Department of Information Systems and Technologies

CTIS259 Database Management Systems and Applications

2025 – 2026 Fall

Lab Guide 17

Instructor : Nimet Ceren SERİM
Assistant : Engin Zafer KIRAÇBEDEL, Hatice Zehra YILMAZ

Week: 11
Date: 27-28.11.2025

Aim of this lab session: **1.** PL/SQL Programming Language

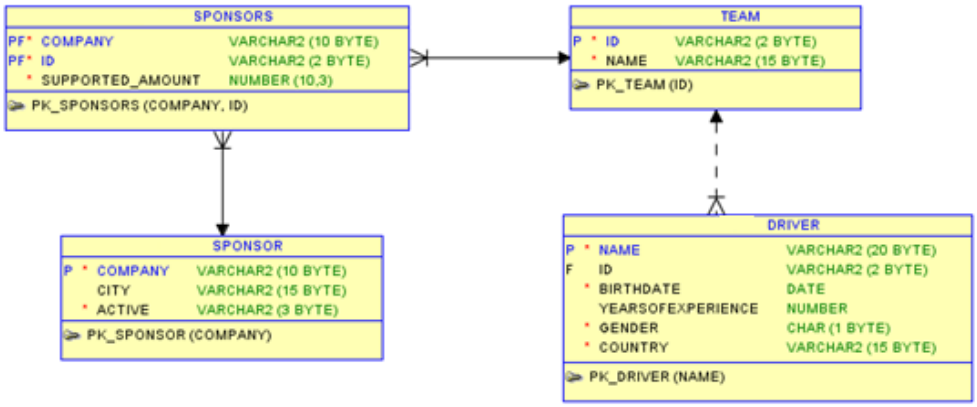
ORACLE Server Configurations:
IP Address: 139.179.33.231
Port number: 1522
SID: orclctis

Please USE student (fStudentID) accounts

PART I

The objective of this guide is to make you practice on PL/SQL environments by creating anonymous and named blocks. The sample database schema **Formula Race Organization** will be used for this purpose.

- There is a Formula Racing organization where there are teams, drivers and sponsors. Sponsors may sponsor one or more teams. A team may also be sponsored by one or more sponsors. Each team must have at least one sponsor and one driver. A driver may be involved in only a single team.
- Below is a Physical Data Model for this E-R data model. Examine the data and data types in the tables and relationships among the tables. Also notice the primary keys and foreign keys in the tables.
- And below is given a relational database instance.



Sponsor

Company	City	Active
VESTEL	İzmir	No
PO	Ankara	Yes
CMS	İstanbul	No
OYAK	Ankara	Yes

Team

Id	Name
T1	Ferrari
T2	Renault
T3	McLaren
T4	Toyota
T5	Toro Rosso

Sponsors

Company	Id	Supported_Amount
VESTEL	T5	500000.250
VESTEL	T3	650000.000
PO	T1	475000.300
OYAK	T2	640000750
PO	T5	800000.000
PO	T3	630000.500

Driver

Name	Id	BirthDate	YearsOfExperience	Gender	Country
Artam	T2	30.06.1981	2	M	Turkey
Alonso	T2	29.07.1981	5	M	Spain
Trulli	T5	13.07.1974	10	M	Italy
aikkonen	T3	17.10.1979	6	M	Finland
Kuling	T1	11.01.1982	NULL	F	Malta
Webber	T4	27.08.1976	5	M	Australia
Derin	T3	14.06.1981	1	F	Turkey

STEP I
Download and run the database creation and data insertion scripts (FormulaDB) from moodle website.
(Folders: LabGuides/Lab17)
Note that the order of tables in this process is important since there are Foreign Key Constraints on some of the tables.

STEP II
Create an anonymous block which calculates the number of drivers in the Driver table and prints the result.

```
SET SERVEROUTPUT ON;
DECLARE
cnt       NUMBER(2);
BEGIN
  SELECT COUNT(*) INTO cnt FROM Driver;

  IF cnt = 0 THEN
     dbms_output.put_line('There are no drivers.');
  ELSE
     dbms_output.put_line('There are ' || cnt || ' drivers.');
  END IF;
END;
/
```

Important:
- Remember that anonymous blocks are just run once and you cannot call them.
- Please notice the difference between single quote types: the character **'** is a single quote character accepted by Oracle. However, the character **’** and other similar smart single quote characters are not accepted.
- Whenever you like to print an output, you must issue "**set serveroutput**"**command** at the beginning of the anonymous blocks.

STEP III If you would like to perform parametric operations in PL/SQL, you should use variables with '&' prefix in your scripts. In order to exercise this issue, assume that you would like to create an anonymous block such that it gets the sponsor name from the user and calculates the average of the supported amount for that sponsor. The calculated average is displayed with some rating depending on the amount:

```
SET SERVEROUTPUT ON;
DECLARE
avg_amount      NUMBER(15,5);
user_company VARCHAR2(10) := '&user_company';
BEGIN
  SELECT AVG(supported_amount) INTO avg_amount
 FROM Sponsors WHERE Company=user_company;
  IF avg_amount = 0 THEN
     dbms_output.put_line('No support is given by ' || user_company);
  ELSIF avg_amount < 500000 THEN
     dbms_output.put_line(user_company || ': Fairly Good Amount with ' || avg_amount);
  ELSE
     dbms_output.put_line(user_company || ': Among the Top Sponsors with ' || avg_amount);
  END IF;
END;
/
```

When the user is prompted for data entry, **&user_company** must have a string value. Try OYAK as the parameter value. The script runs successfully.
The selection criterion is case-sensitive. Try 'Oyak' as the parameter. The SELECT command returns NULL as the value, and hence **avg_amount** is NULL. A result is displayed with empty value.

In order to remove this anomaly, handle this case by updating your script as follows:

```
SET SERVEROUTPUT ON;
DECLARE
avg_amount NUMBER(15,5);
user_company VARCHAR2(10):='&user_company';
BEGIN
  SELECT AVG(supported_amount) INTO avg_amount
 FROM Sponsors WHERE Company=user_company;
  IF avg_amount IS NULL THEN
          dbms_output.put_line(user_company || ' is not among sponsors.');
  ELSIF avg_amount = 0 THEN
     dbms_output.put_line('No support is given by ' || user_company);
  ELSIF avg_amount < 500000 THEN
     dbms_output.put_line(user_company || ': Fairly Good Amount with ' || avg_amount);
  ELSE
     dbms_output.put_line(user_company || ': Among the Top Sponsors with ' || avg_amount);
  END IF;
END;
/
```

Note: Note the following points:

1. **&user_company** is a variable and serves as a parameter. It is not declared anywhere.
2. In the IF Statement, **ELSIF** is used instead of conventional ELSEIF.
3. The parameter variable can be used in the other parts of the scripts than its first usage point.

In this part, you will convert the anonymous block into a stored procedure. The procedure will accept the company name and output the results.

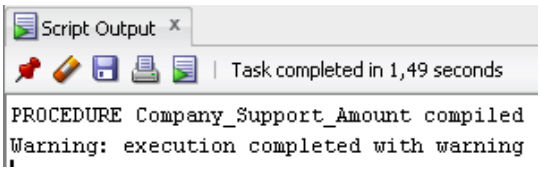
PART II

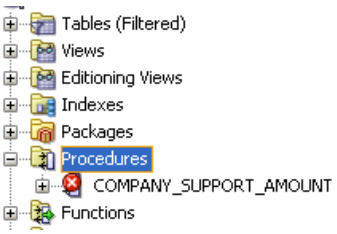
STEP I Start converting your script by:

1. Writing a "**CREATE or REPLACE PROCEDURE**" command,
2. Declaring **user_company** as **IN** (input parameter) and then its type,
3. Remove "**DECLARE**" statement and put "**IS**" in place of it.
4. Remove "&" from all occurrences of **&user_company** since it is not an externally provided variable.

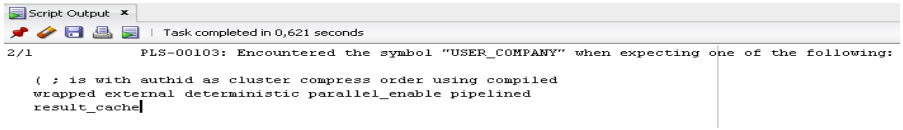
```
CREATE OR REPLACE PROCEDURE Company_Support_Amount
user_company IN VARCHAR2(10);
IS
avg_amount NUMBER(15,5);
BEGIN
  SELECT AVG(supported_amount) INTO avg_amount
 FROM Sponsors WHERE Company=user_company;
  IF avg_amount IS NULL THEN
          dbms_output.put_line(user_company || ' is not among sponsors.');
  ELSIF avg_amount = 0 THEN
     dbms_output.put_line('No support is given by ' || user_company);
  ELSIF avg_amount < 500000 THEN
     dbms_output.put_line(user_company ||       ': Fairly Good Amount with ' || avg_amount);
  ELSE
     dbms_output.put_line(user_company ||       ': Among the Top Sponsors with ' || avg_amount);
  END IF;
END;
/
```

5. Execute it now. You should get a compilation error. In order to see what the error is, execute the command and examine the error. (When you create a procedure or function, you may see them on the **connections part** under the **Procedures/Functions**)






```
SHOW ERRORS PROCEDURE Company_Support_Amount;
```



6. The input/output parameters should be enclosed in parenthesis. Furthermore, the length of parameters is not indicated. So, remove (10) from VARCHAR2 data type.

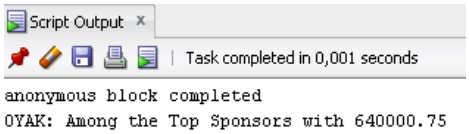
```
CREATE OR REPLACE PROCEDURE Company_Support_Amount
(user_company IN VARCHAR2)
IS
avg_amount NUMBER(15,5);
BEGIN
  SELECT AVG(supported_amount) INTO avg_amount
 FROM Sponsors WHERE Company=user_company;

  IF avg_amount IS NULL THEN
          dbms_output.put_line(user_company || ' is not among sponsors.');
  ELSIF avg_amount = 0 THEN
     dbms_output.put_line('No support is given by ' || user_company);
  ELSIF avg_amount < 500000 THEN
     dbms_output.put_line(user_company || ': Fairly Good Amount with ' || avg_amount);
  ELSE
     dbms_output.put_line(user_company || ': Among the Top Sponsors with ' || avg_amount);
  END IF;
END;
/
```

7. Now your procedure is ready for use.

STEP II

In order to call your procedure, write the following anonymous block and observe the result.

```
SET SERVEROUTPUT ON;
BEGIN
  Company_Support_Amount('OYAK');
END;
/
```

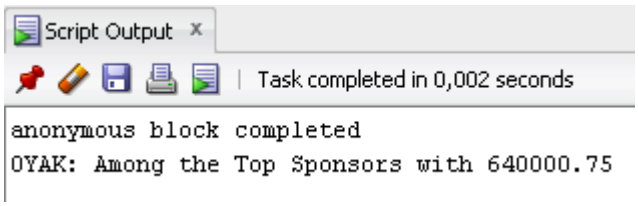


```
CREATE OR REPLACE PROCEDURE Company_Support_Amount
(user_company IN VARCHAR2, the_result OUT VARCHAR2)
IS
avg_amount NUMBER(15,5);
BEGIN
  SELECT AVG(supported_amount) INTO avg_amount
 FROM Sponsors WHERE Company=user_company;
  IF avg_amount IS NULL THEN
     the_result  := user_company || ' is not among sponsors.';
  ELSIF avg_amount = 0 THEN
     the_result  := 'No support is given by ' || user_company;
  ELSIF avg_amount < 500000 THEN
     the_result  :=  user_company ||  ': Fairly Good Amount with ' || avg_amount;
  ELSE
     the_result  := user_company || ': Among the Top Sponsors with ' || avg_amount;
  END IF;
END;
/
```

STEP III Instead of outputting the results on the screen, you may want to get the results back through an output parameter. In our example, add an output (string) parameter to the procedure and assign the displayed string to this parameter within the block:

If your procedure is successfully compiled, then execute the following anonymous block where it calls the procedure with two parameters, latter being the output parameter. And then it prints the output string on the screen.

```
SET SERVEROUTPUT ON;
DECLARE
the_output VARCHAR2(50);

BEGIN
  Company_Support_Amount('OYAK', the_output);
  DBMS_OUTPUT.PUT_LINE(the_output);
END;
/
```



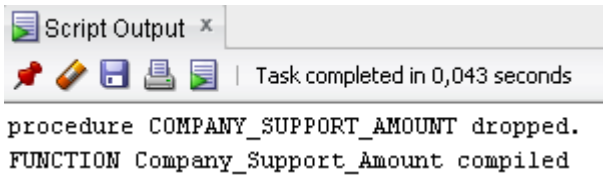
STEP IV
You can turn your procedure into a function by a simple conversion.

Drop the procedure first.

```
DROP PROCEDURE Company_Support_Amount;
```

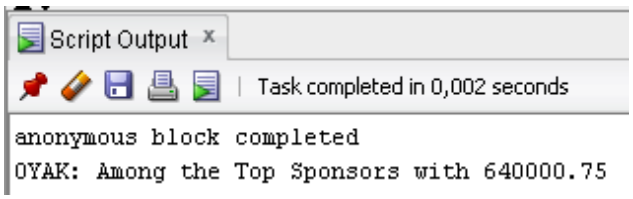
Do the followings:
1. Replace the creating command by "**CREATE or REPLACE FUNCTION**."
2. Remove the output parameter from the procedure .
3. After the parameter enclosing parenthesis, write "**RETURN VARCHAR2**."
4. Replace **"the_result :="** assignment by a "**RETURN**" statement.

```
CREATE OR REPLACE FUNCTION Company_Support_Amount
(user_company IN VARCHAR2) RETURN VARCHAR2
IS
avg_amount NUMBER(15,5);
BEGIN
  SELECT AVG(supported_amount) INTO avg_amount
 FROM Sponsors WHERE Company=user_company;

  IF avg_amount IS NULL THEN
          RETURN user_company || ' is not among sponsors.';
  ELSIF avg_amount = 0 THEN
     RETURN 'No support is given by ' || user_company;
  ELSIF avg_amount < 500000 THEN
     RETURN user_company ||  ': Fairly Good Amount with ' || avg_amount;
  ELSE
     RETURN user_company || ': Among the Top Sponsors with ' || avg_amount;
  END IF;
END;
/
```



If your function is successfully compiled, then execute the following anonymous block where it calls the function, the result is stored in a local variable, and it is printed on the screen.

```
SET SERVEROUTPUT ON;
DECLARE
the_output VARCHAR2(50);

BEGIN
  the_output := Company_Support_Amount('OYAK');
  DBMS_OUTPUT.PUT_LINE(the_output);
END;
/
```



PART III

Handling errors is a vital issue in software engineering. The same applies to PL/SQL scripts since you develop a software code segment to carry out some specific task. In this part, you will experience the need for exception handling and how you should write handlers in PL/SQL.

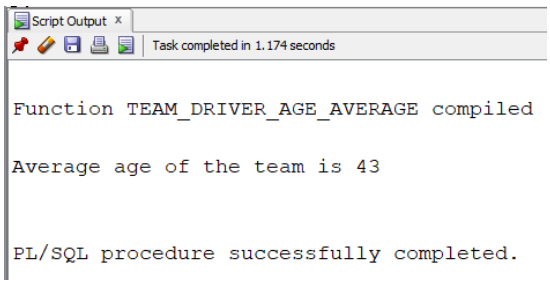
STEP I

Assume that you would like to get the drivers of a specific team and computes ages of the drivers, take averages of their ages and return the result through a function. Below is the first attempt to do it.

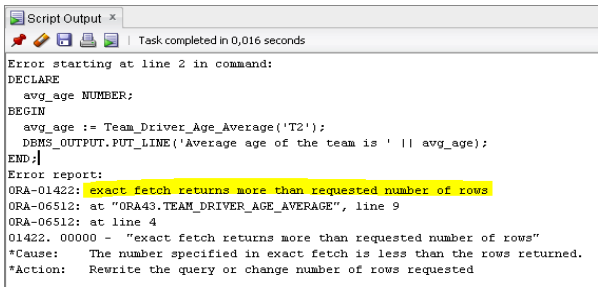
```
CREATE OR REPLACE FUNCTION Team_Driver_Age_Average
(team_id IN VARCHAR2)
RETURN NUMBER
IS
total_age        INTEGER:=0;
avg_age          NUMBER:= 0;
driver_age       NUMBER(2);
BEGIN
  SELECT (EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM BirthDate))
 INTO driver_age
 FROM Driver WHERE ID=team_id;

   total_age := total_age + 1;
   avg_age := avg_age + driver_age;
   avg_age := avg_age / total_age;
   RETURN avg_age;
END;
/
```

Now try calling this function for various teams:

```
SET SERVEROUTPUT ON;
DECLARE
  avg_age NUMBER;
BEGIN
  avg_age := Team_Driver_Age_Average('T1');
  DBMS_OUTPUT.PUT_LINE('Average age of the team is ' || avg_age);
END;
/
```

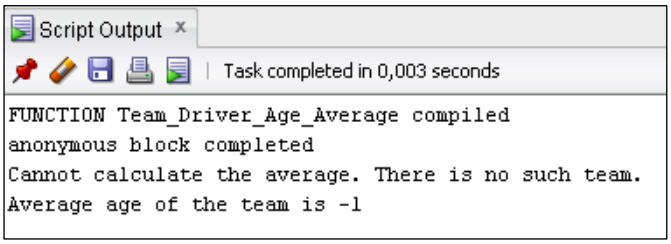



```
SET SERVEROUTPUT ON;
DECLARE
  avg_age NUMBER;
BEGIN
  avg_age := Team_Driver_Age_Average('T2');
  DBMS_OUTPUT.PUT_LINE('Average age of the team is ' || avg_age);
END;
/
```



As you have just observed, for the team T1 everything is OK. But you have received various errors for the others such as **many rows returned and no data found**. Therefore, you should write an exception for handling such errors and informing the caller about the situation. Now you modify your function as follows and call the function for the team '**T2'** again:

```
CREATE OR REPLACE FUNCTION Team_Driver_Age_Average
(team_id IN VARCHAR2)
RETURN NUMBER
IS
total_age        INTEGER:=0;
avg_age          NUMBER:= 0;
driver_age       NUMBER(2);
BEGIN
  SELECT (EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM BirthDate))
 INTO driver_age
 FROM Driver WHERE ID=team_id;

total_age := total_age + 1;
avg_age := avg_age + driver_age;

avg_age := avg_age / total_age;
  RETURN avg_age;
EXCEPTION
WHEN TOO_MANY_ROWS THEN
        dbms_output.put_line('Cannot calculate the average. There are many rows.');
        RETURN -1;
WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('Cannot calculate the average. There is no such team.');
        RETURN -1;
WHEN ZERO_DIVIDE THEN
        dbms_output.put_line('Cannot calculate the average. Divide by 0 occurred.');
        RETURN -1;
END;
/
```



Now call the function for '**T55**':
The function is still not working as intended. However, you have handled possible errors. You may also write your own exceptions and call them whenever necessary.

PART IV

You have handled the exceptions in the previous part. Now it is time to modify the function so that it performs intended functioning. The main source of errors is twofold:

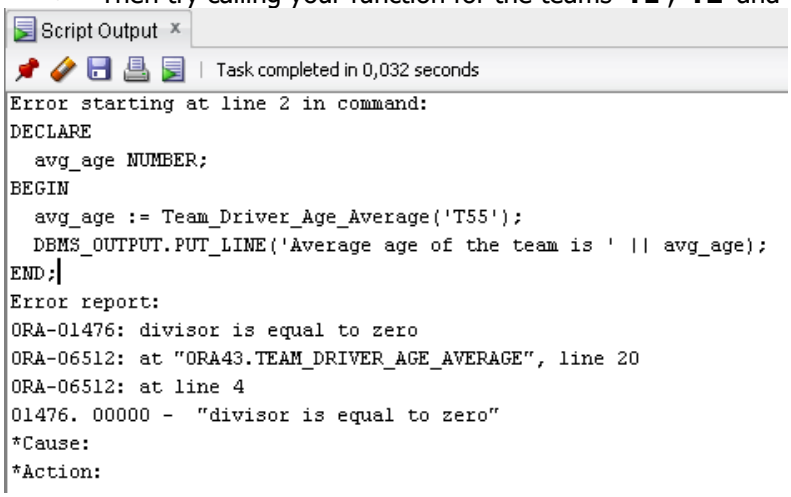
- There may be many drivers in a team. You should compute each driver's age and include it in the average age calculation.
- There may be no driver in the team (or there is no such a team). Then, you should return 0 as the average age or somehow inform the user by printing a message.

STEP I

- To handle the multiple rows that may return as a result of your query, you should use a cursor.
- Reform the SQL statement into a cursor declaration. Do not forget to remove "**INTO**" clause from the SQL statement.
- Write an OPEN, FETCH, CLOSE body with a loop and an exit condition:

```
CREATE OR REPLACE FUNCTION Team_Driver_Age_Average
(team_id IN VARCHAR2)
RETURN NUMBER
IS
total_age          INTEGER:=0;
avg_age            NUMBER:= 0;
driver_age         NUMBER(2);
cursor mycursor is
SELECT (EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM BirthDate))
 FROM Driver WHERE ID=team_id;
BEGIN
  OPEN mycursor;
  LOOP
FETCH mycursor INTO driver_age;
EXIT WHEN mycursor%NOTFOUND;
total_age := total_age + 1;
avg_age := avg_age + driver_age;
  END LOOP;
  CLOSE mycursor;
  avg_age := avg_age / total_age;
  RETURN avg_age;
END;
/
```

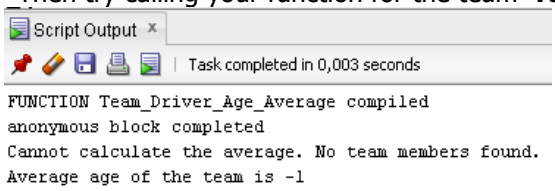
- Then try calling your function for the teams '**T1'**, '**T2**' and '**T55**.' You should get a division by zero error:



You should handle this error. You can write a predefined "**ZERO_DIVIDE**" exception. However, you will now write you own exception handler. To do this, declare and exception, namely "**No_Team_Members**" as follows and raise it when there is no team members.

```
CREATE OR REPLACE FUNCTION Team_Driver_Age_Average
(team_id IN VARCHAR2)
RETURN NUMBER
IS
    No_Team_Members EXCEPTION;
    total_age    INTEGER:=0;
    avg_age      NUMBER:= 0;
    driver_age   NUMBER(2);
    cursor mycursor is
    SELECT (EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM BirthDate))
     FROM Driver WHERE ID=team_id;
BEGIN
    OPEN mycursor;
    LOOP
          FETCH mycursor INTO driver_age;
          EXIT WHEN mycursor%NOTFOUND;
          total_age := total_age + 1;
          avg_age := avg_age + driver_age;
    END LOOP;
    CLOSE mycursor;

    IF  total_age = 0 THEN
         RAISE No_Team_Members;
    END IF;
         avg_age := avg_age / total_age;

    RETURN avg_age;

    EXCEPTION
    WHEN No_Team_Members THEN
         dbms_output.put_line('Cannot calculate the average. No team members found.');
    RETURN -1;
END;
/
```

Then try calling your function for the team '**T55**' and examine the results:



PART V

In this part:

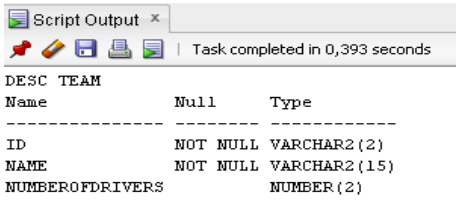
- You will first add a new column, namely "**NumberOfDrivers**" to the Team table and write a stored procedure which will calculate the number of drivers for each team and store this number in this new column.
- Then, you will add a new column, namely Age, to the Driver table.
- Finally, you will be guided to create a trigger for the Driver table such that whenever a new driver is inserted to the Driver table, it will calculate the age of the driver, store it in the Age column and then increase the NumberOfDrivers for the team that driver is involved in.

STEP I Let's first add a new column ("**NumberOfDrivers**") to the Team table.

```
ALTER TABLE TEAM
ADD NUMBEROFDRIVERS NUMBER(2);
```

Check if the column is created properly:

```
DESC TEAM;
```



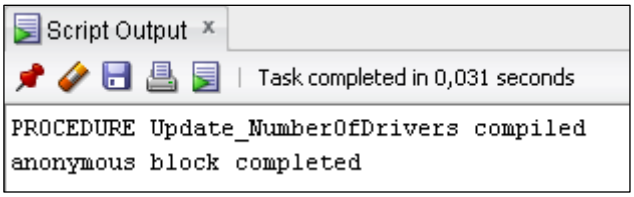
STEP II Write a stored procedure such that for each team in the Team table:

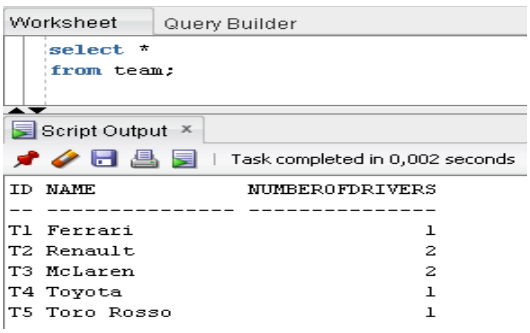
- Will get the number of drivers in the Driver table,
- Update the NumberOfDrivers column in the Team table with this number.
- Commit the changes the update statement makes.

```
CREATE OR REPLACE PROCEDURE Update_NumberOfDrivers
IS
team_id          VARCHAR2(2);
driver_count     NUMBER(3):=0;
cursor mycursor is
SELECT Id FROM Team;
BEGIN
  OPEN mycursor;
  LOOP
FETCH mycursor INTO team_id;
EXIT WHEN mycursor%NOTFOUND;

SELECT COUNT(Id) INTO driver_count FROM Driver WHERE ID=team_id;

UPDATE Team SET NumberOfDrivers = driver_count WHERE ID=team_id;
  END LOOP;
  CLOSE mycursor;
  COMMIT;
END;
/
```

- Execute it on the interface. Make corrections until it is compiled with no error.
- Call the procedure to update the results and issue a select statement to see what happened:

```
SET SERVEROUTPUT ON;
BEGIN
update_numberofdrivers();
END;
/
```





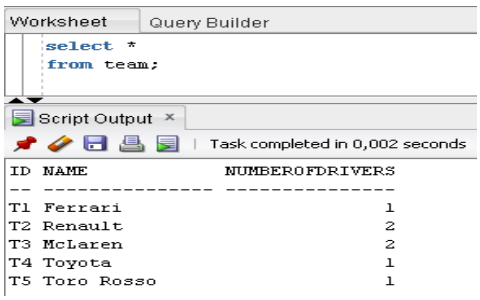


Table showing the difference between Procedure and Function:

Function	Procedure
To perform a calculation and **return a result**.	To **execute** a workflow or an operation.
Mandatory. It must give a value using RETURN.	**None** (Void). Does not return a value (It just performs the task).
Can be used inside queries like SELECT and WHERE.	**Cannot be used** inside SQL queries.
Written inside a variable assignment or a query.	Called standalone via the EXEC command or inside a block.