Exercises for Chapter 18: Bulk SQL

The Labs below provide you with exercises and suggested answers with discussion related to how those answers resulted. The most important thing to realize is whether your answer works. You should figure out the implications of the answers here and what the effects are from any different answers you may come up with.

By the Way

Note that there is no Exercise section for Lab 18.3 Binding Collections in SQL Statements.

Lab 18.1 FORALL Statement

In this exercise, you will modify script ch18_9a.sql created in this chapter. Throughout this lab you will use various options available for the FORALL statement such as SAVE EXCEPTIONS, INDICES OF, and VALUES OF.

Answer the following questions:

1. Modify the script ch18_9a.sql as follows. Select data from the ZIPCODE table for a different state, i.e., 'MA'. Modify the selected records so that they will cause various exceptions in the FORALL statement. Modify the FORALL statement so that it does not fail when an exception occurs. Finally, display detailed exception information.

   Answer: The new version of the script should look similar to the following. Changes are shown in bold.

   For Example *ch18_9b.sql*

```
DECLARE
   -- Declare collection types
   TYPE string_type IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
   TYPE date_type   IS TABLE OF DATE          INDEX BY PLS_INTEGER;

   -- Declare collection variables to be used by the FORALL statement
   zip_tab      string_type;
   city_tab     string_type;
   state_tab    string_type;
   cr_by_tab    string_type;
   cr_date_tab  date_type;
   mod_by_tab   string_type;
   mod_date_tab date_type;

   v_counter PLS_INTEGER := 0;
   v_total   INTEGER := 0;
```

```
   -- Define user-defined exception and associated Oracle error number with it
   errors EXCEPTION;
   PRAGMA EXCEPTION_INIT(errors, -24381);

BEGIN
   -- Populate individual collections
   SELECT *
     BULK COLLECT INTO zip_tab, city_tab, state_tab, cr_by_tab,
        cr_date_tab, mod_by_tab, mod_date_tab
     FROM zipcode
    WHERE state = 'MA';

   -- Modify individual collection records to produce various exceptions
   zip_tab(1)     := NULL;
   city_tab(2)    := RPAD(city_tab(2), 26, ' ');
   state_tab(3)   := SYSDATE;
   cr_by_tab(4)   := RPAD(cr_by_tab(4), 31, ' ');
   cr_date_tab(5) := NULL;

   -- Populate MY_ZIPCODE table
   FORALL i in 1..zip_tab.COUNT SAVE EXCEPTIONS
      INSERT INTO my_zipcode
         (zip, city, state, created_by, created_date, modified_by, modified_date)
      VALUES
         (zip_tab(i), city_tab(i), state_tab(i), cr_by_tab(i), cr_date_tab(i)
         ,mod_by_tab(i), mod_date_tab(i));
   COMMIT;

   -- Check how many records were added to MY_ZIPCODE table
   SELECT COUNT(*)
     INTO v_total
     FROM my_zipcode
    WHERE state = 'MA';

   DBMS_OUTPUT.PUT_LINE (v_total||' records were added to MY_ZIPCODE table');

EXCEPTION
   WHEN errors
   THEN
      -- Display total number of exceptions encountered
      DBMS_OUTPUT.PUT_LINE
         ('There were '||SQL%BULK_EXCEPTIONS.COUNT||' exceptions');

      -- Display detailed exception information
      FOR i in 1.. SQL%BULK_EXCEPTIONS.COUNT
      LOOP
         DBMS_OUTPUT.PUT_LINE ('Record '||
            SQL%BULK_EXCEPTIONS(i).error_index||' caused error '||i||
            ': '||SQL%BULK_EXCEPTIONS(i).ERROR_CODE||' '||
            SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
      END LOOP;

      -- Commit records if any that were inserted successfully
      COMMIT;
```

```
END;
```

In the script above, you declare user-define exception and associate Oracle error number with it via the EXCEPTION_INIT pragma. Next, you populate individual collections with the cursor FOR LOOP against the ZIPCODE table, and then modify them so that they will cause exceptions in the FORALL statement. For example, the first record of the zip_tab collection is set to NULL. This will cause constraint violation as the ZIP column in MY_ZIPCODE table has NOT NULL constraint defined against it. Then, you add SAVE EXCEPTIONS clause to the FORALL statement, and an exception-handling section to the PL/SQL block. In this section, you display total number of errors encountered along with detailed exception information. Note the COMMIT statement in the exception-handling section. This statement is added so that records that are inserted successfully by the FORALL statement are committed when the control of the execution is passed to the exception-handling section of the block.

When run, this version of the script produces the following output:

```
There were 5 exceptions
Record 1 caused error 1: 1400 ORA-01400: cannot insert NULL into ()
Record 2 caused error 2: 12899 ORA-12899: value too large for column  (actual: ,
maximum: )
Record 3 caused error 3: 12899 ORA-12899: value too large for column  (actual: ,
maximum: )
Record 4 caused error 4: 12899 ORA-12899: value too large for column  (actual: ,
maximum: )
Record 5 caused error 5: 1400 ORA-01400: cannot insert NULL into ()
```

2. Modify the script ch18_9b.sql as follows. Do not modify records selected from the ZIPCODE table so that there are no exceptions raised; instead, delete first 3 records from each collection so that they become sparse. Then modify the FORALL statement accordingly.

Answer: This version of the script should look similar to the script below. Modified statements are highlighted in bold.

For Example *ch18_9c.sql*

```
DECLARE
   -- Declare collection types
   TYPE string_type IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
   TYPE date_type   IS TABLE OF DATE          INDEX BY PLS_INTEGER;

   -- Declare collection variables to be used by the FORALL statement
   zip_tab      string_type;
   city_tab     string_type;
   state_tab    string_type;
   cr_by_tab    string_type;
   cr_date_tab  date_type;
   mod_by_tab   string_type;
   mod_date_tab date_type;

   v_counter PLS_INTEGER := 0;
   v_total   INTEGER := 0;

   -- Define user-defined exception and associated Oracle error number with it
   errors EXCEPTION;
```

```
   PRAGMA EXCEPTION_INIT(errors, -24381);

BEGIN
   -- Populate individual collections
   SELECT *
     BULK COLLECT INTO zip_tab, city_tab, state_tab, cr_by_tab,
        cr_date_tab, mod_by_tab, mod_date_tab
     FROM zipcode
    WHERE state = 'MA';

   -- Delete first 3 records from each collection
   zip_tab.DELETE(1,3);
   city_tab.DELETE(1,3);
   state_tab.DELETE(1,3);
   cr_by_tab.DELETE(1,3);
   cr_date_tab.DELETE(1,3);
   mod_by_tab.DELETE(1,3);
   mod_date_tab.DELETE(1,3);

   -- Populate MY_ZIPCODE table
   FORALL i IN INDICES OF zip_tab SAVE EXCEPTIONS
      INSERT INTO my_zipcode
         (zip, city, state, created_by, created_date, modified_by, modified_date)
      VALUES
         (zip_tab(i), city_tab(i), state_tab(i), cr_by_tab(i), cr_date_tab(i)
         ,mod_by_tab(i), mod_date_tab(i));
   COMMIT;

   -- Check how many records were added to MY_ZIPCODE table
   SELECT COUNT(*)
     INTO v_total
     FROM my_zipcode
    WHERE state = 'MA';

   DBMS_OUTPUT.PUT_LINE (v_total||' records were added to MY_ZIPCODE table');

EXCEPTION
   WHEN errors
   THEN
      -- Display total number of exceptions encountered
      DBMS_OUTPUT.PUT_LINE
         ('There were '||SQL%BULK_EXCEPTIONS.COUNT||' exceptions');

      -- Display detailed exception information
      FOR i in 1.. SQL%BULK_EXCEPTIONS.COUNT LOOP
         DBMS_OUTPUT.PUT_LINE ('Record '||
            SQL%BULK_EXCEPTIONS(i).error_index||' caused error '||i||
            ': '||SQL%BULK_EXCEPTIONS(i).ERROR_CODE||' '||
            SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
      END LOOP;

      -- Commit records if any that were inserted successfully
      COMMIT;
```

```
END;
```

This version of the script contains two modifications. First, you delete first three records from each collection. Second, you modify the FORALL statement by replacing lower and upper limits for the counter variable with the INDICES OF clause.

When run, the script produces the following output:

```
2 records were added to MY_ZIPCODE table
```

3. Modify second version of the script, ch18_9b.sql, as follows. Insert records that cause exceptions in a different table called MY_ZIPCODE_EXC.

Answer: The MY_ZIPCODE_EXC table may be created as follows:

```
CREATE TABLE MY_ZIPCODE_EXC
   (ZIP           VARCHAR2(100),
    CITY          VARCHAR2(100),
    STATE         VARCHAR2(100),
    CREATED_BY    VARCHAR2(100),
    CREATED_DATE  DATE,
    MODIFIED_BY   VARCHAR2(100),
    MODIFIED_DATE DATE);
```

Note that even though this table has the same columns as the MY_ZIPCODE table, the column sizes have been increased and all NOT NULL constraints removed. This ensures that records which cause exceptions in the FORALL statement can be inserted in this table.

Next, the script is modified as follows. Changes are shown in bold.

For Example *ch18_9d.sql*

```
DECLARE
   -- Declare collection types
   TYPE string_type  IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
   TYPE date_type    IS TABLE OF DATE          INDEX BY PLS_INTEGER;
   TYPE exc_ind_type IS TABLE OF PLS_INTEGER   INDEX BY PLS_INTEGER;

   -- Declare collection variables to be used by the FORALL statement
   zip_tab      string_type;
   city_tab     string_type;
   state_tab    string_type;
   cr_by_tab    string_type;
   cr_date_tab  date_type;
   mod_by_tab   string_type;
   mod_date_tab date_type;
   exc_ind_tab  exc_ind_type;

   v_counter PLS_INTEGER := 0;
   v_total   INTEGER := 0;

   -- Define user-defined exception and associated Oracle error number with it
   errors EXCEPTION;
   PRAGMA EXCEPTION_INIT(errors, -24381);

BEGIN
   -- Populate individual collections
```

```
   SELECT *
     BULK COLLECT INTO zip_tab, city_tab, state_tab, cr_by_tab,
        cr_date_tab, mod_by_tab, mod_date_tab
     FROM zipcode
    WHERE state = 'MA';

   -- Modify individual collection records to produce various exceptions
   zip_tab(1)     := NULL;
   city_tab(2)    := RPAD(city_tab(2), 26, ' ');
   state_tab(3)   := SYSDATE;
   cr_by_tab(4)   := RPAD(cr_by_tab(4), 31, ' ');
   cr_date_tab(5) := NULL;

   -- Populate MY_ZIPCODE table
   FORALL i in 1..zip_tab.COUNT SAVE EXCEPTIONS
      INSERT INTO my_zipcode
         (zip, city, state, created_by, created_date, modified_by, modified_date)
      VALUES
         (zip_tab(i), city_tab(i), state_tab(i), cr_by_tab(i), cr_date_tab(i)
         ,mod_by_tab(i), mod_date_tab(i));
   COMMIT;

   -- Check how many records were added to MY_ZIPCODE table
   SELECT COUNT(*)
     INTO v_total
     FROM my_zipcode
    WHERE state = 'MA';

   DBMS_OUTPUT.PUT_LINE (v_total||' records were added to MY_ZIPCODE table');

EXCEPTION
   WHEN errors
   THEN
      -- Populate V_EXC_IND_TAB collection to be used in the VALUES OF clause
      FOR i in 1.. SQL%BULK_EXCEPTIONS.COUNT
      LOOP
         exc_ind_tab(i) := SQL%BULK_EXCEPTIONS(i).error_index;
      END LOOP;

      -- Insert records that caused exceptions in the MY_ZIPCODE_EXC table
      FORALL i in VALUES OF exc_ind_tab
         INSERT INTO my_zipcode_exc
            (zip, city, state, created_by, created_date, modified_by, modified_date)
         VALUES
            (zip_tab(i), city_tab(i), state_tab(i), cr_by_tab(i), cr_date_tab(i)
            ,mod_by_tab(i), mod_date_tab(i));

      COMMIT;
END;
```

In this version of the script, you modify exception-handling section so that records causing exceptions in the `FORALL` statement are inserted in the `MY_ZIPCODE_EXC` table created earlier. First, you populate collection `exc_ind_tab` with subscripts of records that caused

exceptions in the FORALL statement. Then, you loop through this collection and insert erroneous records in the MY_ZIPCODE_EXC table. After execution of the script, MY_ZIPCODE_EXC table contains records that caused exceptions.

Lab 18.2 BULK COLLECT Clause

In this exercise, you will create various scripts that will select and modify data in MY_INSTRUCTOR table in bulk.

Create MY_INSTRUCTOR table as follows. Note that if this table already exists, drop it first and then recreate it:

```
DROP TABLE my_instructor;

CREATE TABLE my_instructor AS
SELECT *
  FROM instructor;
```

Answer the following questions:

4. Create the following script: Select instructor ID, first and last names from the MY_INSTRUCTOR table and display it on the screen. Note that the data should be fetched in bulk.

Answer: The newly created script should look similar to the following:

For Example *ch18_14a.sql*

```
DECLARE
   -- Define collection types and variables to be used by the BULK COLLECT clause
   TYPE instructor_id_type IS TABLE OF my_instructor.instructor_id%TYPE;
   TYPE first_name_type    IS TABLE OF my_instructor.first_name%TYPE;
   TYPE last_name_type     IS TABLE OF my_instructor.last_name%TYPE;

   instructor_id_tab instructor_id_type;
   first_name_tab    first_name_type;
   last_name_tab     last_name_type;

BEGIN
   -- Fetch all instructor data at once via BULK COLLECT clause
   SELECT instructor_id, first_name, last_name
     BULK COLLECT INTO instructor_id_tab, first_name_tab, last_name_tab
     FROM my_instructor;

   FOR i IN instructor_id_tab.FIRST..instructor_id_tab.LAST
   LOOP
      DBMS_OUTPUT.PUT_LINE ('instructor_id: '||instructor_id_tab(i));
      DBMS_OUTPUT.PUT_LINE ('first_name:    '||first_name_tab(i));
      DBMS_OUTPUT.PUT_LINE ('last_name:     '||last_name_tab(i));
   END LOOP;
END;
```

The declaration portion of this script contains definitions of three collection types and variables. The executable portion of the script populates collection variables via the SELECT

statement with the BULK COLLECT clause. Finally, it displays data stored in the collection variables by looping through them.

When run this script produces output as follows:

```
instructor_id: 101
first_name:    Fernand
last_name:     Hanks
instructor_id: 102
first_name:    Tom
last_name:     Wojick
instructor_id: 103
first_name:    Nina
last_name:     Schorin
instructor_id: 104
first_name:    Gary
last_name:     Pertez
instructor_id: 105
first_name:    Anita
last_name:     Morris
instructor_id: 106
first_name:    Todd
last_name:     Smythe
instructor_id: 107
first_name:    Marilyn
last_name:     Frantzen
instructor_id: 108
first_name:    Charles
last_name:     Lowry
instructor_id: 109
first_name:    Rick
last_name:     Chow
instructor_id: 110
first_name:    Irene
last_name:     Willig
```

As mentioned previously, the BULK COLLECT clause is similar to the cursor loop in that it does not through NO_DATA_FOUND exception when no rows are returned by the SELECT statement. Consider deleting all rows from the MY_INSTRUCTOR table and then executing this script again. In this case the script produces error as follows:

```
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at line 17
```

Note that the error in the script refers to line 17 which contains FOR LOOP that iterates through the collections and displays the results. Note that the SELECT statement with the BULK COLLECT clause did not cause any errors. To prevent this error from happening, the script can be modified as follows. Changes are shown in bold letters.

For Example *ch18_14b.sql*

```
DECLARE
   -- Define collection types and variables to be used by the
   -- BULK COLLECT clause
   TYPE instructor_id_type IS TABLE OF my_instructor.instructor_id%TYPE;
   TYPE first_name_type    IS TABLE OF my_instructor.first_name%TYPE;
   TYPE last_name_type     IS TABLE OF my_instructor.last_name%TYPE;
```

```
   instructor_id_tab instructor_id_type;
   first_name_tab    first_name_type;
   last_name_tab     last_name_type;

BEGIN
   -- Fetch all instructor data at once via BULK COLLECT clause
   SELECT instructor_id, first_name, last_name
     BULK COLLECT INTO instructor_id_tab, first_name_tab, last_name_tab
     FROM my_instructor;

   IF instructor_id_tab.COUNT > 0
   THEN
      FOR i IN instructor_id_tab.FIRST..instructor_id_tab.LAST
      LOOP
         DBMS_OUTPUT.PUT_LINE ('instructor_id: '||instructor_id_tab(i));
         DBMS_OUTPUT.PUT_LINE ('first_name:    '||first_name_tab(i));
         DBMS_OUTPUT.PUT_LINE ('last_name:     '||last_name_tab(i));
      END LOOP;
   END IF;
END;
```

This version of the script contains `IF-THEN` statement that encloses the `FOR` loop. The `IF-THEN` statement checks if one of the collections is non-empty; thus, preventing the 'numeric or value error'.

Watch Out!

If you have deleted records from the `MY_INSTRUCTOR` table, you need to roll back your changes or populate it with the records from the `INSTRUCTOR` table again before proceeding with rest of the exercises in this Lab.

5. Modify newly created script as follows: fetch no more than five rows at one time from `MY_INSTRUCTOR` table.

Answer: The script should look similar to the following. Modifications are highlighted in bold.

For Example *ch18_14c.sql*

```
DECLARE
   CURSOR instructor_cur IS
      SELECT instructor_id, first_name, last_name
        FROM my_instructor;

   -- Define collection types and variables to be used by the BULK COLLECT clause
   TYPE instructor_id_type IS TABLE OF my_instructor.instructor_id%TYPE;
   TYPE first_name_type    IS TABLE OF my_instructor.first_name%TYPE;
   TYPE last_name_type     IS TABLE OF my_instructor.last_name%TYPE;

   instructor_id_tab instructor_id_type;
   first_name_tab    first_name_type;
   last_name_tab     last_name_type;

   v_limit PLS_INTEGER := 5;
```

```
BEGIN
   OPEN instructor_cur;
   LOOP
      -- Fetch partial instructor data at once via BULK COLLECT clause
      FETCH instructor_cur
       BULK COLLECT INTO instructor_id_tab, first_name_tab, last_name_tab
      LIMIT v_limit;

      EXIT WHEN instructor_id_tab.COUNT = 0;

      FOR i IN instructor_id_tab.FIRST..instructor_id_tab.LAST
      LOOP
         DBMS_OUTPUT.PUT_LINE ('instructor_id: '||instructor_id_tab(i));
         DBMS_OUTPUT.PUT_LINE ('first_name:    '||first_name_tab(i));
         DBMS_OUTPUT.PUT_LINE ('last_name:     '||last_name_tab(i));
      END LOOP;
   END LOOP;
   CLOSE instructor_cur;
END;
```

In this version of the script, you declare a cursor against the MY_INSTRUCTOR table. This enables you to do partial fetch from the MY_INSTRUCTOR table. You process this cursor by fetching 5 records at time via BULK COLLECT clause with the LIMIT option. It ensures that the collection variables contain no more than 5 records in them for each iteration of the cursor loop. Finally, in order to display all results, you move the FOR LOOP inside the cursor FOR LOOP. This version of the script produces output identical to the first version of the script.

6. Modify newly created script as follows: Instead of fetching data from MY_INSTRUCTOR table into individual collections fetch it into a single collection.

   Answer: In order to accomplish this task, the new record type must be declared so that a single collection type can be based on this record type. This is shown below. Changes are shown in bold.

   For Example *ch18_14d.sql*

```
DECLARE
   CURSOR instructor_cur IS
      SELECT instructor_id, first_name, last_name
        FROM my_instructor;

   -- Define record type
   TYPE instructor_rec IS RECORD
      (instructor_id my_instructor.instructor_id%TYPE,
       first_name    my_instructor.first_name%TYPE,
       last_name     my_instructor.last_name%TYPE);

   -- Define collection type and variable to be used by the BULK COLLECT clause
   TYPE instructor_type IS TABLE OF instructor_rec;

   instructor_tab instructor_type;

   v_limit PLS_INTEGER := 5;
```

```
BEGIN
   OPEN instructor_cur;
   LOOP
      -- Fetch partial instructor data at once via BULK COLLECT clause
      FETCH instructor_cur
       BULK COLLECT INTO instructor_tab
      LIMIT v_limit;

      EXIT WHEN instructor_tab.COUNT = 0;

      FOR i IN instructor_tab.FIRST..instructor_tab.LAST
      LOOP
         DBMS_OUTPUT.PUT_LINE ('instructor_id: '||instructor_tab(i).instructor_id);
         DBMS_OUTPUT.PUT_LINE ('first_name:    '||instructor_tab(i).first_name);
         DBMS_OUTPUT.PUT_LINE ('last_name:     '||instructor_tab(i).last_name);
      END LOOP;
   END LOOP;
   CLOSE instructor_cur;
END;
```

In this version of the script, you declare user-defined record type with three fields. Next, you declare a single collection type based on this record type. Then, you fetch the results of the cursor into to collection of records which you then display on the screen.

Next, consider another version that also creates collection of records. In this version, the collection type is based on the row type record returned by the cursor as shown:

For Example *ch18_14e.sql*

```
DECLARE
   CURSOR instructor_cur IS
      SELECT instructor_id, first_name, last_name
        FROM my_instructor;

   -- Define collection type and variable to be used by the BULK COLLECT clause
   TYPE instructor_type IS TABLE OF instructor_cur%ROWTYPE;

   instructor_tab instructor_type;

   v_limit PLS_INTEGER := 5;
BEGIN
   OPEN instructor_cur;
   LOOP
      -- Fetch partial instructor data at once via BULK COLLECT clause
      FETCH instructor_cur
       BULK COLLECT INTO instructor_tab
      LIMIT v_limit;

      EXIT WHEN instructor_tab.COUNT = 0;

      FOR i IN instructor_tab.FIRST..instructor_tab.LAST
      LOOP
         DBMS_OUTPUT.PUT_LINE ('instructor_id: '||instructor_tab(i).instructor_id);
         DBMS_OUTPUT.PUT_LINE ('first_name:    '||instructor_tab(i).first_name);
```

```
         DBMS_OUTPUT.PUT_LINE ('last_name:     '||instructor_tab(i).last_name);
      END LOOP;
   END LOOP;
   CLOSE instructor_cur;
END;
```

7. Create the following script: Delete records from MY_INSTRUCTOR table and display deleted records on the screen.

   Answer: The newly created script should look similar to the following script.

   For Example *ch18_15a.sql*

```
DECLARE
   -- Define collection types and variables to be used by the BULK COLLECT clause
   TYPE instructor_id_type IS TABLE OF my_instructor.instructor_id%TYPE;
   TYPE first_name_type    IS TABLE OF my_instructor.first_name%TYPE;
   TYPE last_name_type     IS TABLE OF my_instructor.last_name%TYPE;

   instructor_id_tab instructor_id_type;
   first_name_tab    first_name_type;
   last_name_tab     last_name_type;

BEGIN
   DELETE FROM MY_INSTRUCTOR
   RETURNING instructor_id, first_name, last_name
   BULK COLLECT INTO instructor_id_tab, first_name_tab, last_name_tab;

   DBMS_OUTPUT.PUT_LINE ('Deleted '||SQL%ROWCOUNT||' rows ');

   IF instructor_id_tab.COUNT > 0
   THEN
      FOR i IN instructor_id_tab.FIRST..instructor_id_tab.LAST
      LOOP
         DBMS_OUTPUT.PUT_LINE ('instructor_id: '||instructor_id_tab(i));
         DBMS_OUTPUT.PUT_LINE ('first_name:    '||first_name_tab(i));
         DBMS_OUTPUT.PUT_LINE ('last_name:     '||last_name_tab(i));
      END LOOP;
   END IF;
   COMMIT;
END;
```

   In this script, you store instructor ID, first and last names in the collections by using RETURNING option with the BULK COLLECT clause. When run, this script produces the following output:

```
Deleted 10 rows
instructor_id: 101
first_name:    Fernand
last_name:     Hanks
instructor_id: 102
first_name:    Tom
last_name:     Wojick
instructor_id: 103
```

```
first_name:    Nina
last_name:     Schorin
instructor_id: 104
first_name:    Gary
last_name:     Pertez
instructor_id: 105
first_name:    Anita
last_name:     Morris
instructor_id: 106
first_name:    Todd
last_name:     Smythe
instructor_id: 107
first_name:    Marilyn
last_name:     Frantzen
instructor_id: 108
first_name:    Charles
last_name:     Lowry
instructor_id: 109
first_name:    Rick
last_name:     Chow
instructor_id: 110
first_name:    Irene
last_name:     Willig
```

Try It Yourself

The projects in this section are meant to have you use 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.

Prior to beginning these exercise create `MY_SECTION` table based on the `SECTION` table. This table should be created empty as follows:

```
CREATE TABLE my_section AS
SELECT *
  FROM section
 WHERE 1 = 2;
```

1) Create the following script. Populate `MY_SECTION` table via the `FORALL` statement with `SAVE EXCEPTIONS` clause. Once `MY_SECTION` is populated, display how many records were inserted.

   Answer: The script should look similar to the following:

   For Example *ch18_16a.sql*

```
DECLARE
   -- Declare collection types
   TYPE number_type IS TABLE of NUMBER        INDEX BY PLS_INTEGER;
   TYPE string_type IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
   TYPE date_type   IS TABLE OF DATE          INDEX BY PLS_INTEGER;

   -- Declare collection variables to be used by the FORALL statement
   section_id_tab      number_type;
   course_no_tab       number_type;
```

```
   section_no_tab      number_type;
   start_date_time_tab date_type;
   location_tab        string_type;
   instructor_id_tab   number_type;
   capacity_tab        number_type;
   cr_by_tab           string_type;
   cr_date_tab         date_type;
   mod_by_tab          string_type;
   mod_date_tab        date_type;

   v_counter PLS_INTEGER := 0;
   v_total   INTEGER := 0;

   -- Define user-defined exception and associated Oracle error number with it
   errors EXCEPTION;
   PRAGMA EXCEPTION_INIT(errors, -24381);

BEGIN
   -- Populate individual collections
   SELECT *
     BULK COLLECT INTO section_id_tab, course_no_tab, section_no_tab
                      ,start_date_time_tab, location_tab, instructor_id_tab
                      ,capacity_tab, cr_by_tab, cr_date_tab, mod_by_tab
                      ,mod_date_tab
     FROM section;

   -- Populate MY_SECTION table
   FORALL i in 1..section_id_tab.COUNT SAVE EXCEPTIONS
      INSERT INTO my_section
         (section_id, course_no, section_no, start_date_time,
          location, instructor_id, capacity, created_by,
          created_date, modified_by, modified_date)
      VALUES
         (section_id_tab(i), course_no_tab(i), section_no_tab(i),
          start_date_time_tab(i), location_tab(i),
          instructor_id_tab(i), capacity_tab(i), cr_by_tab(i),
          cr_date_tab(i), mod_by_tab(i), mod_date_tab(i));
   COMMIT;

   -- Check how many records were added to MY_SECTION table
   SELECT COUNT(*)
     INTO v_total
     FROM my_section;

   DBMS_OUTPUT.PUT_LINE (v_total||' records were added to MY_SECTION table');

EXCEPTION
   WHEN errors
   THEN
      -- Display total number of exceptions encountered
      DBMS_OUTPUT.PUT_LINE
         ('There were '||SQL%BULK_EXCEPTIONS.COUNT||' exceptions');

      -- Display detailed exception information
```

```
      FOR i in 1.. SQL%BULK_EXCEPTIONS.COUNT
      LOOP
         DBMS_OUTPUT.PUT_LINE ('Record '||
            SQL%BULK_EXCEPTIONS(i).error_index||' caused error '||i||
            ': '||SQL%BULK_EXCEPTIONS(i).ERROR_CODE||' '||
            SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
      END LOOP;

      -- Commit records if any that were inserted successfully
      COMMIT;
END;
```

This script populates MY_SECTION table with records selected from the SECTION table. To enable use of the FORALL statement, it employs 11 collections. Note that there are only 3 collection types associated with these collections. This is because the individual collections store only 3 data types, NUMBER, VARCHAR2, and DATE.

The script uses cursor SELECT statement with the BULK COLLECT INTO clause to populate the individual collections, and then uses them with the FORALL statement with the SAVE EXCEPTIONS option to populate MY_SECTION table. To enable the SAVE EXCEPTIONS option, this script declares user-defined exception and associates Oracle error number with it. This script also contains exception-handling section where user-defined exception is processed. This section displays how many exceptions were encountered by the FORALL statement as well as detailed exception information. Note the COMMIT statement in the exception-handling section. This statement is added so that records that are inserted successfully by the FORALL statement are committed when the control of the execution is passed to the exception-handling section of the block.

When run, this script produces output as shown:

```
78 records were added to MY_SECTION table
```

2) Modify the script created in the previous exercise (step 1 above). In addition to displaying total number of records inserted in the MY_SECTION table, display how many records were inserted for each course. Use BULK COLLECT statement to accomplish this step.

Watch Out!

In order to get the correct results you should delete all rows from MY_SECTION table prior to executing this version of the script.

Answer: New version of the script should look similar to the following. All changes are shown in bold.

For Example *ch18_16b.sql*

```
DECLARE
   -- Declare collection types
   TYPE number_type IS TABLE of NUMBER        INDEX BY PLS_INTEGER;
   TYPE string_type IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;
   TYPE date_type   IS TABLE OF DATE          INDEX BY PLS_INTEGER;

   -- Declare collection variables to be used by the FORALL statement
   section_id_tab      number_type;
   course_no_tab       number_type;
   section_no_tab      number_type;
```

```
   start_date_time_tab date_type;
   location_tab        string_type;
   instructor_id_tab   number_type;
   capacity_tab        number_type;
   cr_by_tab           string_type;
   cr_date_tab         date_type;
   mod_by_tab          string_type;
   mod_date_tab        date_type;
   total_recs_tab      number_type;

   v_counter PLS_INTEGER := 0;
   v_total   INTEGER := 0;

   -- Define user-defined exception and associated Oracle error number with it
   errors EXCEPTION;
   PRAGMA EXCEPTION_INIT(errors, -24381);

BEGIN
   -- Populate individual collections
   SELECT *
     BULK COLLECT INTO section_id_tab, course_no_tab, section_no_tab
                      ,start_date_time_tab, location_tab, instructor_id_tab
                      ,capacity_tab, cr_by_tab, cr_date_tab, mod_by_tab
                      ,mod_date_tab
     FROM section;

   -- Populate MY_SECTION table
   FORALL i in 1..section_id_tab.COUNT SAVE EXCEPTIONS
      INSERT INTO my_section
         (section_id, course_no, section_no, start_date_time,
          location, instructor_id, capacity, created_by,
          created_date, modified_by, modified_date)
      VALUES
         (section_id_tab(i), course_no_tab(i), section_no_tab(i),
          start_date_time_tab(i), location_tab(i),
          instructor_id_tab(i), capacity_tab(i), cr_by_tab(i),
          cr_date_tab(i), mod_by_tab(i), mod_date_tab(i));
   COMMIT;

   -- Check how many records were added to MY_SECTION table
   SELECT COUNT(*)
     INTO v_total
     FROM my_section;

   DBMS_OUTPUT.PUT_LINE
      (v_total||' records were added to MY_SECTION table');

   -- Check how many records were inserted for each course
   -- and display this information
   -- Fetch data from MY_SECTION table via BULK COLLECT clause
   SELECT course_no, COUNT(*)
     BULK COLLECT INTO course_no_tab, total_recs_tab
     FROM my_section
   GROUP BY course_no;
```

```
   IF course_no_tab.COUNT > 0 THEN
      FOR i IN course_no_tab.FIRST..course_no_tab.LAST
      LOOP
         DBMS_OUTPUT.PUT_LINE
            ('course_no: '||course_no_tab(i)||
             ', total sections: '||total_recs_tab(i));
      END LOOP;
   END IF;

EXCEPTION
   WHEN errors
   THEN
      -- Display total number of exceptions encountered
      DBMS_OUTPUT.PUT_LINE
         ('There were '||SQL%BULK_EXCEPTIONS.COUNT||' exceptions');

      -- Display detailed exception information
      FOR i in 1.. SQL%BULK_EXCEPTIONS.COUNT
      LOOP
         DBMS_OUTPUT.PUT_LINE ('Record '||
            SQL%BULK_EXCEPTIONS(i).error_index||' caused error '||i||
            ': '||SQL%BULK_EXCEPTIONS(i).ERROR_CODE||' '||
            SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
      END LOOP;

      -- Commit records if any that were inserted successfully
      COMMIT;
END;
```

In this version of the script, you define one more collection `total_recs_tab` in the declaration portion of the PL/SQL block. This collection is used to store total number of sections for each course. In the executable portion of the PL/SQL block, you add a `SELECT` statement with `BULK COLLECT` clause that repopulates `course_no_tab` and initializes the `total_recs_tab`. Next, if the `course_no_tab` collection contains data, you display course numbers and total number of sections for each course on the screen.

When run, this version of the script produces output as follows:

```
78 records were added to MY_SECTION table
course_no: 25, total sections: 9
course_no: 310, total sections: 1
course_no: 100, total sections: 5
course_no: 147, total sections: 1
course_no: 330, total sections: 1
course_no: 450, total sections: 1
course_no: 134, total sections: 3
course_no: 144, total sections: 1
course_no: 120, total sections: 6
course_no: 20, total sections: 4
course_no: 210, total sections: 1
course_no: 220, total sections: 1
course_no: 132, total sections: 2
course_no: 230, total sections: 2
course_no: 240, total sections: 2
```

```
course_no: 350, total sections: 3
course_no: 135, total sections: 4
course_no: 125, total sections: 5
course_no: 130, total sections: 4
course_no: 140, total sections: 3
course_no: 146, total sections: 2
course_no: 420, total sections: 1
course_no: 142, total sections: 3
course_no: 145, total sections: 2
course_no: 124, total sections: 4
course_no: 10, total sections: 1
course_no: 204, total sections: 1
course_no: 122, total sections: 5
```

3) Create the following script. Delete all records from the `MY_SECTION` table and display how many records were deleted for each course. Use `BULK COLLECT` with the `RETURNING` option.

Answer: This script should look similar to the following:

For Example *ch18_17a.sql*

```
DECLARE
   -- Define collection types and variables to be used by the BULK COLLECT clause
   TYPE section_id_type IS TABLE OF my_section.section_id%TYPE;

   section_id_tab section_id_type;

BEGIN
   FOR rec IN (SELECT UNIQUE course_no
                FROM my_section)
   LOOP
      DELETE FROM MY_SECTION
       WHERE course_no = rec.course_no
      RETURNING section_id
      BULK COLLECT INTO section_id_tab;

      DBMS_OUTPUT.PUT_LINE ('Deleted '||SQL%ROWCOUNT||
         ' rows for course '||rec.course_no);

      IF section_id_tab.COUNT > 0
      THEN
         FOR i IN section_id_tab.FIRST..section_id_tab.LAST
         LOOP
            DBMS_OUTPUT.PUT_LINE ('section_id: '||section_id_tab(i));
         END LOOP;
         DBMS_OUTPUT.PUT_LINE ('================================');
      END IF;
      COMMIT;
   END LOOP;
END;
```

In this script you declare a single collection `section_id_tab`. Note that there is no need to declare a collection to store course numbers. This is because the records from `MY_SECTION` table are deleted for each course number instead of all at once. To accomplish this, you introduce cursor `FOR LOOP` that selects unique course numbers from `MY_SECTION` table. Next, for each course number you delete records from `MY_SECTION` table returning corresponding section IDs and collecting them in the `section_id_tab`. Next, you display how many records were deleted for a given course number along with individual section IDs for this course.

Note that even though the collection `section_id_tab` is repopulated for each iteration of the cursor loop, there is no need to reinitialize it (in other words empty it out). This is because the `DELETE` statement does it implicitly.

Consider the partial output produced by this script:

```
Deleted 9 rows for course 25
section_id: 85
section_id: 86
section_id: 87
section_id: 88
section_id: 89
section_id: 90
section_id: 91
section_id: 92
section_id: 93
================================
Deleted 1 rows for course 310
section_id: 103
================================
Deleted 5 rows for course 100
section_id: 141
section_id: 142
section_id: 143
section_id: 144
section_id: 145
================================
…
```