Managing trains (pl/sql edition)

@Adrian Zărnoianu

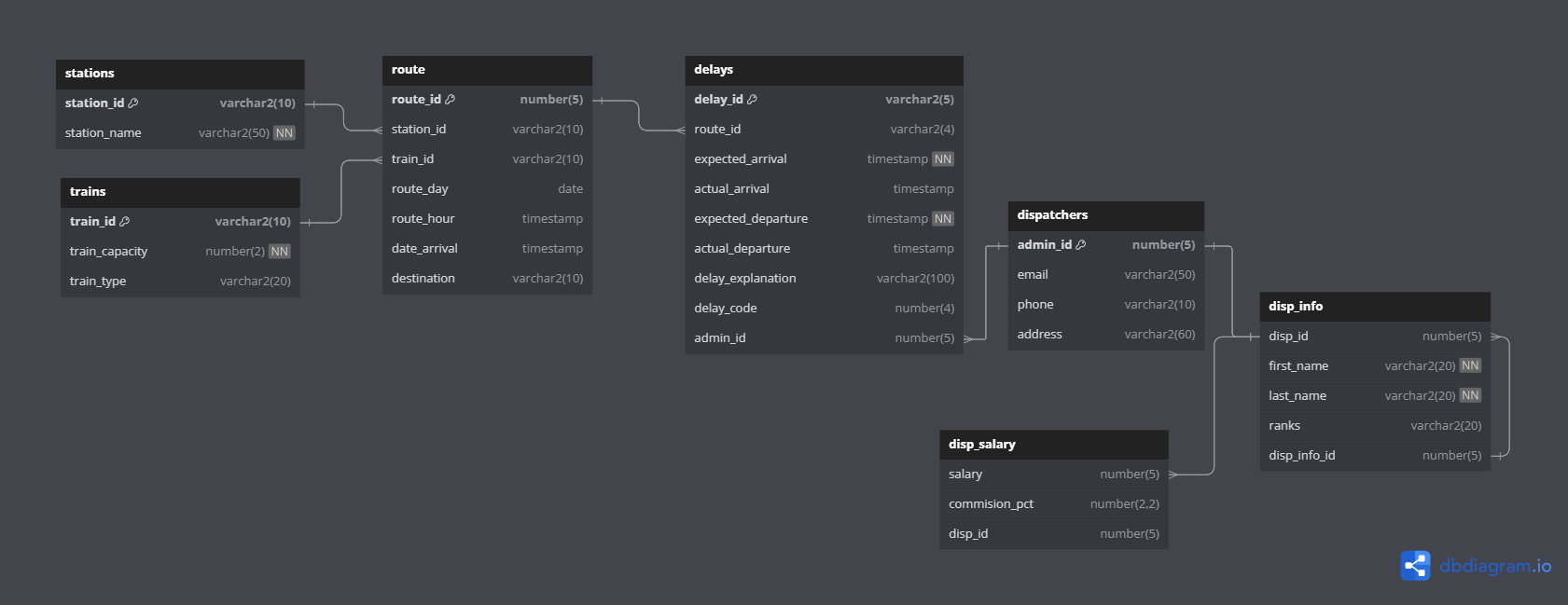
This is the project from the first semester (the tables and inserts from the Database course) alongside new problems and the PL/SQL blocks and commands I added using the theory learned from the DBMS course that solve them.

The database I made follows the trains and their routes, and is made such that users (the database admin, dispatchers, operators and guests) could update it with reports about the trains, which include delay codes and explanations.

The reasons I chose this subject in the first semester are the high frequency of students (including me) who travel by train, and probably the frustration some of us have when the trains we travel by have long delays out of nowhere. This project couldn’t have been done without the help of one of my friends, who knows CFR employees and managed to explain to me the process behind the delay system.

Thus, this database is heavily inspired by Cronos (their site where they manage the trains’ status). However, it is simplified, so a part of the actual real-life problems may not be applied in this project.

# The database query (made with dbdiagram.io):



The tables have not been modified since the first semester. However, I have used another software this time to make the database diagram, now including also the 1:M/1:1 relationship.

The TRAIN table holds basic information about the trains that can be used for routes.

The STATIONS table has station details, so that in the ROUTE table, only registered stations can be used.

The ROUTE table has 3 foreign keys: the train\_id, and two station\_id’s, that represent the station the train leaves from and the station the train goes to. The train\_id is used so that it can be identified in the DELAYS table which of the trains have delays.

The DELAYS table has information about which train, on which route and how much it delayed at departure and arrival. If the trains go out of order, or are cancelled they have special codes.

The DISPATCHERS, DISP\_SALARY AND DISP\_INFO tables have the users, their ranks, and the salaries they have. The DISP\_INFO table has another field for the superiors of the users.

# ALTERNATIVE AND REPETITIVE STRUCTURES

--display the first 4 delay explanations(the delay ids in the table start from 10)

DECLARE

    V\_DELAY\_EXP DELAYS.DELAY\_EXPLANATION%TYPE;

    I NUMBER:=10;

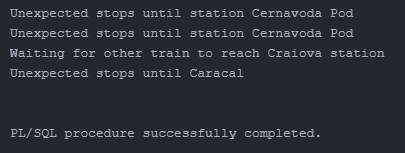
BEGIN

    WHILE I<14 LOOP

        SELECT DELAY\_EXPLANATION INTO V\_DELAY\_EXP FROM DELAYS WHERE DELAY\_ID = I;

        DBMS\_OUTPUT.PUT\_LINE(V\_DELAY\_EXP);

        I:= I+1;

    END LOOP;

END;

/

--update the train\_capacity number to a train

--if there is no train like the input, write a message

DECLARE

    V\_INPUT\_TRAIN TRAINS.TRAIN\_ID%TYPE := :TRAIN\_ID;

BEGIN

    UPDATE TRAINS

    SET TRAIN\_CAPACITY = 3

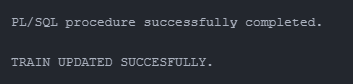
    WHERE TRAIN\_ID = V\_INPUT\_TRAIN;

    IF SQL%NOTFOUND

        THEN DBMS\_OUTPUT.PUT\_LINE('THERE IS NO TRAIN WITH ID '|| V\_INPUT\_TRAIN);

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('TRAIN UPDATED SUCCESFULLY.');

    END IF;

END;

/

(for input R-E9213) 🡪

# DATa collections

The project makes use of varrays, nested tables and index by tables.

NOTE: most of the exercises combine more than two requirements, and I will point out whenever there is a smaller requirement within the PL/SQL blocks

--display (using a varray of records) all the trains and when they leave

DECLARE

    TYPE R\_REC IS RECORD(

        TRAIN\_ID ROUTE.TRAIN\_ID%TYPE,

        ROUTE\_HOUR ROUTE.ROUTE\_HOUR%TYPE

    );

    TYPE T\_ARRAY IS VARRAY(30) OF R\_REC;

    V T\_ARRAY;

    I PLS\_INTEGER;

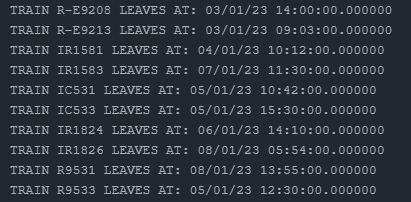
BEGIN

    SELECT TRAIN\_ID, ROUTE\_HOUR BULK COLLECT INTO V

    FROM ROUTE;

    FOR I IN V.FIRST..V.LAST LOOP

        DBMS\_OUTPUT.PUT\_LINE('TRAIN ' || V(I).TRAIN\_ID|| ' LEAVES AT: '||V(I).ROUTE\_HOUR);

    END LOOP;

END;

/

FOR loop, using the I variable

Note: this was part of an exercise which used a procedure, the procedure definition will appear later in this project \/

-- create a procedure, which takes the dispatcher id as a parameter and displays their salary status

-- if the salary is <=5000 the salary is low, if the salary is<=9000, the status is medium, else is high

-- in the anonymous block, call the procedure for a specific id. if it doesn't exist, then raise the proper exception (input is 41172) (the program uses a record)

DECLARE

    V\_DISP\_ID DISP\_SALARY.DISP\_ID%TYPE := &DISP\_ID;

    TYPE R\_INFO IS RECORD(

        FIRST\_NAME DISP\_INFO.FIRST\_NAME%TYPE,

        LAST\_NAME DISP\_INFO.LAST\_NAME%TYPE

    );

    T R\_INFO;

BEGIN

    SELECT FIRST\_NAME, LAST\_NAME INTO T FROM DISP\_INFO

    WHERE DISP\_ID = V\_DISP\_ID;

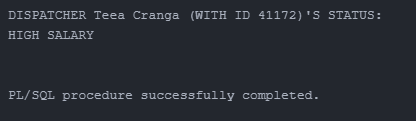
    DBMS\_OUTPUT.PUT\_LINE('DISPATCHER '||T.FIRST\_NAME||' '||T.LAST\_NAME||' (WITH ID '||V\_DISP\_ID||')''S STATUS:');

    SHOW\_SALARY\_STATS(V\_DISP\_ID);

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

  DBMS\_OUTPUT.PUT\_LINE('THIS DISPATCHER DOES NOT EXIST.');

END;

/

Implicit exception added here

# Exception handling

--select the trains which leave from a certain station(in this case bucu-n)

--if there are more than two, raise an exception

--if the station doesn't exist/no train leaves from there, raise NO\_DATA\_FOUND exception

DECLARE

    V\_TRAIN NUMBER;

BEGIN

    SELECT TRAIN\_ID INTO V\_TRAIN FROM ROUTE WHERE STATION\_ID='BUCU-N';

    --SELECT TRAIN\_ID INTO V\_TRAIN FROM ROUTE WHERE STATION\_ID='IASI'; --for no\_data\_found

    EXCEPTION

        WHEN TOO\_MANY\_ROWS THEN

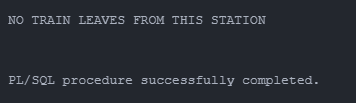
        DBMS\_OUTPUT.PUT\_LINE('THERE ARE MORE THAN TWO TRAINS WHICH LEAVE FROM BUCU-N');

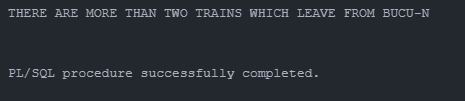
        WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('NO TRAIN LEAVES FROM THIS STATION');

END;

/



this happens when I use

the second select 🡪

this happens when

I use the first select 🡪

NOTE: The following exclusive exception is from a procedure, so I won’t include the procedure later

--make a procedure which shows the email of a certain dispatcher using the id. if the id doesn't exist, raise an exception

CREATE OR REPLACE PROCEDURE GET\_DISP\_DATA(P\_DISP\_ID DISPATCHERS.ADMIN\_ID%TYPE) IS

    CURSOR CUR IS SELECT FIRST\_NAME, LAST\_NAME, EMAIL, PHONE FROM DISPATCHERS D

    JOIN DISP\_INFO I ON ADMIN\_ID = DISP\_ID

    WHERE ADMIN\_ID = P\_DISP\_ID;

    R CUR%ROWTYPE;

    EXCEP EXCEPTION;

    PRAGMA EXCEPTION\_INIT(EXCEP,-20000);

BEGIN

    OPEN CUR;

    FETCH CUR INTO R;

    IF CUR%NOTFOUND THEN RAISE EXCEP;

    END IF;

    DBMS\_OUTPUT.PUT\_LINE(R.FIRST\_NAME||' '||R.LAST\_NAME||'''S DETAILS: '||R.EMAIL||' '||R.PHONE);

CLOSE CUR;

EXCEPTION

WHEN EXCEP THEN

    DBMS\_OUTPUT.PUT\_LINE('DISPATCHER NOT FOUND');

END;

/

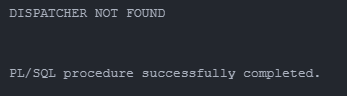
BEGIN

--GET\_DISP\_DATA(10003534534532); --this will raise the exception

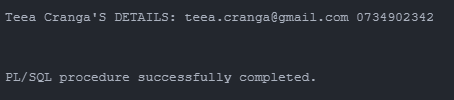
GET\_DISP\_DATA(41172);

END;

/

If I use the first call of the procedure, it will raise the exception

Otherwise, it will show the details as in the image below:



# cursor management

--make an explicit cursor which explains all the data in the trains table

DECLARE

    CURSOR CUR\_TRAINS IS

    SELECT TRAIN\_TYPE, TRAIN\_ID, TRAIN\_CAPACITY

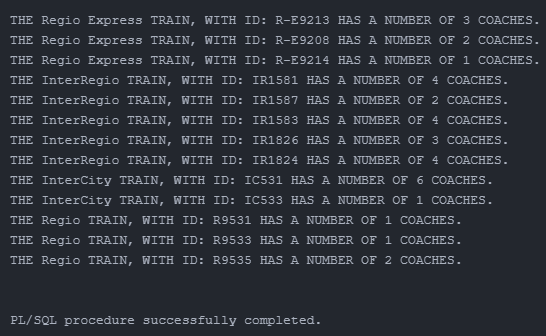
    FROM TRAINS;

BEGIN

    FOR V\_TRAIN\_RECORD IN CUR\_TRAINS LOOP

        DBMS\_OUTPUT.PUT\_LINE('THE ' || V\_TRAIN\_RECORD.TRAIN\_TYPE ||' TRAIN, WITH ID: ' || V\_TRAIN\_RECORD.TRAIN\_ID || ' HAS A NUMBER OF ' || V\_TRAIN\_RECORD.TRAIN\_CAPACITY || ' COACHES.');

    END LOOP;

END;

/

--create an explicit cursor which takes the user\_ids of those who reported delays on a specific train (use a parameter)

--input is 1502 in the pl/sql block

DECLARE

    CURSOR CUR(P\_ROUTE\_ID DELAYS.ROUTE\_ID%TYPE) IS

    SELECT ADMIN\_ID FROM DELAYS

    WHERE ROUTE\_ID = P\_ROUTE\_ID;

    R CUR%ROWTYPE;

BEGIN

    OPEN CUR(1502);

    LOOP

        FETCH CUR INTO R;

        EXIT WHEN CUR%NOTFOUND;

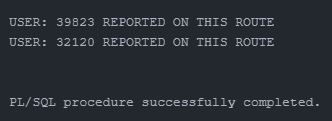
        DBMS\_OUTPUT.PUT\_LINE('USER: '||R.ADMIN\_ID||' REPORTED ON THIS ROUTE');

    END LOOP;

    CLOSE CUR;

END;

/



# functions, procedures, packages

--create a procedure which displays the train and how many minutes of delay it had at both arrival and departure

--if the train has no delay at either of the actions, replace with -1

CREATE OR REPLACE PROCEDURE DELAYS\_TRAIN(P\_TRAIN\_ID ROUTE.TRAIN\_ID%TYPE) IS

    CURSOR CUR IS SELECT R.TRAIN\_ID, R.ROUTE\_ID, NVL(ABS(EXTRACT(MINUTE FROM D.ACTUAL\_ARRIVAL-D.EXPECTED\_ARRIVAL)),-1) ARRIVAL,

    NVL(ABS(EXTRACT(MINUTE FROM D.ACTUAL\_DEPARTURE-D.EXPECTED\_DEPARTURE)),-1) DEPARTURE

    FROM DELAYS D, ROUTE R

    WHERE D.ROUTE\_ID=R.ROUTE\_ID

    AND R.TRAIN\_ID = P\_TRAIN\_ID

    ORDER BY R.TRAIN\_ID;

    R CUR%ROWTYPE;

BEGIN

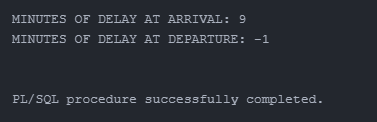
    OPEN CUR;

    FETCH CUR INTO R;

    DBMS\_OUTPUT.PUT\_LINE('MINUTES OF DELAY AT ARRIVAL: '|| R.ARRIVAL||CHR(10)||'MINUTES OF DELAY AT DEPARTURE: '||R.DEPARTURE);

END;

/

BEGIN

    DELAYS\_TRAIN('IR1583');

END;

/

NOTE: the procedure call block and its output are above, at the exception chapter

-- create a procedure, which takes the dispatcher id as a parameter and displays their salary status

-- if the salary is <=5000 the salary is low, if the salary is<=9000, the status is medium, else is high

-- in the anonymous block, call the procedure for a specific id. if it doesn't exist, then raise the proper exception (input is 41172)

CREATE OR REPLACE PROCEDURE SHOW\_SALARY\_STATS(P\_DISP\_ID DISP\_SALARY.SALARY%TYPE) IS

V\_SALARY NUMBER;

BEGIN

    SELECT SALARY INTO V\_SALARY

    FROM DISP\_SALARY

    WHERE DISP\_ID=P\_DISP\_ID; CASE used

    CASE

    WHEN V\_SALARY<=5000 THEN

        DBMS\_OUTPUT.PUT\_LINE('LOW SALARY');

    WHEN V\_SALARY<=9000 THEN

        DBMS\_OUTPUT.PUT\_LINE('MEDIUM SALARY');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('HIGH SALARY');

    END CASE;

END;

/

-- create a function which returns true/false depending on the status of that train

--(if the train has the delay code 1003/1004, it's cancelled/broken, else, it will reach the final station soon)

--create a table of records to store the delays and the routes which they belong to and show how many trains were cancelled/broken during their rides

CREATE OR REPLACE FUNCTION IS\_CANCELLED\_OR\_BROKEN(P\_DELAY\_ID DELAYS.DELAY\_ID%TYPE) RETURN BOOLEAN IS

V\_DELAY DELAYS.DELAY\_CODE%TYPE;

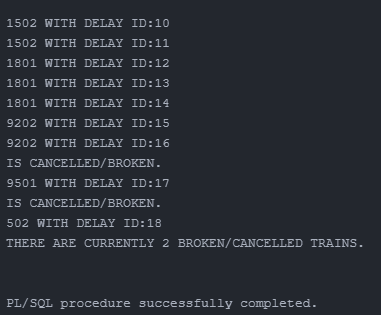
BEGIN

    SELECT DELAY\_CODE

    INTO V\_DELAY nested table, with index by

    FROM DELAYS

    WHERE DELAY\_ID = P\_DELAY\_ID;

    IF V\_DELAY IN(1003,1004) THEN

        RETURN TRUE;

    ELSE RETURN FALSE;

    END IF;

END;

/

DECLARE

    TYPE RECORD\_TRAINS IS RECORD(

        R\_ROUTE\_ID DELAYS.ROUTE\_ID%TYPE,

        R\_DELAY\_ID DELAYS.ROUTE\_ID%TYPE

    );

    TYPE T\_NAME IS TABLE OF RECORD\_TRAINS INDEX BY PLS\_INTEGER;

    V T\_NAME;

    I PLS\_INTEGER;

    V\_NUMBER\_CANCELLED\_BROKEN NUMBER := 0;

BEGIN

    SELECT ROUTE\_ID, DELAY\_ID BULK COLLECT INTO V

    FROM DELAYS;

    FOR I IN V.FIRST .. V.LAST LOOP

        DBMS\_OUTPUT.PUT\_LINE(V(I).R\_ROUTE\_ID||' WITH DELAY ID:'||V(I).R\_DELAY\_ID);

        IF IS\_CANCELLED\_OR\_BROKEN(V(I).R\_DELAY\_ID) THEN

            DBMS\_OUTPUT.PUT\_LINE('IS CANCELLED/BROKEN.');

            V\_NUMBER\_CANCELLED\_BROKEN:=V\_NUMBER\_CANCELLED\_BROKEN+1;

        END IF;

    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('THERE ARE CURRENTLY '||V\_NUMBER\_CANCELLED\_BROKEN||' BROKEN/CANCELLED TRAINS.');

END;

/

--create two functions: one of them returns the station name the train leaves from and one which returns the name of the station in which it arrives to.

--use them in a plsql block

CREATE OR REPLACE FUNCTION GET\_STATION\_NAME(P\_ROUTE\_ID IN ROUTE.ROUTE\_ID%TYPE) RETURN VARCHAR2 IS

    P\_STATION\_NAME STATIONS.STATION\_NAME%TYPE;

BEGIN

    SELECT STATION\_NAME INTO P\_STATION\_NAME FROM STATIONS

    WHERE STATION\_ID = (SELECT STATION\_ID FROM ROUTE WHERE ROUTE\_ID = P\_ROUTE\_ID);

    RETURN P\_STATION\_NAME;

END;

/

BEGIN

    DBMS\_OUTPUT.PUT\_LINE(GET\_STATION\_NAME(9201));

END;

/

CREATE OR REPLACE FUNCTION GET\_DEST\_NAME(P\_ROUTE\_ID IN ROUTE.ROUTE\_ID%TYPE) RETURN VARCHAR2 IS

    P\_STATION\_NAME STATIONS.STATION\_NAME%TYPE;

BEGIN

    SELECT STATION\_NAME INTO P\_STATION\_NAME FROM STATIONS

    WHERE STATION\_ID = (SELECT DESTINATION FROM ROUTE WHERE ROUTE\_ID = P\_ROUTE\_ID);

    RETURN P\_STATION\_NAME;

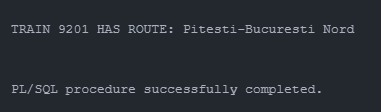
END;

/

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('TRAIN 9201 HAS ROUTE: '||GET\_STATION\_NAME(9201)||'-'||GET\_DEST\_NAME(9201));

END;

/

## THE PACKAGe definition

--create a package

CREATE OR REPLACE PACKAGE PACK\_UTILS AS

    --gets the full name of the dispatcher

    FUNCTION GET\_FULL\_NAME(P\_DISP\_ID IN DISP\_INFO.DISP\_ID%TYPE) RETURN VARCHAR2;

    --gets the main superior of the user

    FUNCTION GET\_SUPERIOR(P\_DISP\_ID IN DISP\_INFO.DISP\_ID%TYPE) RETURN VARCHAR2;

    --prints the rank of the user

    PROCEDURE PRINT\_RANK(P\_DISP\_ID DISP\_INFO.DISP\_ID%TYPE);

END;

/

CREATE OR REPLACE PACKAGE BODY PACK\_UTILS AS

    FUNCTION GET\_FULL\_NAME(P\_DISP\_ID IN DISP\_INFO.DISP\_ID%TYPE) RETURN VARCHAR2 IS

        V\_FULL\_NAME VARCHAR2(200);

        V\_FIRST\_NAME VARCHAR2(100);

        V\_LAST\_NAME VARCHAR2(100);

    BEGIN

        SELECT FIRST\_NAME, LAST\_NAME

        INTO V\_FIRST\_NAME, V\_LAST\_NAME

        FROM DISP\_INFO

        WHERE DISP\_ID = P\_DISP\_ID;

        V\_FULL\_NAME := V\_FIRST\_NAME||' '||V\_LAST\_NAME;

        RETURN V\_FULL\_NAME;

    END GET\_FULL\_NAME;

    FUNCTION GET\_SUPERIOR(P\_DISP\_ID IN DISP\_INFO.DISP\_ID%TYPE) RETURN VARCHAR2 IS

        V\_SUPERIOR\_ID DISP\_INFO.DISP\_ID%TYPE;

        V\_FIRST\_NAME VARCHAR2(100);

        V\_LAST\_NAME VARCHAR2(100);

        V\_FULL\_INFO VARCHAR2(200);

    BEGIN

        SELECT DISP\_INFO\_ID

        INTO V\_SUPERIOR\_ID

        FROM DISP\_INFO

        WHERE DISP\_ID = P\_DISP\_ID;

        SELECT FIRST\_NAME, LAST\_NAME

        INTO V\_FIRST\_NAME, V\_LAST\_NAME

        FROM DISP\_INFO

        WHERE DISP\_ID=V\_SUPERIOR\_ID;

        V\_FULL\_INFO := ' WITH SUPERIOR: '|| V\_SUPERIOR\_ID||' '||V\_FIRST\_NAME||' '||V\_LAST\_NAME;

        RETURN V\_FULL\_INFO;

        EXCEPTION

            WHEN NO\_DATA\_FOUND THEN

            RETURN 'THE USER IS THE ADMIN.';

    END GET\_SUPERIOR;

    PROCEDURE PRINT\_RANK(P\_DISP\_ID DISP\_INFO.DISP\_ID%TYPE) IS

        V\_RANKS VARCHAR2(20);

    BEGIN

        SELECT RANKS INTO V\_RANKS FROM DISP\_INFO

        WHERE DISP\_ID = P\_DISP\_ID;

        DBMS\_OUTPUT.PUT\_LINE('THE USER''S RANK IS: '|| V\_RANKS);

    END PRINT\_RANK;

END;

/

--show all the users, their ranks and their superiors

DECLARE

    FULL\_NAME VARCHAR2(200);

    SUP\_NAME VARCHAR2(200);

BEGIN

    SELECT

    PACK\_UTILS.GET\_FULL\_NAME(DISP\_ID),

    PACK\_UTILS.GET\_SUPERIOR(DISP\_ID) INTO FULL\_NAME, SUP\_NAME

    FROM DISP\_INFO

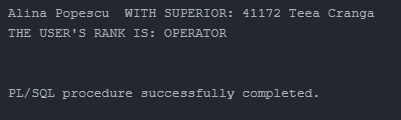
    WHERE DISP\_ID = 21031;

    DBMS\_OUTPUT.PUT\_LINE(FULL\_NAME|| ' '||SUP\_NAME);

    PACK\_UTILS.PRINT\_RANK(21031);

END;

/



--create a trigger to prevent modifying the salary of guests

--they have the commission pct of 0.1

CREATE OR REPLACE TRIGGER CH\_SALARY\_GUESTS AFTER UPDATE OF SALARY ON DISP\_SALARY FOR EACH ROW

BEGIN

    IF :NEW.SALARY<>:OLD.SALARY AND :NEW.COMMISSION\_PCT = 0.1 THEN

    RAISE\_APPLICATION\_ERROR(-20001,'WE CAN''T CHANGE THE SALARY');

   END IF;

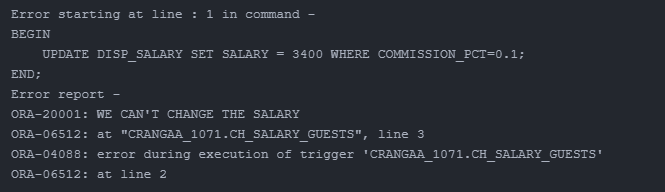
END;

/

BEGIN

    UPDATE DISP\_SALARY SET SALARY = 3400 WHERE COMMISSION\_PCT=0.1;

END;

/

--create a trigger which prevents adding trains with more thean 10 coaches or less or equal to 0

CREATE OR REPLACE TRIGGER CH\_TRAIN\_COACHES BEFORE INSERT ON TRAINS FOR EACH ROW

BEGIN

    IF :NEW.TRAIN\_CAPACITY < 0 OR :NEW.TRAIN\_CAPACITY > 10 THEN

        DBMS\_OUTPUT.PUT\_LINE('THE TRAIN HAS AN INVALID TRAIN CAPACITY.');

    END IF;

END;

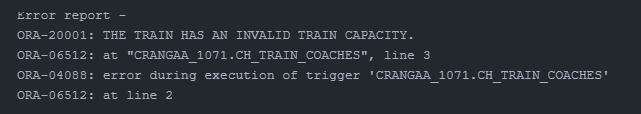
/

BEGIN

    INSERT INTO TRAINS VALUES('R10310', 11, 'Regio Express');

END;

/



--create a trigger which prevents adding duplicate dispatchers in the disp\_info table

CREATE OR REPLACE TRIGGER DUP\_USER AFTER INSERT OR UPDATE ON DISP\_INFO

DECLARE

    V\_DUPLICATES NUMBER;

BEGIN

    SELECT COUNT(\*) INTO V\_DUPLICATES FROM DISP\_INFO

    WHERE PACK\_UTILS.GET\_FULL\_NAME(DISP\_ID) IN

    (SELECT PACK\_UTILS.GET\_FULL\_NAME(DISP\_ID) AS "FULL NAME"

     FROM DISP\_INFO

     GROUP BY PACK\_UTILS.GET\_FULL\_NAME(DISP\_ID)

     HAVING COUNT(\*) > 1

    );

   IF V\_DUPLICATES > 0 THEN

      RAISE\_APPLICATION\_ERROR(-20001, 'DUPLICATE DISPATCHER NAME FOUND!');

   END IF;

END;

/

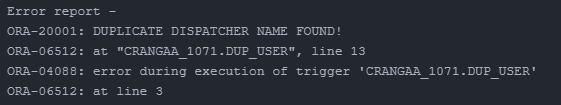
BEGIN

    INSERT INTO DISPATCHERS VALUES('23423', 'ABC@GMAIL.COM', '03243242','ASDASDAS');

    INSERT INTO DISP\_INFO VALUES('23423','OPERATOR','21031', 'Alina','Vlaicu');

END;

/



--create a trigger which prevents modifying delays

CREATE OR REPLACE TRIGGER INS\_SAME\_STATION AFTER INSERT OR UPDATE ON ROUTE

DECLARE

    CURSOR CUR IS SELECT STATION\_ID, DESTINATION

    FROM ROUTE;

    R CUR%ROWTYPE;

BEGIN

    OPEN CUR;

    LOOP

        FETCH CUR INTO R;

        EXIT WHEN CUR%NOTFOUND;

        IF R.STATION\_ID = R.DESTINATION THEN

            RAISE\_APPLICATION\_ERROR(-20001,'THE TRAIN CANNOT GO TO THE SAME STATION WHERE IT DEPARTS FROM.');

        END IF;

    END LOOP;

    CLOSE CUR;

END;

/

BEGIN

    INSERT INTO ROUTE VALUES(9203,'PITE','R-E9208', TO\_DATE('05-01-2023','dd-mm-yyyy'),TO\_TIMESTAMP('05-01-2023 10:42','dd-mm-yyyy hh24:mi'),TO\_TIMESTAMP('05-01-2023 12:34','dd-mm-yyyy hh24:mi'),'PITE');

END;

/

