

DECLARING VARIABLES

-- CHAPTER 16 --

- PL/SQL is an extension to SQL with design features of programming languages.
- Data manipulation and query statements of SQL are included within procedural units of code. PL/SQL offers modern software engineering features such as data encapsulation, exception handling, information hiding, and object orientation, and so brings state-of-the-art programming to the Oracle Server and Toolset.
- **Benefits of PL/SQL:**
- PL/SQL plays a central role to both the Oracle Server (through stored procedures, stored functions, database triggers, and packages) and Oracle development tools (through Oracle Developer component triggers).
- Oracle Developer applications make use of shared libraries that hold code (procedures and functions) and can be accessed locally or remotely. Oracle Developer consists of Oracle Forms, Oracle Reports, and Oracle Graphics.
- SQL datatypes can also be used in PL/SQL. Combined with the direct access that SQL provides, these shared datatypes integrate PL/SQL with the Oracle Server data dictionary. PL/SQL bridges the gap between convenient access to database technology and the need for procedural programming capabilities.
- **PL/SQL in Oracle Tools**
- Many Oracle tools, including Oracle Developer, have their own PL/SQL engine, which is independent of the engine present in the Oracle Server.
- The engine filters out SQL statements and sends them individually to the SQL statement executor in the Oracle Server. It processes the remaining procedural statements in the procedural statement executor, which is in the PL/SQL engine.
- The procedural statement executor processes data that is local to the application (that is already inside the client environment, rather than the database). This reduces work sent to the Oracle Server and the number of memory cursors required.
- PL/SQL Block structure:

```
DECLARE - Optional
    Variables, cursors, user-defined exceptions
BEGIN - Mandatory
    -SQL statements
    - PL/SQL statements
EXCEPTION - Optional
    Actions to perform when errors occur
END; - Mandatory
```

- Note that only the BEGIN and END keywords are required. Basically, a PL/SQL block is composed of three types sections:
 - DECLARATIVE:
 - Contains all variables, constants, cursors, and user defined exceptions that are referenced in the executable and declarative sections.
 - EXECUTABLE:
 - Contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block

- EXCEPTION HANDLING:
 - Specifies the actions to perform when errors and abnormal conditions arise in the executable section.
- There are three types of code blocks:
 - Anonymous


```
[DECLARE]
BEGIN
    --Statements
[EXCEPTION]
END;
```
 - Procedure


```
PROCEDURE name
IS
BEGIN
    --Statements
[EXCEPTION]
END;
```
 - Function


```
PROCEDURE name
RETURN datatype
IS
BEGIN
    --Statements
RETURN value;
[EXCEPTION]
END;
```
- **Block Types:**
 - Every unit of PL/SQL comprises one or more blocks. These blocks can be entirely separate or nested one within another. The basic unit (procedures and functions, also known as subprograms, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. Therefore, one block can represent a small part of another block, which in turn can be part of the whole unit of code. Of the two types of PL/SQL constructs available, anonymous blocks and subprograms, only anonymous blocks are covered in this course.
- **Anonymous Blocks:**
 - Anonymous blocks are unnamed blocks. They are declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at runtime. You can embed an anonymous block within a precompiler program and within SQL*Plus or Server Manager. Triggers in Oracle Developer components consist of such blocks.
- **Subprograms:**
 - Subprograms are named PL/SQL blocks that can take parameters and can be invoked. You can declare them either as procedures or as functions. Generally you use a procedure to perform an action and a function to compute a value.
 - You can store subprograms at the server or application level. Using Oracle Developer components (Forms, Reports, and Graphics), you can declare procedures and functions as part of the application (a form or report) and call them from other procedures, functions, and triggers within the same application whenever necessary.
 - Note that the only difference between a function and a procedure is that a function MUST return a value.
- **Use of Variables:**

- Temporary storage of data
- Manipulation of stored values
- Reusability
- Ease of maintenance
- **Handling Variables in PL/SQL:**
 - Declare and initialize variables in the declaration section
 - You can declare variables in the declarative part of any PL/SQL block, subprogram or package. Declarations allocate storage space for a value, specify its datatype, and name the storage location so that you can reference it. Declarations can also be assigned an initial value and impose the NOT NULL constraint.
 - Assign new values to variables in the executable section.
 - The existing value of the variable is replaced with a new one
 - Forward references are not allowed. You must declare a variable before referencing it in other statements, including other declarative statements.
 - Pass values into PL/SQL subprograms through parameters.
 - These are three parameter modes, IN (default), OUT, and IN OUT. You use the IN parameter to pass values to the subprogram being called. You use the OUT parameter to return values to the caller of a subprogram. And you use the IN OUT parameter to pass initial values to the subprogram being called and to return updated values to the caller. IN and OUT subprogram parameters are covered in another course.
 - View the results of a PL/SQL block through output variables.
- Types of Variables:
 - Scalar
 - Hold a single value. The main datatypes are those that correspond to column types in Oracle Server tables; PL/SQL also supports Boolean variables.
 - Composite
 - Datatypes such as records that allow groups of fields to be defined and manipulated in PL/SQL blocks.
 - Reference
 - These hold values called pointers that designate other program items. Reference datatypes are not covered in this course!
 - LOB (large objects)
 - Also called 'locators' specify the location of large objects (graphic images for example) that are stored out of line. LOB datatypes are only briefly mentioned in this course.
 - Non- PL/SQL variables: Bind and host variables.
 - Include host language variables declared in precompiler programs, screen fields in Form applications, and SQL*Plus host variables.
- SQL*Plus host (or 'bind') variables can be used to pass runtime values out of the PL/SQL block back to the SQL*Plus environment. You can reference them in a PL/SQL block with a preceding colon.
- Declaring PL/SQL Variables:

Identifier [CONSTANT] *datatype* [NOT NULL]
[:= | DEFAULT *expr*];

- You need to declare all PL/SQL identifiers in the declaration section before referencing them in the PL/SQL block. You have the option to assign an initial value.
 - *identifier*

- The name of the variable
 - CONSTANT
 - Constrains the variable so that its value cannot change; constants must be initialized.
 - *datatype*
 - is a scalar, composite, reference, or LOB datatype
 - NOT NULL
 - Constrains the variable so that it must contain a value.
 - *expr*
 - is any PL/SQL expression that can be literal, another variable, or an expression involving operators and functions.
- In CONSTANT declarations, the keyword CONSTANT must precede the type specifier. The following declaration names a constant of NUMBER subtype REAL and assigns the value of 50000 to the constant. A constant must be initialized in its declaration; otherwise, you get a compilation error when the declaration is elaborated (compiled).

```
v_sal CONSTANT REAL := 50000.00;
```

- Naming Rules:
 - Two variables can have the same name provided that they are located in different blocks.
 - The variable name used must not be a keyword or the name of a table column used in the block.
 - Name identifiers must not exceed 30 characters in length.
- Assigning Values to Variables
 - Use the ‘:=’ assignment operator...
- Note: To assign a value into a variable from the database, use a SELECT or FETCH statement.
- Initializing a NOT NULL variable:

```
v_location VARCHAR2(13) NOT NULL := 'CHICAGO';
```

- Initializing using the DEFAULT keyword:

```
g_manager NUMBER(4) DEFAULT 7839;
```

- Scalar Datatypes:
 - VARCHAR2(max_length)
 - NUMBER
 - DATE
 - CHAR
 - LONG
 - LONG RAW
 - BOOLEAN
 - BINARY_INTEGER
 - PLS_INTEGER
- The %TYPE attribute:
 - When you declare PL/SQL variables to hold column values, you must ensure that the variable is of the correct datatype and precision. If it is not a PL/SQL, error will occur during execution.

- Rather than hard coding the datatype and precision of a variable, you can use the %TYPE attribute to declare a variable according to another previously declared variable or database column. The %TYPE attribute is most often used when the value stored in the variable will be derived from a table in the database or if the variable is destined to be written to . To use the attribute in place of the datatype required in the variable declaration, prefix with the database table and column name. If referring to a previously declared variable, prefix the variable to the name of the attribute.
- PL/SQL determines the datatype and size of the variable when the block is compiled, so it is always compatible with the column used to populate it. This is a definite advantage for writing and maintaining code, because there is no need to be concerned with column datatype changes made at the database level. You can also declare a variable according to another previously declared variable by prefixing the variable with the name of the attribute.
- Declaring a Boolean expression:
 - Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable.

```
v_comm_sal BOOLEAN := (v_sal1 <v_sal2);
```

- In the previous example, we make use of an expression using a comparison operator to return a Boolean type value.
- **PL/SQL Record Structure:**
 - Composite datatypes (also known as collections) are TABLE, RECORD, NESTED TABLE, and VARRAY. You use the RECORD datatype to treat related but dissimilar data as a logical unit. You use the TABLE datatype to reference and manipulate collections of data as a whole object. Both RECORD and TABLE datatypes are covered in detail later. The NESTED TABLE and VARRAY datatypes are not covered in this course.
- LOB Datatype Variables Include:
 - CLOB
 - Book
 - BLOB
 - Photo
 - BFILE
 - Movie
 - NCLOB
 - The NCLOB (national language character large object) datatype is used to store large blocks of single-byte of fixed-width multi-byte NCHAR in the database, in line or out of line.
- **Bind Variables:**
 - A bind variable is a variable that you declare in a host environment and then use to pass runtime values, either number or character, into or out of one of more PL/SQL programs, which can use it as they would use any other variable.
 - To declare a bind variable in the SQL*Plus environment, you use the command VARIABLE. For example, you declare a variable of type NUMBER and VARCHAR2 as follows:

```
VARIABLE return_code NUMBER
VARIABLE return_msg VARCHAR2(30)
```

- Displaying Bind Variables:

- To display the current value of bind variables in the SQL*Plus environment, you use the command PRINT. However, PRINT cannot be used inside a PL/SQL block as a SQL*Plus command. The following example illustrates a PRINT command:

```
SQL> VARIABLE g_n NUMBER
...
SQL> PRINT g_n
```

- DBMS_OUTPUT.PUT_LINE
 - An Oracle-supplied package procedure
 - An alternative for displaying data from a PL/SQL block
 - Must be enabled in SQL*Plus with SET SERVEROUTPUT ON