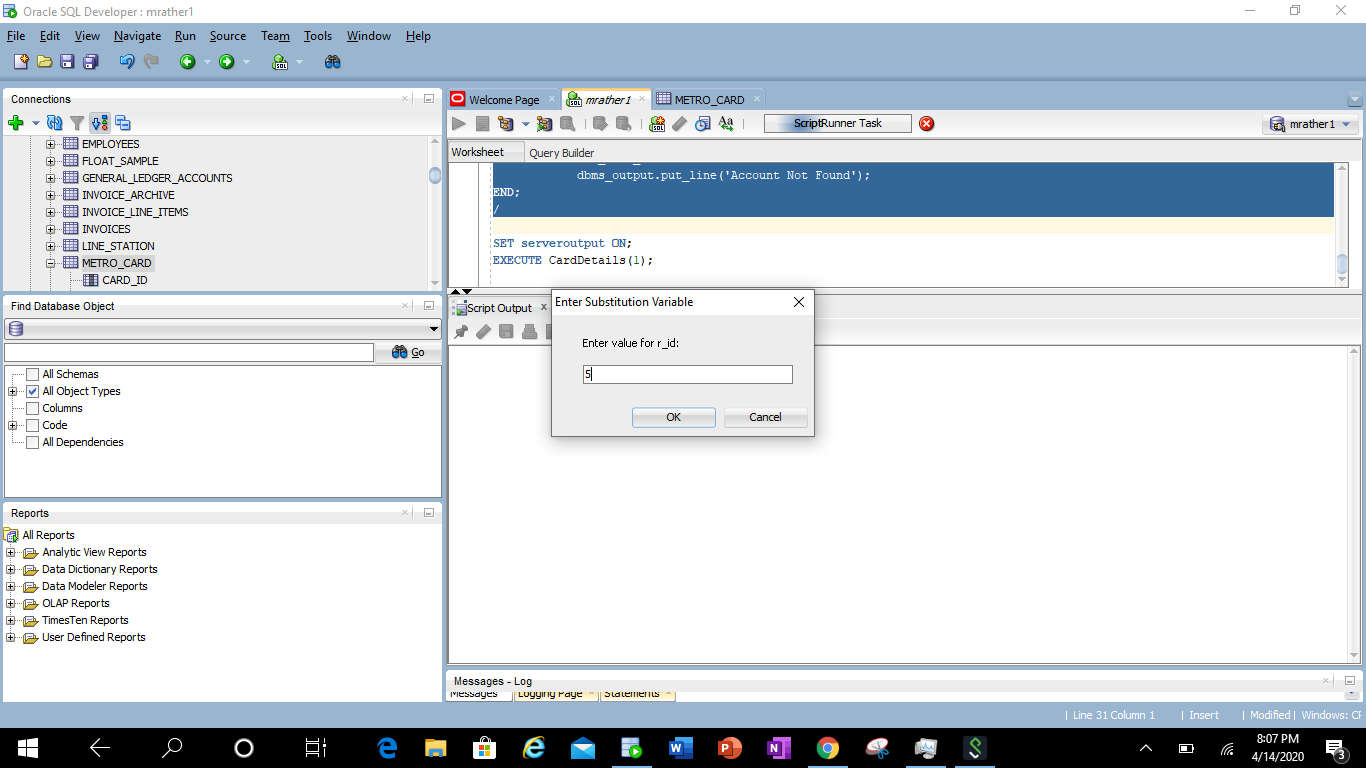
**Display of Procedure**

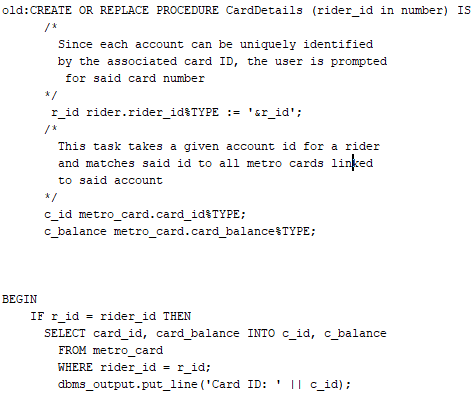
SET serveroutput ON;

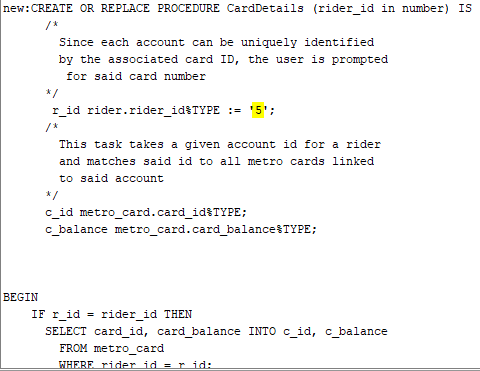
execute CardDetails(2);

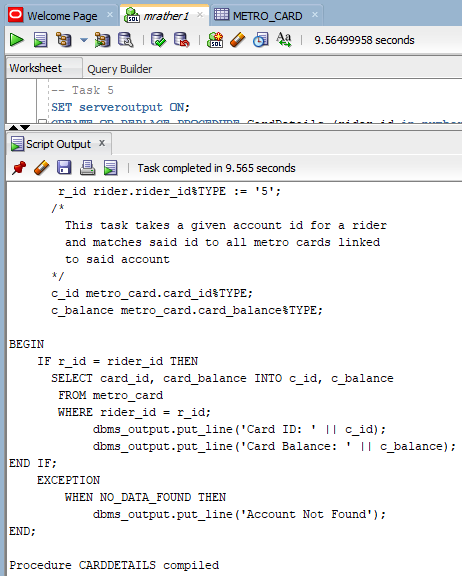


***RIDER 5: Invalid Test Scenario***





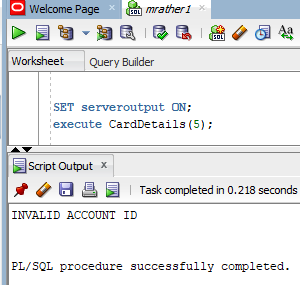




**Display of Procedure**

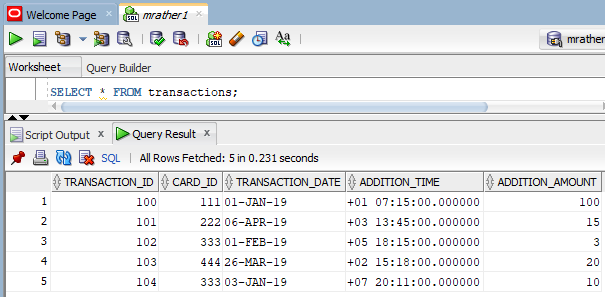
SET serveroutput ON;

execute CardDetails(5);

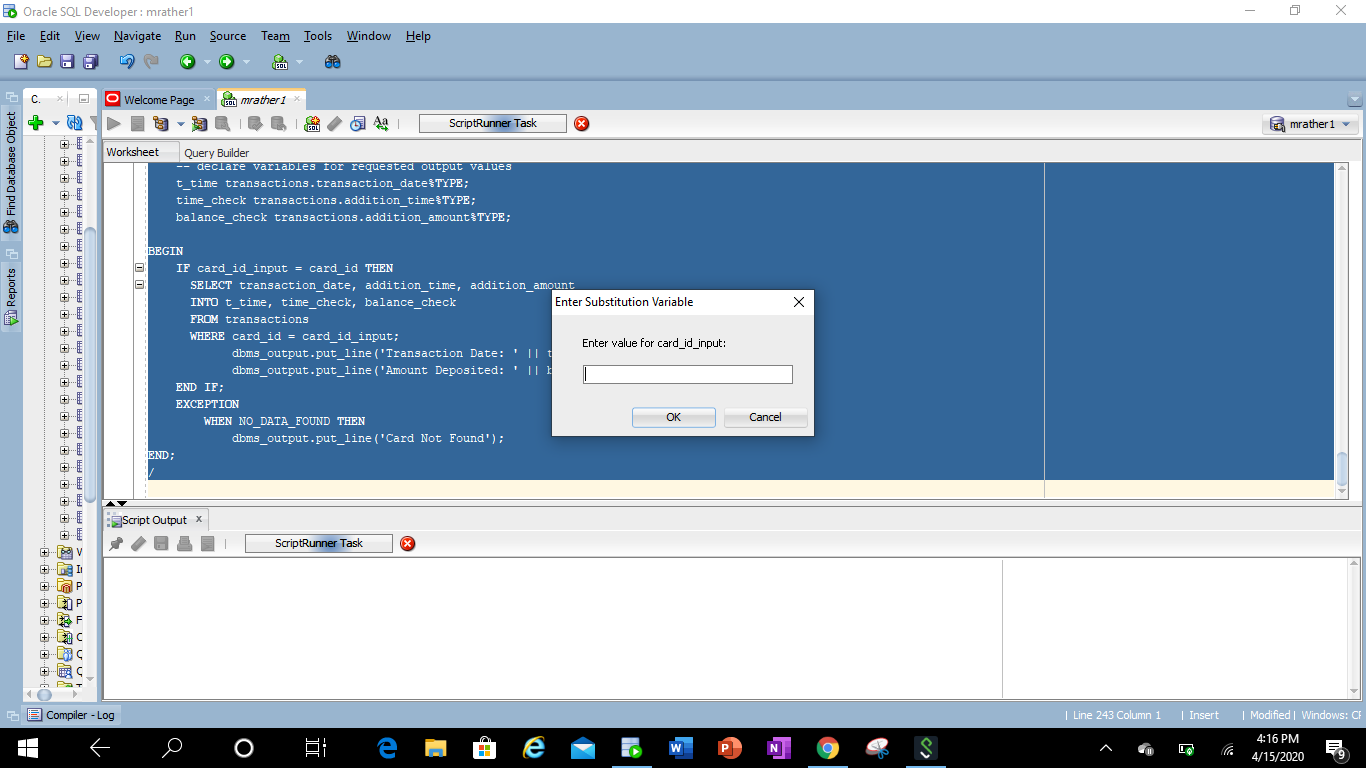


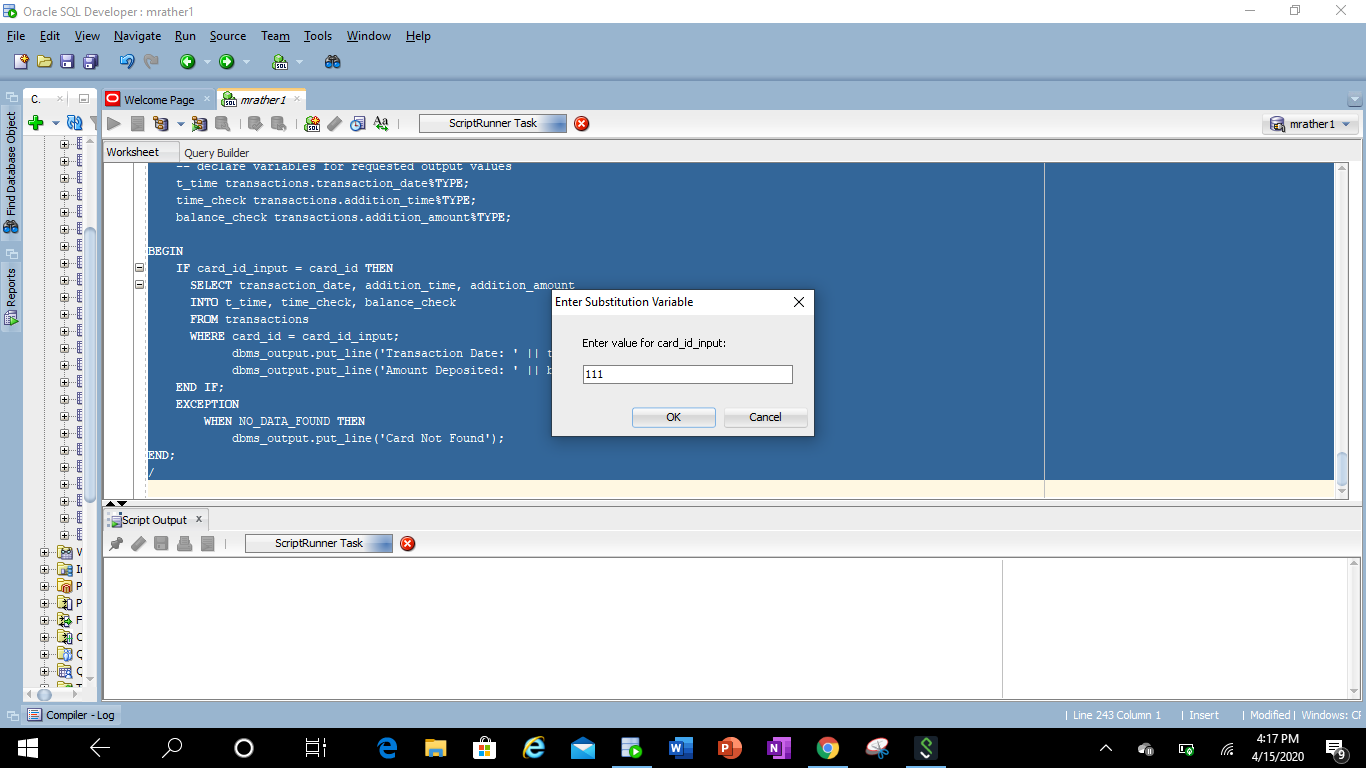
**Task 6:**

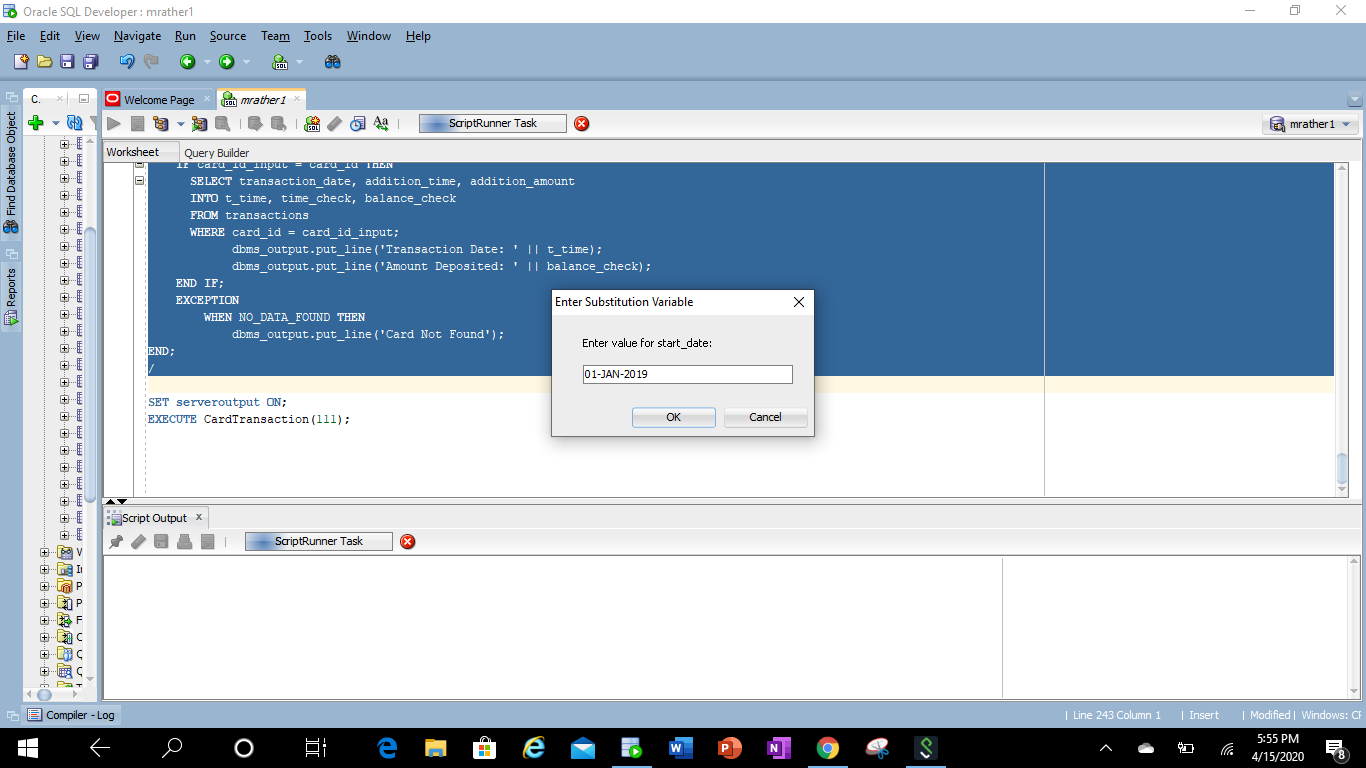
*SELECT \* FROM transactions;*

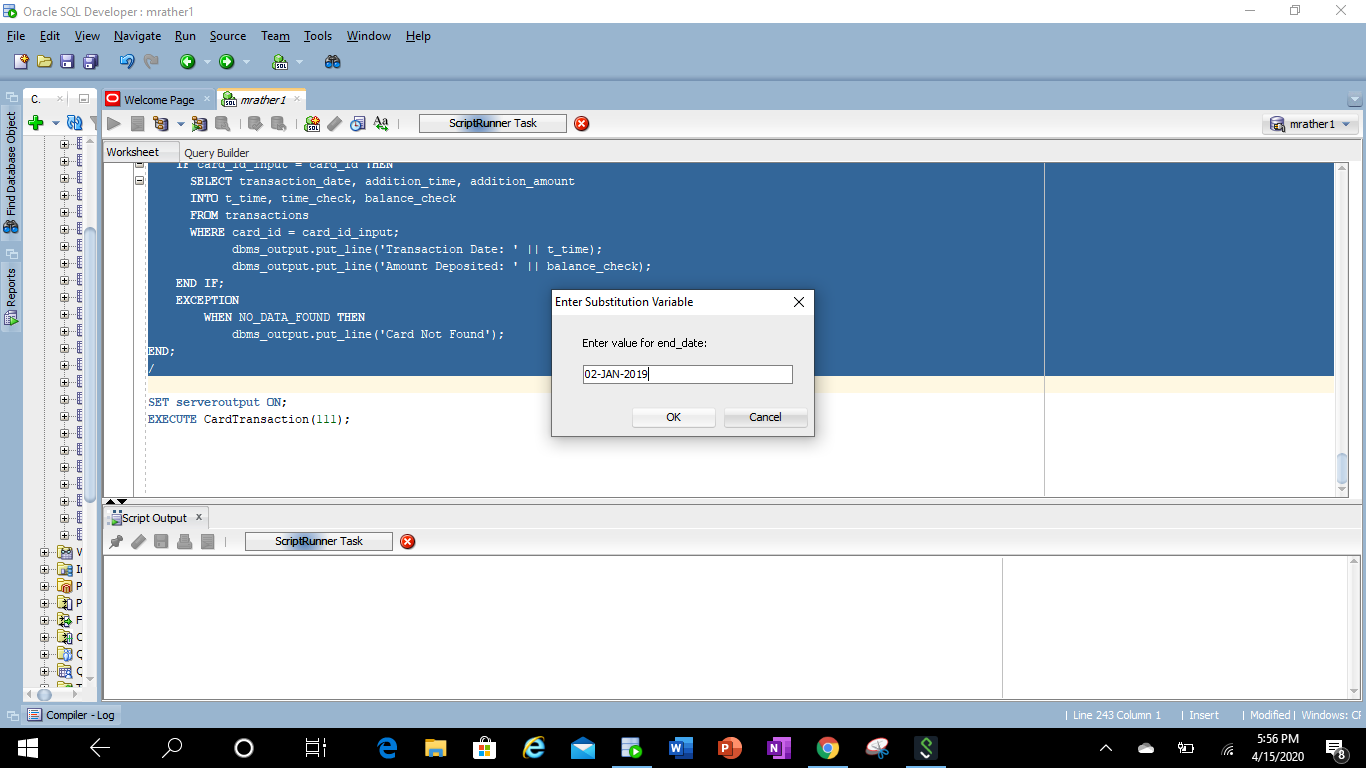


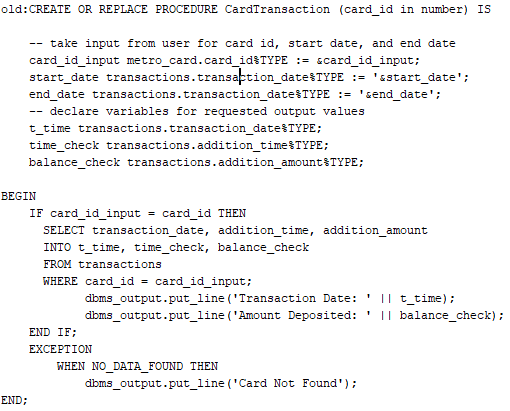
***CARD 1: Valid Test Scenario***

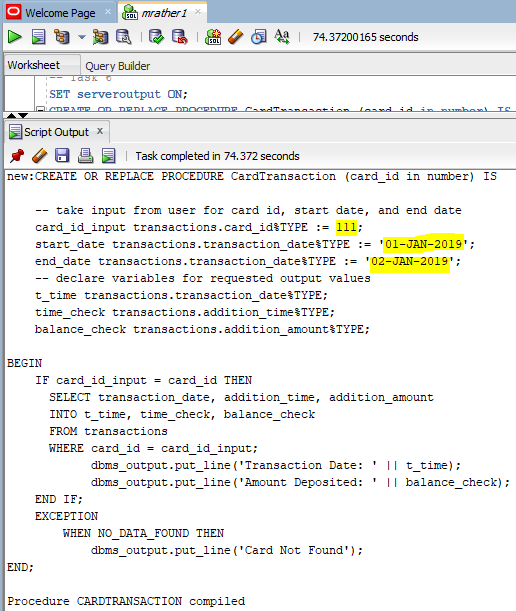








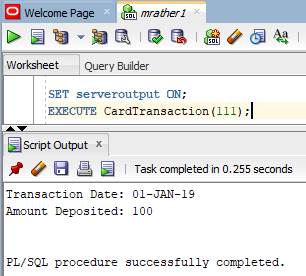




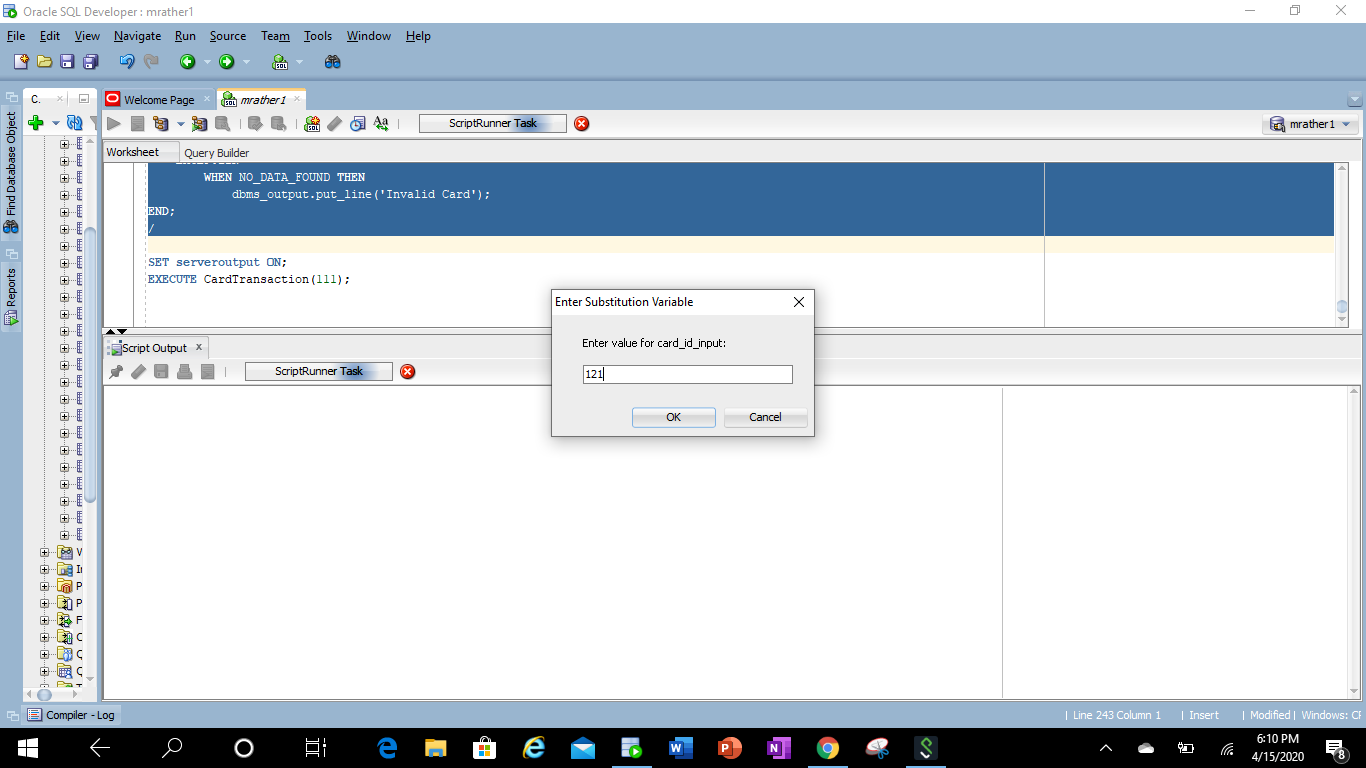
**Display of Procedure**

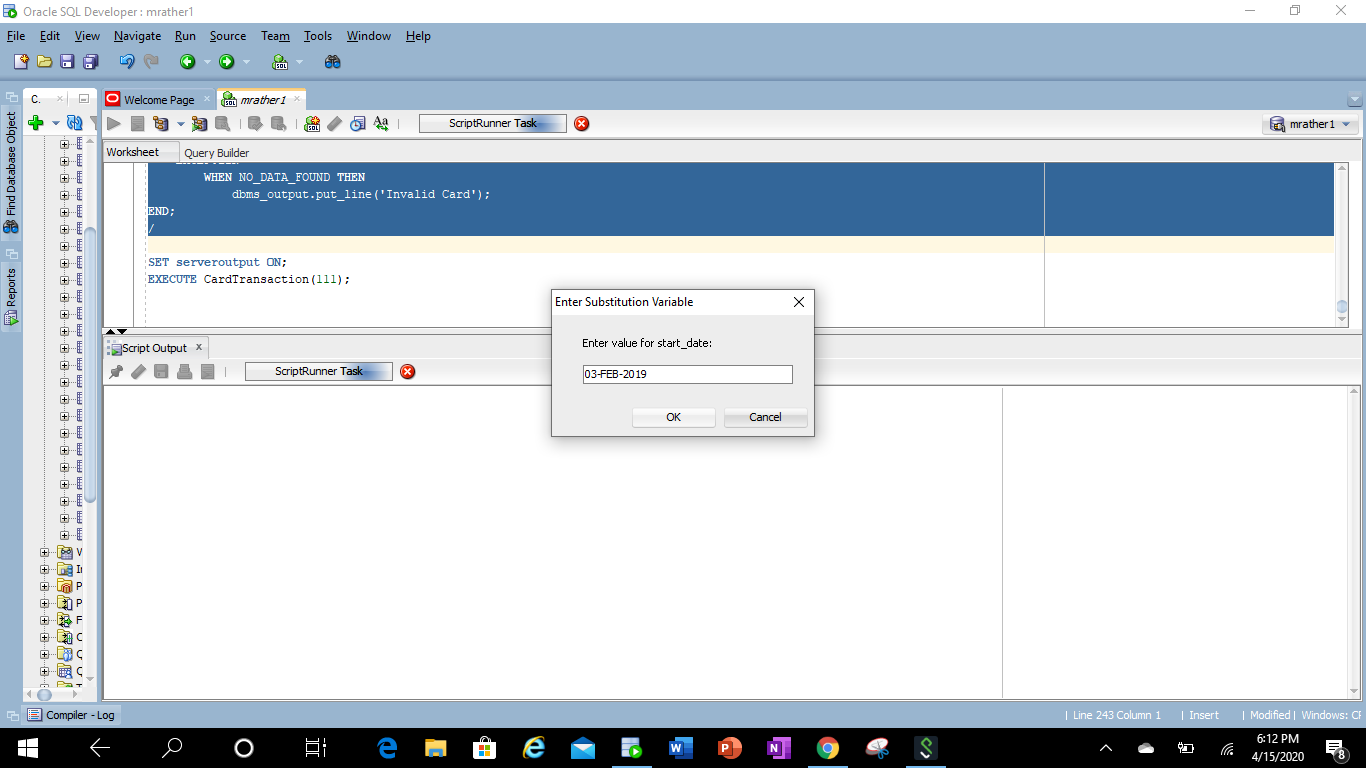
SET serveroutput ON;

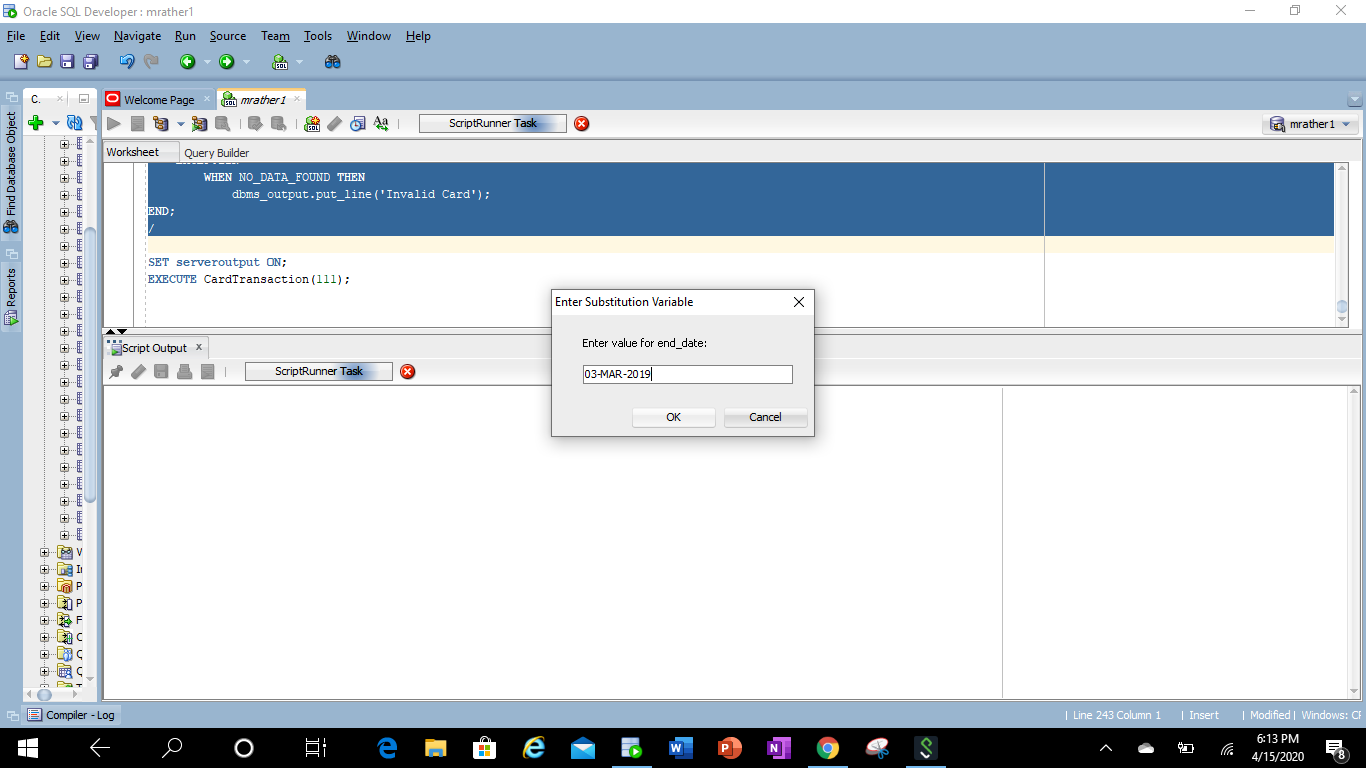
EXECUTE CardTransaction(111);

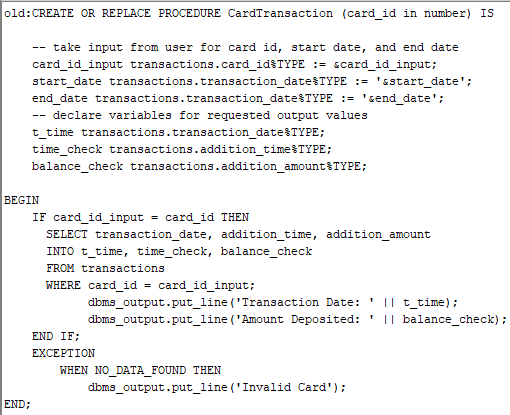


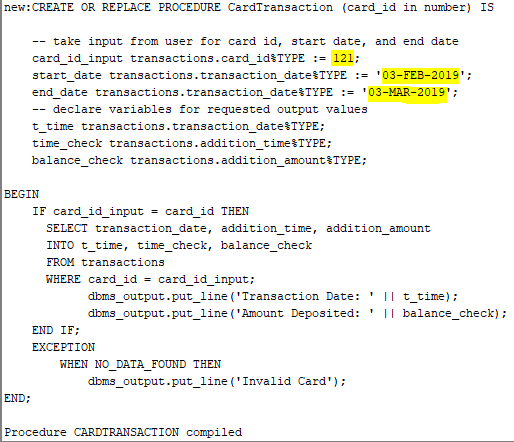
***CARD 2: Invalid Test Scenario***







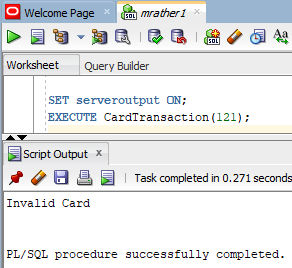




**Display of Procedure**

SET serveroutput ON;

EXECUTE CardTransaction(121);



**Task 7:**

Set serveroutput on;

Exec

**Task 8:**

Set serveroutput on;

Exec checkdeparture(greenbelt, 1500)

SET serveroutput ON;

CREATE OR REPLACE PROCEDURE checkdeparture(station\_name, arrival\_time )

DECLARE

-- take input from user for time, and station name

time\_input schedule\_station.arrival\_time%TYPE := '&time\_input';

station\_input station.station\_name%TYPE := '&station\_input';

-- declare variables for requested output values

schedule\_id\_output schedule.schedule\_id%TYPE;

line\_name\_output train\_line.line\_name%TYPE;

direction\_output schedule.direction%TYPE;

station\_name\_check station.station\_name%TYPE;

BEGIN

SELECT station.station\_name

INTO station\_name\_check

FROM station

WHERE station\_name = station\_input;

FOR result\_info IN (

SELECT schedule\_id, direction

INTO line\_name\_output, direction\_output

FROM schedule, train\_line

WHERE schedule.line\_id = train\_line.line\_id

)

LOOP

dbms\_output.put\_line('Line: ' || line\_name\_output ||

' Direction: ' || direction\_output);

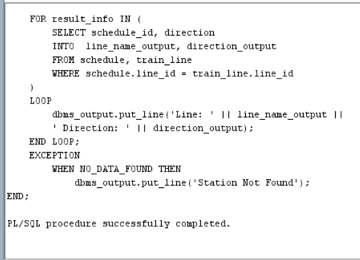
END LOOP;

EXCEPTION

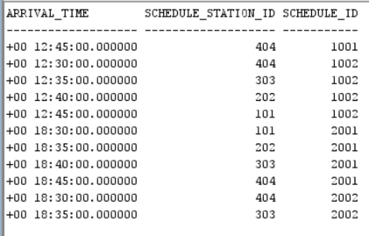
WHEN NO\_DATA\_FOUND THEN

dbms\_output.put\_line('Station Not Found');

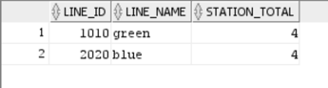
END;



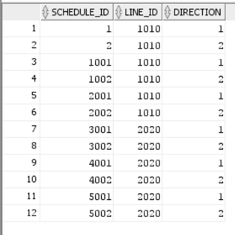
Schedule\_Station table



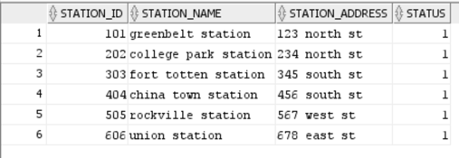
Train\_line table



Schedule table



Station table



**Task 9:**

**--Test 1 increasing**

Set serveroutput on;

DECLARE

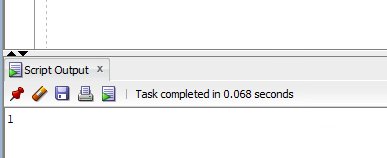
message NUMBER;

BEGIN

message := task9('greenbelt station', 'college park station', 1010);

DBMS\_OUTPUT.put\_line(message);

END;



--Test 2 decreasing

Set serveroutput on;

DECLARE

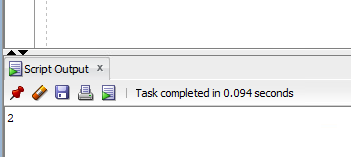
message NUMBER;

BEGIN

message := task9('college park station', 'greenbelt station', 1010);

DBMS\_OUTPUT.put\_line(message);

END;

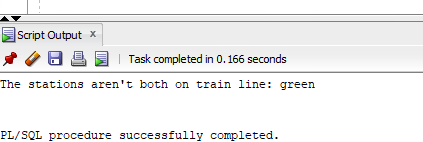


**Task 10:**

--different line test

Set serveroutput on;

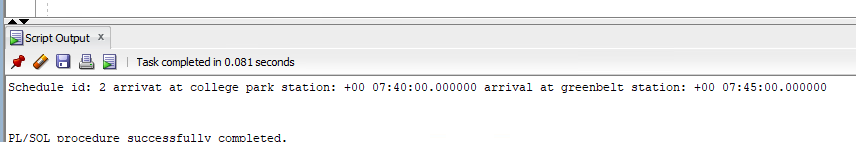
Exec Task10(INTERVAL '0 7:40:00.00' DAY TO SECOND, INTERVAL '50' MINUTE, 'college park station', 'rockville station', 'green');



--same line test

Set serveroutput on;

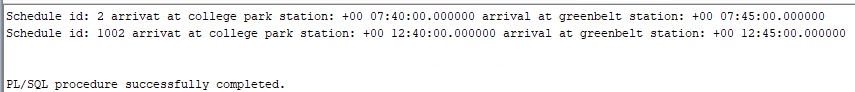
Exec Task10(INTERVAL '0 7:40:00.00' DAY TO SECOND, INTERVAL '50' MINUTE,'college park station', 'greenbelt station', 'green');



--same line test

Set serveroutput on;

Exec Task10 (INTERVAL '0 7:40:00.00' DAY TO SECOND, INTERVAL '500' MINUTE,'college park station', 'greenbelt station', 'green');

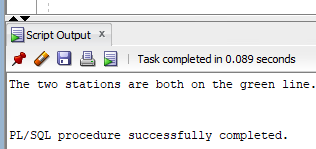


**Task 11:**

--same line test

Set serveroutput on;

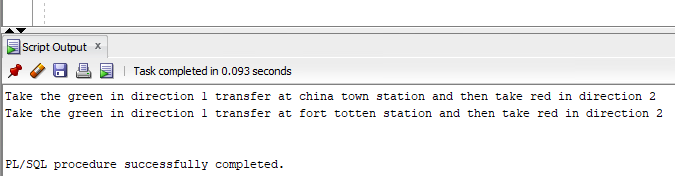
Exec task11( 'college park station', 'greenbelt station');



--different line test

Set serveroutput on;

Exec task11('college park station', 'rockville station');

****

**Task 12:**

Set serveroutput on;

Exec task12;

