Oracle Academy
Introduction to Database Programming with PL/SQL
Instructor Resource Guide

INSTRUCTOR NOTES FOR SLIDES

General Note On Autocommit in Application Express:
The default setting for the APEX SQL command processor is that AUTOCOMMIT is turned on. When AUTOCOMMIT is turned off, DML commands produce permanent changes only when the COMMIT command is issued either within a PL/SQL block or by itself.

If the user logs off normally or closes the browser window before COMMITting the changes, the changes will be rolled back. This behavior is different from SQL*Plus and iSQL*Plus.

This is important especially for Section 9 Lesson 4 Practice, "Persistent State of Package Variables" where students are asked to turn off AUTOCOMMIT. However, a correct solution to this practice will not contain any DML commands.

SECTION 3 LESSON 1 – Review of SQL DML

Slide 1: Review of SQL DML
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me –Data Manipulation Language (DML)
These DML operations are very special. They allow changes to the tables by adding and updating rows. Programmers must take care to know what their programs are trying to accomplish, as any of the DML commands issued will change the data in the affected tables.

Slide 5: Tell Me / Show Me – INSERT
No instructor notes for this slide

Slide 6: Tell Me / Show Me – INSERT (continued)
No instructor notes for this slide

Slide 7: Tell Me / Show Me – INSERT (continued)
Although this is possible, it is not recommended. The ART table has four columns. What if the DBA later adds a fifth column using a DDL ALTER TABLE statement? The INSERT statement above would not work anymore.

Slide 8: Tell Me / Show Me – UPDATE
Remind students that:

- if the WHERE clause is omitted, all rows will be modified
- More than one column can be modified. Example:

```
  UPDATE art
     SET description = 'marble sculpture',
              artist = 'Michaelangelo'
     WHERE ….
```

Slide 9: Tell Me / Show Me – DELETE
Remind students that all rows in the table are deleted if you omit the WHERE clause.

Slide 10: Tell Me / Show Me – MERGE
Merge allows two functions, INSERT and UPDATE to be completed when needed. Once again, make sure the coding is written correctly so that the data integrity of the tables is maintained.

1. Ask the students to identify the differences in the table structures between the two tables.
   Answer: The ITEMS table has a column named ITEM_ID, whereas the ART table has a column named ID.

2. Ask the students to identify any duplicate data (duplicate IDs).
   Answer: The Leonardo da Vinci piece “Mona Lisa” is found in both tables, but the descriptions are different.

3. Ask the students as to how to handle the duplicate data.
   One answer: Update the ART table with the description in the ITEMS table.

Slide 11: Tell Me / Show Me – MERGE
No instructor notes for this slide

Slide 12: Tell Me / Show Me – Terminology
Data Manipulation Language (DML) -- When you change data in an object (for example, by inserting or deleting rows).
Data Definition Language (DDL) -- When you create, change, or delete an object in a database.
INSERT – Statement used to add new rows to a table.
UPDATE -- Statement used to modify existing rows in a table.
DELETE – Statement used to remove existing rows in a table.
MERGE – Statement used to INSERT and/or UPDATE a target table, based on matching values in a source table.

Slide 13: Summary
No instructor notes for this slide

Slide 14: Try It / Solve It
No instructor notes for this slide

SECTION 3 LESSON 2 - Retrieving Data in PL/SQL

Slide 1: Retrieving Data in PL/SQL
No instructor notes for this slide

Slide 2: What Will I Learn?
No instructor notes for this slide

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – SQL Statements in PL//SQL
PL/SQL is tightly integrated with SQL and therefore the database. As a quick reminder, SELECT returns data from one or more tables, INSERT creates new rows in a table, UPDATE changes existing data in a table, DELETE removes an entire row from a table and MERGE in a combination of INSERT and UPDATE. Any changes done by one or more statements can be saved in the Database by a COMMIT statement, whilst statements can be undone by a ROLLBACK statement. SAVEPOINTS can be created in between DML statements to allow a user to ROLLBACK to a particular SAVEPOINT, rather than ROLLING back all transactions since the last issued COMMIT or ROLLBACK.

Slide 5: Tell Me / Show Me – SQL Statements in PL/SQL (continued)
These statements cannot be run between a BEGIN and END of PL/SQL code unless "wrapped". These statements make a major change to the database. We will learn about how to execute DDL and DCL statements within PL/SQL in a later lesson.

Consider the following example:

```
BEGIN
  CREATE TABLE my_emp_table
  AS SELECT * FROM employees;
END;
```

The above example tries to use a DDL statement directly in the block. When you execute the block, you will see the following error:

```
ORA-06550: line 2, column 3:
PLS-00103: Encountered the symbol "CREATE" when expecting one of the following:
……
```

The indirect way of working with DDL statements is to use Dynamic SQL with the EXECUTE IMMEDIATE statement. This is explained in Section 9.

Slide 6: Tell Me / Show Me – SELECT Statements in PL/SQL
Use the SELECT statement to retrieve data from the database. In the syntax:

- *select_list* Is a list of at least one column and can include SQL expressions, row functions, or group functions
- *variable_name* Is a scalar variable that holds a retrieved value
- *record_name* Is the PL/SQL record that holds the retrieved values

- *table* Specifies the database table name
- *condition* Is composed of column names, expressions, constants, and comparison operators, including PL/SQL variables and constants

Slide 7: Tell Me / Show Me – SELECT Statements in PL/SQL (continued)
Remember the SELECT statement is used to retrieve data. The second clause INTO matches variables with the data retrieved.

A SELECT..INTO retrieves a single row of data into the variables. SELECT statements within a PL/SQL block fall into the ANSI classification of embedded SQL, for which the following rule applies: queries must return exactly one row. A query that returns more than one row or no rows generates an error.

Slide 8: Tell Me / Show Me – Retrieving Data in PL/SQL
In the example in the slide, the v_emp_hiredate and v_emp_salary variables are declared in the declarative section of the PL/SQL block. In the executable section, the values of the columns hire_date and salary for the employee with the employee_id 100 is retrieved from the employees table and stored in the v_emp_hiredate and v_emp_salary variables, respectively. Observe how the INTO clause, along with the SELECT statement, retrieves the database column values into the PL/SQL variables.

Note: The SELECT statement is retrieving hire_date and then salary and therefore the variables in the INTO clause also must be in the same order. For example, if you interchange v_emp_hiredate and v_emp_salary in the above statement, the statement will result in an error.

Slide 9: Tell Me / Show Me – Retrieving Data in PL/SQL
A SELECT statement with the INTO clause can retrieve only one row at a time. If your requirement is to retrieve multiple rows and operate on the data, then you can make use of explicit cursors. Students will learn about cursors in Section 5.

If students are curious about exception handling or cursors, or both, below are two sample programs involving these topics.
Exception Handling

```
DECLARE
   v_salary employees.salary%TYPE;
BEGIN
   SELECT salary INTO v_salary
     FROM employees;
   DBMS_OUTPUT.PUT_LINE(' Salary is : ' || v_salary);
EXCEPTION
   WHEN TOO_MANY_ROWS THEN
       DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
```

Cursors

```
DECLARE
   CURSOR c1 IS
      SELECT first_name, last_name, salary
         FROM employees
         WHERE salary>10000;
BEGIN
   FOR v_sal IN c1 LOOP
      DBMS_OUTPUT.PUT_LINE(v_sal.first_name || ' ' || v_sal.last_name || '''s salary is: 
' || v_sal.salary);
  END LOOP;
END;
```

Slide 10: Tell Me / Show Me – Retrieving Data in PL/SQL (continued)
In the example in the slide, the v_sum_sal and v_deptno variables are declared in the declarative section of the PL/SQL block. In the executable section, the total salary for the employees in the department with the department_id 60 is computed using the SQL aggregate function SUM. The calculated total salary is assigned to the v_sum_sal variable.

Point out that the SELECT statement in the slide will always return exactly one row, even if there are many (or no) employees in department 60. The "must return exactly one row" rule states that the **output** from the SELECT must be exactly one row.

Note: Group functions cannot be used in PL/SQL syntax. They are used in SQL statements within a PL/SQL block as shown in the example. You cannot use them as follows:

```
v_sum_sal := SUM(employees.salary);
```

Slide 11: Tell Me / Show Me – Guidelines for Retrieving Data in PL/SQL
Bullet point 3: the WHERE clause is not needed if the table can contain only one row, for example the DUAL table.

Slide 12: Tell Me / Show Me – Guidelines for Naming Conventions
The example shown in the slide is defined as follows: Retrieve the hire date from the employees table for employee_id 176. This example raises an unhandled run-time exception because in the WHERE clause, the PL/SQL variable name EMPLOYEE_ID is the same as that of the database column name in the employees table. Explain that for any table and any column, the condition "WHERE column_name = column_name" is TRUE for all rows with a non-null value in the column.

If students struggle with this, demonstrate the following SELECT statement directly, ie not in a PL/SQL block:

```
SELECT hire_date FROM employees
  WHERE employee_id = employee_id;
```

Slide 13: Tell Me / Show Me – Guidelines for Naming Conventions (continued)
To test this, run the following statements:

```
CREATE TABLE emp_dup AS SELECT * from employees;

SELECT first_name, last_name FROM emp_dup;

DECLARE
   last_name VARCHAR2(25) := 'King';
BEGIN
   DELETE FROM emp_dup WHERE last_name = last_name;
END;

SELECT first_name, last_name FROM emp_dup;
```

Answer
The DELETE statement removes all employees from the employees table as opposed to just the records that contain the last name “King”.This is because the Oracle server cannot differentiate between the database column last_name and the declared variable last_name.

Slide 14: Tell Me / Show Me – Guidelines for Naming Conventions (continued)
Avoid ambiguity in the WHERE clause by using a naming convention that distinguishes database column names from PL/SQL variable names.

- Database columns and identifiers should have distinct names (for example, ensure all variable names begin with v_).
- Syntax errors can arise because PL/SQL checks the database first for a column in the table.

Note: There is no possibility for ambiguity in the SELECT clause because any identifier in the SELECT clause must be a database column name. There is no possibility for ambiguity in the INTO clause because identifiers in the INTO clause must be PL/SQL variables. There is the possibility of a confusion only in the WHERE clause.

Slide 15: Summary
No instructor notes for this slide

Slide 16: Try It / Solve It
No instructor notes for this slide

SECTION 3 LESSON 3 - Manipulating Data in PL/SQL

<u>Slide 1: Manipulating Data in PL/SQL</u>
No instructor notes for this slide

<u>Slide 2: What Will I Learn?</u>
Starting in this lesson, we will be using DML statements within PL/SQL blocks to modify the contents of tables. It is very important that students do NOT modify their existing tables (such as EMPLOYEES and DEPARTMENTS), because they will be needed later in the course.

Therefore all the code examples in these lessons, and the practice exercises, use copies of these tables. The examples in this lesson use the COPY_EMP table.

If students have not already created the COPY_EMP table, ask them to execute this SQL statement:

```
CREATE TABLE copy_emp
   AS SELECT *
   FROM employees;
```

<u>Slide 3: Why Learn It?</u>
No instructor notes for this slide

<u>Slide 4: Tell Me / Show Me – Manipulating Data Using PL/SQL</u>
Point out that a DML statement within a PL/SQL block can modify many rows (unlike a SELECT statement, which must read exactly one row).

<u>Slide 5: Tell Me / Show Me – Manipulating Data Using PL/SQL (continued)</u>
No instructor notes for this slide

<u>Slide 6: Tell Me / Show Me – Inserting Data</u>
Important Note: The data in the EMPLOYEES table needs to remain unchanged for later in the course. Therefore we use the COPY_EMP table in all these DML examples.

Inserting Data
In the example in the slide, an INSERT statement is used within a PL/SQL block to insert a record into the COPY_EMP table. While using the INSERT command in a PL/SQL block, you can:

- Use SQL functions, such as USER and SYSDATE
- Generate primary key values by using existing database sequences
- Derive values in the PL/SQL block.

Slide 7: Tell Me / Show Me – Updating Data
There may be ambiguity in the SET clause of the UPDATE statement because although the identifier on the left of the assignment operator is always a database column, the identifier on the right can be either a database column or a PL/SQL variable. Recall that if column names and identifier names are identical in the WHERE clause, then the Oracle server looks to the database first for the name.

Remember that the WHERE clause is used to determine which rows are affected. If no rows are modified, then no error occurs (unlike the SELECT statement in PL/SQL).

Note: PL/SQL variable assignments always use :=, and SQL column assignments always use =.

Slide 8: Tell Me / Show Me – Deleting Data
The DELETE statement removes unwanted rows from a table. If the WHERE clause is not used, then all the rows in a table will be removed, provided that no integrity constraints are violated.

Instructor Note
Refer to the code file for more information and queries illustrating integrity constraints.

Slide 9: Tell Me / Show Me – Merging Rows
The MERGE statement inserts or updates rows in one table by using data from another table. Each row is inserted or updated in the target table, depending on an equijoin condition.

The example shown matches the employee_id in the COPY_EMP table to the employee_id in the EMPLOYEES table. If a match is found, then the row is updated to match the row in the EMPLOYEES table. If the row is not found, then it is inserted into the copy_emp table.

Slide 10: Tell Me / Show Me – Getting information from a Cursor
No instructor notes for this slide

Slide 11: Tell Me / Show Me – What is a Cursor?
The word “cursor” has several meanings in Oracle. It is sometimes used to mean a pointer to the private memory area, rather than the memory area itself. It is also used to refer to an area of shared memory. In thıs course, we focus only on its meaning in the PL/SQL environment.

Slide 12: Tell Me / Show Me – Implicit and Explicit Cursors
Implicit cursors: Implicit cursors are created and managed by the Oracle server. Oracle uses an implicit cursor for each SELECT, UPDATE, DELETE, or INSERT statement you execute in a program. You do not have access to them. The Oracle server creates such a cursor when it has to execute a SQL statement. The remaining pages in this lesson discuss implicit cursors.

Explicit cursors: Explicit cursors are declared by the programmer. As a programmer you may want to retrieve multiple rows from a database table, have a pointer to each row that is retrieved, and work on the rows one at a time. In such cases, you can declare cursors explicitly depending on your business requirements. You declare these cursors in the declarative section of a PL/SQL block. Explicit cursors are discussed in Section 5.

<u>Slide 13: Tell Me / Show Me – Cursor Attributes for Implicit Cursors</u>
You can test the attributes — SQL%ROWCOUNT, SQL%FOUND, and SQL%NOTFOUND — in the executable section of a block to gather information after the appropriate command. PL/SQL does not return an error if a DML statement does not affect any rows in the underlying table. However, if a SELECT statement does not retrieve any rows, PL/SQL returns an exception.

Observe that the attributes are prefixed with the automatic name of the implicit cursor: “SQL”.

The SQL%NOTFOUND attribute is opposite to SQL%FOUND. This attribute may be used as the exit condition in a loop. It is useful in UPDATE or DELETE statements when no rows are changed because exceptions are not returned in these cases.

You will learn about explicit cursor attributes later in the course.

<u>Slide 14: Tell Me / Show Me – Using Implicit Cursor Attributes: Example 1</u>
The example in the slide deletes all rows with department_id 50 from the copy_emp table. Using the SQL%ROWCOUNT attribute, you can display the number of rows deleted.

<u>Slide 15: Tell Me / Show Me – Using Implicit Cursor Attributes: Example 2</u>
No instructor notes for this slide

<u>Slide 16: Tell Me / Show Me – Using Implicit Cursor Attributes: Good Practice Guidelines</u>
Since every embedded SQL statement creates an implicit cursor, and all implicit cursors are called “SQL”, each SQL statement in turn will modify the value in SQL%ROWCOUNT as it executes. Therefore the SQL%ROWCOUNT value returned from the UPDATE statement will be overwritten by the INSERT statement.

Therefore, to use an implicit cursor attribute in a later SQL statement, it is good practice to save the value in an explicitly declared variable before it can be automatically overwritten. Better code in the slide would be:

```
DECLARE
   v_rowcount    INTEGER;
BEGIN
   UPDATE    copy_emp
      SET        salary = salary + 100
      WHERE  job_id = 'ST_CLERK';
   v_rowcount := SQL%ROWCOUNT;
     /* This is not a SQL statement and therefore
      it will not alter SQL%ROWCOUNT */
   INSERT INTO results (num_rows)
     VALUES (v_rowcount);
END;
```

Slide 17: Tell Me / Show Me – Terminology
INSERT - Statement adds new rows to the table.
UPDATE - Statement modifies existing rows in the table.
DELETE - Statement removes rows from the table.
MERGE - Statement selects rows from one table to update and/or insert into another table. The decision whether to update or insert into the target table is based on a condition in the ON clause.

Implicit cursors: Implicit cursors are created and managed by the Oracle server. Oracle uses an implicit cursor for each SELECT, UPDATE, DELETE, or INSERT statement you execute in a program. You do not have access to them. The Oracle server creates such a cursor when it has to execute a SQL statement.

Explicit cursors: Explicit cursors are declared by the programmer. As a programmer you may want to retrieve multiple rows from a database table, have a pointer to each row that is retrieved, and work on the rows one at a time. In such cases, you can declare cursors explicitly depending on your business requirements. You declare these cursors in the declarative section of a PL/SQL block. Explicit cursors are discussed later in the course.

Slide 18: Summary
No instructor notes for this slide

Slide 19: Try It / Solve It
No instructor notes for this slide

SECTION 3 LESSON 4 - Using Transaction Control Statements

Slide 1: Using Transaction Control Statements
No instructor notes for this slide

Slide 2: What Will I Learn?
In **iSQLPlus** and most other Oracle development tools, the changes made by SQL DML statements are not committed until a COMMIT statement is explicitly executed. However in **Application Express**, DML statements are automatically committed by default. You can change this behavior in SQL Workshop -> SQL Commands by un-checking the "Autocommit" check box.

Slide 3: Why Learn It?
No instructor notes for this slide

Slide 4: Tell Me / Show Me – Database Transaction
No instructor notes for this slide

Slide 5: Tell Me / Show Me – Example of a Transaction
No instructor notes for this slide

Slide 6: Tell Me / Show Me – Example of a Transaction (continued)
No instructor notes for this slide

Slide 7: Tell Me / Show Me – Example of a Transaction (continued)
No instructor notes for this slide

Slide 8: Tell Me / Show Me – Example of a Transaction (continued)
No instructor notes for this slide

Slide 9: Tell Me / Show Me – Transaction Control Statements
Remind students to turn off the **Autocommit** check box in Application Express, otherwise each DML statement will immediately be autocommitted and they will not be able to test the ROLLBACK command.

Slide 10: Tell Me / Show Me – COMMIT
The note at the end of the slide means that transactions are not automatically committed when a PL/SQL block terminates. This means that (for example) when using nested blocks, a transaction which is started in an inner block can later be committed in an outer block.

Slide 11: Tell Me / Show Me – ROLLBACK
Rollback and savepoint are fantastic commands. They are used to restore the data to its original values, when and if, the transaction does not accomplish the goal.

Students may notice that this code example looks pointless: what is the purpose of coding and executing the first INSERT statement if we are going to reverse out the change?

In real life, we usually ROLLBACK if an exception occurs, or as a result of a conditional test (IF ….. ELSE …..).

Slide 12: Tell Me / Show Me – SAVEPOINT
SAVEPOINTs are useful in application programs. If a procedure contains several functions, then you can create a SAVEPOINT before each function begins. Then, if a function fails, it is easy to return the data to its state before the function began and re-run the function with revised parameters or perform a recovery action.

Slide 13: Tell Me / Show Me – Terminology
Transaction -- An inseparable list of database operations, which must be executed either in its entirety or not at all.
COMMIT – Statement used to make database changes permanent.
END – Keyword used to signal the end of a PL/SQL block, not the end of a transaction.
ROLLBACK – Used for discarding any changes that were made to the database after the last COMMIT.
SAVEPOINT – Used to mark an intermediate point in transaction processing.

Slide 14: Summary
No instructor notes for this slide

Slide 15: Try It / Solve It
No instructor notes for this slide

PRACTICE SOLUTIONS

SECTION 3 LESSON 1 - Review of SQL DML

Terminology

1. **_DELETE_________________** Statement used to remove existing rows in a table.
2. **_INSERT__________________** Statement used to add new rows to a table.
3. **_MERGE__________________** Statement used to INSERT and/or UPDATE a target table, based on matching values in a source table.
4. **_UPDATE__________________** Statement used to modify existing rows in a table.
5. **_DDL______________________** When you create, change, or delete an object in a database.
6. **__DML________________** When you change data in an object (for example, by inserting or deleting rows).

Try It/Solve It

1. DELETE FROM students;

   This SQL statement will:
   A. Not execute due to wrong syntax
   B. Delete the first row from STUDENTS
   C. Delete all rows from STUDENTS
   D. None of the above

2. INSERT INTO STUDENTS (id, last_name, first_name)
   VALUES (29,'Perez','Jessica');

   This SQL statement:
   A. Does an explicit insert
   B. Does an implicit insert

Use the following table for questions 3 through 8.

grocery_items		
product_id	**brand**	**description**
110	Colgate	Toothpaste
111	Ivory	Soap
112	Heinz	Ketchup

3. Write a SQL statement to create the above table.

```
CREATE TABLE grocery_items
  (product_id  NUMBER(3) PRIMARY KEY,
   brand       VARCHAR2(20),
   description VARCHAR2(20));
```

4. Write and execute three SQL statements to explicitly add the above data to the table.

```
INSERT INTO grocery_items (product_id, brand, description)
  VALUES (110,'Colgate','Toothpaste');

INSERT INTO grocery_items (product_id, brand, description)
  VALUES (111,'Ivory','Soap');

INSERT INTO grocery_items (product_id, brand, description)
  VALUES (112,'Heinz','Ketchup');
```

5. Write and execute a SQL statement that will explicitly add your favorite beverage to the table.

```
INSERT INTO grocery_items (product_id, brand, description)
  VALUES (to be provided by student);
```

Example: INSERT INTO grocery_items (product_id, brand, description) VALUES (199,'Coke','Soda');

6. Write and execute a SQL statement that modifies the description for Heinz ketchup to “tomato catsup”.

```
UPDATE grocery_items
  SET description  = 'tomato catsup'
  WHERE product_id = 112;
```

7. Write and execute a SQL statement that will implicitly add your favorite candy to the table.

```
INSERT INTO grocery_items
  VALUES (to be provided by student);
```

Example: INSERT INTO grocery_items VALUES (200,'Mars','Candy');

Note: the values must match the order of columns in the database

8. Write and execute a SQL statement that changes the soap brand from "Ivory" to "Dove."

```
UPDATE grocery_items
  SET brand  = 'Dove'
  WHERE product_id = 111;
```

Use the following table for questions 9 through 14.

new_items		
product_id	**brand**	**description**
110	Colgate	Dental paste
175	Dew	Soda
275	Palmolive	Dish detergent

9. Write and execute SQL statements to create the new_items table and populate it with the data in the table.

```
CREATE TABLE new_items
  (product_id   NUMBER(3) PRIMARY KEY,
   brand        VARCHAR2(20),
   description  VARCHAR2(20));

INSERT INTO new_items (product_id, brand, description)
  VALUES (110,'Colgate','Dental paste');

INSERT INTO new_items (product_id, brand, description)
  VALUES (175,'Dew','Soda');

INSERT INTO new_items (product_id, brand, description)
  VALUES (275,'Palmolive','Dish detergent');
```

10. Write a SQL statement that will update the grocery_items table with the brand and description from the new_items table when the product ID values match. If they don't match, add a new row to the grocery_items table. DO NOT EXECUTE YOUR STATEMENT YET.

```
MERGE INTO grocery_items g
  USING new_items i
    ON (g.product_id = i.product_id)
  WHEN MATCHED
    THEN UPDATE SET
      g.brand   = i.brand,
      g.description   = i.description
  WHEN NOT MATCHED
    THEN INSERT
      VALUES(i.product_id, i.brand, i.description);
```

11. How many rows will be updated by the SQL statement in question 10?

1 row will be updated: product_id 110

12. How many rows will be inserted by the SQL statement in question 10?

2 rows will be inserted: product_ids 175 and 275

13. Which of the following is true about the SQL statement in question 10?

   A. **new_items is the source table and grocery_items is the target table.**
   B. grocery_items is the source table and new_items is the target table.

14. Execute the SQL statement you wrote in question 10, then SELECT all data from the target table to verify your answers to questions 11 and 12.

```
SELECT *
FROM grocery_items;
```

This should show that two rows (175 and 275) have been inserted, while row 110 has been updated.

SECTION 3 LESSON 2 - Retrieving Data in PL/SQL

Terminology
No new vocabulary for this lesson.

Try It/Solve It
1. State whether each of the following SQL statements can be included directly in a PL/SQL block.

Statement	**Valid in PL/SQL**	**Not Valid in PL/SQL**
ALTER USER SET password='oracle';		**X**
CREATE TABLE test (a NUMBER);		**X**
DROP TABLE test;		**X**
SELECT emp_id INTO v_id FROM employees;	**X**	
GRANT SELECT ON employees TO PUBLIC;		**X**
INSERT INTO grocery_items (product_id, brand, description) VALUES (199,'Coke','Soda');	**X**	
REVOKE UPDATE ON employees FROM PUBLIC;		**X**
ALTER TABLE employees RENAME COLUMN employee_id TO emp_id;		**X**
DELETE FROM grocery_items WHERE description='Soap';	**X**	

2. Create a PL/SQL block that selects the maximum department_id in the departments table and stores it in the v_max_deptno variable. Display the maximum department_id.

   A. Declare a variable v_max_deptno of the same datatype as the department_id column.

   B. Start the executable section with the keyword BEGIN and include a SELECT statement to retrieve the highest department_id from the departments table.

   C. Display the variable v_max_deptno and end the executable block.

   D. Execute your block.

```
DECLARE
  v_max_deptno   departments.department_id%TYPE;
BEGIN
  SELECT MAX(department_id) INTO v_max_deptno
    FROM departments;
  DBMS_OUTPUT.PUT_LINE('The maximum department_id is : ' ||
                      v_max_deptno);
END;
```

3. The following code is supposed to display the lowest and highest elevations for a country name entered by the user. However, the code does not work. Fix the code by following the guidelines for retrieving data that you learned in this lesson.

```
DECLARE
  v_country_name      wf_countries.country_name%TYPE
                      := 'United States of America';
  v_lowest_elevation  wf_countries.lowest_elevation%TYPE;
  v_highest_elevation wf_countries.highest_elevation%TYPE;
BEGIN
  SELECT lowest_elevation, highest_elevation
    FROM wf_countries;
  DBMS_OUTPUT.PUT_LINE('The lowest elevation in
          '||v_country_name||' is '||v_lowest_elevation
          ||' and the highest elevation is '||
          v_highest_elevation||'.');
END;
```

```
DECLARE
  v_country_name      wf_countries.country_name%TYPE
                        := 'United States of America';
  v_lowest_elevation  wf_countries.lowest_elevation%TYPE;
  v_highest_elevation wf_countries.highest_elevation%TYPE;
BEGIN
   /* The SELECT statement must have an INTO clause */
 SELECT lowest_elevation, highest_elevation
   INTO v_lowest_elevation, v_highest_elevation
   FROM wf_countries
      /*only one row can be returned */
    WHERE country_name = v_country_name;
DBMS_OUTPUT.PUT_LINE('The lowest elevation in
           '||v_country_name||' is '||v_lowest_elevation
           ||' and the highest elevation is '||
           v_highest_elevation||'.');
END;
```

4. Enter and run the following anonymous block, observing that it executes successfully.

```
DECLARE
  v_emp_lname    employees.last_name%TYPE;
  v_emp_salary   employees.salary%TYPE;
BEGIN
  SELECT last_name, salary INTO v_emp_lname, v_emp_salary
    FROM employees
    WHERE job_id = 'AD_PRES';
  DBMS_OUTPUT.PUT_LINE(v_emp_lname||' '||v_emp_salary);
END;
```

A. Now modify the block to use ‘IT_PROG’ instead of ‘AD_PRES’ and re-run it. Why does it fail this time?

A SELECT statement in a PL/SQL block must return exactly one row. There is more than one IT_PROG in the table.

B. Now modify the block to use ‘IT_PRAG’ instead of ‘IT_PROG’ and re-run it. Why does it still fail?

There is no IT_PRAG in the table.

5. Use the following code to answer this question:

```
DECLARE
  last_name VARCHAR2(25) := 'Fay';
BEGIN
  UPDATE emp_dup SET first_name = 'Jennifer'
    WHERE last_name = last_name;
END;
```

What do you think would happen if you ran the above code? Write your answer here and then follow the steps below to test your theory.

The variable last_name has the same name as the database column. Therefore every row in the table would be updated, because table column names take precedence over PL/SQL variable names.

A. Create a table called emp_dup that is a duplicate of employees.

```
CREATE TABLE emp_dup
  AS SELECT * from employees;
```

B. Select the first_name and last_name values for all rows in emp_dup.

```
SELECT first_name, last_name from emp_dup;
```

C. Run the above anonymous PLSQL block.

D. Select the first_name and last_name columns from emp_dup again to confirm your theory.

```
SELECT first_name, last_name from emp_dup;
```

All of the employees have the first name, Jennifer.

E. Now we are going to correct the code so that it changes only the first name for the employee whose last name is “Chen”. Drop emp_dup and re-create it.

```
DROP TABLE emp_dup;

CREATE TABLE emp_dup
  AS SELECT * from employees;
```

F. Modify the code so that for the employee whose last_name=”Fay”, the first_name is updated to Jennifer. Run your modified block.

```
DECLARE
  v_last_name VARCHAR2(25) := 'Fay';
BEGIN
  update emp_dup SET first_name = 'Jennifer'
    WHERE last_name = v_last_name;
END;
```

G. Confirm that your update statement worked correctly.

```
SELECT first_name, last_name from emp_dup;
```

Extension Exercise

1. Is it possible to name a column in a table the same name as the table? Create a table to test this question. Don't forget to populate the table with data.

Students' answers will vary, but YES it is possible (although not recommended).

```
CREATE TABLE test (test NUMBER(1));

INSERT INTO test values (1);
INSERT INTO test values (2);
INSERT INTO test values (3);
```

2. Is it possible to have a column, table, and variable, all with the same name? Using the table you created in the question above, write a PL/SQL block to test your theory.

Students' code will vary. If you use the %TYPE statement, then it is not possible to have a variable match the names of the column and table. But without %TYPE, yes it is possible (although again not recommended).

```
-- the following fails
DECLARE
  test    test.test%TYPE;
BEGIN
  SELECT test INTO test
    FROM test
    WHERE test=1;
END;

-- the following succeeds
DECLARE
  test    NUMBER(1);
BEGIN
  SELECT test INTO test
    FROM test
    WHERE test=1;
END;
```

SECTION 3 LESSON 3 - Manipulating Data in PL/SQL

Terminology

1. **__Implicit cursors_________** Defined automatically by Oracle for all SQL data manipulation statements, and for queries that return only one row.
2. **__Explicit cursors___________** Defined by the programmer for queries that return more than one row.
3. **___MERGE________________** Statement selects rows from one table to update and/or insert into another table. The decision whether to update or insert into the target table is based on a condition in the ON clause.
4. **___INSERT_______________** Statement adds new rows to the table.
5. **___DELETE_______________** Statement removes rows from the table.
6. **___UPDATE_______________** Statement modifies existing rows in the table.

Try It/Solve It

1. True or False: When you use DML in a PL/SQL block, Oracle uses explicit cursors to track the data changes.

Answer: False

2. **_Explicit_____________** cursors are created by the programmer.

3. **_Implicit_____________** cursors are created by the Oracle server.

4. SQL%FOUND, SQL%NOTFOUND, and SQL%ROWCOUNT are **_CURSOR ATTRIBUTES__**, and are available when you use **___implicit____________** cursors.

The following questions use a copy of the departments table. Execute the following SQL statement to create the copy table.

```
CREATE TABLE new_depts AS SELECT * FROM departments;
```

5. Examine and run the following PL/SQL code, which obtains and displays the maximum department_id from new_depts.

```
DECLARE
  v_max_deptno    new_depts.department_id%TYPE;
BEGIN
  SELECT MAX(department_id) INTO v_max_deptno
    FROM new_depts;
  DBMS_OUTPUT.PUT_LINE('The maximum department id is: '||
                       v_max_deptno);
END;
```

The maximum department id is: 190

6. Modify the code to declare two additional variables, (assigning a new department name to one of them) by adding the following two lines to your Declaration section:

```
v_dept_name   new_depts.department_name%TYPE
                 := 'A New Department' ;
v_dept_id     new_depts.department_id%TYPE;
```

7. Modify the code to add 10 to the current maximum department number and assign the result to v_dept_id.

v_dept_id := v_max_deptno+10;   -- AFTER the SELECT statement !

8. Modify the code to include an INSERT statement to insert a new row into the new_depts table, using v_dept_id and v_dept_name to populate the department_id and department_name columns. Insert NULL into the location_id and manager_id columns. Save your code.

By now the block should look like this:

```
DECLARE
  v_max_deptno        new_depts.department_id%TYPE;
  v_dept_name   new_depts.department_name%TYPE
                   := 'A New Department' ;
  v_dept_id     new_depts.department_id%TYPE;
BEGIN
  SELECT MAX(department_id) INTO v_max_deptno
    FROM new_depts;
  v_dept_id := v_max_deptno + 10;
  INSERT INTO new_depts  (department_id, department_name)
    VALUES (v_dept_id, v_dept_name);
  DBMS_OUTPUT.PUT_LINE('The maximum department id is: '||
                   v_max_deptno);
END;
```

9. Execute the block and check that the new row has been inserted.

SELECT * FROM new_depts;

10. Now modify the code to use SQL%ROWCOUNT to display the number of rows inserted, and execute the block again.

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

11. Now modify the block, removing the INSERT statement and adding a statement that will UPDATE all rows with location_id = 1700 to location_id = 1400. Execute the block again to see how many rows were updated.

Answer: 4 rows affected
By now the block should look like this:

```
DECLARE
  v_max_deptno         new_depts.department_id%TYPE;
  v_dept_name   new_depts.department_name%TYPE
                          := 'A New Department' ;
  v_dept_id     new_depts.department_id%TYPE;
BEGIN
  SELECT MAX(department_id) INTO v_max_deptno
    FROM new_depts;
  UPDATE new_depts SET location_id = 1400
    WHERE location_id = 1700;
  DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows affected.');
  DBMS_OUTPUT.PUT_LINE('The maximum department id is: '||
                      v_max_deptno);
END;
```

SECTION 3 LESSON 4 - Using Transaction Control Statements

Terminology

1. **__Transaction______________** An inseparable list of database operations, which must be executed either in its entirety or not at all.
2. **__ROLLBACK_____________** Used for discarding any changes that were made to the database after the last COMMIT.
3. **__SAVEPOINT_____________** Used to mark an intermediate point in transaction processing.
4. **__END____________________** Keyword used to signal the end of a PL/SQL block, not the end of a transaction.
5. **__COMMIT________________** Statement used to make database changes permanent.

Try It/Solve It

For all the questions in this Practice, you need to disable the Autocommit feature in Application Express, so that you can COMMIT and ROLLBACK transactions explicitly. Uncheck the Autocommit check box in the top left corner of the SQL Commands window.

1. How many transactions are shown in the following code? Explain your reasoning.

```
BEGIN
  INSERT INTO my_savings (account_id, amount)
        VALUES (10377,200);
  INSERT INTO my_checking(account_id, amount)
        VALUES (10378,100);
END;
```

There is one (uncommitted) transaction. Even though there are two INSERT statements, there is only one transaction.

2. Create the endangered_species table by running the following statement in Application Express:

```
CREATE TABLE endangered_species
 (species_id          NUMBER(4)
                    CONSTRAINT es_spec_pk PRIMARY KEY,
  common_name       VARCHAR2(30)
                    CONSTRAINT es_com_name_nn NOT NULL,
  scientific_name VARCHAR2(30)
                    CONSTRAINT es_sci_name_nn NOT NULL);
```

3. Examine the following block. If you were to run this block, what data do you think would be saved in the database?

```
BEGIN
  INSERT INTO endangered_species
    VALUES (100, 'Polar Bear','Ursus maritimus');
  SAVEPOINT sp_100;
  INSERT INTO endangered_species
    VALUES (200, 'Spotted Owl','Strix occidentalis');
  SAVEPOINT sp_200;
  INSERT INTO endangered_species
    VALUES (300, 'Asiatic Black Bear','Ursus thibetanus');
  ROLLBACK TO sp_100;
  COMMIT;
END;
```

Only one row: 100 Polar Bear Ursus maritimus

4. Run the block to test your theory. Select from the table to confirm the result.

SELECT * FROM endangered_species;

5. Examine the following block. If you were to run this block, what data do you think would be saved in the database?

```
BEGIN
  INSERT INTO endangered_species
    VALUES (400, 'Blue Gound Beetle','Carabus intricatus');
  SAVEPOINT sp_400;
  INSERT INTO endangered_species
    VALUES (500, 'Little Spotted Cat','Leopardus tigrinus');
  ROLLBACK;
  INSERT INTO endangered_species
    VALUES (600, 'Veined Tongue-Fern','Elaphoglossum nervosum');
  ROLLBACK TO sp_400;
END;
```

Error. This code would result in an error because SAVEPOINT sp_400 is removed by the first ROLLBACK statement (on line 7).

6. Run the block to test your theory.