****

USMAN INSTITUTE OF TECHNOLOGY

Department of Computer Science

**CS311 Introduction to Database Systems**

**Fall 2019**

**Lab # 14**

**Objective:** Working onStored Procedure

.

SHOAIB AHMED

**Name of Student:**

17B-033-SE

A

**Roll No: Section:**

24-JAN-2020

**Date of Experiment:**

31-JAN-2020

**Date Submitted:**

Marks Obtained/Remarks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Stored procedures (subprograms) can be compiled and stored in an Oracle Database, ready to be executed. Once compiled, it is a schema object known as a stored procedure, which can be referenced or called any number of times by multiple applications connected to Oracle Database. Stored procedures can accept parameters when they are executed (called). To execute a stored procedure or function, you only need to include its object name.

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.

**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

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure\_body > --PL/SQL here

END procedure\_name;

**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 |

* The REPLACE option indicates that if the procedure exists, it will be dropped and replaced with the new version created by the statement.
* 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.

**Creating a Stored Procedure using SQL\*Plus**

* Create a procedure sing 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. |

Calling

environment

Procedure



IN parameter



OUT parameter



IN OUT parameter

DECLARE)

(

BEGIN

EXCEPTION

END

;

**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

**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; |

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

1. 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

1. 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.