Chapter 5

Advanced Conditional Control in PL/SQL

Mayko Freitas da Silva

Introduction

In this chapter, you will dive deeper into conditional control structures in PL/SQL. You will explore CASE statements, CASE expressions, and two useful functions: NULLIF and COALESCE. These tools will help you write more efficient and flexible code when working with Oracle databases.

In this chapter, you will learn about:

- CASE statements and their variations
- CASE expressions and their applications
- NULLIF and COALESCE functions

Lab 5.1: CASE Statements

After this lab, you will be able to:

- Use simple CASE statements
- Use searched CASE statements
- Understand the differences between CASE and searched CASE statements

Understanding CASE Statements

Imagine you're a traffic controller managing a busy intersection. The traffic light is your selector, and the actions drivers take are like the different WHEN clauses in a CASE statement. Let's explore how this works in PL/SQL.

A simple CASE statement looks like this:

```
CASE
   WHEN condition1 THEN result1;
   WHEN condition2 THEN result2;
   ...
   ELSE default_result;
END CASE;
```

Let's see this in action with an example using the HR schema:

```
DECLARE
   v_job_id VARCHAR2(10) := 'IT_PROG';
   v_job_category VARCHAR2(50);
BEGIN
   CASE v_job_id
      WHEN 'IT_PROG' THEN v_job_category := 'Technology';
      WHEN 'SA_REP' THEN v_job_category := 'Sales';
      WHEN 'FI_ACCOUNT' THEN v_job_category := 'Finance';
      ELSE v_job_category := 'Other';
   END CASE;

   DBMS_OUTPUT.PUT_LINE('Job Category: ' || v_job_category);
END;
```

In this example, you're categorizing jobs based on their job ID. The job ID is your selector, and each WHEN clause checks for a specific job ID and assigns a category.

Searched CASE Statements

Now, let's look at a searched CASE statement. It's like a more flexible traffic system where you can make decisions based on multiple factors, not just the color of the light.

A searched CASE statement has this structure:

```
CASE
   WHEN condition1 THEN result1;
   WHEN condition2 THEN result2;
   ...
   ELSE default_result;
END CASE;
```

Here's an example using employee salaries from the HR schema:

```
DECLARE
   v_employee_id NUMBER := 103;
   v_salary NUMBER;
   v_salary_category VARCHAR2(20);
BEGIN
   SELECT salary INTO v_salary
   FROM employees
   WHERE employee_id = v_employee_id;

   CASE
      WHEN v_salary < 5000 THEN v_salary_category := 'Low';
      WHEN v_salary BETWEEN 5000 AND 10000 THEN v_salary_category := 'Medi
um';
      WHEN v_salary > 10000 THEN v_salary_category := 'High';
      ELSE v_salary_category := 'Unknown';
   END CASE;

   DBMS_OUTPUT.PUT_LINE('Salary Category: ' || v_salary_category);
END;
```

In this searched CASE statement, you're categorizing an employee's salary without needing a specific selector value.

Comparing CASE and Searched CASE Statements

The main difference between CASE and searched CASE statements is flexibility. Simple CASE statements compare a selector against specific values, while searched CASE statements can evaluate complex conditions.

Best Practices for CASE Statement Order

Remember, CASE statements evaluate conditions in order. It's like a line of people waiting for a bus - the first person who meets the criteria gets on. Always put your most specific conditions first and more general ones later.

Lab 5.2: CASE Expressions

After this lab, you will be able to:

- Use CASE expressions in SQL queries
- Understand the syntax differences between CASE statements and expressions

Understanding CASE Expressions

Think of a CASE expression as a vending machine. You input a code (the selector or condition), and it returns a specific item (the result of the expression). Unlike CASE statements, CASE expressions return a value that can be used in a larger query or assignment.

Here's the basic structure of a CASE expression:

```
CASE [selector]
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   ELSE default_result
END
```

Let's see a CASE expression in action using the HR schema:

```
SELECT
   employee_id,
   first_name,
   last_name,
   CASE
      WHEN department_id = 60 THEN 'IT'
      WHEN department_id = 80 THEN 'Sales'
      WHEN department_id = 100 THEN 'Finance'
      ELSE 'Other'
   END AS department_category
FROM employees
WHERE ROWNUM <= 5;
```

This query categorizes employees based on their department ID, returning the category as part of the SELECT statement.

Using CASE Expressions in Queries

CASE expressions are particularly useful in SELECT statements, WHERE clauses, and ORDER BY clauses. Here's an example that uses a CASE expression to order employees by their job type:

```
SELECT
   employee_id,
   first_name,
   last_name,
   job_id
FROM employees
ORDER BY
   CASE
      WHEN job_id LIKE 'SA%' THEN 1
      WHEN job_id LIKE 'IT%' THEN 2
      WHEN job_id LIKE 'FI%' THEN 3
      ELSE 4
   END,
   last_name
FETCH FIRST 10 ROWS ONLY;
```

This query orders employees first by their job type (Sales, then IT, then Finance, then others), and then by their last name.

Lab 5.3: NULLIF and COALESCE Functions

After this lab, you will be able to:

- Use the NULLIF function
- Use the COALESCE function
- Compare NULLIF and COALESCE to equivalent CASE expressions

NULLIF Function

The NULLIF function is like a game of "Spot the Difference." If the two expressions you give it are the same, it returns NULL. If they're different, it returns the first expression.

Here's the syntax:

```
NULLIF(expression1, expression2)
```

Here's some visual examples:

```
NULLIF(A, B)

Case 1: A and B are different
  A   |   B   | Result
------+-------+--------
  X   |   Y   |   X

Case 2: A and B are the same
  A   |   B   | Result
------+-------+--------
  X   |   X   | NULL

Flowchart:

     +--------------------+
     |    NULLIF(A, B)    |
     +--------------------+
               |
               V
          +------------+
          | A == B ?   |
          +------------+
           /          \
         Yes           No
         /               \
        V                 V
  +------+          +--------+
  | NULL |          |   A    |
  +------+          +--------+
```

Let's see an example using the HR schema:

```
SELECT
   employee_id,
   first_name,
   last_name,
   NULLIF(manager_id, employee_id) AS different_manager
FROM employees
WHERE ROWNUM <= 5;
```

This query returns NULL for any employee whose manager_id is the same as their employee_id (which might indicate they manage themselves).

COALESCE Function

The COALESCE function is like having a series of backup plans. It returns the first non-NULL expression in a list.

Here's the syntax:

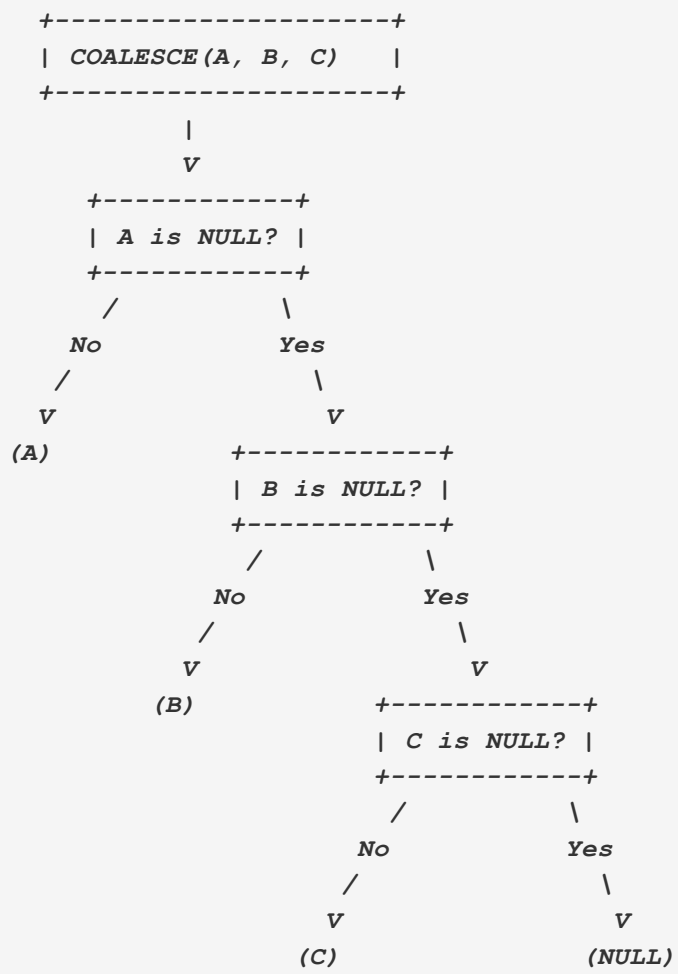
```
COALESCE(expression1, expression2, ..., expressionN)
```

Let's imagine we're checking three values: A, B, and C.

```
COALESCE(A, B, C)

Case 1: First non-NULL value is A
 A | B | C | Result
---+---+---+--------
 X | NULL| NULL| X

Case 2: First non-NULL value is B
 A | B | C | Result
---+---+---+--------
NULL | Y | NULL| Y

Case 3: First non-NULL value is C
 A | B | C | Result
---+---+---+--------
NULL | NULL| Z | Z

Case 4: All values are NULL
 A | B | C | Result
---+---+---+--------
NULL | NULL| NULL| NULL
```

Let's see a flowchart.

Flowchart:



```
No Yes
 / \
 V V
(C) (NULL)
```

This visual shows:

- COALESCE checks each value in order (A, then B, then C).
- It returns the first non-NULL value it finds.
- If all values are NULL, it returns NULL.

The flowchart demonstrates the decision process: it checks each value for NULL, returning the first non-NULL value it encounters. If all are NULL, it reaches the end and returns NULL.

Let's use COALESCE with the HR schema:

```
SELECT
  employee_id,
  COALESCE(commission_pct, salary/1000, 0) AS bonus_rate
FROM employees
WHERE ROWNUM <= 5;
```

This query tries to assign a bonus rate: first it checks commission_pct, then 10% of salary, and if both are NULL, it assigns 0.

Comparing NULLIF and COALESCE to CASE Expressions

Both NULLIF and COALESCE can be rewritten as CASE expressions.

For example,

```
NULLIF(a, b)
```

Is equivalent to:

```
CASE
 WHEN a = b THEN NULL
 ELSE a
END
```

And

```
COALESCE(a, b, c)
```

Is equivalent to:

```
CASE
 WHEN a IS NOT NULL THEN a
 WHEN b IS NOT NULL THEN b
 ELSE c
END
```

These functions provide a more concise way to write common CASE expression patterns.

Summary

In this chapter, you've learned about advanced conditional control structures in PL/SQL. You now understand how to use CASE statements and expressions to make your code more flexible and readable. You've also explored the NULLIF and COALESCE functions, which provide shorthand ways to handle common conditional scenarios.

Remember, these tools are like different types of traffic lights or vending machines - they all help you control the flow of your program, but each has its own strengths and best use cases. Practice using these structures with the HR schema to become more comfortable with when and how to apply each one.

Advanced Conditional Control in PL/SQL