Practices for Lesson 8: Using Dynamic SQL

Chapter 8

Practices for Lesson 8: Overview

Overview

In this practice, you create a package that uses Native Dynamic SQL to create or drop a table, and to populate, modify, and delete rows from the table. In addition, you create a package that compiles the PL/SQL code in your schema, either all the PL/SQL code or only code that has an `INVALID` status in the `USER_OBJECTS` table.

Note:

1. Before starting this practice, execute `/home/oracle/labs/plpu/code_ex/cleanup_scripts/cleanup_08.sql` script.
2. If you missed a step in a practice, please run the appropriate solution script for that practice step before proceeding to the next step or the next practice.

Practice 8-1: Using Native Dynamic SQL

Overview

In this practice, you create a package that uses Native Dynamic SQL to create or drop a table, and to populate, modify, and delete rows from the table. In addition, you create a package that compiles the PL/SQL code in your schema, either all the PL/SQL code or only code that has an `INVALID` status in the `USER_OBJECTS` table.

Note: Execute `cleanup_08.sql script` from `/home/oracle/labs/plpu/code_ex/cleanup_scripts/` before performing the following tasks.

Task

1. Create a package called `TABLE_PKG` that uses Native Dynamic SQL to create or drop a table, and to populate, modify, and delete rows from the table. The subprograms should manage optional default parameters with `NULL` values.
   a. Create a package specification with the following procedures:
   ```
   PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
   PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
     VARCHAR2, p_cols VARCHAR2 := NULL)
   PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
     VARCHAR2, p_conditions VARCHAR2 := NULL)
   PROCEDURE del_row(p_table_name VARCHAR2,
       p_conditions VARCHAR2 := NULL);
   PROCEDURE remove(p_table_name VARCHAR2)
   ```
   b. Create the package body that accepts the parameters and dynamically constructs the appropriate SQL statements that are executed using Native Dynamic SQL, except for the remove procedure. This procedure should be written using the `DBMS_SQL` package.
   c. Execute the `MAKE` package procedure to create a table as follows:
   ```
   make('my_contacts', 'id number(4), name varchar2(40)');
   ```
   d. Describe the `MY_CONTACTS` table structure.
   e. Execute the `ADD_ROW` package procedure to add the following rows. Enable `SERVEROUTPUT`.
   ```
   add_row('my_contacts','1,''Lauran Serhal''','id, name');
   add_row('my_contacts','2,''Nancy''','id, name');
   add_row('my_contacts','3,''Sunitha Patel''','id,name');
   add_row('my_contacts','4,''Valli Pataballa''','id,name');
   ```
   f. Query the `MY_CONTACTS` table contents to verify the additions.
   g. Execute the `DEL_ROW` package procedure to delete a contact with an ID value of `3`.
   h. Execute the `UPD_ROW` procedure with the following row data:
   ```
   upd_row('my_contacts','name=''Nancy Greenberg''','id=2');
   ```
   i. Query the `MY_CONTACTS` table contents to verify the changes.
   j. Drop the table by using the remove procedure and describe the `MY_CONTACTS` table.

2. Create a `COMPILE_PKG` package that compiles the PL/SQL code in your schema.
   a. In the specification, create a package procedure called `MAKE` that accepts the name of a PL/SQL program unit to be compiled.
   b. In the package body, include the following:
      1) The `EXECUTE` procedure used in the `TABLE_PKG` procedure in step 1 of this practice.
      2) A private function named `GET_TYPE` to determine the PL/SQL object type from the data dictionary.
         – The function returns the type name (use `PACKAGE` for a package with a body) if the object exists; otherwise, it should return a `NULL`.
         – In the `WHERE` clause condition, add the following to the condition to ensure that only one row is returned if the name represents a `PACKAGE`, which may also have a `PACKAGE BODY`. In this case, you can only compile the complete package, but not the specification or body as separate components:

```
rownum = 1
```

      3) Create the `MAKE` procedure by using the following information:
         – The `MAKE` procedure accepts one argument, name, which represents the object name.
         – The `MAKE` procedure should call the `GET_TYPE` function. If the object exists, `MAKE` dynamically compiles it with the `ALTER` statement.
   c. Use the `COMPILE_PKG.MAKE` procedure to compile the following:
      1) The `EMPLOYEE_REPORT` procedure
      2) The `EMP_PKG` package
      3) A nonexistent object called `EMP_DATA`

Solution 8-1: Using Native Dynamic SQL

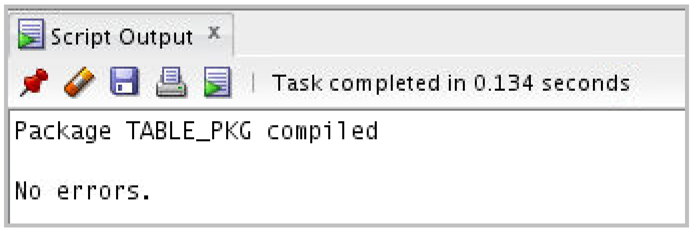
In this practice, you create a package that uses Native Dynamic SQL to create or drop a table, and to populate, modify, and delete rows from the table. In addition, you create a package that compiles the PL/SQL code in your schema, either all the PL/SQL code or only code that has an `INVALID` status in the `USER_OBJECTS` table.

1. Create a package called `TABLE_PKG` that uses Native Dynamic SQL to create or drop a table, and to populate, modify, and delete rows from the table. The subprograms should manage optional default parameters with `NULL` values.
   a. Create a package specification with the following procedures:

```
PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
  VARCHAR2, p_cols VARCHAR2 := NULL)
PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
  VARCHAR2, p_conditions VARCHAR2 := NULL)
PROCEDURE del_row(p_table_name VARCHAR2,
    p_conditions VARCHAR2 := NULL);
PROCEDURE remove(p_table_name VARCHAR2)
```

Open the `/home/oracle/labs/plpu/solns/sol_08.sql` script. Uncomment and select the code under Task 1_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create and compile the package specification. The code and the result are displayed as follows:

```
CREATE OR REPLACE PACKAGE table_pkg IS
  PROCEDURE make(p_table_name VARCHAR2, p_col_specs
      VARCHAR2);
  PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
      VARCHAR2, p_cols VARCHAR2 := NULL);
  PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
      VARCHAR2, p_conditions VARCHAR2 := NULL);
  PROCEDURE del_row(p_table_name VARCHAR2, p_conditions
      VARCHAR2 := NULL);
  PROCEDURE remove(p_table_name VARCHAR2);
END table_pkg;
/
SHOW ERRORS
```

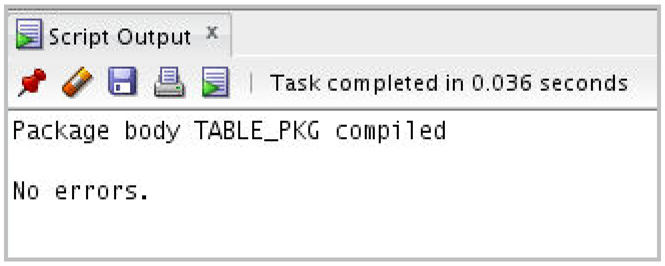


```
Package TABLE_PKG compiled

No errors.
```

b. Create the package body that accepts the parameters and dynamically constructs the appropriate SQL statements that are executed using Native Dynamic SQL, except for the remove procedure. This procedure should be written using the `DBMS_SQL` package.

   Uncomment and select the code under Task 1_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create and compile the package specification. The code and the result are shown below.

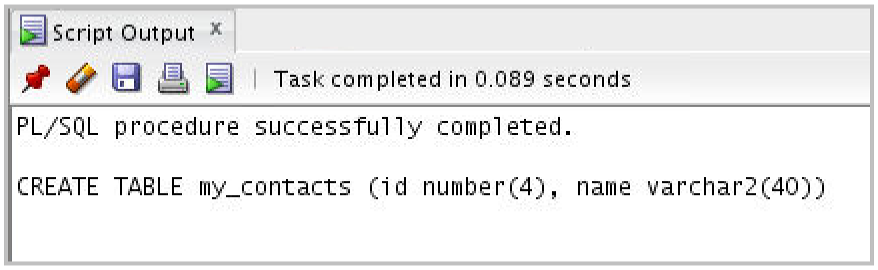
```
CREATE OR REPLACE PACKAGE BODY table_pkg IS
  PROCEDURE execute(p_stmt VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_stmt);
    EXECUTE IMMEDIATE p_stmt;
  END;

  PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
  IS
    v_stmt VARCHAR2(200) := 'CREATE TABLE '|| p_table_name ||
                        ' (' || p_col_specs || ')';
  BEGIN
    execute(v_stmt);
  END;

  PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                    VARCHAR2, p_cols VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'INSERT INTO '|| p_table_name;
  BEGIN
    IF p_cols IS NOT NULL THEN
       v_stmt := v_stmt || ' (' || p_cols || ')';
    END IF;
    v_stmt := v_stmt || ' VALUES (' || p_col_values || ')';
    execute(v_stmt);
  END;

  PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                   VARCHAR2, p_conditions VARCHAR2 := NULL) IS


   v_stmt VARCHAR2(200) := 'UPDATE '|| p_table_name || ' SET '
|| p_set_values;
  BEGIN
    IF p_conditions IS NOT NULL THEN
       v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
  END;


  PROCEDURE del_row(p_table_name VARCHAR2, p_conditions

                    VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'DELETE FROM '|| p_table_name;
  BEGIN
    IF p_conditions IS NOT NULL THEN
      v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
  END;

  PROCEDURE remove(p_table_name VARCHAR2) IS
    cur_id INTEGER;
    v_stmt VARCHAR2(100) := 'DROP TABLE '||p_table_name;
  BEGIN
    cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_OUTPUT.PUT_LINE(v_stmt);
    DBMS_SQL.PARSE(cur_id, v_stmt, DBMS_SQL.NATIVE);
    -- Parse executes DDL statements,no EXECUTE is required.
    DBMS_SQL.CLOSE_CURSOR(cur_id);
  END;

END table_pkg;
/
SHOW ERRORS
```



```
Package body TABLE_PKG compiled

No errors.
```

c. Execute the `MAKE` package procedure to create a table as follows:
`make('my_contacts', 'id number(4), name varchar2(40)');`
Uncomment and select the code under Task 1_c. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create the package specification. The code and the results are displayed as follows:

```
EXECUTE table_pkg.make('my_contacts', 'id number(4), name
varchar2(40)')
```



Script Output ×

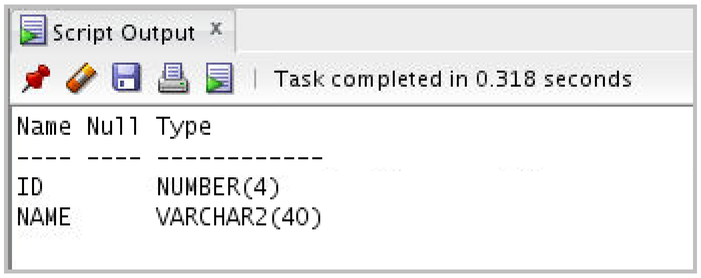
Task completed in 0.089 seconds

```
PL/SQL procedure successfully completed.

CREATE TABLE my_contacts (id number(4), name varchar2(40))
```

d. Describe the `MY_CONTACTS` table structure.

```
DESCRIBE my_contacts
```

The result is displayed as follows:



Script Output ×

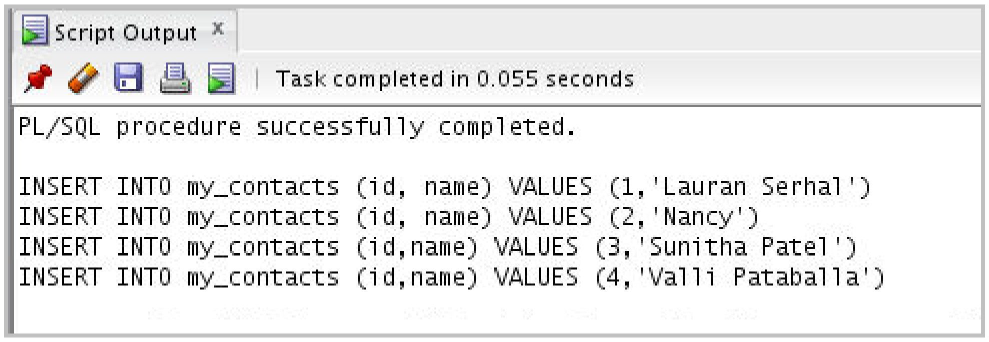
Task completed in 0.318 seconds

```
Name Null Type
---- ---- ------------
ID        NUMBER(4)
NAME      VARCHAR2(40)
```

e. Execute the `ADD_ROW` package procedure to add the following rows.

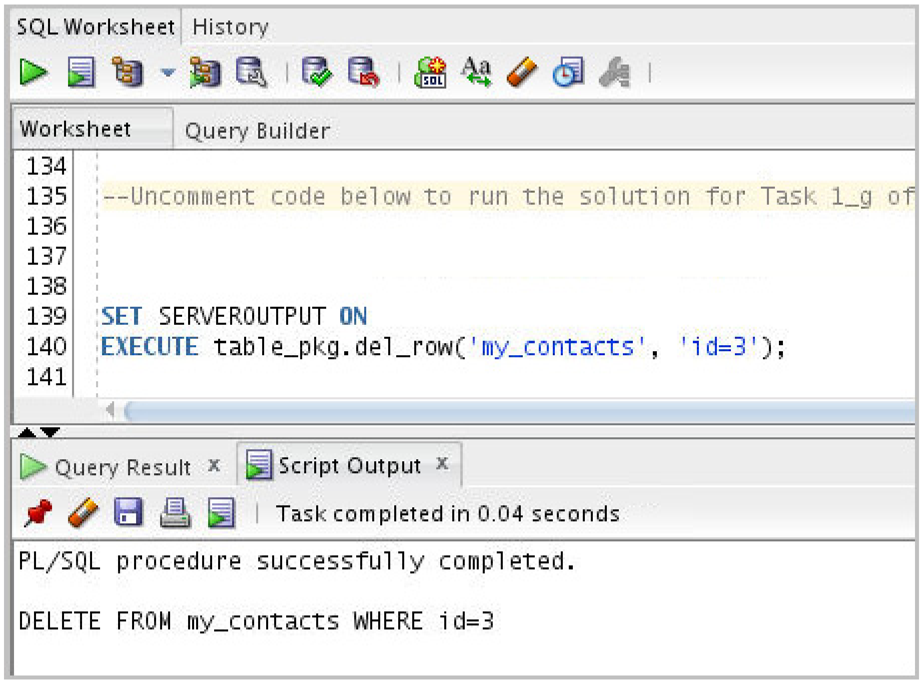
```
SET SERVEROUTPUT ON

BEGIN
  table_pkg.add_row('my_contacts','1,''Lauran Serhal''','id,
   name');
  table_pkg.add_row('my_contacts','2,''Nancy''','id, name');
  table_pkg.add_row('my_contacts','3,''Sunitha
   Patel''','id,name');
  table_pkg.add_row('my_contacts','4,''Valli
   Pataballa''','id,name');
END;
/
```

Uncomment and select the code under Task 1_e. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to execute the script.



```
PL/SQL procedure successfully completed.

INSERT INTO my_contacts (id, name) VALUES (1,'Lauran Serhal')
INSERT INTO my_contacts (id, name) VALUES (2,'Nancy')
INSERT INTO my_contacts (id,name) VALUES (3,'Sunitha Patel')
INSERT INTO my_contacts (id,name) VALUES (4,'Valli Pataballa')
```

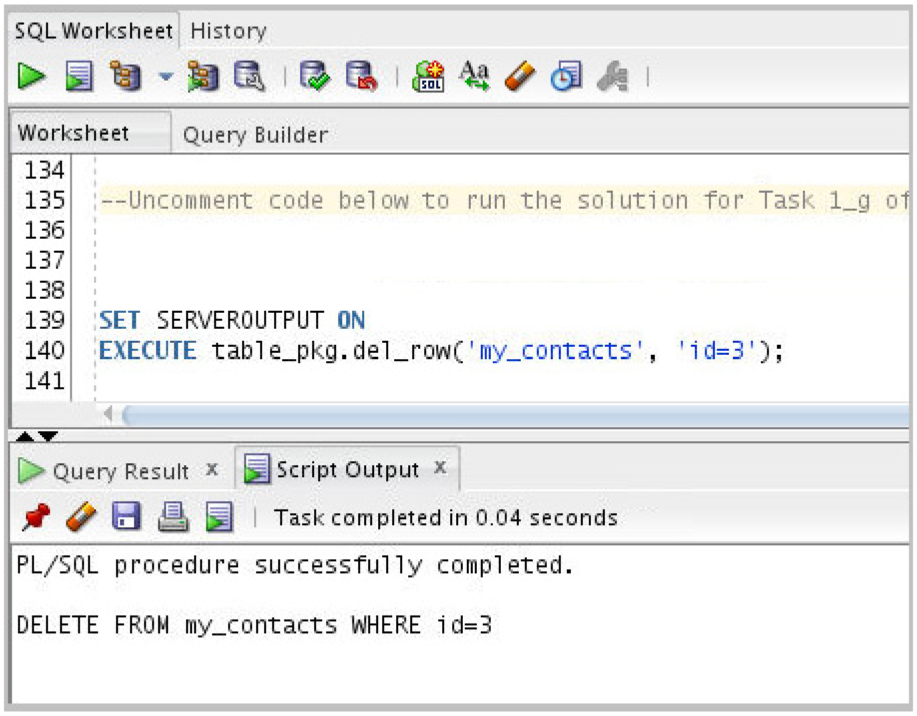
f. Query the `MY_CONTACTS` table contents to verify the additions.
The code and the results are displayed as follows:



```
--Uncomment code below to run the solution for Task 1_g of

SET SERVEROUTPUT ON
EXECUTE table_pkg.del_row('my_contacts', 'id=3');
```

```
PL/SQL procedure successfully completed.

DELETE FROM my_contacts WHERE id=3
```

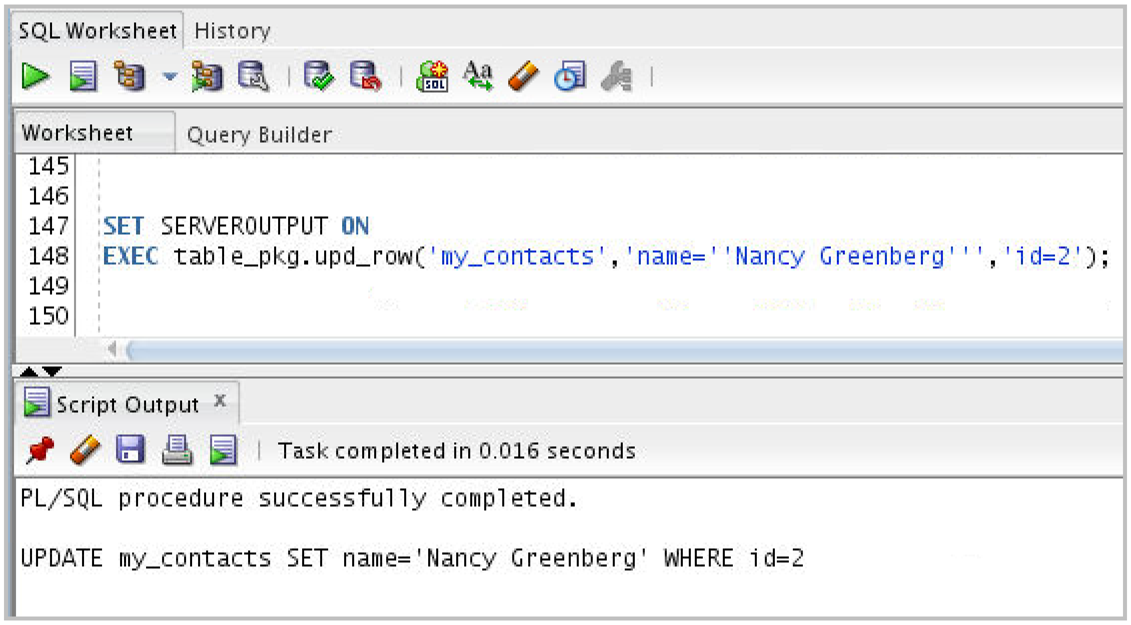
g. Execute the `DEL_ROW` package procedure to delete a contact with an ID value of `3`.
The code and the results are displayed as follows:



```
--Uncomment code below to run the solution for Task 1_g of


SET SERVEROUTPUT ON
EXECUTE table_pkg.del_row('my_contacts', 'id=3');
```

```
PL/SQL procedure successfully completed.

DELETE FROM my_contacts WHERE id=3
```

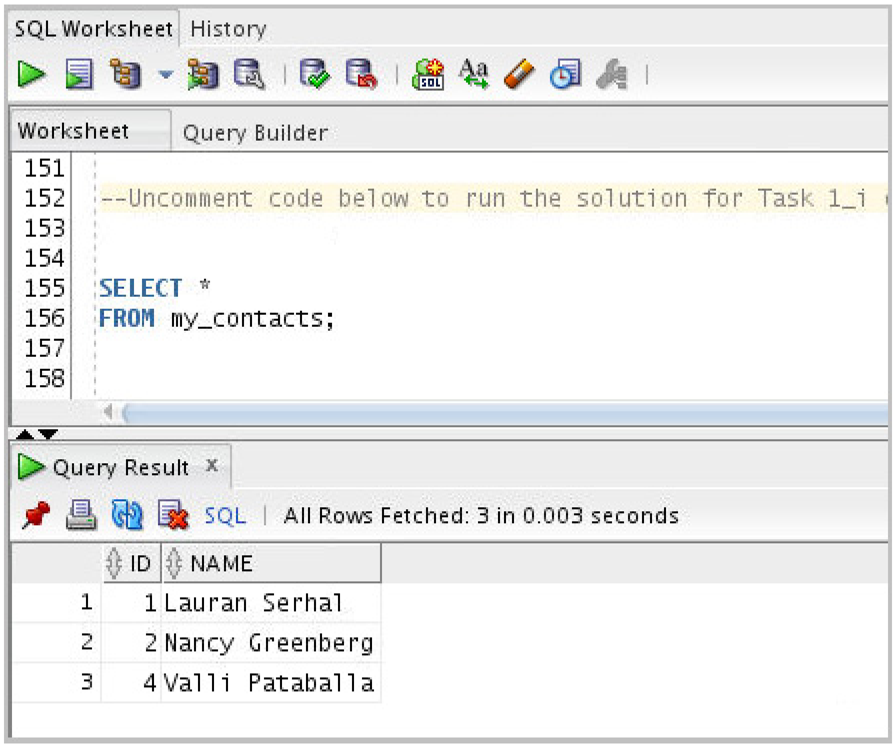
h. Execute the `UPD_ROW` procedure with the following row data:
`upd_row('my_contacts','name=''Nancy Greenberg''','id=2');`
The code and the results are displayed as follows:



```
SET SERVEROUTPUT ON
EXEC table_pkg.upd_row('my_contacts','name=''Nancy Greenberg''','id=2');
```

```
PL/SQL procedure successfully completed.

UPDATE my_contacts SET name='Nancy Greenberg' WHERE id=2
```

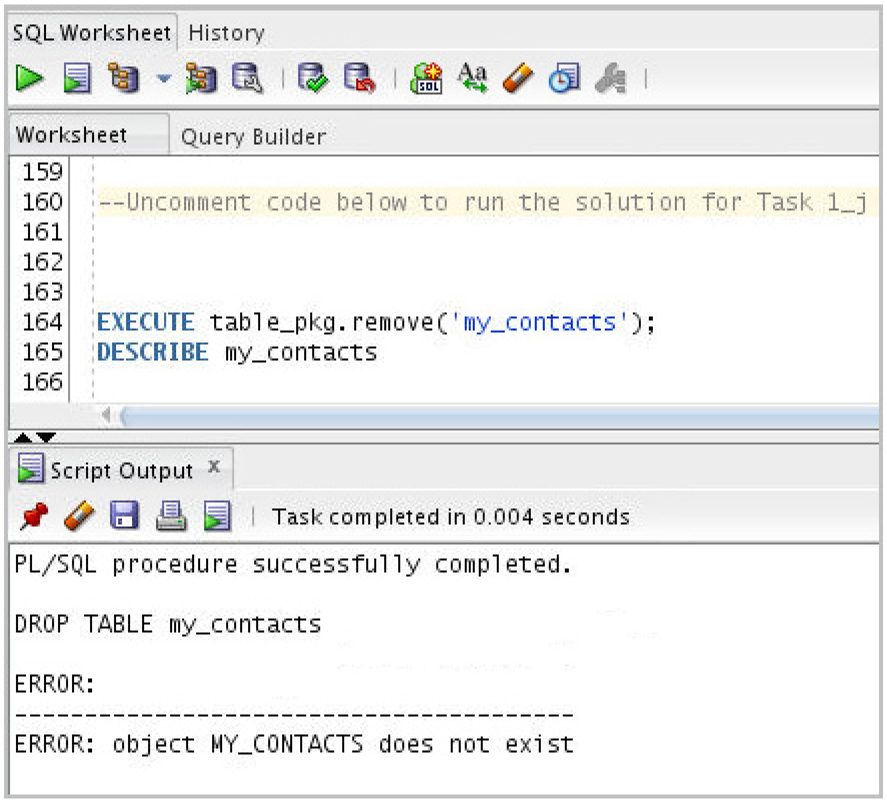
i. Query the `MY_CONTACTS` table contents to verify the changes.
The code and the results are displayed as follows:



```
--Uncomment code below to run the solution for Task 1_i

SELECT *
FROM my_contacts;
```

```
   ID NAME
1   1 Lauran Serhal
2   2 Nancy Greenberg
3   4 Valli Pataballa
```

j. Drop the table by using remove procedure and describe the `MY_CONTACTS` table.
The code and the results are displayed as follows:



```
--Uncomment code below to run the solution for Task 1_j



EXECUTE table_pkg.remove('my_contacts');
DESCRIBE my_contacts
```

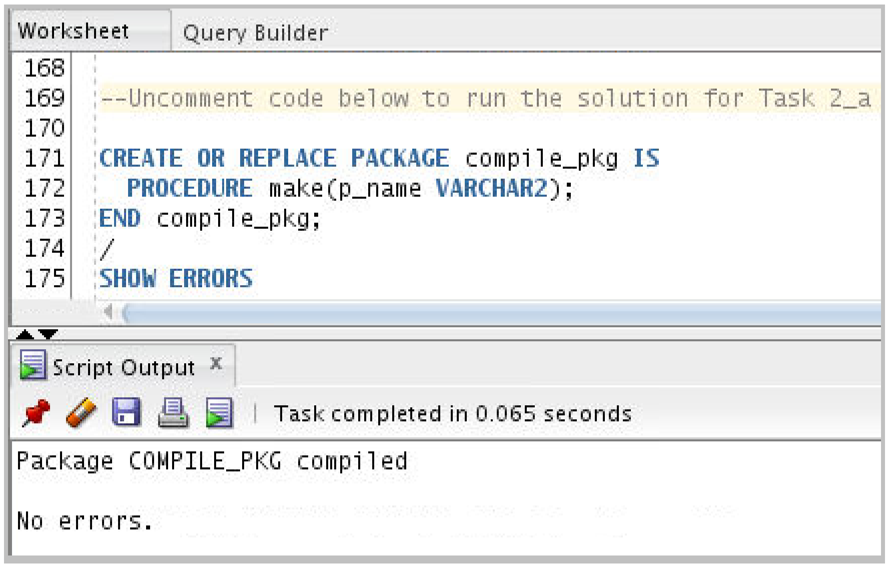
```
PL/SQL procedure successfully completed.

DROP TABLE my_contacts

ERROR:
----------------------------------------
ERROR: object MY_CONTACTS does not exist
```

2. Create a `COMPILE_PKG` package that compiles the PL/SQL code in your schema.

a. In the specification, create a package procedure called MAKE that accepts the name of a PL/SQL program unit to be compiled.

Uncomment and select the code under Task 2_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create and compile the package specification. The code and the results are shown below.

```
CREATE OR REPLACE PACKAGE compile_pkg IS
  PROCEDURE make(p_name VARCHAR2);
END compile_pkg;
/
SHOW ERRORS
```



```
--Uncomment code below to run the solution for Task 2_a

CREATE OR REPLACE PACKAGE compile_pkg IS
  PROCEDURE make(p_name VARCHAR2);
END compile_pkg;
/
SHOW ERRORS
```

```
Package COMPILE_PKG compiled

No errors.
```

b. In the package body, include the following:

1) The EXECUTE procedure used in the TABLE_PKG procedure in step 1 of this practice.

2) A private function named GET_TYPE to determine the PL/SQL object type from the data dictionary.

   – The function returns the type name (use PACKAGE for a package with a body) if the object exists; otherwise, it should return a NULL.

   – In the WHERE clause condition, add the following to the condition to ensure that only one row is returned if the name represents a PACKAGE, which may also have a PACKAGE BODY. In this case, you can only compile the complete package, but not the specification or body as separate components:

     `rownum = 1`

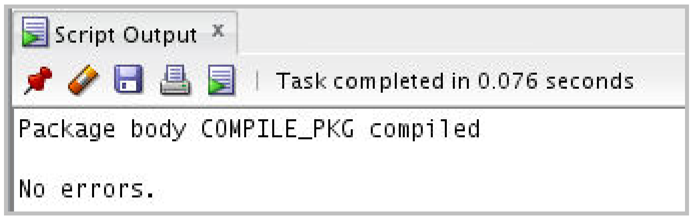
3) Create the MAKE procedure by using the following information:

   – The MAKE procedure accepts one argument, name, which represents the object name.

   – The MAKE procedure should call the GET_TYPE function. If the object exists, MAKE dynamically compiles it with the ALTER statement.

Uncomment and select the code under Task 2_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to create and compile the package body. The code and the results are displayed as follows:

```
CREATE OR REPLACE PACKAGE BODY compile_pkg IS

  PROCEDURE execute(p_stmt VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_stmt);
    EXECUTE IMMEDIATE p_stmt;
  END;

  FUNCTION get_type(p_name VARCHAR2) RETURN VARCHAR2 IS
    v_proc_type VARCHAR2(30) := NULL;
  BEGIN

    -- The ROWNUM = 1 is added to the condition
    -- to ensure only one row is returned if the
    -- name represents a PACKAGE, which may also
    -- have a PACKAGE BODY. In this case, we can
    -- only compile the complete package, but not
    -- the specification or body as separate
    -- components.

    SELECT object_type INTO v_proc_type
    FROM user_objects
    WHERE object_name = UPPER(p_name)
    AND ROWNUM = 1;
    RETURN v_proc_type;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN NULL;
  END;

  PROCEDURE make(p_name VARCHAR2) IS
    v_stmt       VARCHAR2(100);
    v_proc_type  VARCHAR2(30) := get_type(p_name);
  BEGIN
    IF v_proc_type IS NOT NULL THEN
      v_stmt := 'ALTER '|| v_proc_type ||' '|| p_name ||'
COMPILE';
      execute(v_stmt);
    ELSE
      RAISE_APPLICATION_ERROR(-20001,
         'Subprogram '''|| p_name ||''' does not exist');
    END IF;
  END make;
END compile_pkg;
/
SHOW ERRORS
```



Script Output

Task completed in 0.076 seconds

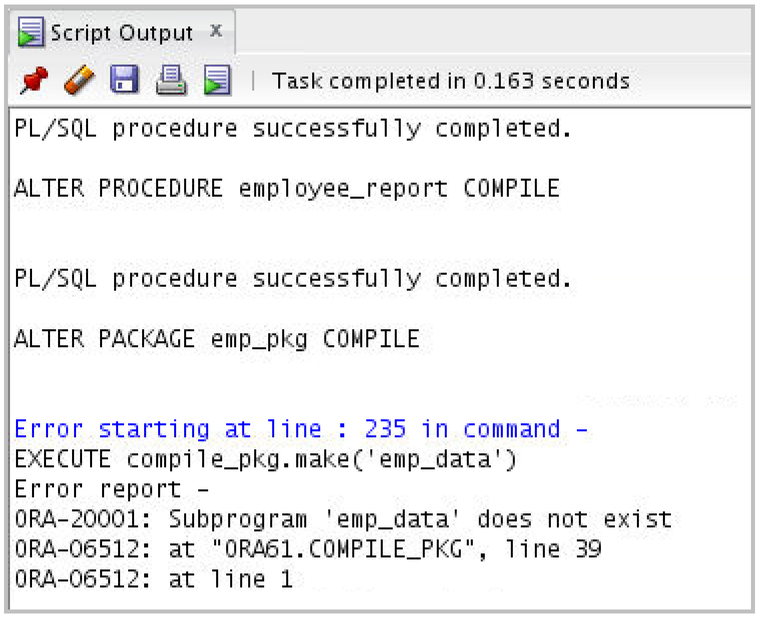
```
Package body COMPILE_PKG compiled

No errors.
```

c. Use the `COMPILE_PKG.MAKE` procedure to compile the following:

1) The `EMPLOYEE_REPORT` procedure
2) The `EMP_PKG` package
3) A nonexistent object called `EMP_DATA`

Uncomment and select the code under task 2_c. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to execute the package's procedure. The result is shown below.

```
SET SERVEROUTPUT ON
EXECUTE compile_pkg.make('employee_report')
EXECUTE compile_pkg.make('emp_pkg')
EXECUTE compile_pkg.make('emp_data')
```



Script Output

Task completed in 0.163 seconds

```
PL/SQL procedure successfully completed.

ALTER PROCEDURE employee_report COMPILE


PL/SQL procedure successfully completed.

ALTER PACKAGE emp_pkg COMPILE


Error starting at line : 235 in command -
EXECUTE compile_pkg.make('emp_data')
Error report -
ORA-20001: Subprogram 'emp_data' does not exist
ORA-06512: at "ORA61.COMPILE_PKG", line 39
ORA-06512: at line 1
```