ISYS 365 - PL/SQL Basics

Week 03
Agenda

- PL/SQL Block Structure
- Declaration Syntax
- Variable Scope
- IF-THEN-ELSE
- CASE
PL/SQL Block Structure

- Basic building block of PL/SQL programs
  - Three possible sections of a block
    - Declarative section (optional)
    - Executable section (required)
      - Delimiters: BEGIN, END
    - Exception handling (optional)

- Blocks can be
  - Anonymous
  - Named
Declaration Syntax

Syntax

- variable_name type [CONSTANT] [NOT NULL] [:= initial value];
  - If an initial value is not assigned, then the initial value is NULL
  - If the NOT NULL is used, then an initial value MUST be defined
  - Can only declare one variable per line

Example:

DECLARE -- anonymous block
  v_Description VARCHAR2(50);
  v_NumberSeats NUMBER := 45;
  v.Counter BINARY_INTEGER :=0;
  v.FirstName VARCHAR2(20) DEFAULT 'Scott';
  v.MinimumStudentID CONSTANT NUMBER(5) := 10000;
  v.PI CONSTANT NUMBER(3,2) := 3.14;
BEGIN
  -- process data here
EXCEPTION
  -- Error handling would go here
END;/
Declaration Syntax

The following are ILLEGAL variable declarations

Example:

```
DECLARE
    v_TempVar        NUMBER NOT NULL;
    v_FirstName, v_LastName  VARCHAR2(20);
BEGIN
    -- process data here
EXCEPTION
    -- Error handling would go here
END;
/
```

Why are the above variable declarations illegal?
How can the above illegal variable declarations be corrected?
PL/SQL variables are often used to manipulate data in a database. So, the data type should be the same as the table column.

```
DECLARE
  v_FirstName   VARCHAR2(20);
  v_LastName    VARCHAR2(20);
END;
```

What happens if the table was altered such that last_name is now VARCHAR2(25)?

Rather than hardcode the variable type, use the %TYPE attribute.

```
DECLARE
  v_FirstName   students.first_name%TYPE;
  v_LastName    students.last_name%TYPE;
END;
```
Variable Scope

- Variable scope
  - The portion of the program in which the variable can be accessed
  - The PL/SQL engine will free the memory used for the variable when a variable goes out of scope

```
DECLARE
  v_Number NUMBER(3,2);
BEGIN
  DECLARE
    v_Character VARCHAR2(10);
    ...
    END;
END;
```
IF boolean_expression1 THEN
  sequence_of_statements;
[ELSIF boolean_expression2 THEN
  sequence_of_statements]
[ELSE
  sequence_of_statements]
END IF;

Note for the use of **ELSIF**, not **ELSEIF**
See examples!
DECLARE
    v_Major    students.major%TYPE;
    v_CourseName VARCHAR2(10);
BEGIN
    -- Retrieve the major for a given student
    SELECT major
        INTO v_Major
    FROM students
    WHERE ID = 10011;

    -- Based on the major, choose a course
    IF v_Major = 'Computer Science' THEN
        v_CourseName := 'CS 101';
    ELSIF v_Major = 'Economics' THEN
        v_CourseName := 'ECN 203';
    ELSIF v_Major = 'History' THEN
        v_CourseName := 'HIS 101';
    ELSIF v_Major = 'Music' THEN
        v_CourseName := 'MUS 100';
    ELSIF v_Major = 'Nutrition' THEN
        v_CourseName := 'NUT 307';
    ELSE
        v_CourseName := 'Unknown';
    END IF;

    DBMS_OUTPUT.PUT_LINE(v_CourseName);
END;/
CASE

- Anything written as an IF-THEN-ELSE construct can be written as a CASE statement
- Form 1: specify test expression after CASE keyword

```plaintext
CASE test_var
    WHEN value1 THEN sequence_of_statements1;
    WHEN value2 THEN sequence_of_statements2;
    ...
    WHEN valuen THEN sequence_of_statementsn;
    [ELSE else_sequence;]
END CASE;
```
See examples!
DECLARE
    v_Major    students.major%TYPE;
    v_CourseName VARCHAR2(10);
BEGIN
    -- Retrieve the major for a given student
    SELECT major       INTO v_Major
    FROM students    WHERE ID = 10011;

    -- Based on the major, choose a course
    CASE  v_Major
        WHEN 'Computer Science' THEN
            v_CourseName := 'CS 101';
        WHEN 'Economics' THEN
            v_CourseName := 'ECN 203';
        WHEN 'History' THEN
            v_CourseName := 'HIS 101';
        WHEN 'Music' THEN
            v_CourseName := 'MUS 100';
        WHEN 'Nutrition' THEN
            v_CourseName := 'NUT 307';
        ELSE
            v_CourseName := 'Unknown';
    END CASE;

    DBMS_OUTPUT.PUT_LINE(v_CourseName);
END;/

DECLARE
    v_TestVar NUMBER := 1;
BEGIN
    -- This CASE statement is labeled.
    <<MyCase>>
    CASE v_TestVar
    WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('One!');
    WHEN 2 THEN DBMS_OUTPUT.PUT_LINE('Two!');
    WHEN 3 THEN DBMS_OUTPUT.PUT_LINE('Three!');
    WHEN 4 THEN DBMS_OUTPUT.PUT_LINE('Four!');
    END CASE MyCase;
END;
/

CASE

Form 2: no test expression after the CASE keyword

CASE

  WHEN boolean_expression1 THEN
  sequence_of_statements;

  WHEN boolean_expression2 THEN
  sequence_of_statements;

  ...

  [ELSE
  sequence_of_statements; ]

END CASE;

See examples!
DECLARE
    v_Test1 NUMBER := 2;
    v_Test2 VARCHAR2(20) := 'Goodbye';
BEGIN
    CASE
        WHEN v_Test1 = 1 THEN
            DBMS_OUTPUT.PUT_LINE('One!');
            DBMS_OUTPUT.PUT_LINE('Another one!');
        WHEN v_Test1 > 1 THEN
            DBMS_OUTPUT.PUT_LINE('> 1!');
            DBMS_OUTPUT.PUT_LINE('Still > 1!');
        WHEN v_Test2 = 'Goodbye' THEN
            DBMS_OUTPUT.PUT_LINE('Goodbye!');
            DBMS_OUTPUT.PUT_LINE('Adios!');
        ELSE
            DBMS_OUTPUT.PUT_LINE('No match');
    END CASE;
END;
/
 Agenda

- LOOPS
  - Simple Loops
  - WHILE Loops
  - Numeric FOR Loops
  - Class Exercises

- Records
Summary of last class

- PL/SQL block
  - Sections
  - Anonymous vs. Named
- Variables
  - Declaration, Anchored declaration
  - Variable Scope
- IF-THEN-ELSE
- CASE
PL/SQL Loops

- Used to execute a sequence of statements repeatedly

- When the number of iterations is unknown
  - Simple loops: executes at least once
  - WHILE loops: executes while the condition is true

- When the number of iterations is known in advance
  - Numeric FOR Loops: executes a specific number of times
Simple Loops

Syntax:

```
DECLARE
    v_Counter  BI NARY_ I NTEGER :=0;
BEGIN
    LOOP
        sequence_of_statements;
        EXIT WHEN condition;
    END LOOP;
EXCEPTION
    -- Error handling would go here
END;
/
```

See example!
Example

REM This is an example of a simple loop.

DECLARE
  v_Counter BINARY_INTEGER := 1;
BEGIN
  LOOP
    -- Insert a row into temp_table with the current value of the loop counter.
    INSERT INTO temp_table
    VALUES (v_Counter, 'Loop index');
    v_Counter := v_Counter + 1;
    -- Exit condition - when the loop counter > 50 we will break out of the loop.
    IF v_Counter > 50 THEN
      EXIT;
    END IF;
  END LOOP;
END;
/

WHILE Loops

Syntax:

DECLARE
    v_Counter  BIT RARY_INTEGER := 1;
BEGIN
    WHILE condition LOOP
        sequence_of_statements;
        [EXIT WHEN exit_condition;]
    END LOOP;
EXCEPTION
    -- Error handling would go here
END;
/

See example!
Example

ORDER"REM while1.sql

DECLARE
  v_Counter BINARY_INTEGER := 1;
BEGIN
  -- Test the loop counter before each loop iteration to
  -- insure that it is still less than 50.
  WHILE v_Counter <= 50 LOOP
    INSERT INTO temp_table
    VALUES (v_Counter, 'Loop index');
    v_Counter := v_Counter + 1;
  END LOOP;
END;
/

Example 2

REM while2.sql (Chapter 3, Oracle9i PL/SQL Programming by Scott Urman)

REM The WHILE loop in this example has a NULL condition—error!

DECLARE
  v_Counter BINARY_INTEGER;
BEGIN
  -- This condition will evaluate to NULL, since v_Counter
  -- is initialized to NULL by default.
  WHILE v_Counter <= 50 LOOP
    INSERT INTO temp_table
    VALUES (v_Counter, 'Loop index');
    v_Counter := v_Counter + 1;
  END LOOP;
END;
/

Numeric FOR Loops

- Syntax:
  -- Note that index variable MUST NOT be declared

BEGIN
  FOR loop_counter IN [REVERSE] low_bound..high_bound LOOP
    sequence_of_statements;
    [EXIT WHEN exit_condition;]
    END LOOP;
  EXCEPTION
    -- Error handling would go here
  END;
/

Numeric FOR Loops

- Do not explicitly declare loop index in FOR loops

- If the REVERSE keyword is used, the low_bound is still listed before the high_bound

- Low_bound and high_bound can be any expression that can evaluate to a numeric value

See examples!
Example

- REM forloop.sql
- REM Chapter 3, Oracle9i PL/SQL Programming by Scott Urman
- REM This is an example of a FOR loop.

- BEGIN
- FOR v_Counter IN 1..50 LOOP
- INSERT INTO temp_table
-  VALUES (v_Counter, 'Loop Index');
- END LOOP;
- END;
- /


Example

DECLARE
  v_Counter NUMBER := 7;
BEGIN
  -- Inserts the value 7 into temp_table.
  INSERT INTO temp_table (num_col)
  VALUES (v_Counter);
  -- This loop redeclares v_Counter as a BINARY_INTEGER, which hides
  -- the NUMBER declaration of v_Counter.
  FOR v_Counter IN 20..30 LOOP
    -- Inside the loop, v_Counter ranges from 20 to 30.
    INSERT INTO temp_table (num_col)
    VALUES (v_Counter);
  END LOOP;
  -- Inserts another 7 into temp_table.
  INSERT INTO temp_table (num_col)
  VALUES (v_Counter);
END;
/

Find error!

- BEGIN
- FOR v_Counter IN 20..30 LOOP
  - -- Inside the loop, v_Counter ranges from 20 to 30.
  - INSERT INTO temp_table (num_col)
  - VALUES (v_Counter);
  - END LOOP;
- INSERT INTO temp_table (num_col)
  - VALUES (v_Counter);
- END;
- /
Exercises (Brief)

In-class exercises:

- WHILE loop
  - Write an anonymous PL/SQL WHILE loop in which you...
    - declare a counter variable with an initial value of 10
    - For each iteration, output a message that states the words ‘Loop is at ’ followed by the iteration number
    - Exit the WHILE loop when the counter is greater than 20

- Numeric FOR loop
  - Write a named PL/SQL block that contains a numeric FOR loop in which you...
    - Output a message that states the words ‘Loop is at ‘ followed by the iteration number
    - The loop goes from 10 to 20
**Records**

- User-defined composite types

- Provides a way to treat separate but logically related variables as a unit

- Similar to “structures” in C

- Use the dot notation to refer to fields within a record
  - `v_StudentInfo.FirstName := ‘John’;`

- In order to copy one record to another record, both records must be of the same record type
- Syntax:

```sql
DECLARE
  TYPE t_StudentRecord IS RECORD(
    Student_ID NUMBER(5),
    FirstName VARCHAR2(20),
    LastName VARCHAR2(20));

  v_StudentInfo t_StudentRecord;
BEGIN
  -- Process data here
  EXCEPTION
    -- Error handling would go here
END;
/*
Use the dot notation to refer to fields within a record.
Example:
```
```sql
v_StudentInfo.FirstName := 'John';
```
To declare a record with the same field types as a database row, use %ROWTYPE

Syntax:

```sql
DECLARE
    v_StudentInfo student%ROWTYPE;
BEGIN
    -- Process data here
EXCEPTION
    -- Error handling would go here
END;
/
```

**NOTE:**

Any NOT NULL constraints that are defined for a column within the table will not be applicable to the declared record when %ROWTYPE is used.
DECLARE
-- Define a record to match some fields in the students table. -- Note the use of %TYPE for the fields.

    TYPE t_StudentRecord IS RECORD (
    FirstName  students.first_name%TYPE,
    LastName   students.last_name%TYPE,
    Major      students.major%TYPE);

-- Declare a variable to receive the data.
    v_Student  t_StudentRecord;

BEGIN
-- Retrieve information about student with ID 10,000.
-- Note how the query is returning columns which match the fields in v_Student.

    SELECT first_name, last_name, major
    INTO v_Student
    FROM students
    WHERE ID = 10000;
END;
Illegal Assignment

- REM This block shows legal and illegal record assignments.

DECLARE

- TYPE t_Rec1Type IS RECORD
  - Field1 NUMBER,
  - Field2 VARCHAR2(5));

- TYPE t_Rec2Type IS RECORD
  - Field1 NUMBER,
  - Field2 VARCHAR2(5));

- v_Rec1 t_Rec1Type;
- v_Rec2 t_Rec2Type;

BEGIN

  /* Even though v_Rec1 and v_Rec2 have the same field names
   and field types, the record types themselves are different.
   This is an illegal assignment which raises PLS-382. */

  v_Rec1 := v_Rec2;

  /* However, the fields are the same type, so the following
   are legal assignments. */

  v_Rec1.Field1 := v_Rec2.Field1;
  v_Rec2.Field2 := v_Rec2.Field2;

END;