Database Programming with PL/SQL

7-1
Handling Exceptions
Objectives

This lesson covers the following objectives:

• Describe several advantages of including exception handling code in PL/SQL

• Describe the purpose of an EXCEPTION section in a PL/SQL block

• Create PL/SQL code to include an EXCEPTION section

• List several guidelines for exception handling
Purpose

• You have learned to write PL/SQL blocks with a declarative section and an executable section.

• All the SQL and PL/SQL code that must be executed is written in the executable block.

• Thus far, you have assumed that the code works fine if you take care of compile time errors.

• However, the code can cause some unanticipated errors at run time.

• In this lesson, you learn how to deal with such errors in the PL/SQL block.
What is an Exception?

• An exception occurs when an error is discovered during the execution of a program that disrupts the normal operation of the program.

• There are many possible causes of exceptions: a user makes a spelling mistake while typing; a program does not work correctly; an advertised web page does not exist; and so on.

• Can you think of errors that you have come across while using a web site or application?
Exceptions in PL/SQL

This example works fine. But what if `v_country_name` was 'Korea, South' instead of 'Republic of Korea'?

```plsql
DECLARE
    v_country_name countries.country_name%TYPE
        := 'Republic of Korea';
    v_elevation countries.highest_elevation%TYPE;
BEGIN
    SELECT highest_elevation
        INTO v_elevation
        FROM countries
        WHERE country_name = v_country_name;
    DBMS_OUTPUT.PUT_LINE(v_elevation);
END;
```

Republic of Korea
Statement processed.
When our `v_country_name` is not found, our code results in an error.

```sql
DECLARE
    v_country_name countries.country_name%TYPE := 'Korea, South';
    v_elevation    countries.highest_elevation%TYPE;
BEGIN
    SELECT highest_elevation
    INTO v_elevation
    FROM countries
    WHERE country_name = v_country_name;
END;
```

ORA-01403: no data found
Exceptions in PL/SQL

• The code does not work as expected.
• No data was found for 'Korea, South' because the country name is actually stored as 'Republic of Korea.'
• This type of error in PL/SQL is called an exception.
• When code does not work as expected, PL/SQL raises an exception.
• When an exception occurs, we say that an exception has been "raised."
• When an exception is raised, the rest of the execution section of the PL/SQL block is not executed.
What Is an Exception Handler?

• An exception handler is code that defines the recovery actions to be performed when an exception is raised (that is, when an error occurs).

• When writing code, programmers need to anticipate the types of errors that can occur during the execution of that code.

• They need to include exception handlers in their code to address these errors. In a sense, exception handlers allow programmers to "bulletproof" their code.
What Is an Exception Handler?

What types of errors might programmers want to account for by using an exception handler?

• System errors (for example, a hard disk is full)
• Data errors (for example, trying to duplicate a primary key value)
• User action errors (for example, data entry error)
• Many other possibilities!
Why is Exception Handling Important?

Some reasons include:

• Protects the user from errors (frequent errors, unhelpful error messages, and software crashes can frustrate users/customers, and this is not good).

• Protects the database from errors (data can be lost or overwritten).

• Errors can be costly, in time and resources (processes may slow as operations are repeated or errors are investigated).
Handling Exceptions with PL/SQL

- A block always terminates when PL/SQL raises an exception, but you can specify an exception handler to perform final actions before the block ends.

```sql
DECLARE
    v_country_name countries.country_name%TYPE := 'Korea, South';
    v_elevation    countries.highest_elevation%TYPE;
BEGIN
    SELECT highest_elevation INTO v_elevation
    FROM countries WHERE country_name = v_country_name;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('Country name, ' || v_country_name || ',
            cannot be found. Re-enter the country name using the correct
            spelling.');
END;
```
Handling Exceptions with PL/SQL

• The exception section begins with the keyword **EXCEPTION**.

```sql
DECLARE
    v_country_name countries.country_name%TYPE := 'Korea, South';
    v_elevation    countries.highest_elevation%TYPE;
BEGIN
    SELECT highest_elevation INTO v_elevation
    FROM countries WHERE country_name = v_country_name;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('Country name, ' || v_country_name || ' ,
cannot be found. Re-enter the country name using the correct spelling.'));
END;
```
Handling Exceptions with PL/SQL

• When an exception is handled, the PL/SQL program does not terminate abruptly.

• When an exception is raised, control immediately shifts to the exception section and the appropriate handler in the exception section is executed.

• The PL/SQL block terminates with normal, successful completion.

Country name, Korea, South, cannot be found. Re-enter the country name using the correct spelling.

Statement processed.
Handling Exceptions with PL/SQL

The code at point A does not execute because the `SELECT` statement failed.

```plsql
DECLARE
    v_country_name countries.country_name%TYPE := 'Korea, South';
    v_elevation countries.highest_elevation%TYPE;
BEGIN
    SELECT highest_elevation INTO v_elevation
    FROM countries WHERE country_name = v_country_name;
    DBMS_OUTPUT.PUT_LINE(v_elevation); -- Point A
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('Country name, ' || v_country_name || ',
cannot be found. Re-enter the country name using the correct
spelling.');
END;
```
Handling Exceptions with PL/SQL

• The following is another example.
• The select statement in the block is retrieving the last_name of Stock Clerks.

DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
    FROM employees WHERE job_id = 'ST_CLERK';
  DBMS_OUTPUT.PUT_LINE('The last name of the ST_CLERK is : '||v_lname);
END;

• However, an exception is raised because more than one ST_CLERK exists in the data.

ORA-01422: exact fetch returns more than requested number of rows
Handling Exceptions with PL/SQL

• The following code includes a handler for the predefined Oracle server error called TOO_MANY_ROWS.

• You will learn more about predefined server errors in the next lesson.

```plsql
DECLARE
    v_lname employees.last_name%TYPE;
BEGIN
    SELECT last_name INTO v_lname
    FROM employees WHERE job_id = 'ST_CLERK';
    DBMS_OUTPUT.PUT_LINE('The last name of the ST_CLERK is: ' || v_lname);
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE ('Your select statement retrieved multiple rows. Consider using a cursor.');
END;
```
Trapping Exceptions

• You can handle or "trap" any error by including a corresponding handler within the exception-handling section of the PL/SQL block.

• Syntax:

```sql
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```
Trapping Exceptions

• Each handler consists of a **WHEN** clause, which specifies an exception name (**exception1**, **exception 2**, etc.), followed by **THEN** and one or more statements to be executed when that exception is raised (**statement1**, **statement 2**, etc.).

• You can include any number of handlers within an **EXCEPTION** section to handle different exceptions.

```
EXCEPTION
    WHEN exception1 [OR exception2 . . .] THEN
        statement1;
        statement2;
        . . .
    [WHEN OTHERS THEN
        statement1;
        statement2;
        . . .]
```
Trapping

• In the syntax, **OTHERS** is an optional exception-handling clause that traps any exceptions that have not been explicitly handled.

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```
The **OTHERS** Exception Handler

- The exception-handling section traps only those exceptions that are specified; any other exceptions are not trapped unless you use the **OTHERS** exception handler.
- The **OTHERS** handler traps all the exceptions that are not already trapped.
- If used, **OTHERS** must be the last exception handler that is defined.
The **OTHERS** Exception Handler

Consider the following example:

```
BEGIN
  ...
  ...
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;
    ...
  WHEN TOO_MANY_ROWS THEN
    statement3;
    statement4;
    ...
  WHEN OTHERS THEN
    statement5;
    statement6;
    ...
END;
```
Guidelines for Trapping Exceptions

Follow these guidelines when trapping exceptions:

• Always add exception handlers whenever there is a possibility of an error occurring.

• Errors are especially likely during calculations, string manipulation, and SQL database operations.

• Handle named exceptions whenever possible, instead of using OTHERS in exception handlers.

• Learn the names and causes of the predefined exceptions.

• Test your code with different combinations of bad data to see what potential errors arise.
Guidelines for Trapping Exceptions

• Write out debugging information in your exception handlers.

• Carefully consider whether each exception handler should commit the transaction, roll it back, or let it continue.

• No matter how severe the error is, you want to leave the database in a consistent state and avoid storing any bad data.
Terminology

Key terms used in this lesson included:

• Exception
• Exception handler
Summary

In this lesson, you should have learned how to:

• Describe several advantages of including exception handling code in PL/SQL

• Describe the purpose of an EXCEPTION section in a PL/SQL block

• Create PL/SQL code to include an EXCEPTION section

• List several guidelines for exception handling