Exercises for Chapter 19: Procedures

Try It Yourself

The projects in this section are meant to have you utilize all of the skills that you have acquired throughout this chapter. Here are some exercises that will help you test the depth of your understanding.

Part 1

1. Write a procedure with no parameters. The procedure will let you know if the current day is a weekend or a weekday. Additionally, it will let you know the user name and current time. It will also let you know how many valid and invalid procedures are in the database.

Answer: The procedure should look similar to the following:

```
CREATE OR REPLACE PROCEDURE current_status
AS
   v_day_type CHAR(1);
   v_user     VARCHAR2(30);
   v_valid    NUMBER;
   v_invalid  NUMBER;
BEGIN
   SELECT SUBSTR(TO_CHAR(sysdate, 'DAY'), 0, 1)
     INTO v_day_type
     FROM dual;

   IF v_day_type = 'S' THEN
      DBMS_OUTPUT.PUT_LINE ('Today is a weekend.');
   ELSE
      DBMS_OUTPUT.PUT_LINE ('Today is a weekday.');
   END IF;
   --
   DBMS_OUTPUT.PUT_LINE('The time is: '||
     TO_CHAR(sysdate, 'HH:MI AM'));
   --
   SELECT user
     INTO v_user
     FROM dual;
   DBMS_OUTPUT.PUT_LINE ('The current user is '||v_user);
   --
   SELECT NVL(COUNT(*), 0)
```

```
   INTO v_valid
   FROM user_objects
 WHERE status = 'VALID'
   AND object_type = 'PROCEDURE';
 DBMS_OUTPUT.PUT_LINE
   ('There are '||v_valid||' valid procedures.');
 --
 SELECT NVL(COUNT(*), 0)
   INTO v_invalid
   FROM user_objects
 WHERE status = 'INVALID'
   AND object_type = 'PROCEDURE';
 DBMS_OUTPUT.PUT_LINE
   ('There are '||v_invalid||' invalid procedures.');
END;

SET SERVEROUTPUT ON
EXEC current_status;
```

2. Write a procedure that takes in a zip code, city, and state and inserts the values into the ZIPCODE table. There should be a check to see if the zip code is already in the database. If it is, an exception will be raised and an error message will be displayed. Write an anonymous block that uses the procedure and inserts your zip code.

Answer: The script should look similar to the following:

```
CREATE OR REPLACE PROCEDURE insert_zip
  (I_ZIPCODE IN zipcode.zip%TYPE,
   I_CITY    IN zipcode.city%TYPE,
   I_STATE   IN zipcode.state%TYPE)
AS
   v_zipcode zipcode.zip%TYPE;
   v_city    zipcode.city%TYPE;
   v_state   zipcode.state%TYPE;
   v_dummy   zipcode.zip%TYPE;
BEGIN
   v_zipcode := i_zipcode;
   v_city    := i_city;
   v_state   := i_state;
--
   SELECT zip
     INTO v_dummy
     FROM zipcode
    WHERE zip = v_zipcode;
--
   DBMS_OUTPUT.PUT_LINE('The zipcode '||v_zipcode||
      ' is already in the database and cannot be'||
      ' reinserted.');
--
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      INSERT INTO ZIPCODE
      VALUES (v_zipcode, v_city, v_state, user, sysdate,
              user, sysdate);
```

```
     WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE ('There was an unknown error '||
             'in insert_zip.');
END;

SET SERVEROUTPUT ON
BEGIN
   insert_zip (10035, 'No Where', 'ZZ');
END;

BEGIN
   insert_zip (99999, 'No Where', 'ZZ');
END;

ROLLBACK;
```

Part 2

3. Create a stored procedure based on the script ch17_1c.sql (version 3), created in first lab of chapter 17. The procedure should accept two parameters to hold a table name and an ID, and return six parameters with first name, last name, street, city, state, and zip information.

Answer: The procedure should look similar to the procedure shown below. All changes are highlighted in bold.

```
CREATE OR REPLACE PROCEDURE get_name_address
   (table_name_in   IN VARCHAR2
   ,id_in           IN NUMBER
   ,first_name_out OUT VARCHAR2
   ,last_name_out  OUT VARCHAR2
   ,street_out     OUT VARCHAR2
   ,city_out       OUT VARCHAR2
   ,state_out      OUT VARCHAR2
   ,zip_out        OUT VARCHAR2)
AS
   sql_stmt VARCHAR2(200);
BEGIN
   sql_stmt := 'SELECT a.first_name, a.last_name, a.street_address'||
               '     ,b.city, b.state, b.zip'                       ||
               ' FROM '||table_name_in||' a, zipcode b'             ||
               ' WHERE a.zip = b.zip'                               ||
               '   AND '||table_name_in||'_id = :1';
   EXECUTE IMMEDIATE sql_stmt
   INTO first_name_out, last_name_out, street_out, city_out, state_out,
        zip_out
   USING id_in;
END get_name_address;
```

The procedure above contains two IN parameters whose values are used by the dynamic SQL statement, and six OUT parameters that hold date returned by the SELECT statement. Once created, the procedure can be tested with the following PL/SQL block:

```
SET SERVEROUTPUT ON
DECLARE
   v_table_name VARCHAR2(20) := '&sv_table_name';
   v_id         NUMBER := &sv_id;
   v_first_name VARCHAR2(25);
   v_last_name  VARCHAR2(25);
   v_street     VARCHAR2(50);
   v_city       VARCHAR2(25);
   v_state      VARCHAR2(2);
   v_zip        VARCHAR2(5);
BEGIN
   get_name_address (v_table_name, v_id, v_first_name, v_last_name,
                     v_street, v_city, v_state, v_zip);

   DBMS_OUTPUT.PUT_LINE ('First Name: '||v_first_name);
   DBMS_OUTPUT.PUT_LINE ('Last Name:  '||v_last_name);
   DBMS_OUTPUT.PUT_LINE ('Street:     '||v_street);
   DBMS_OUTPUT.PUT_LINE ('City:       '||v_city);
   DBMS_OUTPUT.PUT_LINE ('State:      '||v_state);
   DBMS_OUTPUT.PUT_LINE ('Zip Code:   '||v_zip);
END;
```

When run, this script produces the following output (the first run is
against the STUDENT table, and the second run is against the
INSTRUCTOR table):

```
Enter value for sv_table_name: student
old   2:    v_table_name VARCHAR2(20) := '&sv_table_name';
new   2:    v_table_name VARCHAR2(20) := 'student';
Enter value for sv_id: 105
old   3:    v_id NUMBER := &sv_id;
new   3:    v_id NUMBER := 105;
First Name: Angel
Last Name:  Moskowitz
Street:     320 John St.
City:       Ft. Lee
State:      NJ
Zip Code:   07024


PL/SQL procedure successfully completed.


Enter value for sv_table_name: instructor
old   2:    v_table_name VARCHAR2(20) := '&sv_table_name';
new   2:    v_table_name VARCHAR2(20) := 'instructor';
Enter value for sv_id: 105
old   3:    v_id NUMBER := &sv_id;
new   3:    v_id NUMBER := 105;
First Name: Anita
Last Name:  Morris
Street:     34 Maiden Lane
City:       New York
State:      NY
Zip Code:   10015
```

```
PL/SQL procedure successfully completed.
```

4. Modify procedure created in the previous exercise. Instead of using six parameters to hold name and address information, the procedure should return a user-defined record that contains six fields that hold name and address information. Note: you may want to create a package where you define record type. This record may be used later, for example, when the procedure is invoked in a PL/SQL block.

Answer: The package should look similar to the package shown below. All changes to the procedure are highlighted in bold.

```
CREATE OR REPLACE PACKAGE dynamic_sql_pkg AS

   -- Create user-defined record type
   TYPE name_addr_rec_type IS RECORD
      (first_name VARCHAR2(25),
       last_name  VARCHAR2(25),
       street     VARCHAR2(50),
       city       VARCHAR2(25),
       state      VARCHAR2(2),
       zip        VARCHAR2(5));

   PROCEDURE get_name_address (table_name_in  IN VARCHAR2
                              ,id_in          IN NUMBER
                              ,name_addr_rec OUT name_addr_rec_type);
END dynamic_sql_pkg;
/


CREATE OR REPLACE PACKAGE BODY dynamic_sql_pkg AS

PROCEDURE get_name_address (table_name_in  IN VARCHAR2
                           ,id_in          IN NUMBER
                           ,name_addr_rec OUT name_addr_rec_type)
IS
   sql_stmt VARCHAR2(200);
BEGIN
   sql_stmt := 'SELECT a.first_name, a.last_name, a.street_address'||
               '      ,b.city, b.state, b.zip'                     ||
               ' FROM '||table_name_in||' a, zipcode b'            ||
               ' WHERE a.zip = b.zip'                              ||
               '   AND '||table_name_in||'_id = :1';
   EXECUTE IMMEDIATE sql_stmt
   INTO name_addr_rec
   USING id_in;
END get_name_address;

END dynamic_sql_pkg;
```

In the package specification created above, you declare a user-defined record type. This record type is used by the procedure for its OUT parameter, name_addr_rec. Once the package is created, its procedure can be tested with the following PL/SQL block (changes are shown in bold):

```
SET SERVEROUTPUT ON
DECLARE
    v_table_name VARCHAR2(20) := '&sv_table_name';
    v_id NUMBER := &sv_id;
    name_addr_rec DYNAMIC_SQL_PKG.NAME_ADDR_REC_TYPE;
BEGIN
    dynamic_sql_pkg.get_name_address (v_table_name, v_id, name_addr_rec);

    DBMS_OUTPUT.PUT_LINE ('First Name: '||name_addr_rec.first_name);
    DBMS_OUTPUT.PUT_LINE ('Last Name:  '||name_addr_rec.last_name);
    DBMS_OUTPUT.PUT_LINE ('Street:     '||name_addr_rec.street);
    DBMS_OUTPUT.PUT_LINE ('City:       '||name_addr_rec.city);
    DBMS_OUTPUT.PUT_LINE ('State:      '||name_addr_rec.state);
    DBMS_OUTPUT.PUT_LINE ('Zip Code:   '||name_addr_rec.zip);

END;
```

Notice that instead of declaring six variables, you declare one variable of the user-defined record type, name_addr_rec_type. Because this record type has been defined in the package DYNAMIC_SQL_PKG, the name of the record type is prefixed by the name of the package. Similarly, the name of package has been added to the procedure call statement.

When run, this script produces the output shown below (the first output is against the STUDENT table, and the second output is against the INSTRUCTOR table):

```
Enter value for sv_table_name: student
old   2:    v_table_name VARCHAR2(20) := '&sv_table_name';
new   2:    v_table_name VARCHAR2(20) := 'student';
Enter value for sv_id: 105
old   3:    v_id NUMBER := &sv_id;
new   3:    v_id NUMBER := 105;
First Name: Angel
Last Name:  Moskowitz
Street:     320 John St.
City:       Ft. Lee
State:      NJ
Zip Code:   07024


PL/SQL procedure successfully completed.


Enter value for sv_table_name: instructor
old   2:    v_table_name VARCHAR2(20) := '&sv_table_name';
new   2:    v_table_name VARCHAR2(20) := 'instructor';
Enter value for sv_id: 105
old   3:    v_id NUMBER := &sv_id;
new   3:    v_id NUMBER := 105;
First Name: Anita
Last Name:  Morris
Street:     34 Maiden Lane
City:       New York
```

```
State:      NY
Zip Code:   10015

PL/SQL procedure successfully completed.
```