F24-95736
Advanced Relational Database Management
Professor: Randy Trzeciak

PL/SQL Lab 4

In the previous lab, we acquired the following capabilities:

1. **Creating** a simple relational schema, using the SQL DDL (Data Definition Language) provided by Oracle.
2. Defined **primary key** and **foreign key constraints** to the tables we created in step 1.
3. Learned how to create and use **database sequences** for primary keys.
4. Ran **sample SQL queries** off the small database we have created.
5. Created and executed an **anonymous PL/SQL** block of code that accessed the database.
6. Used **cursors** in PL/SQL blocks.
7. Used **exceptions** in PL/SQL blocks, and finally
8. **Dropped** the schema that we created in step 1. (*i.e.,* destroyed the tables, the constraints and all data in the tables).

In this lab we shall continue from where we left off, and learn to do the following:

1. Write and use **stand-alone procedures** for our database
2. Write and use **stand-alone functions** for our database
3. Use **IN, OUT** and **IN OUT parameters** for procedures
4. Use **IN parameters** for functions
5. Create a **package specification** and **package body** containing procedures and functions
6. **Call** on this package in another procedure

Before we get started, please do the following:

1. Log into SQL Developer.
2. Drop the schema, by typing `start <pathname>\lab4_drop.txt` hit enter at the SQL> prompt.
3. Recreate the schema by typing `start <pathname>\lab4_create.txt` hit enter at the SQL> prompt.
4. Insert data into the schema by typing `start <pathname>\lab4_insert.txt` hit enter at the SQL> prompt.

For this lab, we should create a **folder** . Call it by a convenient name like *plsqllab4*. **All the files we create today should be saved in this folder**.

Now we are ready to start writing procedures and functions!!

Creating file *stndalone_proc1.txt*:

Using a text editor please type in the following file. Remember to keep saving the file frequently in your folder. Call the file *stndalone_proc1*. The file extension should be *.txt*.

```
REM This file is part of PL/SQL lab 4 in 95-736
REM This file contains a stand-alone procedure called
REM display_repairs that lists all the contents of
REM the table REPAIRS
REM The procedure has no parameters

CREATE OR REPLACE PROCEDURE display_repairs AS
 repr_record repairs%ROWTYPE;
 CURSOR c1 is
    SELECT * FROM repairs;

 BEGIN
   OPEN c1;
   DBMS_OUTPUT.PUT('  '||'MECHANIC'||'   '||'CAR   '||'   '||'DATE  ');
   LOOP
     FETCH c1 INTO repr_record;
     EXIT WHEN c1%NOTFOUND;
     DBMS_OUTPUT.NEW_LINE;
     DBMS_OUTPUT.PUT('  '||repr_record.mech_num||'        '||
                     repr_record.car_num||'      '||
                     repr_record.repair_date);
     DBMS_OUTPUT.NEW_LINE;
   END LOOP;
   CLOSE c1;
 END;
/
```

In this file, we have used a **cursor** to fetch all the rows of the SQL query into **a local variable** (which is of the same type as a **row** in the table *repairs*). Then, we print out the contents of this local variable, and get the next row of the cursor into the local variable. The CREATE OR REPLACE statement (at the start of the procedure) simply means that if the procedure already exists, it is to be overwritten. This is useful if we are going to be making constant changes and recompiling the procedure. In this class, it is best to use this statement, as opposed to merely CREATE.

1. Start a spool file with the name of Lab4

At the SQL> prompt, type in `SET SERVEROUTPUT ON`
This activates the DBMS_OUTPUT package.
Now let us compile and run this procedure.
At the SQL> prompt, type in `start (or @)<pathname>\stndalone_proc1.txt` hit enter. This compiles the procedure. When we have successfully compiled the procedure, it is ready to run.

To run this procedure (which is called *display_repairs*), in SQL Developer type in: `execute display_repairs;` and hit enter. This will execute the procedure. Note the output. Also, we should note that the name of the procedure is **different** from the file in which it is contained.

Now let us write a procedure that has an IN parameter.

<u>Creating file *stndalone_proc2.txt*:</u>

Using a text editor, type in the following file, saving it as *stndalone_proc2.txt* in your folder.

```
REM This file is part of PL/SQL lab 4 in 95-736
REM This file contains a stand-alone procedure called car_info_by_make
REM that lists information about all cars in the database
REM of a certain make. The procedure uses an IN parameter: car_make.
REM NOTE how we parametrize the cursor to fetch rows of information
REM about the car make required.

CREATE    OR    REPLACE    PROCEDURE    car_info_by_make(car_make    IN
cars.car_model%TYPE)
 AS
 car_record cars%ROWTYPE;
 CURSOR c1 (car_make1 cars.car_model%TYPE) is
    SELECT * FROM cars
    WHERE cars.car_model LIKE '%'||car_make1||'%';

 BEGIN
   OPEN c1(car_make);
   DBMS_OUTPUT.PUT('  '||'NUMBER'||' '||'MODEL'||'               '||'YEAR
');
   DBMS_OUTPUT.NEW_LINE;
   LOOP
     FETCH c1 INTO car_record;
     EXIT WHEN c1%NOTFOUND;
     DBMS_OUTPUT.NEW_LINE;
     DBMS_OUTPUT.PUT('  '||car_record.car_num||'      '||
                     car_record.car_model||'          '||
                     car_record.car_year);
     DBMS_OUTPUT.NEW_LINE;
   END LOOP;
   CLOSE c1;
 END;
/
```

Here, we pass a **parameter to the procedure**, and this parameter is then passed to the cursor. The cursor fetches all the rows in the database pertaining to cars whose model information pattern-matches the actual parameter we have passed.

Now, execute `<pathname>\stndalone_proc2.txt`.

Once the procedure is compiled, execute it as follows. At the SQL> prompt, type in:
```
execute car_info_by_make('Ferrari');
```
and hit enter. This will give you information on all Ferraris in the database.

Now let us create a procedure with IN OUT parameters, and another procedure that calls it.

Creating file *stndalone_proc34.txt*:
Using a text editor, type in the following file, saving it as *stndalone_proc34.txt* in your folder.
```
REM This file is part of PL/SQL lab 4 in 95-736
REM This file contains a stand-alone procedure called calc_sal_stats
REM It calculates the mean, maximum and minimum salaries of
REM all employees in the database, and returns them to the
REM calling procedure. The procedure uses 3 IN OUT parameters:
REM mean1, max1 and min1.

CREATE OR REPLACE
PROCEDURE calc_sal_stats(   mean1 IN OUT mechanics.mech_salary%TYPE,
                            max1 IN OUT mechanics.mech_salary%TYPE,
                            min1 IN OUT mechanics.mech_salary%TYPE)
 AS
  temp_sal mechanics.mech_salary%TYPE;
  total_sal mechanics.mech_salary%TYPE := 0.00;
  CURSOR c1 is
    SELECT mech_salary FROM mechanics;

 BEGIN
```

```
   OPEN c1;
   LOOP
     FETCH c1 INTO temp_sal;
     EXIT WHEN c1%NOTFOUND;
     total_sal:=total_sal+temp_sal;
   END LOOP;
   mean1:=total_sal/(c1%ROWCOUNT);
   CLOSE c1;
   SELECT MAX(mech_salary) INTO max1
   FROM mechanics;
   SELECT MIN(mech_salary) INTO min1
   FROM mechanics;

 END;
/

REM This is procedure print_sal_stats.
REM It calls the  stand-alone procedure calc_sal_stats
REM It prints the mean, maximum and minimum salaries of
REM all mechanics in the database.
REM The procedure uses no parameters.

CREATE OR REPLACE PROCEDURE print_sal_stats
AS
  mean_sal mechanics.mech_salary%TYPE:=-1;
  max_sal mechanics.mech_salary%TYPE:=-1;
  min_sal mechanics.mech_salary%TYPE:=-1;

 BEGIN
  calc_sal_stats(mean_sal, max_sal, min_sal);
/*Procedure call above. Note that procedure calc_sal_stats
must have been compiled before we can compile this procedure*/
   DBMS_OUTPUT.PUT('MEAN SALARY:'||'  '||mean_sal);
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.PUT('MAX SALARY:'||'  '||max_sal);
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.PUT('MIN SALARY:'||'  '||min_sal);
   DBMS_OUTPUT.NEW_LINE;
 EXCEPTION
   WHEN ZERO_DIVIDE THEN
       DBMS_OUTPUT.PUT_LINE('No rows in mechanics table');
       ROLLBACK;
   WHEN OTHERS THEN
       ROLLBACK;
 END;
/
```

The first procedure (*calc_sal_stats*) calculates the mean, maximum and minimum salaries of mechanics, and returns the values in the initialized memory locations that are passed into the procedure. The second procedure basically calls the first. Now let us compile these procedures.

In SQL Developer execute `<pathname>\stndalone_proc34.txt` .
Next, `execute print_sal_stats`.
This executes the procedure. Note the output.
Now let us write some functions! The first function has no parameters.

Creating file *stndalone_fn1.txt*:
Using a text editor, type in the following file, saving it as *stndalone_fn1.txt* in your folder.

```
REM This file is part of PL/SQL lab 4 in 95-736
REM This file contains a stand-alone function called calc_num_jobs_
REM all
REM It calculates the number of repair jobs done
REM in the last 3 years.
REM  The function has no parameters.
REM The file also contains a procedure that calls the function and
REM prints out the value returned

CREATE OR REPLACE
FUNCTION calc_num_jobs_all RETURN NUMBER
```

```
 AS
num_jobs NUMBER:=0;
 BEGIN
   SELECT count(*) INTO num_jobs
          FROM  repairs
         WHERE MONTHS_BETWEEN(SYSDATE, repairs.repair_date)
            <=48;
   RETURN num_jobs;
 END;
/

REM Make sure you type SET SERVEROUTPUT ON before running the procedure
REM below

CREATE OR REPLACE
PROCEDURE print_jobs
AS
BEGIN
  DBMS_OUTPUT.PUT('Number of repair jobs: '||calc_num_jobs_all);
  DBMS_OUTPUT.NEW_LINE;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT('No Jobs done!');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT('Some other exception occurred');

END;
/
```

Now compile this file, and execute procedure *print_jobs*. Note the result. Include the output in your lab 4 submission. The number of repair jobs in the last 3 years are printed. Next, we shall create a function with parameters.

Creating file *stndalone_fn2.txt*:
Using a text editor, type in the following file, saving it as *stndalone_fn2.txt* in your folder.

```
REM This file is part of PL/SQL lab 4 in 95-736
REM This file contains a stand-alone function called calc_num_jobs_
REM for_model
REM It calculates the number of repair jobs done
REM in the last 3 years for a particular type of car
REM  The function has one parameter.
REM The file also contains a procedure that calls the function and
REM prints out the value returned

CREATE OR REPLACE
FUNCTION calc_num_jobs_for_model(model IN cars.car_model%TYPE) RETURN
NUMBER
 AS
num_jobs NUMBER:=0;
 BEGIN
   SELECT count(*) INTO num_jobs
          FROM  repairs,cars
         WHERE MONTHS_BETWEEN(SYSDATE, repairs.repair_date)
            <=48
         AND repairs.car_num=cars.car_num
         AND cars.car_model LIKE '%'||model||'%';
   RETURN num_jobs;
 END;
/

REM Make sure you type SET SERVEROUTPUT ON before running the procedure
REM below

CREATE OR REPLACE
PROCEDURE print_jobs_model(model_par IN cars.car_model%TYPE)
AS
BEGIN
  DBMS_OUTPUT.PUT('Number of repair jobs: '||
                   calc_num_jobs_for_model(model_par));
  DBMS_OUTPUT.NEW_LINE;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
```

```
      DBMS_OUTPUT.PUT('No Jobs done!!');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT('Some other exception occured');

END;
/
```

Now compile the file, and execute procedure *print_jobs_model*, `execute print_jobs_model('Ferrari')` hit enter at the SQL> prompt. Note the results.

Now let us create a package.

<u>Creating file *pack1.txt:*</u>

Using the notepad editor, type in the following file, saving it as *pack1.txt* in your folder.

```
REM This file is part of PL/SQL lab 4 in 95-736
REM This file contains a package called mech_manager
REM It has a function for adding a new mechanic, and a procedure
REM for raising the salary of an existing mechanic
REM Make sure you SET SERVEROUTPUT ON

CREATE OR REPLACE PACKAGE mech_manager
AS
FUNCTION add_mech(name mechanics.mech_name%TYPE, salary mechanics.
                  mech_salary%TYPE, specialty
mechanics.mech_specialty%TYPE)
RETURN mechanics.mech_num%TYPE;
PROCEDURE sal_chnge(mech_number IN mechanics.mech_num%TYPE, salary
                    IN mechanics.mech_salary%TYPE);
END mech_manager;
/

REM The body follows

CREATE OR REPLACE PACKAGE BODY mech_manager
AS
```

```
/*Function Follows*/
FUNCTION add_mech(name mechanics.mech_name%TYPE, salary mechanics.
                  mech_salary%TYPE, specialty
mechanics.mech_specialty%TYPE)
RETURN mechanics.mech_num%TYPE IS

new_mech_no mechanics.mech_num%TYPE;

BEGIN
  SELECT mech_sequence.NEXTVAL INTO new_mech_no FROm dual;
  INSERT INTO mechanics VALUES(new_mech_no, name, salary, specialty);
  RETURN (new_mech_no);
END add_mech;

/*PROCEDURE FOLLOWS*/
PROCEDURE sal_chnge(mech_number IN mechanics.mech_num%TYPE,
                    salary IN mechanics.mech_salary%TYPE)
IS
no_number EXCEPTION;
BEGIN
 UPDATE mechanics
   SET mech_salary=salary
   WHERE mech_num=mech_number;
 IF SQL%NOTFOUND THEN
     RAISE no_number;
 END IF;
EXCEPTION
    WHEN no_number THEN
      DBMS_OUTPUT.PUT_LINE('NO such mechanic');
      ROLLBACK;
END sal_chnge;

END mech_manager;
/
```

Now compile this package.

<u>Creating file *call_package.txt*:</u>

Next type in the following file, saving it as *call_package.txt* in your folder.

```
REM This file is part of PL/SQL lab 4. It calls a package
REM called mech_manager that we have created.
REM Note that the package must be successfully compiled
REM before we can run this procedure


CREATE OR REPLACE PROCEDURE create_mech(name mechanics.mech_name%TYPE,
salary mechanics.
                  mech_salary%TYPE,
                  specialty mechanics.mech_specialty%TYPE)
AS
num_returned mechanics.mech_num%TYPE;
BEGIN
  num_returned:=mech_manager.add_mech(name,salary,specialty);
  DBMS_OUTPUT.PUT('The new number is:  '||num_returned);
  DBMS_OUTPUT.NEW_LINE;
END;
/
```

Now compile this procedure. Run it, as follows:

```
execute create_mech('Devon Bush', 60000.00, 'Ferrari Specialist');
```

To see the newly inserted row, type in

```
select * from mechanics;
```

at the SQL> prompt, and hit enter.

Close your spool file using the FILE menu item or by entering the following at the SQL prompt:

Congratulations, we have finished all the exercises in PL/SQL lab 4! For a list of capabilities you have acquired, please refer back to page 1. Please submit spool file Lab4.lst as part of submission.

These 2 labs have shown us one way that we can build complete database applications, that run off the SQL prompt, without using a graphical user interface front end. The way is: **first**, create the schema. **Next**, write a bunch of key procedures as standalone procedures (*e.g.,* add_an_employee(…), salary_raise(….), delete_an_employee(…), add_new_customer(…)). The key standalone procedures represent business functions and should be grouped into packages, maybe one package each for a small set of similar employees. This will create an **application that users can run off the SQL> prompt**. The application will consist of the list of standalone procedures.