USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science

CS311 Introduction to Database Systems

Lab#13

Objective:

- **Stored Procedures**

Name of Student: **Muhammad Waleed**

Roll No: **20B-115-SE** **Sec.** **B**

Date of Experiment:

.....

Marks Obtained/Remarks: _____

Signature: _____

STORED PROCEDURES

Stored procedure is a named PL/SQL block that can take parameters and be invoked.

Generally speaking, a procedure is used to perform an action. A procedure has a header, a declarative part, an executable part, and an optional exception-handling part.

Procedures promote reusability and maintainability. Once validated, they can be used in any number of applications. If the definition changes, only the procedure is affected, this greatly simplifies maintenance.

Syntax for Creating Procedures

We create new procedures with the CREATE PROCEDURE statement, which may declare a list of arguments (sometimes referred to as parameters), and must define the actions to be performed by the standard PL/SQL block.

```
CREATE [OR REPLACE] PROCEDURE procedure_name
        (argument1  [mode1]  datatype1,
         argument2  [mode2]  datatype2,
         …..
IS [AS]
PL/SQL Block;
```

Syntax Definitions

Parameter	Description
Procedure_name	Name of the procedure
Argument	Name of a PL/SQL variable whose value is passed to, populated by the calling environment, or both, depending, on the *mode* being used
Mode	Type of argument IN (default) OUT IN OUT
Datatype	Datatype of the argument
PL/SQL block	Procedural body that defines the action performed by the procedure

- PL/SQL block starts with either BEGIN or the declaration of local variables and end with either END or END procedure name. We cannot reference host or bind variables in the PL/SQL block of a stored procedure.
- The REPLACE option indicates that if the procedure exists, it will be dropped and replaced with the new version created by the statement.

Creating a Stored Procedure using SQL*Plus

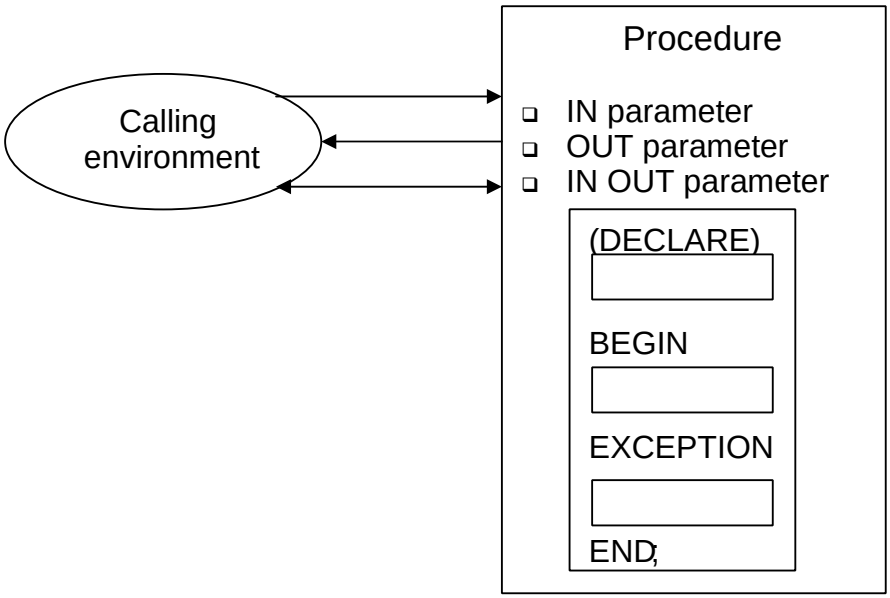
- Enter the text of the CREATE PROCEDURE statement in a system editor or word processor and save it as a script file (*.sql extension*)
- From SQL*Plus, run the script file to compile the source code into p-code and store both in the database.
- Invoke the procedure from an oracle server environment to determine whether it executes without error.

Procedural Parameter Modes

We can transfer values to and from the calling environment through parameters. Choose one of the following three modes for each parameter: IN, OUT, or IN OUT. Attempts to change the value of an IN parameter will result in an error.

DATATYPE can only be the %TYPE definition, %ROWTYPE definition, or an explicit datatype with no size specification.

Type of Parameter	Description
IN (default)	Passes a constant value from the calling environment into the procedure
OUT	Passes a value from the procedure to the calling environment
IN OUT	Passes a value from the calling environment into the procedure and a possibly different value from the procedure back to the calling environment using the same parameter.



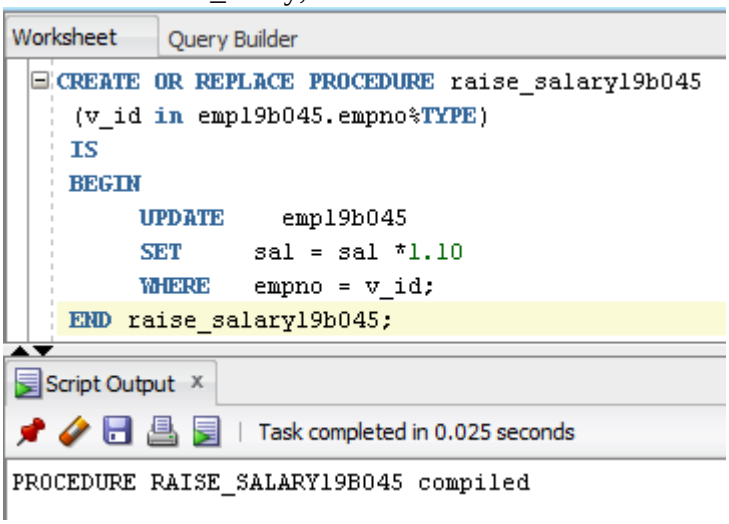
Parameter Modes for Formal Parameters

IN	OUT	IN OUT
Default	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant or initialized variable	Must be a variable	Must be a variable

IN Parameters

The example below shows a procedure with one IN parameter. Running this statement in SQL*Plus creates the RAISE_SALARY procedure. When invoked, RAISE_SALARY takes the parameter for the employee number and updates the employee's record with a salary increase of 10 percent.
To invoked a procedure in SQL*Plus, use the EXECUTE command.

```
SQL> CREATE OR REPLACE PROCEDURE raise_salary
        (v_id in emp.empno%TYPE)
        IS
        BEGIN
            UPDATE     emp
            SET        sal = sal *1.10
            WHERE      empno = v_id;
        END raise_salary;
```



Procedure created

SQL> EXECUTE raise_salary (7369)
PL/SQL procedure successfully completed

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17-DEC-80	800	(null)	20

```
EXECUTE raise_salary19b045 (7369);
```

Query Result | Script Output

Task completed in 0.016 seconds

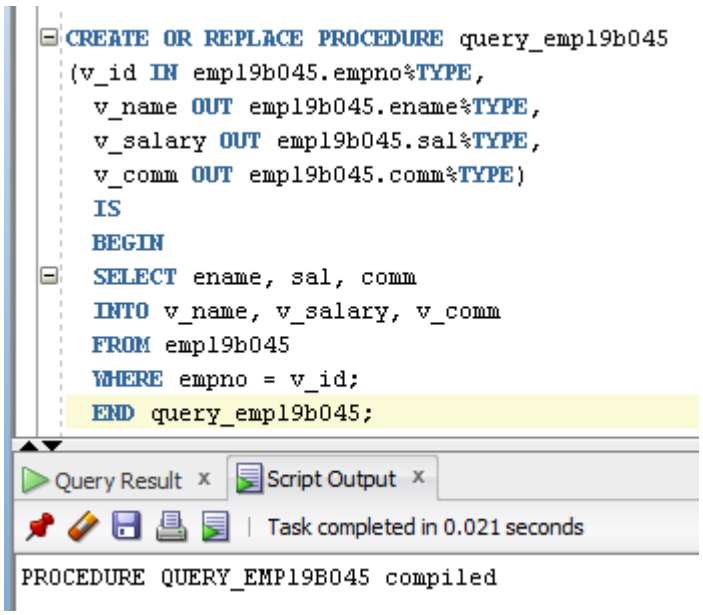
```
anonymous block completed
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17-DEC-80	880	(null)	20

OUT Parameters

The example below shows a procedure, QUERY_EMP, with one IN and three OUT parameters that will return a value to the calling environment. Running this statement in SQL*Plus creates the RAISE_SALARY procedure.

```
SQL>CREATE OR REPLACE PROCEDURE query_emp
        (v_id IN emp.empno%TYPE,
        v_name OUT emp.ename%TYPE,
        v_salary OUT emp.sal%TYPE,
        v_comm OUT emp.comm%TYPE)
        IS
        BEGIN
            SELECT ename, sal, comm.
            INTO v_name, v_salary, v_comm.
            FROM emp
            WHERE empno = v_id;
        END query_emp;
/
```



Procedure created

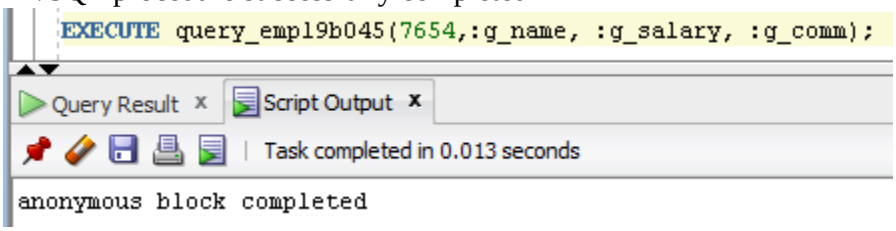
View the value of OUT parameters with SQL*Plus

- Create host variables in SQL*Plus using the VARIABLE syntax.
- Invoke the QUERY_EMP procedure, supplying these host variables as the OUT parameters. Note the use of the colon (:) to reference the host variables in the EXECUTE syntax.
- To view the values passed from the procedure to the calling environment, use the PRINT syntax. Only one variable can be supplied to each PRINT command

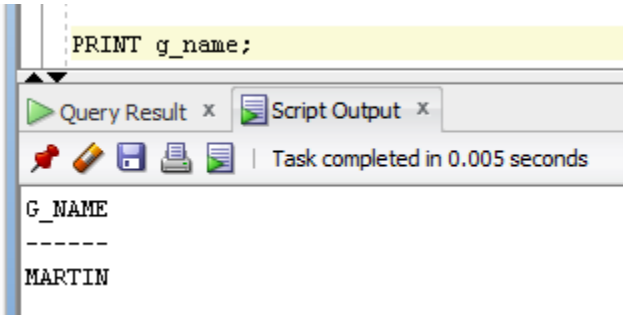
.

```
SQL> VARIABLE g_name varchar2(15)
SQL> VARIABLE g_salary number
SQL> VARIABLE g_comm number
```

```
SQL> EXECUTE query_emp (7654,: g_name,
:g_salary, :g_comm.)
PL/SQL procedure successfully completed
```




```
SQL> PRINT g_name
G_NAME
-------------------------
MARTIN
```



Viewing IN OUT Parameters with SQL*Plus

1. Create a host variable using the VARIABLE syntax.
2. Populate the host variable with a value, using an anonymous PL/SQL block.
3. Invoke the FORMAT_PHONE procedure supplying the host variable as the IN OUT parameter. Note the use of the colon (:) to reference the host variable in the EXECUTE syntax.
4. To view the value passed back to the calling environment, use the PRINT syntax.

EXERCISES

1. Describe the different types of parameters used in procedures.

1. Procedure with no parameters:

It is a procedure that doesn't take any input.

2. Procedure with IN parameter:

An IN parameter is used to take a parameter as input such as an attribute. When we define an IN parameter in a procedure, the calling program has to pass an argument to the stored procedure

3. Procedure with OUT parameter:

An OUT parameter is used to pass a parameter as output or display like the select operator. The value of an OUT parameter can be changed inside the procedure and its new value is passed back to the calling program

4. Procedure with IN-OUT parameter:

An INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter and pass the new value back to the calling program

2. Write down the different modes for IN and IN OUT parameters.

IN mode

It is the default mode. When we define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. The value of an IN parameter is protected which means that even the value of the IN parameter is changed inside the stored procedure; its original value is retained after the stored procedure ends.

INOUT mode

An INOUT parameter is the combination of IN and OUT parameters which means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter and pass the new value back to the calling program.

3. Create a stored procedure to get the project ID and print the project title, client name, duration (in days) and status.

```
--task 1--
CREATE OR REPLACE PROCEDURE info_project045
(v_id IN project045.pid%TYPE,
  v_title OUT project045.title%TYPE,
  v_client OUT project045.client%TYPE,
  v_days OUT project045.duration_weeks%TYPE,
  v_status OUT project045.status%TYPE)
  IS
  BEGIN
  SELECT title, client, duration_weeks*7 , status
  INTO v_title, v_client, v_days,v_status
  FROM project045
  WHERE pid  = v_id;
  END info_project045;
```

Script Output

Task completed in 0.03 seconds

```
PROCEDURE INFO_PROJECT045 compiled
```

```
EXECUTE info_project045(2125,:g_title, :g_client, :g_days, :g_status);
```

Script Output

Task completed in 0.016 seconds

```
anonymous block completed
```

```
PRINT g_title
PRINT g_client
PRINT g_days
PRINT g_status
```

Script Output

Task completed i

```
G_TITLE
----------
SOCIAL APP

G_CLIENT
----
Zain

G_DAYS
--
56

G_STATUS
-----------
IN PROGRESS
```

4. Create a stored procedure that takes the employee number and designation. Change the designation of the employee to the new value passed and then print the new grade as follows:-
   New Grade: xxx

```
CREATE OR REPLACE PROCEDURE info_employee045
 (v_empno IN employee045.empno%TYPE,
 v_designation IN employee045.designation%TYPE,
 v_grade OUT grade045.grade%TYPE
 )
 IS
 BEGIN
 UPDATE employee045
 SET designation = v_designation
 WHERE empno = v_empno;
 SELECT g.grade into v_grade
 from employee045 e inner join grade045 g on(e.designation = g.designation)
 WHERE e.empno = v_empno;
 END info_employee045 ;
```

Script Output | Task completed in 0.03 seconds

```
PROCEDURE INFO_EMPLOYEE045 compiled
```

BEFORE

```
select e.designation, g.grade from employee045 e inner join grade045 g on (e.designation = g.designation) where e.empno = 1;
```

Query Result | SQL | All Rows Fetched: 1 in 0.004 seconds

	DESIGNATION	GRADE
1	MANAGER	15

```
EXECUTE info_employee045(1,'DESIGNER',:g_grade);
```

Query Result | Script Output | Task completed in 0.012 seconds

```
anonymous block completed
```

```
select e.designation, g.grade
from employee045 e inner join grade045 g on (e.designation = g.designation)
where e.empno = 1;
```

Script Output | Query... | SQL | All Rows Fetched: 1 in 0.001 seconds

	DESIGNATION	GRADE
1	DESIGNER	13

```
PRINT g_grade
```

Script Output | Query Result

Task completed in 0.011 seconds

```
anonymous block completed
G_GRADE
--
13
```

5. Create a stored procedure that has two arguments TID and duration. The first argument takes training code for a training program from the calling environment and the second argument should return the duration of training in weeks to the calling environment. Then print the duration in the calling environment.

```
CREATE OR REPLACE PROCEDURE stored_duration045
(v_code IN TRAINING045.TCODE%TYPE,
  v_duration OUT TRAINING045.title%TYPE)
  IS
  BEGIN
  SELECT round(((to_date(enddate,'DD-MM-YYYY')-to_date(startdate,'DD-MM-YYYY'))/7),1)
  INTO v_duration
  FROM TRAINING045
  WHERE tcode = v_code;
END stored_duration045;
```

Query Result | Script Output

Task completed in 0.479 seconds

```
PROCEDURE STORED_DURATION045 compiled
```

```
EXECUTE stored_duration045 ('T101', :g_duration);
```

Query Result | Script Output

Task completed in 0.049 seconds

```
anonymous block completed
```

Query Result | Script Output

Task completed in 0.016 sec

```
G_DURATION
---
4.4
```

* * * * *