PL / SQL Basics

Chapter 3
PL / SQL Basics

- PL / SQL block
- Lexical units
- Variable declarations
- PL / SQL types
- Expressions and operators
- PL / SQL control structures
- PL / SQL style guide
PL / SQL Block
Basic Block Structure

- All blocks have three (3) sections
  - Declarative
    - Where all variables, cursors, types, local procedures and functions are declared
  - Executable
    - Where the work of the block is performed
    - Contains SQL and procedural statements
  - Exception
    - Errors are handled
    - Code here is not run unless an error occurs
Lexical Units

- Identifiers
- Delimiters
- Literals
- Comments
Lexical Units

- A lexical unit is a sequence of characters, the character set includes:
  - Upper and lowercase letters: A-Z, a-z
  - White space: tabs, spaces, and carriage returns
  - Math symbols: +-*/<>=
  - Punctuation symbols: () {} [] ? ~ ; : . ’ “ @ # % $ ^ & _ |
Lexical Units
Identifiers

- Used to name PL/SQL objects
- Consists of a letter followed by any sequence of characters such as:
  - Letters, numbers, $, underscore, #
  - Maximum length is thirty (30) characters
  - Other characters are not permitted
Lexical Units

Identifiers ~ Examples

- Some proper identifiers:
  - y
  - v_StudentID
  - TempName
  - t1
  - t2_
  - social_security_
  - YTD
Lexical Units
Identifiers ~ Examples

- Some improper identifiers:
  - m+n
  - _temp_
  - First Name
  - OhBoyThisIsReallyALongIdentifierName
  - 1CannotStartWithADigit
Lexical Units
Identifiers ~ Reserved Words

- There are many identifiers known as reserved words (keywords)
- They have special meaning to PL/SQL
- You cannot use them as identifiers
  - For example the word BEGIN is used to start a PL/SQL block and cannot be used as a variable name
Lexical Units

Delimiters

- Symbols used to separate identifiers
  - either a single character
  - sequence of characters
- Examples of delimiters are:
  - + * > <> <= ; || (and others...)
Lexical Units

Literals

- A literal is a value that is not an identifier
- There are three (3) types of literals:
  - Character
    - ‘This is a character literal’
  - Numeric
    - 123  -9  +17  0  -6.9  3  9.87e-3  -95.12e7
  - Boolean
    - True  False  Null
Lexical Units

Comments

- It is important to document your code with comments to make it clear to others and yourself
- They are ignored by the PL/SQL compiler
- There are two kinds of comments
  - Single-line
  - Multiline
Variable Declarations

- Declaration syntax
- Variable initialization
Variable Declarations

Declaration Syntax

- Variables are declared in the declarative section of a block
- Each variable has a specific type of data that it can hold
- General syntax is:
  - `variable_name type [CONSTANT] [NOT NULL] [:=value];`
Variable Declarations
Variable Initialization

- Variables are initialized in their declaration statement
  - `v_Counter` NUMBER := 100;
  - `v_Fname` VARCHAR2(10) DEFAULT 'Scott';
- If a variable is not initialized it is set to NULL
- NULL simply means missing or unknown value
PL / SQL Types

- Scalar types
- Composite types
- Reference types
- LOB types
- Using %TYPE
- User-defined subtypes
- Converting between Datatypes
- Variable scope and visibility
PL / SQL Types
Scalar Types

- Valid scalar types consist of the same as the types for database columns
- Scalar types can be divided into seven families:
  - numeric
  - character
  - raw
  - date
  - rowid
  - boolean
  - trusted
PL / SQL Types

Composite Types

- A composite type is one that has components within it
- The three (3) composite types available in PL/SQL are:
  - Records
  - Tables
  - varrays
PL / SQL Types

Reference Types

- A variable that is declared as a reference type can point (refer to) different storage locations over the life of the program
  - REF CURSOR
  - REF OBJECT
PL / SQL Types
LOB Types

- This type is used to store large objects
- A large object can be either binary or character value up to 4 gigabytes
- Can contain unstructured data
PL / SQL Types
Using %TYPE

- Often a PL/SQL variable will be used to manipulate data stored in a database table
  - Variable needs to have the same type as the table column
  - Variable needs to be changed as the table column is altered
  - Time consuming and error prone
PL / SQL Types
Using %TYPE

- By using %TYPE when declaring a variable
  - Variable will be defined with the same type as the associated table column
  - Type is determined each time a block is run or
  - When procedures and functions are compiled
PL / SQL Types

User-Defined Subtypes

- A subtype is a PL/SQL type that is based on an existing type.
- Used to give an alternative name which describes its intended use.
- PL/SQL defines several subtypes.
- DECIMAL and INTEGER are predefined subtypes of NUMBER.
Explicit Scalar Datatype Conversion

- Built-in conversion functions in SQL are also available in PL/SQL
- Converts explicitly between variables using formats

Examples are:
- TO_CHAR - converts numeric and date
- TO_DATE - converts character
- TO_NUMBER - converts character
PL / SQL Types

Converting Between Datatypes

- Implicit Scalar Datatype Conversion
  - PL/SQL will convert between families when possible
  - Good programming practice suggests using explicit conversions
  - Formats are not used and does not clearly show the intent of the program
PL / SQL Types
Variable Scope and Visibility

- The scope of a variable is the portion of the program in which it can be accessed.
- The visibility of a variable is the portion of the program where the variable can be accessed without having to qualify the reference.
Expressions and Operators

- Assignment
- Expressions
Expressions and Operators
Assignment

- The syntax for an assignment is:
  - variable := expression;
- Performed in the executable and exception handling sections
- lvalues must refer to actual storage locations
- rvalues can be storage locations or literals
- An rvalue is read from while an lvalue is written to
Expressions and Operators

Expressions

- PL/SQL expressions are rvalues
- These expressions have two components
  - An operand is the argument to an operator
  - An operator is what operates on the operands
- Precedence of operators determines the order of evaluation of the expression
  - Please Excuse My Dear Aunt Sally
Expressions and Operators

Expressions

- Numeric expressions
  - Evaluated from left to right applying the precedence order of operators

- Character expressions
  - Only operator is concatenation operator `||`
  - Joins one or more strings (or arguments that can be implicitly converted)

- Boolean expressions
  - A boolean expression is any expression that evaluates to a boolean value (True, False, Null)
PL / SQL Control Structures

- IF-THEN-ELSE
- Loops
- GOTOs and Labels
- PRAGMAs
PL / SQL Control Structures

IF-THEN-ELSE

The syntax for an IF-THEN-ELSE statement is:

```pl/sql
IF boolean_expression1 THEN
    sequence_of_statements1;
[ELSIF boolean_expression2 THEN
    sequence_of_statements2;]
...
[ELSE
    sequence_of_statements3;] END IF;
```
Loops are divided into four (4) categories:

- Simple loops
- WHILE loops
- Numeric FOR loops
- Cursor FOR loops
PL / SQL Control Structures
Loops ~ Simple Loops

- The basic kind of loop has the syntax:
  
  ```
  LOOP
  sequence_of_statements;
  END LOOP;
  ```

- These loops will execute forever unless you include an EXIT statement
  - Checks a condition for ending the loop since it has no stopping condition, its syntax is:
    ```
    EXIT [WHEN condition];
    ```
PL / SQL Control Structures
Loops ~ WHILE Loops

- The syntax for a WHILE loop is:
  ```plsql
  WHILE condition LOOP
    sequence_of_statements;
  END LOOP;
  ```

- Condition is checked before each cycle of the loop
  - If True, loop body is executed
  - Otherwise loop is passed over
WHILE LOOP Example

DECLARE v_a number(2) := 1;
BEGIN
WHILE v_a < 10 LOOP
    dbms_output.put_line('value of v_a: ' || v_a);
    v_a := v_a + 1;
END LOOP;
END; /

PL / SQL Control Structures
Loops ~ Numeric FOR Loops

- The syntax for a numeric FOR loop is:

```plsql
FOR loop_counter IN [REVERSE] lo_bound .. hi_bound LOOP
    sequence_of_statements;
END LOOP;
```

- Numeric FOR loops have a defined number of cycles
- Do not declare the loop index
- Loop increment (decrement) is always one
- In order to have a different increment you would need to include additional code in the loop body
PL / SQL Control Structures
Loops ~ Cursor FOR Loops

- A cursor FOR loop is a loop that is associated:
  - With an explicit cursor
  - A SELECT statement incorporated directly within the loop boundary
- Used when you need to fetch and process records from a cursor
- Replaces the cursor operations such as:
  - Open
  - Fetch
  - End of records
  - Close
CREATE OR REPLACE PROCEDURE PrintStudents(
p_Major IN students.major%TYPE) AS
CURSOR c_Students IS
    SELECT first_name, last_name
    FROM students
    WHERE major = p_Major;
BEGIN
    FOR v_StudentRec IN c_Students LOOP
        DBMS_OUTPUT.PUT_LINE(v_StudentRec.first_name || ' ' ||
            v_StudentRec.last_name);
    END LOOP;
END;
/

The syntax of a GOTO statement is:

GOTO label;

- Where label is a label defined in the PL/SQL block
- Labels are enclosed in double angle brackets

When a GOTO statement is evaluated, program control passes to the statement identified by the label

If your code is properly structured you will never have to use a GOTO statement
PL / SQL Control Structures

PRAGMAs

- The PRAGMA keyword signifies a preprocessor statement
  - PRAGMAs are processed at compile time
  - They do not execute during runtime
  - Passes information to the compiler
Example: Types of Pragmas

- Until Oracle 11g, PL/SQL programming language could make use of five types of pragma(s) which are listed as below.
- AUTONOMOUS_TRANSACTION - Compiler allows schema objects like subprograms, PL/SQL blocks, or triggers to commit their transaction within their scope.
- RESTRICT_REFERENCES - Allows the compiler to scan the database purity during package compilation. It can be used in packages only.
- EXCEPTION_INIT - Compiler associates a user defined exception name with an exception number.
- SERIALLY_REUSABLE - Compiler deduces the package state from session level to just a server call. It can be used in Package spec only.
- PRAGMA_INLINE - New member of oracle pragma family. It is used to explicitly inline a program unit.
PL / SQL Style Guide

- Good style means that:
  - It will take less time to understand what the program is doing
  - Modifications can be made easier and will be less error prone
  - Uniformity of code increases productivity

- Areas of where style can be applied are:
  - Comments
  - Variable Names
  - Capitalization
  - Indentation
  - General Guidelines
PL / SQL Style Guide

Comments

- At the start of each block or procedure
- With each variable declaration
- Before each major section of a block
  - Comments should be meaningful and not restate what the code obviously shows
  - It’s possible to have too many comments
PL / SQL Style Guide

Variable Names

- Make the variable names descriptive
  
  ```plsql
  t  NUMBER;
  v_StudentID  NUMBER(5);
  ```

- A variable name can tell us its use:
  
  ```plsql
  v_VariableName  Program variable
  e_ExceptionName  User-defined exception
  t_TypeName  User-defined type
  p_ParameterName  Parameter to a procedure or function
  ```
PL / SQL Style Guide

Capitalization

- PL/SQL is not case sensitive
- Proper use of case will improve program readability
  - reserved words in uppercase
  - built-in functions in uppercase
  - predefined types in uppercase
  - sql keywords in uppercase
  - database objects in lowercase
  - variable names are in mixed case, with capital letter for each word in the name
PL / SQL Style Guide

Indentation

- Use of white space (carriage returns, spaces, and tabs) is necessary to provide readability of your code

- Indent:
  - inside blocks
  - SQL continuation statements
  - IF-THEN-ELSE blocks
As you write more code you will develop a pronounced level of style.

It's a good idea to show your code to another programmer and have it reviewed.

Be consistent in making your code understandable.
In Conclusion

- We have covered the basic syntax and structure of PL/SQL
  - Block
  - Variables
  - Datatypes (scalar, composite, reference)
  - Expressions and operators
  - Datatype conversions
  - Control structures
  - Programming style