Department of Information Systems and Technologies

CTIS259 Database Management Systems and Applications

2025 – 2026 Fall

Lab Guide 16

Instructor	: Nimet Ceren SERİM	**Week:**	11
Assistant	: Engin Zafer KIRAÇBEDEL, Hatice Z. YILMAZ	**Date:**	24-25.11.2025

Aim of this lab session: **1.** PL/SQL Programming Language
2. Creating other Schema Objects (view, sequence)

ORACLE Server Configurations:

IP Address: 139.179.33.231

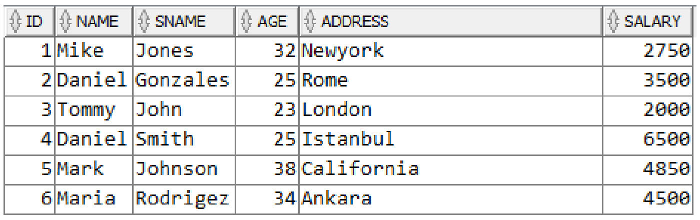
Port number: 1522

SID: orclctis

Please USE student (fYourStudentId) accounts

➢ Create the table **personnel** and insert data. Table structure and data is as follows: (You may download the script from moodle **`Labguides/Lab16/Personnel.zip`**)

```
Name    Null     Type
------- -------- ------------
ID      NOT NULL NUMBER(3)
NAME    NOT NULL VARCHAR2(20)
SNAME   NOT NULL VARCHAR2(20)
AGE     NOT NULL NUMBER(38)
ADDRESS          CHAR(25)
SALARY           NUMBER(10,2)
```



ID	NAME	SNAME	AGE	ADDRESS	SALARY
1	Mike	Jones	32	Newyork	2750
2	Daniel	Gonzales	25	Rome	3500
3	Tommy	John	23	London	2000
4	Daniel	Smith	25	Istanbul	6500
5	Mark	Johnson	38	California	4850
6	Maria	Rodrigez	34	Ankara	4500

Implicit Cursors

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The following table provides the description of the most used attributes;

Attribute & Description
%FOUND Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%NOTFOUND The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ISOPEN Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
%ROWCOUNT Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as **sql%attribute_name** .

PLSQL_Lab16_1.sql

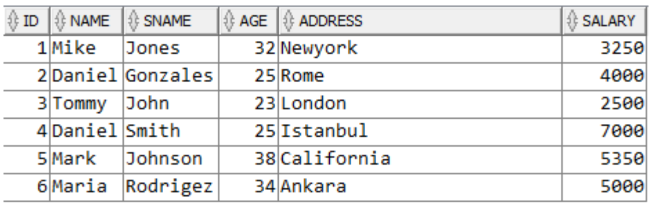
➢ Write a PL/SQL program which will update the table and increase the salary of each personnel by **500** and use the **SQL%ROWCOUNT** attribute to determine the number of rows affected.

```
SET SERVEROUTPUT ON;
DECLARE
   total_rows number(2);
BEGIN
   UPDATE personnel
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('There is no personnel in the list');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line('There are '||total_rows || ' personnel');
      commit;
   END IF;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result –

```
6 personnel selected
PL/SQL procedure successfully completed.
```

➢ Display the content of the personnel table.

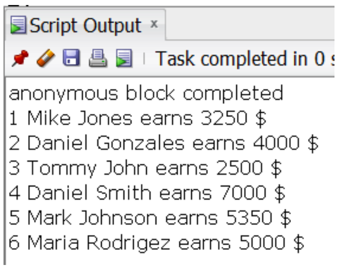


ID	NAME	SNAME	AGE	ADDRESS	SALARY
1	Mike	Jones	32	Newyork	3250
2	Daniel	Gonzales	25	Rome	4000
3	Tommy	John	23	London	2500
4	Daniel	Smith	25	Istanbul	7000
5	Mark	Johnson	38	California	5350
6	Maria	Rodrigez	34	Ankara	5000

Following is a complete example to illustrate the concepts of **<u>explicit cursor</u>**s.

PLSQL_Lab16_2.sql

➢ Display the id, name, surname and salary for all **personnel** as in the example run.



```
anonymous block completed
1 Mike Jones earns 3250 $
2 Daniel Gonzales earns 4000 $
3 Tommy John earns 2500 $
4 Daniel Smith earns 7000 $
5 Mark Johnson earns 5350 $
6 Maria Rodrigez earns 5000 $
```

```
SET SERVEROUTPUT ON;
DECLARE
   CURSOR personnel_cur is
      SELECT id, name, sname, salary
      FROM personnel;
   c_rec personnel_cur%rowtype;
BEGIN
   OPEN personnel_cur;
   LOOP
      FETCH personnel_cur into c_rec;
      EXIT WHEN personnel_cur%notfound;
      DBMS_OUTPUT.put_line(c_rec.id || ' ' || c_rec.name||' '||c_rec.sname||
                     ' earns '||c_rec.salary||' $');
   END LOOP;
   CLOSE personnel_cur;
END;
/
```

Please USE ORAxx accounts

➢ **`PLSQL_Lab16_3.sql`**

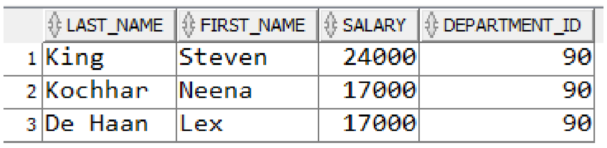
➢ Write the following procedures;

- **`disp_employee`**: takes the department_id as input parameter and displays the last_name, first_name and salary of all employees in the given department.

- **`emp_sal:`** takes the department_id and the raise amount as input parameters and updates the salary for the employees who work in that department. The procedure displays a warning message or the number of updated records.
(use `sql%notfound, sql%found, sql%rowcount`)

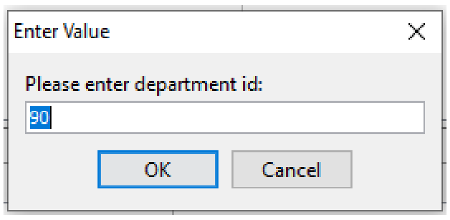
➢ Write a program block that displays prompts to the user and reads the department_id and the raise amount. The program will update the salaries of the employees who work in that department and displays their information using the procedures above.

The list of employees in the department **90**.



	LAST_NAME	FIRST_NAME	SALARY	DEPARTMENT_ID
1	King	Steven	24000	90
2	Kochhar	Neena	17000	90
3	De Haan	Lex	17000	90

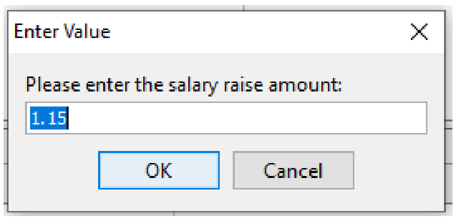
<u>**Example Run**</u>:



Enter Value

Please enter department id:

90



Enter Value

Please enter the salary raise amount:

1.15

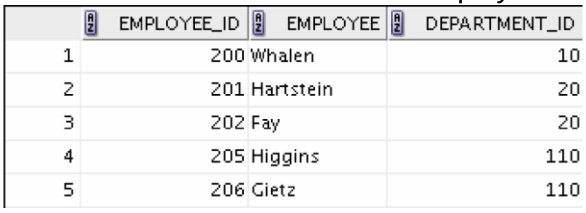
```
PL/SQL procedure successfully completed.

3 records have been updated!
King, Steven: 27600
Kochhar, Neena: 19550
De Haan, Lex: 19550
```

Practice 12-1: Creating Other Schema Objects

Part 1

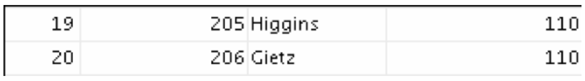
1. The staff in the HR department wants to hide some of the data in the EMPLOYEES table.
Create a view called EMPLOYEES_VU based on the employee numbers, employee last names, and department numbers from the EMPLOYEES table. The heading for the employee name should be EMPLOYEE.

2. Confirm that the view works. Display the contents of the EMPLOYEES_VU view.



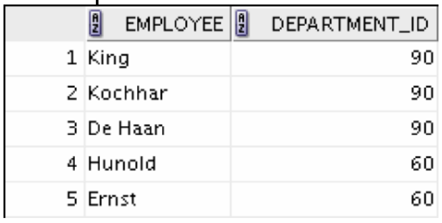
	EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
5	206	Gietz	110

…



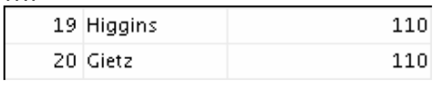
19	205	Higgins	110
20	206	Gietz	110

3. Using your EMPLOYEES_VU view, write a query for the HR department to display all employee names and department numbers.



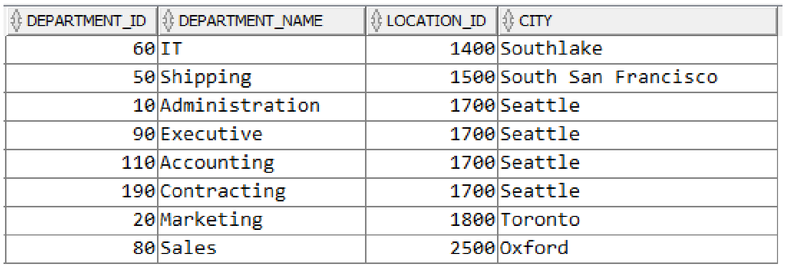
	EMPLOYEE	DEPARTMENT_ID
1	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60

….



19	Higgins	110
20	Gietz	110

4. Create a view called locations_view based on the department number, department name with their location information (location_id and city) from the DEPARTMENTS and LOCATIONS table. Display the contents of the view locations_view.



DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

Part 2

If you didn't create the table **"dept"** in the previous labs, create the table as follows;

```
create table dept
as
select department_id id, department_name name
from departments;

alter table dept add constraint dept_pk primary key(id);
```

7. You need a sequence that can be used with the PRIMARY KEY column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence DEPT_ID_SEQ.

8. To test your sequence, write a script to insert two rows in the DEPT table. Name your script lab_12_08.sql. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.