An

AppITop™ Solutions

Training Guide

PL/SQL for Beginners

Workbook

Workbook

This workbook should be worked through with the associated Training Guide, *PL/SQL for Beginners*.

Each section of the workbook corresponds to a section in the Training Guide. There are a number of questions and exercises to perform in each section.

All the answers are at the back of this workbook, feel free to take a look whilst working through, sometimes a quick look at the answer is just enough to jog your memory.

Good Luck!

Exercise Hints

When you write any PL/SQL from within SQL*Plus, remember to use the following SQL*Plus commands:-

`ed`	Invoke editor
`save`	Save contents of SQL buffer to a file
`get`	Load file into SQL buffer
`start`	Load and execute file
`@`	Execute file
`/`	Execute contents of SQL buffer

Using the above commands will save you time and allow you to keep a copy of your work.

Ampersand substitution variables can be used in a PL/SQL block just like they can in a SQL program, for example:-

```
DECLARE
      l_number NUMBER := &1;
BEGIN
      DBMS_OUTPUT.put_line(l_number);
END;
```

The above code will prompt you to enter a value, this value is put into `l_number`.

Section One

Introduction to PL/SQL

Quiz

1. What does PL/SQL stand for?

2. What is PL/SQL?

3. What types of PL/SQL are there?

4. Name 4 features of PL/SQL

Section Two

PL/SQL Fundamentals

Quiz

1. How is a statement terminated?

2. What keywords are used to determine a PL/SQL block?

3. Names 3 Block Types.

4. What does the following code display?

```
DECLARE
      l_var NUMBER := 10;
BEGIN
      DECLARE
            l_var NUMBER := 20;
      BEGIN
            DBMS_OUTPUT.put_line(l_var);
      END;
END;
```

5. What are the two types of comment that can be used in PL/SQL?

6. Name 3 scalar variable datatypes

7. What is a Constant?

8. What is wrong with the following code?

```
l_status_type CONSTANT NUMBER;
```

9. What does `%TYPE` do and where is it used?

10. What is wrong with the following code?

```
l_description VARCHAR2(5) := 'Description';
```

11. What is the value of `l_x`?

```
l_a    NUMBER := 10;
l_b    NUMBER := 15;
l_x    NUMBER := l_a + l_b;
```

12. Given the following declarations:-

```
l_x   NUMBER := 50;
l_y   NUMBER := 25;
l_z   NUMBER;
l_a   NUMBER;
```

What is the value of `l_a`:-

```
l_a := l_x + l_y;
l_a := l_x / 2 + l_y;
l_a := l_y + l_z;
l_a := NVL(l_z,100) + l_x;
```

13. What is the difference between the `=` and `:=` operators?

14. Given the following declarations:-

```
l_x   NUMBER := 50;
l_y   NUMBER := 20;
l_s   VARCHAR2(3) := '50';
l_b   BOOLEAN;
```

Is `l_b` TRUE or FALSE?

```
l_b := l_x > l_y;
l_b := 100 < l_y + l_x;
l_b := l_s;
l_b := l_s > l_x;
l_b := l_y < l_x AND l_x = l_s;
l_b := 30 BETWEEN l_x AND l_y;
```

15. How can you view compile (syntax) errors in a PL/SQL block?

16. What has to be done before `DBMS_OUTPUT` can be used?

17. What does the following code print

```
DECLARE
      l_x NUMBER := 50/10;
      l_y NUMBER := 6;
BEGIN
      IF l_x < l_y AND l_x BETWEEN l_y-l_y+1 AND l_y+l_x THEN
            DBMS_OUTPUT.put_line('TRUE');
      ELSE
            DBMS_OUTPUT.put_line('FALSE');
      END IF;
END;
```

18. What is the difference between a `FOR` loop and a `WHILE` loop?

19. What is wrong with the following code?

```
DECLARE
      l_done BOOLEAN := FALSE;
BEGIN
      WHILE NOT l_done
            l_total := l_total + 10;
      END LOOP;
END;
```

20. What will the following code print?

```
<<block1>>
DECLARE
      l_x NUMBER := 10;
      l_y NUMBER := 20;
BEGIN
      DBMS_OUTPUT.put_line(l_x);

      <<block2>>
      DECLARE
            l_x NUMBER := 50;
            l_y NUMBER := 60;
            l_z NUMBER;
      BEGIN
            DBMS_OUTPUT.put_line(l_x);
            DBMS_OUTPUT.put_line(block1.l_y);

            l_z := (l_x + block1.l_x) / block1.l_y;

            DBMS_OUTPUT.put_line(block2.l_z);
      END;
END;
```

21. Why should the use of the `GOTO` statement be avoided?

Exercises

1. Create a program that accepts two numbers from substitution variables, display one of the following messages:-

```
first is greater than second
first is less than second
first is the same as second
```

2. Create a program that accepts a single number. Display the message `Hello World` X times, where X is the number entered.

3. Try running the above program and entering a negative number, what happens? Change the code to print a message if a number less than 1 is entered that informs the user they must enter a number greater than or equal to 1.

Section Three

SQL within PL/SQL

Quiz

1. What kind of SQL statements cannot be directly run from within a PL/SQL program?

2. What is the INTO clause for?

3. What **could** go wrong with the following code?
```
DECLARE
      l_name      VARCHAR2(10);
BEGIN
      SELECT ename
      INTO   l_name
      FROM   emp
      WHERE  job = 'CLERK';
END;
```

4. What is an Implicit Cursor?

5. What other DML statements can be used in PL/SQL besides `SELECT`?

6. What is wrong with the following code?
```
DECLARE
      job emp.job%TYPE := 'CLERK';
BEGIN
      UPDATE emp
      SET sal = sal * 1.25
      WHERE job = job;
END;
```

7. What is `SQL%NOTFOUND`?

Exercises

1. Write a program that gives all employees in department 10 a 15% pay increase. Display a message displaying how many employees were awarded the increase.

2. Create a PL/SQL block that accepts a new job title and an old job title. Find all employees in the old job and give them the new job. Ensure a valid message is given back to the user even if no employees changed job.

Section Four

Exceptions

Quiz

1. What is an Exception?

2. What is an Exception Handler?

3. What keyword defines the start of an Exception Handler?

4. Where in a block should an Exception Handler appear?

5. Look at the following code:-

```
DECLARE
      l_name emp.ename%TYPE;
      l_empno emp.empno%TYPE := &1;
BEGIN
      SELECT      ename
      INTO        l_name
      FROM  emp
      WHERE empno = l_empno;

      DBMS_OUTPUT.put_line('Employee name = '||l_name);
END;
```

   What would you have to add to the code to display `UNKNOWN EMPLOYEE` if the employee number entered did not exist?

6. What do the functions `SQLCODE` and `SQLERRM` do?

7. What is Exception Propagation?

8. What does the `EXCEPTION_INIT` pragma allow you to do?

9. What does `WHEN OTHERS` allow you to do?

Exercises

1. Write the code for question 5 on the quiz

2. Create a program that accepts two numbers. If the first is larger than the second raise an exception called `e_bigger` and display an appropriate message.

3. Create a program that sets the `comments` column on the `transactions` table to `THIS IS A COMMENT LINE`.

4. Change the above program to handle the error raised. Display an appropriate message including the value of `SQLERRM` and insert the same message into the messages table.

Section Five

Explicit Cursors

Quiz

1. What is an Explicit Cursor?

2. Names 3 keywords used with explicit cursors.

3. What has to be done before rows can be fetched from a cursor?

4. How many errors can you see in the following code?

```
DECLARE
      CURSOR employee_cur(p_deptno emp.deptno)
      IS
            SELECT ename
            ,      job
            FROM   emp
            WHERE  deptno = l_deptno;

BEGIN

      OPEN employee_cur(10);

      LOOP
            FETCH r_employee INTO employee_cur;

            EXIT employee_cur.NOTFOUND;

      END LOOP;

      CLOSE;
END;
```

5. What is a Cursor `FOR` Loop?

6. Name 3 explicit cursor attributes.

7. What does `WHERE CURRENT OF` allow you to do?

Exercises

1. Create a program that mimics selecting all columns and rows from the `dept` table. There is no need to format the output, just select all columns and all rows. Use a cursor `FOR` loop.

2. Create a program that copies all departments to a table called `old_dept`. Do not use a cursor `FOR` loop. Display how many rows were copied.

Section Six

Stored Procedures & Functions

Quiz

1. What are differences between stored subprograms and anonymous blocks?

2. What is the difference between a function and a procedure?

3. What is an argument list?

4. What is the difference between Positional and Named notation?

5. What determines how an argument/parameter can be used, i.e., whether it can be used to pass values, return values,…etc.

6. Describe what each parameter mode does

7. What is the difference between Actual and Formal parameters?

8. How many errors can you find in the following code:-

```
REPLACE OR CREATE FUNCTION DelEmp(p_empno emp.empno)
BEGIN
      DELETE emp
      WHERE  empno := p_empno;
END;
```

9. How might you invoke a procedure with the following declaration?

```
PROCEDURE EmpIno( p_empno IN  emp.empno%TYPE
            ,     p_ename OUT emp.ename%TYPE
            ,     p_sal   OUT emp.sal%TYPE);
```

10. What is a local subprogram?

11. When might you use a local subprogram?

12. What does `RAISE_APPLICATION_ERROR` do?

Exercises

1. Create a procedure that deletes rows from the `old_emp` table. It should accept 1 parameter, job; only delete the employee's with that job. Display how many employees were deleted. Write a program to invoke the procedure.

2. Change the above procedure so that it returns the number of employees removed via an `OUT` parameter. Write a program to invoke the procedure and display how many employees were deleted.

3. Convert the above program to a function. Instead of using an `OUT` parameter for the number of employees deleted, use the functions return value. Write a program to invoke the function and display how many employees were deleted.

Section Seven

Packages

Quiz

1. What is a package?

2. What two parts make up a package?

3. What should appear in a package specification?

4. What is the difference between a package specification and package header?

5. What should appear in a package body?

6. What does subprogram-overloading mean?

7. When might you use subprogram overloading?

8. What are private package objects and where are they defined?

9. When invoking a packaged procedure or function, what do you need to do that is different to a stored procedure or function?

10. What is the pragma `RESTRICT_REFERENCES` used for?

11. Can the following function be invoked from a `SELECT` statement?

```
FUNCTION CapName( p_empno IN emp.empno%TYPE
            ,     p_ename OUT emp.ename%TYPE)
RETURN VARCHAR2
IS
      l_ename emp.ename%TYPE;
BEGIN
      SELECT ename
      INTO l_ename
      FROM emp
      WHERE empno = p_empno;

      RETURN l_ename;
END;
```

Exercises

Read the following specification:-

Our developers require some software that will act as an API (Application Programming Interface) for the `items` table. We need to protect our data and want to ensure no developers writes any code that will directly access this table.

Here is the structure of the items table: -

Column	**DataType**	**Description**
`item_id`	`NUMBER`	Internal ID for item
`item_number`	`VARCHAR2(10)`	User item number
`description`	`VARCHAR2(30)`	Item description
`status`	`VARCHAR2(1)`	[T]est or [L]ive
`cost`	`NUMBER`	Standard cost of item

We need the `item_id` column to be a sequential number (use `items_item_id_s` sequence)

The following business rules must be applied:-

- An item is created as a test item and with a zero cost.
- A procedure or function must be called to promote the item from test to live. An item cannot be made live with a zero cost.
- Only test items can be removed

We need an API to provide the developer the following facilities:-

- Create new items
- Promote items from test to live
- Remove items
- Change item cost

All API functions and procedures should work with the `item_id`.

Create a package to implement the above. Remember, try and work out the requirements for the package first. Determine your public and private procedures/functions and any data that might be needed.

Section Eight

Triggers

Quiz

1. What is a trigger?

2. When might triggers be used?

3. What are the 12 trigger types?

4. What is the trigger condition and why is it used?

5. How can you reference the column value of a row being updated, both before and after the update?

6. Assume we have created a trigger that fires on `INSERT` or `UPDATE` of a table. How can I make the trigger act differently depending on the triggering event?

7. How many errors can you find in the following code?

```
CREATE OR REPLACE TRIGGER set_stock
      BEFORE INSERT ON transactions
      FOR EVERY ROW
      WHEN (:new.transaction_type IN ('ISS','RCT'))
BEGIN
      UPDATE stock
      SET quantity = quantity + new.quantity
      WHERE item = :new.item_id;

      COMMIT;
END;
```

8. Generally, what kind of things cause triggers to fire?

Exercises

To compliment the package developed in the last section, the user has come up with the following addition to the specification.

> When items are removed using the new API you provided, we need to ensure the item is archived in a table called `items_archive`.
>
> We also want any changes in item cost to be audited, record the details of each change in the `audit_cost` table.

Implement the above specification using triggers.

NOTE
The above changes could just as easily be implemented within the package created in the last section. Remember, you provided an API to the items table so ALL changes to the data are controlled through the package, in theory, ALL developers should use the package. Implementing the changes using triggers is a more secure method because even if any changes to the data are made not using the package, the triggers will still do their job.

Answers

Section 1 Quiz

1. Procedural Language/Structured Query Language

2. PL/SQL is Oracle's procedural extension to SQL

3. Two Types, Client and Server

4. Any one of:-
   Has Variables & Constants
   Uses SQL
   Flow Control
   Many built-in functions
   Cursor Management
   Block Structure
   Exception Handling
   Composite Types
   Stored Code

Section 2 Quiz

1. With a semicolon (;).

2. `BEGIN` and `END.`

3. Anonymous, Named, Subprograms and Triggers.

4. 20

5. Single line (`--`) and multiple line (`/* */`).

6. `NUMBER`, `VARCHAR2`, `BOOLEAN`, `DATE.`

7. A constant is a variable declared with the `CONSTANT` keyword, its value cannot be changed.

8. It is constrained with the `CONSTANT` keyword but it has not been initialised.

9. `%TYPE` is used for anchoring the datatype of a variable to another object, this could be another variable or a column on a table. They are used within the declarative section of a program.

10. `l_description` is declared as a `VARCHAR2` of 5 digits, the string literal `Description` is more than 5 digits, this will cause the following error to occur:-
`ORA-06502: PL/SQL: numeric or value error`.

11. 25.

13. 75, 50, NULL and 150.

14. `=` is the equality operator, `:=` is the assignment operator.

15. `l_b := l_x > l_y;` = TRUE
`l_b := 100 < l_y + l_x;` = FALSE

`l_b := l_s;`	= Illegal
`l_b := l_s > l_x;`	= FALSE
`l_b := l_y < l_x AND l_x = l_s;`	= TRUE
`l_b := 30 BETWEEN l_x AND l_y;`	= FALSE

16. Use the SQL*Plus command:-
```
SHOW ERR[ORS]
```

17. Use SQL*Plus command:-
```
SET SERVEROUT[PUT] ON [SIZE x]
```

18. TRUE

19. A `FOR` is used when the number of iterations is known in advance. A `WHILE` loop is generally used when the number of iterations is not known in advance.

20. TRUE.

21. 10, 50, 20 and 3.

22. `GOTO` can make your code unstructured and hard to read/ debug.

Section 2 Exercises

1.
```
DECLARE
   l_number1 NUMBER := &1;
   l_number2 NUMBER := &2;

BEGIN

   IF l_number1 > l_number2 THEN
      DBMS_OUTPUT.put_line('first is greater than second');
   ELSIF l_number1 < l_number2 THEN
      DBMS_OUTPUT.put_line('first is less than second');
   ELSE
      DBMS_OUTPUT.put_line('first is same as second');
   END IF;

END;
```

2.
```
DECLARE
   l_times NUMBER := &1;

BEGIN

   FOR l_loop IN 1..l_times
   LOOP
      DBMS_OUTPUT.put_line('Hello World');
   END LOOP;

END;
```

3. If a negative number is entered, nothing happens, the loop never actually starts.
```
DECLARE
   l_times NUMBER := &1;

BEGIN

   IF l_times < 1 THEN
      DBMS_OUTPUT.put_line('Number must be at least 1');
   ELSE
      FOR l_loop IN 1..l_times
      LOOP
         DBMS_OUTPUT.put_line('Hello World');
      END LOOP;
   END IF;

END;
```

Section 3 Quiz

1. DDL cannot be used in PL/SQL directly, only DML.

2. The `INTO` clause is used to tell PL/SQL where to put data retrieved from a cursor.

3. The variable `l_name` should be anchored to a database table, if the name ever increased beyond 10 digits then an error would occur. The implicit cursor could possibly return more than one row or no rows.

4. All data is selected using cursors, an implicit cursor is simply a `SELECT` statement (or any other DML) that does not make direct use of any cursor commands such as `OPEN`, `FETCH`,…etc.

5. `INSERT`, `UPDATE` and `DELETE`.

6. The PL/SQL variable `job` is the same as a column on the `emp` table, this has the effect of making the `UPDATE` statement update all rows on the `emp` table because the statement reads, 'Where job on emp is equal to job on emp', this is TRUE for all rows.

7. `SQL%NOTFOUND` is an implicit cursor attribute, it is used to determine if the last DML statement affected any rows.

Section 3 Exercises

1.
```
BEGIN
      UPDATE emp
      SET sal = sal * 1.15
      WHERE deptno = 10;

      DBMS_OUTPUT.put_line(TO_CHAR(SQL%ROWCOUNT)||
            ' employee(s) updated');
END;
```

2.
```
DECLARE
      l_old_job emp.job%TYPE := '&1';
      l_new_job emp.job%TYPE := '&2';

BEGIN
      UPDATE emp
      SET job = l_new_job
      WHERE job = l_old_job;

      IF SQL%FOUND THEN
            DBMS_OUTPUT.put_line(TO_CHAR(SQL%ROWCOUNT)||
            ' employee(s) changed job');
      ELSE
            DBMS_OUTPUT.put_line('No employee found with job'||
                  ' of '||l_old_job);
      END IF;
END;
```

Section 4 Quiz

1. An Exception is an identifier within PL/SQL that can be used to trap for a specific condition. Exceptions are typically associated with an error. Exceptions are either raised automatically by PL/SQL or they can be raised explicitly.

2. An Exception Handler is a section of PL/SQL code that is there purely to deal with any raised exceptions.

3. `EXCEPTION`

4. The exception section should appear at the end of a block.

5.
```
DECLARE
       l_name emp.ename%TYPE;
       l_empno emp.empno%TYPE := &1;
BEGIN
       SELECT  ename
       INTO           l_name
       FROM    emp
       WHERE   empno = l_empno;

       DBMS_OUTPUT.put_line('Employee name = '||l_name);
EXCEPTION
       WHEN NO_DATA_FOUND THEN
              DBMS_OUTPUT.put_line('UNKNOWN EMPLOYEE');
END;
```

6. `SQLCODE` returns the last error number, `SQLERRM` returns the last error message and it includes the error code.

7. If an exception is raised within a block of PL/SQL, and this block does not explicitly handle that exception, then the exception is passed to the enclosing block, this continues until either the exception is handled or control is passed to the calling environment. This is called exception propagation.

8. It allows you to associate an error code with a declared exception.

9. `WHEN OTHERS` is used to handle all unhandled exceptions.

Section 4 Exercises

1. See answer to question 5 in the quiz.

2.
```
DECLARE
      l_number1 NUMBER := &1;
      l_number2 NUMBER := &2;

      e_bigger EXCEPTION;
BEGIN
      IF l_number1 > l_number2 THEN
            RAISE e_bigger;
      END IF;

      DBMS_OUTPUT.put_line('first is not bigger than second');

EXCEPTION
      WHEN e_bigger THEN
            DBMS_OUTPUT.put_line
                  ('EXCEPTION : first is bigger than second');
END;
```

3.
```
BEGIN
      UPDATE transactions
      SET comments = 'THIS IS A COMMENT LINE';
END;
```

4.
```
DECLARE
      l_error VARCHAR2(100);
      e_too_big EXCEPTION;
      PRAGMA EXCEPTION_INIT(e_too_big,-1401);

BEGIN
      UPDATE transactions
      SET comments = 'THIS IS A COMMENT LINE';

EXCEPTION
      WHEN e_to_big THEN
            l_error := 'Error : Could not update '||
                  'transactions table - '||SQLERRM;
            DBMS_OUTPUT.put_line(l_error);
            INSERT INTO messages
                        (logged_at
                  ,      message) VALUES
                  (      SYSDATE
                  ,      l_error);
END;
```

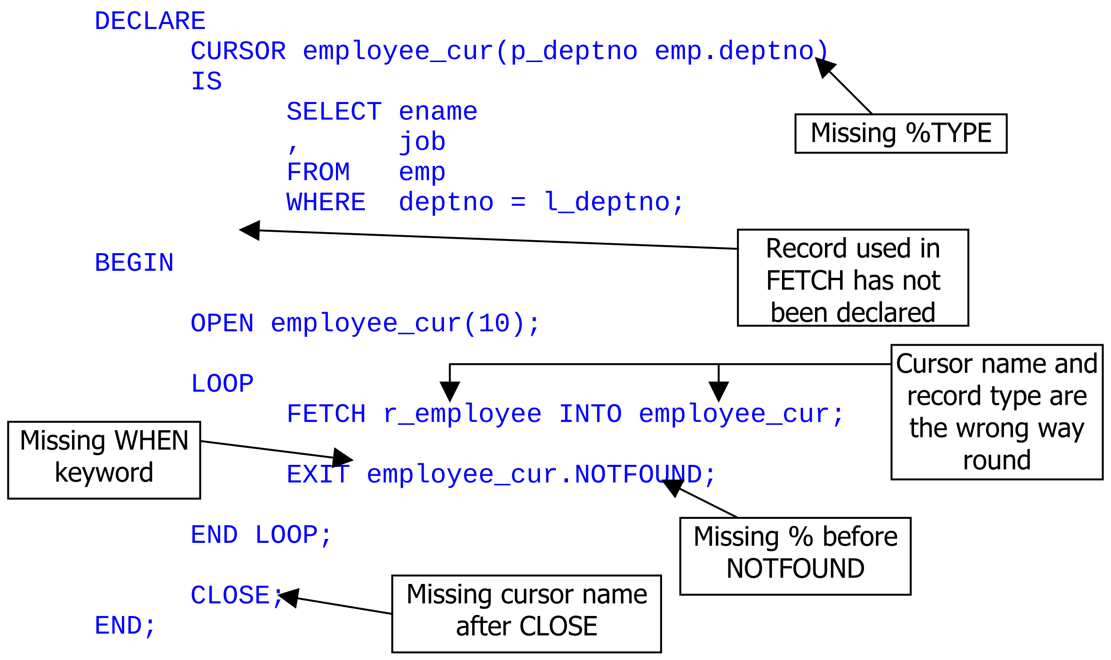
Section 5 Quiz

1. An Explicit Cursor is a named construct within PL/SQL that is used to retrieve data from the database.

2. Any one of:- `CURSOR`, `OPEN`, `FETCH`, `CLOSE` or `FOR`.

3. The cursor has to be declared and opened, unless you are using a cursor `FOR` loop with a `SELECT` sub-statement.

4. 6 errors.



```
DECLARE
      CURSOR employee_cur(p_deptno emp.deptno)
      IS
            SELECT ename
            ,      job
            FROM   emp
            WHERE  deptno = l_deptno;

BEGIN

      OPEN employee_cur(10);

      LOOP
            FETCH r_employee INTO employee_cur;

            EXIT employee_cur.NOTFOUND;

      END LOOP;

      CLOSE;
END;
```

Missing %TYPE
Record used in FETCH has not been declared
Cursor name and record type are the wrong way round
Missing WHEN keyword
Missing % before NOTFOUND
Missing cursor name after CLOSE

5. A cursor `FOR` loop is a convenient way to work with explicit cursors. They do the opening, fetching and closing for you.

6. `ISOPEN`, `NOTFOUND`, `FOUND` or `ROWCOUNT`.

7. `WHERE CURRENT OF` allows you to reference the last `FETCH`ed row from a cursor without having to specify any column names. It is typically used to `UPDATE` or `DELETE` rows and must be used in conjunction with `FOR UPDATE` in the cursor declaration.

Section 5 Exercises

1.
```
DECLARE
      CURSOR dept_cur
      IS
            SELECT      deptno
                  ,     dname
                  ,     loc
            FROM  dept;

BEGIN
      FOR r_dept in dept_cur
      LOOP
            DBMS_OUTPUT.put_line(r_dept.deptno);
            DBMS_OUTPUT.put_line(r_dept.dname);
            DBMS_OUTPUT.put_line(r_dept.loc);
      END LOOP;
END;
```

2.
```
DECLARE
      CURSOR dept_cur
      IS
            SELECT      deptno
                  ,     dname
                  ,     loc
            FROM  dept;

      r_dept dept_cur%ROWTYPE;

BEGIN
      OPEN dept_cur;

      LOOP
            FETCH dept_cur INTO r_dept;

            EXIT WHEN dept_cur%NOTFOUND;

            INSERT INTO old_dept
                  (deptno,dname,loc) VALUES
                  (r_dept.deptno,r_dept.dname,r_dept.loc);
      END LOOP;

      DBMS_OUTPUT.put_line(TO_CHAR(dept_cur%ROWCOUNT)||
                  ' department(s) copied');

      CLOSE dept_cur;
END;
```

Section 6 Quiz

1. Stored programs are stored within the database in compiled form and executed on the database, whereas anonymous blocks are usually held in a host file and are parsed and compiled at runtime, they are explicitly executed in a client tool, typically SQL*Plus.

2. A procedure acts like a PL/SQL statement where as a function returns a value and is used as part of an expression.

3. The arguments define what parameters a function or procedure accepts.

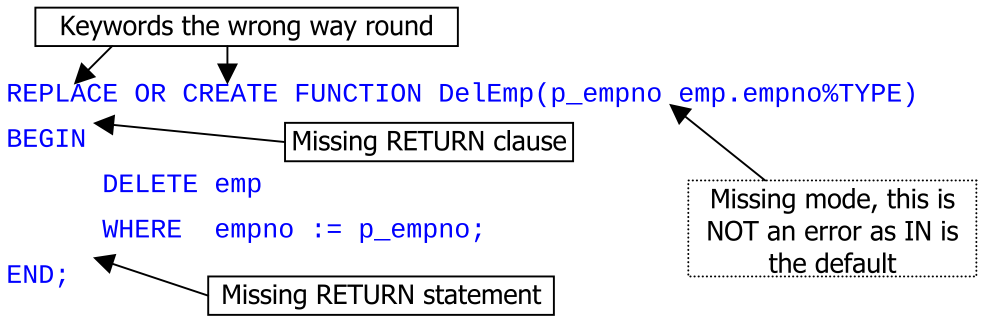
4. Parameters are passed to a function/procedure in two ways; Positional Notation matches Actual parameters with Formal parameters based wholly on the position in the argument list, whereas Named notation allows you to specify parameters in any order, this is achieved by pre-fixing the Actual parameter with the Formal parameter name.

5. The parameter MODE; `IN`, `OUT` or `IN OUT`

6. `IN` = Allows parameters to be passed into a subprogram, they are read only.
   `OUT` = Allows parameters to be passed back to the calling programs, they are write only.
   `IN OUT` = Allows both read and write.

7. An Actual parameter is the parameter passed within the calling program. A Formal parameters is the variable used within the subprogram itself.

8. 3 errors



```
REPLACE OR CREATE FUNCTION DelEmp(p_empno emp.empno%TYPE)
BEGIN
      DELETE emp
      WHERE  empno := p_empno;
END;
```

Keywords the wrong way round
Missing RETURN clause
Missing mode, this is NOT an error as IN is the default
Missing RETURN statement

9.

```
DECLARE
      l_ename      emp.ename%TYPE;
      l_sal        emp.sal%TYPE;
BEGIN
      EmpInfo(     7900
            ,      l_ename
            ,      l_sal);
END;
```

10. A local subprogram is a function or procedure that is local to a block. They are defined in the declarative section of a block after all other declarations.

11. A typical use for local subprograms is to create helper subprograms, these are subprograms which are only of use to the block in which they are defined.

12. `RAISE_APPLICATION_ERROR` allows you to pass error information back to a calling program.

Exercises

1.

```
CREATE OR REPLACE PROCEDURE DelEmp(p_job IN emp.job%TYPE)
IS
BEGIN
      DELETE old_emp
      WHERE job = p_job;

      DBMS_OUTPUT.put_line(TO_CHAR(SQL%ROWCOUNT)||' removed');
END;
```

To invoke the procedure:-

```
BEGIN
      DelEmp('CLERK');
END;
```

2.

```
CREATE OR REPLACE PROCEDURE DelEmp( p_job   IN  emp.job%TYPE
                                ,      p_count OUT NUMBER)
IS
BEGIN
      DELETE old_emp
      WHERE job = p_job;

      p_count := SQL%ROWCOUNT;
END;
```

To invoke the procedure:-

```
DECLARE
      l_count NUMBER;
BEGIN
      DelEmp('CLERK',l_count);
      DBMS_OUTPUT.put_line(l_count);
END;
```

3.

```
CREATE OR REPLACE FUNCTION DelEmp(p_job IN emp.job%TYPE)
      RETURN NUMBER
IS
BEGIN
      DELETE old_emp
      WHERE job = p_job;

      RETURN SQL%ROWCOUNT;
END;
```

To invoke the function:-

```
DECLARE
      l_count NUMBER;
BEGIN
      l_count := DelEmp('CLERK');
      DBMS_OUTPUT.put_line(l_count);
END;
```

Section 7 Quiz

1. A package is a named PL/SQL block that is stored in compiled form and executed within the database. Packages can contain subprograms and data.

2. A package is made up of two parts, a Specification and a Body.

3. Public object declarations appear in the specification.

4. They are the same thing.

5. Private data and subprograms as well as the definitions for public subprograms.

6. Subprogram overloading allows you to create more than one subprogram with the same name but with different arguments. If allows you to create subprograms that act differently depending on the data they are supplied, though to the user, it appears as if a single subprogram is being used.

7. A common use for subprogram overloading is to provide a single function that can act on different types of data.

8. A private package object is something that only the package itself can use. These are declared and defined in the package body.

9. Qualify the subprogram name with the package name.

10. It is used to inform the compiler of the purity level of a packaged function.

11. No, because the function has an `OUT` parameter, these are not allowed when invoking a function from DML.

Section 7 Exercises

Here is the finished package, though I have not included any exception handling and some of the code could probably have been written in a more generic/complete way, I have tried to keep it simple as the main concern here is the creation of the actual package and not what it does.

```
CREATE OR REPLACE PACKAGE items_api
IS
      --
      -- Public procedure declarations
      --
      PROCEDURE add(    p_item_number     IN items.item_number%TYPE
                  ,     p_description     IN items.description%TYPE);

      PROCEDURE promote(p_item_id IN items.item_id%TYPE);

      PROCEDURE remove(p_item_id IN items.item_id%TYPE);

      PROCEDURE chg_cost(p_item_id  IN items.item_id%TYPE
                  ,     p_new_cost  IN items.cost%TYPE);
END items_api;

CREATE OR REPLACE PACKAGE BODY items_api
IS
      --
      -- Private data
      --
      c_test_status CONSTANT VARCHAR2(1) := 'T';
      c_live_status CONSTANT VARCHAR2(1) := 'L';

      --
      -- Private procedures/functions
      --
      PROCEDURE p(p_text IN VARCHAR2)
      IS
      BEGIN
            DBMS_OUTPUT.put_line(p_text);
      END;

      --
      -- Public procedure/function definitions
      --

      -- Procedure to create a new item
      PROCEDURE add(    p_item_number     IN items.item_number%TYPE
                  ,     p_description     IN items.description%TYPE)
      IS
            c_new_cost   CONSTANT NUMBER := 0; -- Starting cost

      BEGIN
            INSERT INTO items
            (     item_id
            ,     item_number
```

```
        ,      description
        ,      status
        ,      cost ) VALUES
        (      items_item_id_s.NEXTVAL
        ,      p_item_number
        ,      p_description
        ,      c_test_status
        ,      c_new_cost);

        p('Item created');
END;

-- Procedure to promote an item
PROCEDURE promote(p_item_id IN items.item_id%TYPE)
IS
        CURSOR items_cur(p_item_id items.item_id%TYPE)
        IS
               SELECT status
               ,      cost
               FROM   items
               WHERE  item_id = p_item_id
               FOR UPDATE;

        r_items items_cur%ROWTYPE;
BEGIN
        OPEN items_cur(p_item_id);

        FETCH items_cur INTO r_items;

        -- Does item exist?
        IF items_cur%NOTFOUND THEN
               p('Item not found');
        ELSE
               -- Ensure item is not already live
               IF r_items.status = c_live_status THEN
                      p('Item already live');
               ELSE
                      -- ensure cost is not zero
                      IF r_items.cost = 0 THEN
                             p('Cannot promote'||
                                    '. Item cost is zero');
                      ELSE
                             -- Promote item
                             UPDATE items
                             SET status = c_live_status
                             WHERE CURRENT OF items_cur;

                             p('Item promoted');
                      END IF;
               END IF;
        END IF;

        CLOSE items_cur;
END;

-- Procedure to remove an item
PROCEDURE remove(p_item_id IN items.item_id%TYPE)
IS
BEGIN
```

```
            -- Only remove item if status is test
            DELETE items
            WHERE  item_id = p_item_id
            AND    status = c_test_status;

            -- Give feedback
            IF SQL%NOTFOUND THEN
                  p('Test item not found');
            ELSE
                  p('Item deleted');
            END IF;
      END;

      -- Procedure to change item cost
      PROCEDURE chg_cost(p_item_id  IN items.item_id%TYPE
                  ,     p_new_cost  IN items.cost%TYPE)
      IS
      BEGIN
            -- Change cost
            UPDATE items
            SET    cost = p_new_cost
            WHERE  item_id = p_item_id;

            -- Give feedback
            IF SQL%NOTFOUND THEN
                  p('Item not found');
            ELSE
                  p('Cost changed');
            END IF;
      END;

END items_api;
```

Section 8 Quiz

1. A trigger is a named PL/SQL block that fires implicitly when a particular database event occurs.

2. Because triggers are guaranteed to fire, they are perfect for many tasks, some of the more common tasks might be:- Auditing, Archiving, Complex Constraints, Maintain Derived Values,…and many more.

3. ROW and Statement types, BEFORE and AFTER timing types and INSERT, UPDATE and DELETE event types, that is 2 * 2 * 3 which is a total of 12 types.

4. A trigger condition determines if the trigger should fire or not. Trigger conditions are specified using the WHEN clause.

5. Use the old and new keywords as a prefix to the column name.

6. Use the INSERTING and UPDATING functions to determine what the actual triggering event was, then code accordingly.

7. 4 errors:-

```
CREATE OR REPLACE TRIGGER set_stock
      BEFORE INSERT ON transactions
      FOR EVERY ROW
      WHEN (:new.transaction_type IN ('ISS','RCT'))
BEGIN
      UPDATE stock
      SET quantity = quantity + new.quantity
      WHERE item_id = :new.item_id;
      COMMIT;
END;
```

Used EVERY instead of EACH

No need for colon (:) here

Missing colon (:) before new

COMMIT not allowed in a trigger (*unless AUTONOMOUS_TRANSACTION*)

8. DML statements.

Exercises

1.
```
CREATE OR REPLACE TRIGGER items_archive_t
      BEFORE DELETE ON items
      FOR EACH ROW
BEGIN
      INSERT INTO items_archive
      (     item_id
      ,     item_number
      ,     description
      ,     status
,     cost
      ,     date_archived ) VALUES
      (     :old.item_id
      ,     :old.item_number
      ,     :old.description
      ,     :old.status
      ,     :old.cost
      ,     SYSDATE);
END;
```

2.
```
CREATE OR REPLACE TRIGGER audit_cost_t
      BEFORE UPDATE OF cost ON items
      FOR EACH ROW
      WHEN (new.cost <> old.cost)
BEGIN
      INSERT INTO audit_cost
      (     item_id
      ,     old_cost
      ,     new_cost
      ,     date_changed ) VALUES
      (     :new.item_id
      ,     :old.cost
      ,     :new.cost
      ,     SYSDATE);
END;
```