Chapter 08

Mastering Error Management and Built-in Exceptions

Mayko Freitas da Silva

Introduction

In the world of programming, errors are an inevitable part of the development process. Just as a skilled sailor must navigate through both calm and stormy seas, a proficient PL/SQL developer must learn to handle both smooth executions and unexpected errors. This chapter will equip you with the knowledge and tools to effectively manage errors and leverage built-in exceptions in PL/SQL.

Imagine you're piloting a complex spacecraft. Your control panel is filled with various instruments and indicators. Some of these alert you to problems before you even start your journey (like compilation errors), while others only light up when you're in mid-flight (runtime errors). Learning to interpret and respond to these signals is crucial for a successful mission.

In PL/SQL, error management is not just about preventing crashes; it's about creating robust, reliable, and user-friendly applications. By implementing effective error handling, you're essentially building a safety net that catches issues before they escalate into critical failures.

In this chapter, you will embark on a journey to master the art of error management. You'll learn to:

1. Distinguish between compilation errors and runtime errors
2. Implement exception handling to gracefully manage unexpected situations
3. Utilize common built-in exceptions to address specific error scenarios
4. Develop strategies for handling multiple exceptions within a single block of code

By the end of this chapter, you'll have the skills to write PL/SQL code that not only functions well under ideal conditions but also gracefully handles unexpected situations. This knowledge will elevate your programming from merely functional to truly professional and robust.

Let's begin our exploration into the world of PL/SQL error management and built-in exceptions. Fasten your seatbelts – it's going to be an enlightening ride!

Lab 8.1: Understanding Error Handling

After this lab, you will be able to:

- Distinguish between compilation and runtime errors
- Implement basic exception handling

The Two Faces of Errors

In PL/SQL, errors come in two main flavors: compilation errors and runtime errors. Understanding the difference is crucial for effective error management.

Compilation errors occur when the PL/SQL compiler detects issues in your code before it runs. These are like typos in a recipe - they prevent your program from even starting. Common compilation errors include syntax mistakes, undeclared variables, or incorrect data types.

Runtime errors, on the other hand, occur during program execution. These are like a cake falling flat while baking - everything looked fine at the start, but something went wrong during the process. Examples include dividing by zero or trying to access a non-existent database record.

Let's examine a simple program to illustrate these concepts:

```
DECLARE
  v_num1   NUMBER := &sv_num1;
  v_num2   NUMBER := &sv_num2;
  v_result NUMBER;
BEGIN
  v_result := v_num1 / v_num2;
  DBMS_OUTPUT.PUT_LINE('Result: ' || v_result);
END;
```

This code will compile successfully. However, if you run it with v_num1 = 10 and v_num2 = 0, you'll encounter a runtime error:

```
ORA-01476: divisor is equal to zero
```

Catching Errors with Exception Handling

To manage runtime errors, PL/SQL provides exception handling. It's like adding a safety net to your code. Here's how you can modify the previous example:

```
DECLARE
  v_num1   NUMBER := &sv_num1;
  v_num2   NUMBER := &sv_num2;
  v_result NUMBER;
BEGIN
  v_result := v_num1 / v_num2;
  DBMS_OUTPUT.PUT_LINE('Result: ' || v_result);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Error: Cannot divide by zero.');
END;
```

Now, if v_num2 is 0, instead of crashing, your program will display a user-friendly message.

The Structure of Exception Handling

Exception handling in PL/SQL follows this general structure:

```
BEGIN
  -- Your normal code here
EXCEPTION
  WHEN exception1 THEN
    -- Handle exception1
  WHEN exception2 THEN
    -- Handle exception2
  WHEN OTHERS THEN
    -- Handle any other exceptions
END;
```

This structure allows you to catch and handle specific exceptions, as well as provide a catch-all for unexpected errors.

Exercise 8.1

Try running the division code with different values for v_num1 and v_num2. Observe what happens when:

1. Both are valid numbers (e.g., 10 and 2)
2. The second number is 0
3. You enter a non-numeric value for either input

Reflect on how exception handling improves the user experience in each scenario.

Lab 8.2: Exploring Built-in Exceptions

After this lab, you will be able to:

- Identify and use common built-in exceptions
- Handle multiple exceptions in a single block

Common Built-in Exceptions

PL/SQL provides a set of predefined exceptions to handle common error scenarios. These built-in exceptions are automatically raised when specific conditions occur. Let's explore some of the most frequently used ones:

1. NO_DATA_FOUND Raised when a SELECT INTO statement returns no rows.
2. TOO_MANY_ROWS Raised when a SELECT INTO statement returns more than one row.
3. ZERO_DIVIDE Raised when an attempt is made to divide by zero.
4. VALUE_ERROR Raised when there's a data type conversion error or constraint violation.
5. DUP_VAL_ON_INDEX Raised when attempting to insert a duplicate value in a unique index column.

Let's see these exceptions in action with a practical example:

```
DECLARE
  v_student_name VARCHAR2(50);
  v_student_id   NUMBER := &sv_student_id;
BEGIN
  SELECT first_name || ' ' || last_name
  INTO v_student_name
  FROM student
  WHERE student_id = v_student_id;

  DBMS_OUTPUT.PUT_LINE('Student name: ' || v_student_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No student found with ID ' || v_student_id);
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Multiple students found with ID ' || v_student_id);
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('Invalid data type for student ID');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
```

This code demonstrates how to handle multiple exceptions in a single block. It's prepared for various scenarios that might occur when querying the student table.

The OTHERS Clause: Your Safety Net

The OTHERS clause acts as a catch-all for any exceptions not explicitly handled. It's like a default case in a switch statement. While useful, it should be used judiciously:

```
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
```

SQLERRM is a function that returns the error message associated with the most recent exception.

Best Practices for Exception Handling

1. Handle specific exceptions before general ones.
2. Use the OTHERS clause sparingly and always log the specific error.
3. Avoid empty exception handlers.
4. Consider the scope of your exception handlers.

Exercise 8.2

Modify the student lookup code to handle the DUP_VAL_ON_INDEX exception. Then, write a PL/SQL block that attempts to insert a new student record and handles this exception. What real-world scenario might cause this exception to be raised?

I apologize for the confusion. You're absolutely right. Let's focus on the content that matches the original structure. I'll provide the summary section, which should be the next part after Lab 8.2 in the original content.

Summary

In this chapter, you've gained valuable insights into error handling and built-in exceptions in PL/SQL. Here's a recap of the key points you've learned:

1. Error Types:
    - You can now distinguish between compilation errors (caught before program execution) and runtime errors (occurring during program execution).
    - Understanding these differences helps you in writing more robust and error-resistant code.
2. Exception Handling:
    - You've learned how to implement basic exception handling using the EXCEPTION block in PL/SQL.
    - This knowledge allows you to gracefully manage runtime errors, improving the reliability of your programs.
3. Built-in Exceptions:
    - You're now familiar with common built-in exceptions such as NO_DATA_FOUND, TOO_MANY_ROWS, ZERO_DIVIDE, and VALUE_ERROR.
    - You understand when these exceptions are raised and how to handle them in your code.
4. Multiple Exception Handling:
    - You've practiced handling multiple exceptions within a single PL/SQL block.
    - This skill enables you to create more comprehensive error-handling routines.
5. The OTHERS Clause:
    - You've learned about the OTHERS clause and its role as a catch-all for unhandled exceptions.
    - You understand the importance of using this clause judiciously to avoid masking specific errors.
6. Practical Application:
    - Through examples and exercises, you've applied these concepts to real-world scenarios, such as querying a student database.
    - This practical experience prepares you for implementing error handling in your own PL/SQL projects.

Remember, effective error handling is crucial for developing robust, user-friendly PL/SQL applications. It not only prevents unexpected program terminations but also provides meaningful feedback to users and developers alike.

As you continue your journey in PL/SQL development, keep refining your error-handling techniques. Practice identifying potential error scenarios in your code and implementing appropriate exception handlers. This proactive approach will significantly enhance the quality and reliability of your PL/SQL programs.

Mastering Oracle PL/SQL: From Beginner to Advanced