PL/SQL Block Structure | Programming Constructs

Subject 2: Oracle PL/SQL

Chapter 2.2: Block Structure & Programming Constructs

Topic 2.2.1: Basics - PL/SQL Programs

Concept 1: Hello World Program | Attributes | Composite Datatypes

Simple Example:

```
Write a program to display the message on to the screen?
set serveroutput on
begin
dbms_output.put_line('Hello World');
dbms_output.put_line('Welcome to Oracle Programming');
end;
```

1. Anchor Datatypes (Attributes)

* Allow variables to inherit datatypes from database objects or other variables.

Types

1. %TYPE → inherits datatype of a column/variable.
v_name employees.first_name%TYPE;

2. %ROWTYPE → inherits entire row structure (record).
v_emp employees%ROWTYPE;

2. Composite Datatypes

* Datatypes made up of multiple values/components.

Types

1. Record (row-like structure, fields of different types).

```
TYPE emp_record IS RECORD (
emp_id employees.employee_id%TYPE,
emp_name employees.first_name%TYPE
);
v_emp emp_record;
```

2. Table/Associative Array (index-value pairs).
3. VARRAY (bounded array of elements).

3. Collections

Collections

* Group of elements of the same datatype.
* Stored in memory, used in bulk operations.

Types:

1. Associative Arrays (Index-by tables)
2. Nested Tables
3. VARRAY (Variable-Size Array)

Collections: Types

1. Associative Arrays (Index-by tables)

* Indexed by integer or string.
* No fixed size.

```
TYPE num_list IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
v_tab num_list;
v_tab(1) := 100;
```

2. Nested Tables

* Can be stored in database tables.
* Unbounded.

```
TYPE str_list IS TABLE OF VARCHAR2(20);
v_nt str_list := str_list('A', 'B', 'C');
```

3. VARRAY (Variable-Size Array)

* Bounded, ordered collection.

```
TYPE grade_list IS VARRAY(5) OF VARCHAR2(2);
v_grades grade_list := grade_list('A', 'B');
```

Subject 2: Oracle PL/SQL

Chapter 2.2: Block Structure & Programming Constructs

Topic 2.2.1: Basics - PL/SQL Programs

Concept 2: PL/SQL Block Structure & Designing

PL/SQL Block Structure & Designing

General Structure

```
DECLARE
-- variables, constants, cursors
BEGIN
-- executable statements
EXCEPTION
-- error handling
END;
```

Types of Blocks

* Anonymous block (temporary, not stored).
* Named block (procedures, functions, packages, triggers).
* Nested block (blocks inside other blocks).

11. Scope and Visibility

* Scope → where a variable can be referenced in a program.
* Visibility → whether a variable is accessible inside nested blocks.

Rules

* Variables declared in outer block → visible to inner block.
* Variables declared in inner block → NOT visible to outer block.
* If inner block redeclares a variable → shadows outer variable.

Example:

```
DECLARE
v_num NUMBER := 10;  -- outer
BEGIN
DECLARE
v_num NUMBER := 20;  -- inner shadows outer
BEGIN
DBMS_OUTPUT.PUT_LINE(v_num);  -- prints 20
END;
DBMS_OUTPUT.PUT_LINE(v_num);     -- prints 10
END;
```

Constructs of PL/SQL

These are the building blocks of PL/SQL programming.

a) Variables & Constants

v_name VARCHAR2(20) := 'Oracle';
c_tax CONSTANT NUMBER := 0.18;

b) Control Statements

* Conditional: IF-THEN-ELSE, CASE
* Looping: LOOP, FOR, WHILE

Example:

```
FOR i IN 1..5 LOOP
DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
```

c) Cursors

* Control row-by-row processing of queries.
Example:

CURSOR c IS SELECT first_name FROM employees;

d) Exception Handling

* Handles errors gracefully.

Example:

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No record found');
```

Highlights: Features of PL/SQL over SQL
PL/SQL extends SQL with procedural constructs.
SQL alone is non-procedural → PL/SQL adds loops, conditions, variables, exceptions.
PL/SQL is essential for business logic, error handling, modular programming, and performance optimization.
Key tools: Anchored datatypes (%TYPE, %ROWTYPE), Composite types, Collections, Blocks, Scope, and Constructs.

Subject 2: Oracle PL/SQL
Chapter 2.2: Block Structure & Programming Constructs
Topic 2.2.2: PL/SQL Programming Constructs
Concept 1: Assignment Operations | Debugging Statements

1. Assignment Operations

Definition

* Assignment is used to store a value in a variable or constant.
* Done using the := operator (NOT = like in SQL).

Syntax

```
variable_name := expression;
```

Example

```
DECLARE
v_salary NUMBER(10,2);
v_bonus  NUMBER(10,2);
BEGIN
v_salary := 50000;
v_bonus := v_salary * 0.10;  -- 10% bonus
DBMS_OUTPUT.PUT_LINE('Bonus = ' || v_bonus);
END;
```

■ Note: Constants cannot be reassigned after initialization.

2. Debugging Statement

* PL/SQL does not have a built-in debugger in basic SQL*Plus, so debugging is usually done with output statements.
* The main tool is DBMS_OUTPUT.PUT_LINE.

Syntax
DBMS_OUTPUT.PUT_LINE('Message: ' || variable);

Example

```
DECLARE
v_name VARCHAR2(20) := 'Debugging Statement';
BEGIN
DBMS_OUTPUT.PUT_LINE('Debugging: Current value of v_name is ' || v_name);
END;
```

Highlights:
■ Helps to trace values and program flow.

Subject 2: Oracle PL/SQL
Chapter 2.2: Block Structure & Programming Constructs
Topic 2.2.2: PL/SQL Programming Constructs
Concept 2: Flow Control Statements

Flow Control Statements

PL/SQL provides multiple ways to control the flow of execution:

* Conditional → IF, CASE
* Unconditional jump → GOTO, EXIT
* Iterative (loops) → LOOP, WHILE, FOR

1. IF Statements

a) Simple IF

Syntax:
IF condition THEN
statements;
END IF;

Example:

```
DECLARE
v_sal NUMBER := 60000;
BEGIN
IF v_sal > 50000 THEN
DBMS_OUTPUT.PUT_LINE('High Salary');
END IF;
END;
```

b) IF-ELSE

Syntax:
IF condition THEN
statements1;
ELSE
statements2;
END IF;

Example:

```
DECLARE
v_sal NUMBER := 30000;
BEGIN
IF v_sal > 50000 THEN
DBMS_OUTPUT.PUT_LINE('High Salary');
ELSE
DBMS_OUTPUT.PUT_LINE('Normal Salary');
END IF;
END;
```

c) IF-ELSIF Ladder

Syntax:
IF condition1 THEN
statements1;
ELSIF condition2 THEN
statements2;
ELSE
statements3;
END IF;

Example:

```
DECLARE
v_marks NUMBER := 72;
BEGIN
IF v_marks >= 90 THEN
DBMS_OUTPUT.PUT_LINE('Grade A');
ELSIF v_marks >= 75 THEN
DBMS_OUTPUT.PUT_LINE('Grade B');
ELSE
DBMS_OUTPUT.PUT_LINE('Grade C');
END IF;
END;
```

d) Nested IF

* An IF inside another IF.

Example:

```
DECLARE
v_sal NUMBER := 60000;
v_dept NUMBER := 10;
BEGIN
IF v_sal > 50000 THEN
IF v_dept = 10 THEN
DBMS_OUTPUT.PUT_LINE('Manager in Sales Dept');
END IF;
END IF;
END;
```

2. EXIT Statement

* Used to terminate a loop prematurely.

Syntax
EXIT; -- unconditional exit
EXIT WHEN condition; -- conditional exit

Example

```
DECLARE
v_counter NUMBER := 1;
BEGIN
LOOP
DBMS_OUTPUT.PUT_LINE('Counter: ' || v_counter);
v_counter := v_counter + 1;
EXIT WHEN v_counter > 5;  -- stops after 5
END LOOP;
END;
```

3. GOTO Statement

* Used for unconditional jumps inside a block.
* Should be avoided in structured programming (use loops/IF instead).

Syntax:
GOTO label_name;
<<label_name>>
statements;

Example

```
DECLARE
v_num NUMBER := 1;
BEGIN
IF v_num = 1 THEN
GOTO print_label;
END IF;

DBMS_OUTPUT.PUT_LINE('This will be skipped');

<<print_label>>
DBMS_OUTPUT.PUT_LINE('Goto executed');
END;
```

Subject 2: Oracle PL/SQL

Chapter 2.2: Block Structure & Programming Constructs

Topic 2.2.2: PL/SQL Programming Constructs

Concept 3: Iterative Statements | CONTINUE Statement

Iterative Statements

1. PL/SQL provides 3 main types of loops:

a) Simple Loop

* Executes at least once.
* Must have an EXIT condition.

Syntax:
LOOP
statements;
EXIT WHEN condition;
END LOOP;

Example

```
DECLARE
v_num NUMBER := 1;
BEGIN
LOOP
DBMS_OUTPUT.PUT_LINE('Number: ' || v_num);
v_num := v_num + 1;
EXIT WHEN v_num > 5;
END LOOP;
END;
```

b) WHILE Loop

* Executes only if condition is true.

Syntax:
WHILE condition LOOP
statements;
END LOOP;

Example

```
DECLARE
v_num NUMBER := 1;
BEGIN
WHILE v_num <= 5 LOOP
DBMS_OUTPUT.PUT_LINE('Number: ' || v_num);
v_num := v_num + 1;
END LOOP;
END;
```

c) FOR Loop

* Used when number of iterations is known in advance.

Syntax:
FOR counter IN lower..upper LOOP
statements;
END LOOP;

Example

```
BEGIN
FOR i IN 1..5 LOOP
DBMS_OUTPUT.PUT_LINE('Iteration ' || i);
END LOOP;
END;
```

d) Reverse loop:

Syntax:
FOR i IN REVERSE 5..1 LOOP
DBMS_OUTPUT.PUT_LINE(i);
END LOOP;

2. CONTINUE Statement

* Skips current iteration of loop and goes to next iteration.
* Available from Oracle 11g onwards.

Syntax:
CONTINUE;
CONTINUE WHEN condition;

Example

```
BEGIN
FOR i IN 1..5 LOOP
CONTINUE WHEN i = 3;  -- skips 3
DBMS_OUTPUT.PUT_LINE('Value: ' || i);
END LOOP;
END;
```

Subject 2: Oracle PL/SQL

Chapter 2.2: Block Structure & Programming Constructs

Topic 2.2.2: PL/SQL Programming Constructs

Concept 4: Summary - Programming Constructs

■ Summary

Statement	Purpose	Example Use
:=	Assignment	v_sal := 50000;
DBMS_OUTPUT	Debugging	PUT_LINE statement
IF	Condition	Decision making
EXIT	Break loop	Exit after 5 iterations
GOTO	Jump	Move to label
LOOP	Iteration	Infinite unless exited
WHILE	Iteration with condition	Execute while x < 10
FOR	Iteration with counter	FOR i IN 1..5
CONTINUE	Skip iteration	Skip i=3