# CIT 325 W02 Quiz: Chapter 3

* Block Structure
  + PL/SQL program units organize the code into blocks. A block with no name is known as an anonymous block. The anonymous block is the simplest unit in PL/SQL. On the other hand, named blocks are called functions and procedures.
  + 
* Declaration Block
  + It is the optional and starts with the DECLARE keyword and ends with the BEGIN keyword for anonymous blocks. The declaration section allows you to define data types, structures, and variables. You often declare variables in the declaration section by giving them names, data types, and initial values.
* Exception Block
  + This is optional and starts with the EXCEPTION keyword and ends with the END keyword. The exception section is the place where you put the code to handle exceptions. You can either catch or handle exceptions in the exception section.
* Execution Block
  + This is a mandatory section enclosed between the keys BEGIN and END. When using the EXCEPTION block, it will end with the exception keyword. The execution section is the place where you put the execution code or business logic code. You can use both procedural and SQL statements inside the execution section.
* Behavior of Variables in Blocks
  + PL/SQL also supports scalar and composite variables. Scalar variables hold only one thing, while composite variables hold more than one thing. This section covers the scope and behavior of variables in anonymous blocks, nested blocks, local names blocks, and stored named blocks.
* Nested Anonymous Blocks
  + Make sure you understand the examples. The following is an example of nested blocks. Do not worry if you don’t understand the program, just look at the declare, begin, exception, and end keywords.



* + Nested anonymous blocks acts like the blocks in the example in the preceding section. That’s because any program that contains an anonymous block program assumes the SQL\*Plus environment’s role for a stand-alone anonymous block PL/SQL program.

SQL> DECLARE

2 -- Declare local variable.

3 lv\_input VARCHAR2(30) DEFAULT 'OUTER';

4 BEGIN

5 -- Print the value before the inner block.

6 dbms\_output.put\_line('Outer block ['||lv\_input||']');

7

8 -- Nested block.

9 BEGIN

10 -- Print the value before the assignment.

11 dbms\_output.put\_line('Inner block ['||lv\_input||']');

12

13 -- Assign new value to variable.

14 lv\_input := 'INNER';

15

16 -- Print the value after the assignment.

17 dbms\_output.put\_line('Inner block ['||lv\_input||']');

18 END;

19

20 -- Print the value after the nested block.

21 dbms\_output.put\_line('Outer block ['||lv\_input||']');

22 EXCEPTION 23 WHEN OTHERS THEN

24 dbms\_output.put\_line('Exception ['||SQLERRM||']');

25 END;

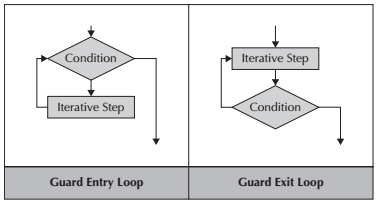
26 /

Line 3 declares the lv\_input variable with an initial value of “Set in the outer block.” The scope of the variable is the outer block of the anonymous block, which means you can assign it new values in the outer or inner blocks. Line 14 in the inner block assigns to the lv\_input variable a new value, “Set in the inner block. ”

* Basic Scalar and Composite Data Types
  + This section introduces the basics about the three most common scalar data types, attribute and table anchoring, and four generic composite data types. The common scalar data types are character, dates, and numbers. Composite data types include SQL UDTs, PL/SQL record types, collections of SQL data types, and collections of SQL data types, and collections of PL/SQL data types. Scalar data types hold only one thing, while composite data types hold more than one thing, such as a structure or collection of data.
* Scalar
  + They hold only one thing at a time and are frequently labeled as primitive; these include numbers, strings, and timestamps. Oracle timestamps are dates precise to one thousandth of a second.
* Composite Data Types
  + You can also define compound variables, alternatively labeled composite variables. There’s not much difference in the words, but Oracle Database 12c documentation uses the term composite variables. So, this book uses “composite variables” to describe arrays, structures, and objects. Composite variables are variables built from primitives in a programming language.
  + They hold collections of data.
* SQL Collections
  + Collections are programming structures that hold sets of variables; they fall into two categories: arrays and lists. Oracle8i Database forward provides three types of collections. Covered in more detail in Chapter 6.
    - Associative Arrays are collection types that associates a unique key with the value of the index.
    - Nested Table is one table placed inside of another.
    - Varray are fixed-size sequential collection of elements of the same type.
  + These collections maybe tables (or lists) of values varrays (or arrays in traditional programming languages). Tables have no upward limit on the number of elements in the collection, which is why they act like lists. Varrays have a maximum number of elements set when you define their types.

SQL collections can exist for scalar data types or SQL UDT elements. Oracle calls SQL collections of scalar columns Attribute Data Types (ADTs). While there are some differences between how ADTs and UDTs are used in the Oracle Call Interface (OCI), the distinct names appear to disambiguate collections of native data types from collections of UDTs.

* Associative Arrays
  + A collection of types that associates a unique key with a value of the index.
* ADT Collection
  + An ADT collection in SQL requires that you define a collection of a SQL base data type, such as a string data type. Not to confuse matters, but the syntax is the same as when you create any object type collection in the database.
* Nested Table or Table
  + The idea of nesting falls apart when the collection isn’t a persistent object type that defines a column in a table. That’s why I chose to use table instead of nested table in most of this book.
* Varray Collections
  + These collections are single-dimensional structures that have a maximum number of elements. The elements all have the same data type. As mentioned, the element data type is the base data type of the varray collection.
  + The prototype for creating a SQL varray collection is
    - TYPE type\_name IS {VARRAY | VARYING ARRAY}(size\_limit) OF data\_type [NOT NULL];
* Conditional Structures
  + Conditional structures check logical conditions and branch program execution. The if, elseif, else, and case statements are conditional structures.
    - **IF-THEN-ELSIF, and ELSE** statements work on a concept of Boolean logic. A Boolean variable or an expression, such as a comparison of values, is the only criterion for an if or elsif statement. While this seems simple, it really isn’t, because truth or untruth has a third case in an Oracle database: a Boolean variable or expression can be true, false, or null. This is called three-valued logic. You can manage three-valued logic by using the NVL built-in function. It allows you to impose an embedded check for a null and return the opposite of the logical condition you attempted to validate.
    - **Three-Valued Logic** means basically that if you find something is true when you look for truth, it is true. By the same token, when you check whether something is false and it is, then it is false. The opposite case isn’t proved. That means when something isn’t true, you can’t assume it is false, and vice versa. The third case is that if something isn’t true, it can be false or null. Likewise, if something isn’t false, it can be true or null. Something is null when a Boolean variable is defined but not declared or when an expression compares something against another variable that is null.
    - **CASE Statement** appears very similar to a switch structure in many programming languages, but it doesn’t perform in the same way because it doesn’t support fall-through. **Fall-through** is the behavior of finding the first true case and then performing all remaining cases. The case statement in PL/SQL performs like an if-elsif-else statement. There are two types of CASE statements: the simple case and the searched case. You can use a CHAR, NCHAR, or VARCHAR2 data type in simple case statements, and you can use any Boolean expression in searched case statements.
    - **Iterative Structures** are blocks that let you repeat a statement or a set of statements. These structures come in two varieties: a guard-on-entry loop and a guard-on-exit loop. Figure 3-2 shows the execution logic for these two types of loops.



* + - **FOR Loop Statements**: You can implement the for loop as a range loop or as a cursor loop. A **range loop** moves through a set of sequential numbers, but you need to know the beginning and ending values. It is a guard-on-exit looping structure. You can navigate through a for loop forward or backward by using an ascending integer range.
    - **WHILE Loop Statements**: A while loop is a guard-on-entry loop: you need to manage both the entry and exit criteria of a while loop. Unlike the for loop, with the while loop you don’t need an index value because you can use other criteria to control the entry and exit criteria. If you use an index, the Oracle Database 11g CONTINUE statement can make control more complex, because it allows you to abort an iteration and return to the top of the loop.
* Exceptions
  + The exception block manages any exceptions that occur while running the execution block. Errors raised in the declaration block are thrown to and managed by the calling scope program.
    - **User-Defined Exceptions**: You can declare user-defined exceptions in either of two ways: declare an EXCEPTION variable only or declare an EXCEPTION variable and EXCEPTION\_INIT compiler directive. The EXCEPTION variable by itself lets you catch a user-defined exception with an OTHERS exception handler and if statement. The if statement checks for the user-defined exception number (oddly enough, 1 is that number). The combination of an EXCEPTION variable and EXCEPTION\_INIT compiler directive lets you create a customer exception handler.
    - **Dynamic User-Defined Exceptions**: let you raise a customized exception by assigning a number in the range of –20,000 to –20,999. The RAISE\_APPLICATION\_ERROR function provides this ability in PL/SQL.
* Functions, Procedures, and Packages
  + PL/SQL stored programming units are typically functions, procedures, packages, and triggers. This functionality enables commonly required code to be written and tested once and then accessed by any application that requires the code. Covered in more detail in chapters 8 and 9.
    - **Functions**: Stored functions are convenient structures because you can call them directly from SQL statements or PL/SQL programs. All stored functions must return a value. You can also use them as right operands because they return a value. Functions are defined in local declaration blocks or the database. You frequently implement them inside stored packages.
    - **Procedures**: As mentioned in the previous section, procedures cannot be right operands. Nor can you use them in SQL statements. You move data into and out of PL/SQL stored procedures through their formal parameter list. Like stored functions, you can also define local named block programs in the declaration section of procedures.
    - **Packages**: Package development starts with planning which shared data types and cursors should be bundled with which functions and procedures. Shared data types let you exchange information using the specifications of scalar, record structure, and collection data types that a package can require. Shared cursors, on the other hand, present the possibility that a query might be reused many times and would be
* Database Triggers
  + Triggers are similar to stored procedures that run implicitly when an INSERT, UPDATE or DELETE statement is issued against the associated table or, in some cases, against a view or when database system actions occur. This topic is covered in more detail in Chapter 12.