ORACLE

Academy

Database Programming with PL/SQL

7-1

Handling Exceptions

ORACLE
Academy

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

ORACLE Academy

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

ORACLE Academy

Computer programs should be written so that even unanticipated errors are handled, no program should ever just crash or quit working.

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?

What is an Exception?

- Some examples of errors you may have seen:
- Entering an incorrect username and/or password
- Forgetting to include the @ in an email address
- Entering a credit card number incorrectly
- Entering an expiration date that has passed
- Selecting more than one row into a single variable
- Receiving “no rows returned” from a select statement

Exceptions in PL/SQL

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

```
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;
```

```
1950

Statement processed.
```

ORACLE Academy

PLSQL 7-1
Handling Exceptions

To confirm the spelling of Republic of Korea in the data, run a statement such as the following:

```
SELECT country_name
  FROM countries
  WHERE country_name LIKE '%Korea%';
```

Exceptions in PL/SQL

- When our v_country_name is not found, our code results in an error

```
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
```

ORACLE Academy

Remember that a SELECT statement in PL/SQL must return exactly one row. This statement returns no rows and therefore raises an exception in the executable section.

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

ORACLE Academy

PL/SQL programs will start to get longer and more complicated due to the exception handling code needed to handle all errors, but that is preferable to programs just crashing or quitting. By including exception handling code, error messages can help the user understand what went wrong and how to correct it.

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!

ORACLE Academy

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)

ORACLE Academy

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

```
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;
```

ORACLE Academy

Handling Exceptions with PL/SQL

- The exception section begins with the keyword EXCEPTION

```
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;
```

ORACLE Academy

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.
```

ORACLE Academy

Only one exception can occur at a time and only one handler (although it may include multiple statements) may be executed.

Handling Exceptions with PL/SQL

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

```
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

- When an exception is raised, the rest of the executable section of the block is NOT executed; instead, the EXCEPTION section is searched for a suitable handler

ORACLE Academy

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;
```

```
ORA-01422:  exact fetch returns more than requested number of rows
```

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

ORACLE Academy

There is no exception handler in this block, therefore the block terminates unsuccessfully, returning an "unhandled exception" status code to the calling environment (Application Express), which then reports the exception as shown.

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

ORACLE Academy

This code will successfully handle the exception inside the block, so PL/SQL returns a "success" status code to the calling environment, which therefore will report Statement Processed (below the display of the PUT_LINE).

Handling Exceptions with PL/SQL

```
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;
```

ORACLE Academy

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:

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

ORACLE Academy

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

Trapping Exceptions

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

ORACLE Academy

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

ORACLE Academy

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;
```

ORACLE Academy

PLSQL 7-1
Handling Exceptions

If the exception NO_DATA_FOUND is raised by the program, then the statements in the corresponding handler are executed.

If the exception TOO_MANY_ROWS is raised, then the statements in the corresponding handler are executed.

However, if some other exception is raised, then the statements in the OTHERS exception handler are executed.

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

ORACLE Academy

Terminology

- Key terms used in this lesson included:
  - Exception
  - Exception handler

ORACLE Academy

- Exception – Occurs when an error is discovered during the execution of a program that disrupts the normal operation of the program.
- Exception Handler – Code that defines the recovery actions to be performed when execution-time errors occur.

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

ORACLE Academy

ORACLE

Academy

ORACLE

Academy

Database Programming with PL/SQL

7-2

Trapping Oracle Server Exceptions

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Describe and provide an example of an error defined by the Oracle server
  - Describe and provide an example of an error defined by the PL/SQL programmer
  - Differentiate between errors that are handled implicitly and explicitly by the Oracle server
  - Write PL/SQL code to trap a predefined Oracle server error

Objectives

- This lesson covers the following objectives:
  - Write PL/SQL code to trap a non-predefined Oracle server error
  - Write PL/SQL code to identify an exception by error code and by error message

ORACLE Academy

Purpose

- PL/SQL error handling is flexible and allows programmers to handle Oracle server errors and errors defined by the programmer
- This lesson discusses Oracle server errors
- User/programmer-defined errors will be discussed in the next lesson
- Oracle server errors can be either predefined or non-predefined

Purpose

- Both types have an error code and a message
- The predefined errors are the most common and they also have a name (ex., NO_DATA_FOUND, TOO_MANY_ROWS, etc.)

Exception Types

- This lesson discusses both predefined and non-predefined Oracle server errors
- An Oracle Server error is an error which is recognized and raised automatically by the Oracle server

Exception	Description	Instructions for Handling
Predefined Oracle server error	Most common PL/SQL errors (about 20 or so that are named)	You need not declare these exceptions. They are predefined by the Oracle server and are raised implicitly (automatically).
Non-predefined Oracle server error	Other PL/SQL errors (no name)	Declare within the declarative section and allow the Oracle Server to raise them implicitly (automatically).
User-defined error	Defined by the programmer	Declare within the declarative section, and raise explicitly.

Handling Exceptions with PL/SQL

- There are two methods for raising an exception:
  - Implicitly (automatically) by the Oracle server:
    - An Oracle error occurs and the associated exception is raised automatically
    - For example, if the error ORA-01403 occurs when no rows are retrieved from the database in a SELECT statement, then PL/SQL raises the exception NO_DATA_FOUND

Handling Exceptions with PL/SQL

- Explicitly by the programmer:
  - Depending on the business functionality your program is implementing, you might have to explicitly raise an exception
  - You raise an exception explicitly by issuing the RAISE statement within the block
  - The exception being raised can be either user-defined or predefined
  - User-defined exceptions are explained in the next lesson

ORACLE Academy

Two Types of Oracle Server Errors

- When an Oracle server error occurs, the Oracle server automatically raises the associated exception, skips the rest of the executable section of the block, and looks for a handler in the exception section
- As mentioned earlier, Oracle server errors can be predefined or non-predefined

ORACLE Academy

Two Types of Oracle Server Errors

- Predefined Oracle server errors:
  - Each of these errors has a predefined name, in addition to a standard Oracle error number (ORA-#####) and message
  - For example, if the error ORA-01403 occurs when no rows are retrieved from the database in a SELECT statement, then PL/SQL raises the predefined exception NO_DATA_FOUND

ORACLE Academy

Two Types of Oracle Server Errors

- Non-predefined Oracle server errors:
  - Each of these errors has a standard Oracle error number (ORA-#####) and error message, but not a predefined name
  - You declare your own names for these so that you can reference these names in the exception section

ORACLE Academy

The EXCEPTION section can refer to exceptions only by name, not by number (i.e., we cannot code WHEN ORA-##### THEN …).

Trapping Predefined Oracle Server Errors

- Reference the predefined name in the exception handling routine
- Sample predefined exceptions:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

ORACLE Academy

Trapping Predefined Oracle Server Errors

- For a complete list of predefined exceptions, see the PL/SQL User's Guide and Reference

ORACLE Academy

Trapping Predefined Oracle Server Errors

- The following example uses the TOO_MANY_ROWS predefined Oracle server error
- Note that it is not declared in the DECLARATION section

```
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);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE ('Your select statement retrieved multiple
                    rows. Consider using a cursor.');
END;
```

ORACLE Academy

Trapping Several Predefined Oracle Server Errors

- This example handles TOO_MANY_ROWS and NO_DATA_FOUND, with an OTHERS handler in case any other error occurs

ORACLE Academy

The OTHERS handler will handle all types of raised exceptions: predefined, non-predefined, and user-defined.

Trapping Several Predefined Oracle Server Errors

```
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);
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE ('Select statement found multiple
rows');
  WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE ('Select statement found no rows');
  WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE ('Another type of error occurred');
END;
```

ORACLE Academy

Trapping Non-Predefined Oracle Server Errors

- Non-predefined exceptions are similar to predefined exceptions, except they do not have predefined names
- They do have a standard Oracle error number (ORA-#####) and error message
- To use specific handlers (rather than handling through an OTHERS clause), you create your own names for them in the DECLARE section and associate the names with the specific ORA-##### numbers using the PRAGMA EXCEPTION_INIT function

ORACLE Academy

Trapping Non-Predefined Oracle Server Errors

- You can trap a non-predefined Oracle server error by declaring it first
- The declared exception is raised implicitly
- In PL/SQL, the PRAGMA EXCEPTION_INIT tells the compiler to associate an exception name with a specific Oracle error number
- This allows you to refer to any Oracle Server exception by a name and to write a specific handler for it

Non-Predefined Error

- Examine the following example

```
BEGIN
 INSERT INTO departments
   (department_id, department_name) VALUES (280, NULL);
END;
```

- The code above results in the error message below

```
ORA-01400: cannot insert NULL into
("US_1217_S19_PLSQL"."DEPARTMENTS"."DEPARTMENT_NAME")
```

Non-Predefined Error

- The INSERT statement tries to insert the value NULL for the department_name column of the departments table
- However, the operation is not successful because department_name is a NOT NULL column
- There is no predefined error name for violating a NOT NULL constraint
- The following slides will demonstrate how to "handle" non-predefined exceptions

Non-Predefined Error

- Declare the name of the exception in the declarative section

```
DECLARE
  e_insert_excep EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
  INSERT INTO departments
 (department_id, department_name)
 VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep
 THEN
   DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

1

```
Syntax:

exception_name EXCEPTION;
```

ORACLE Academy

Non-Predefined Error

- Associate the declared exception name with the standard Oracle server error number using the PRAGMA EXCEPTION_INIT function

```
DECLARE
  e_insert_excep EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
  INSERT INTO departments
 (department_id, department_name)
 VALUES (280, NULL);
EXCEPTION
  WHEN e_insert
 THEN
   DBMS_OUTPUT.
END;
```

2

```
Syntax:

PRAGMA EXCEPTION_INIT(exception_name, -number);
```

ORACLE Academy

A full list of Oracle server error numbers can be found at http://docs.oracle.com/. Search for "database error messages."

Non-Predefined Error

- Reference the declared exception name within a WHEN clause in the exception-handling section

```
DECLARE
  e_insert_excep EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
  INSERT INTO departments
 (department_id, department_name)
 VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep
 THEN
   DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

3

Functions for Trapping Exceptions

- When an exception occurs, you can retrieve the associated error code or error message by using two functions
- Based on the values of the code or the message, you can decide which subsequent actions to take
  - SQLERRM returns character data containing the message associated with the error number
  - SQLCODE returns the numeric value for the error code. (You can assign it to a NUMBER variable)

ORACLE Academy

Functions for Trapping Exceptions

- Note : +100 is an internationally-agreed code when no rows are returned from a query

SQLCODE Value	Description
0	No exception encountered
1	User defined exception
+100	NO_DATA_FOUND exception
Negative number	Another Oracle Server error number

Functions for Trapping Exceptions

- You cannot use SQLCODE or SQLERRM directly in an SQL statement
- Instead, you must assign their values to local variables, then use the variables in the SQL statement, as shown in the following example:

ORACLE Academy

Why can't we use SQLCODE or SQLERRM directly in a SQL statement?

Answer: because that SQL statement (the INSERT INTO error_log in the slide) could also raise an exception, which would change the values of SQLCODE and SQLERRM.

SQLCODE and SQLERRM are often used in a WHEN OTHERS handler. Someone (often the Database Administrator) would be responsible for reading the ERROR_LOG table and taking suitable action.

Functions for Trapping Exceptions

```
DECLARE
  v_error_code   NUMBER;
  v_error_message   VARCHAR2(255);
BEGIN   ...
EXCEPTION
  WHEN OTHERS THEN
 ROLLBACK;
 v_error_code := SQLCODE;
 v_error_message := SQLERRM;
 INSERT INTO error_log(e_user, e_date, error_code,
error_message)
       VALUES(USER, SYSDATE, v_error_code, v_error_message);
END;
```

ORACLE Academy

Terminology

- Key terms used in this lesson included:
  - Non-predefined Oracle server errors
  - Predefined Oracle server errors
  - PRAGMA EXCEPTION_INIT
  - SQLERRM
  - SQLCODE

ORACLE Academy

- Non-predefined Oracle Server Errors: Each of these has a standard Oracle error number (ORA-nnnnn) and error message, but not a predefined name. We declare our own names for these so that we can reference these names in the exception section.

- Predefined Oracle Server Errors: Each of these has a predefined name. For example, if the error ORA-01403 occurs when no rows are retrieved from the database in a SELECT statement, then PL/SQL raises the predefined exception-name NO_DATA_FOUND.

- PRAGMA EXCEPTION_INIT – Tells the compiler to associate an exception name with an Oracle error number. That allows you to refer to any Oracle Server exception by name and to write a specific handler for it.

- SQLERRM – Returns character data containing the message associated with the error number

- SQLCODE – Returns the numeric value for the error code (You can assign it to a NUMBER

variable.)

Summary

- In this lesson, you should have learned how to:
  - Describe and provide an example of an error defined by the Oracle server
  - Describe and provide an example of an error defined by the PL/SQL programmer
  - Differentiate between errors that are handled implicitly and explicitly by the Oracle server
  - Write PL/SQL code to trap a predefined Oracle server error

ORACLE Academy

Summary

- In this lesson, you should have learned how to:
  - Write PL/SQL code to trap a non-predefined Oracle server error
  - Write PL/SQL code to identify an exception by error code and by error message

ORACLE Academy

ORACLE

Academy

ORACLE

Academy

Database Programming with PL/SQL

7-3

Trapping User-Defined Exceptions

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Write PL/SQL code to name a user-defined exception
  - Write PL/SQL code to raise an exception
  - Write PL/SQL code to handle a raised exception
  - Write PL/SQL code to use RAISE_APPLICATION_ERROR

ORACLE Academy

Purpose

- In addition to the predefined Oracle errors, programmers can create their own user-defined errors
- User-defined errors are not automatically raised by the Oracle server, but are defined by the programmer and must be raised by the programmer when they occur
- With a user-defined error, the programmer creates an error code and an error message
- An example of a user-defined error might be INVALID_MANAGER_ID

ORACLE Academy

Can you think of situations where the Oracle server would execute a statement successfully (and therefore not raise an exception automatically), but there is still an "error" from the user's viewpoint.

Answers:

A DML UPDATE or DELETE statement modifies no rows.

A SELECT statement successfully reads a row which should not exist yet.

A Stock Clerk has been identified as a manager, but our business rules state that clerks cannot be managers.

Exception Types

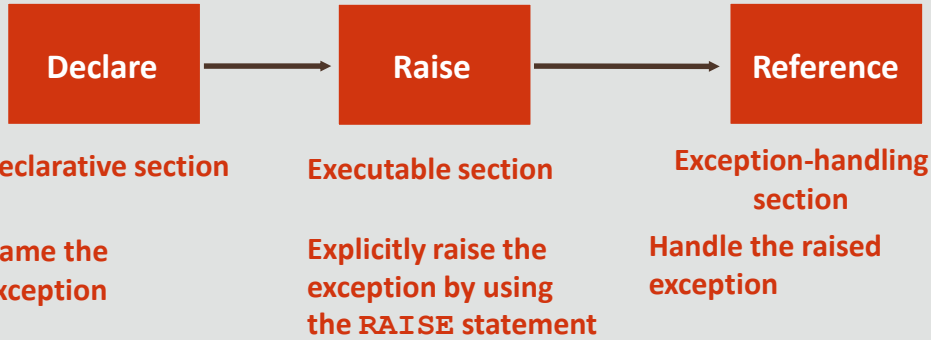
- This lesson discusses user-defined errors

Exception	Description	Instructions for Handling
Predefined Oracle server error	Most common PL/SQL errors (about 20 or so that are named)	You need not declare these exceptions. They are predefined by the Oracle server and are raised implicitly (automatically).
Non-predefined Oracle server error	Other PL/SQL errors (no name)	Declare within the declarative section and allow the Oracle Server to raise them implicitly (automatically).
User-defined error	Defined by the programmer	Declare within the declarative section, and raise explicitly.

ORACLE Academy

Trapping User-Defined Exceptions

- PL/SQL allows you to define your own exceptions
- You define exceptions depending on the requirements of your application



ORACLE Academy

Trapping User-Defined Exceptions

- One example of the need for a user-defined exception is during the input of data
- Assume your program prompts the user for a department number and name so it can update the name of the department

```
DECLARE
  v_name   VARCHAR2(20):= 'Accounting';
  v_deptno NUMBER := 27;
BEGIN
  UPDATE  departments
    SET     department_name = v_name
    WHERE   department_id = v_deptno;
END;
```

ORACLE Academy

Remember that an UPDATE or DELETE statement is treated as successful by the server even if it modifies no rows. Therefore the Oracle server will not automatically raise an exception in this case. If we want to raise an exception, we must do it ourselves.

Trapping User-Defined Exceptions

- What happens if the user enters an invalid department number?
- Oracle doesn't see this as an error
- You will need a user-defined error to catch this situation

```
DECLARE
  v_name   VARCHAR2(20):= 'Accounting';
  v_deptno NUMBER := 27;
BEGIN
  UPDATE  departments
    SET     department_name = v_name
    WHERE   department_id = v_deptno;
END;
```

ORACLE Academy

Trapping User-Defined Exceptions

- What happens when the user enters an invalid department?
- The code as written doesn't produce an Oracle error
- You need to create a user-defined error to handle this situation
- You do this by:
  - 1. Declaring the name of the user-defined exception within the declarative section

```
e_invalid_department EXCEPTION;
```

  - 2. Using the RAISE statement to raise the exception explicitly within the executable section

```
IF SQL%NOTFOUND THEN RAISE e_invalid_department;
```

ORACLE Academy

Trapping User-Defined Exceptions

- You do this by:
  - 3. Referencing the declared exception name within a WHEN clause in the exception-handling section

```
EXCEPTION
  WHEN e_invalid_department  THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
```

- These three "steps" are similar to what we did in the previous lesson with non-predefined Oracle errors
- The differences are, no PRAGMA EXCEPTION_INIT is required and you must explicitly raise the exception using the RAISE command

ORACLE Academy

Trapping User-Defined Exceptions

- The completed code with the "steps" indicated

```
DECLARE
  e_invalid_department EXCEPTION;          1
  v_name VARCHAR2(20):='Accounting';
  v_deptno NUMBER := 27;
BEGIN
  UPDATE  departments
    SET     department_name = v_name
    WHERE   department_id = v_deptno;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;            2
  END IF;
EXCEPTION
  WHEN e_invalid_department                3
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

ORACLE Academy

The RAISE Statement

- You can use the RAISE statement to raise exceptions
- Raising a user-defined exception:

```
IF v_grand_total = 0 THEN
   RAISE e_invalid_total;
ELSE
  DBMS_OUTPUT.PUT_LINE(v_num_students / v_grand_total);
END IF;
```

- Raising an Oracle server error:

```
IF v_grand_total = 0 THEN
   RAISE ZERO_DIVIDE;
ELSE
   DBMS_OUTPUT.PUT_LINE(v_num_students / v_grand_total);
END IF;
```

The RAISE_APPLICATION_ERROR Procedure

- You can use the RAISE_APPLICATION_ERROR procedure to return user-defined error messages from stored subprograms
- The following slides explain the syntax for using RAISE_APPLICATION_ERROR
- The main advantage of using this procedure instead of RAISE, is that RAISE_APPLICATION_ERROR allows you to associate your own error number and meaningful message with the exception

The RAISE_APPLICATION_ERROR Syntax

- The error_number must fall between -20000 and -20999
- This range is reserved by Oracle for programmer use, and is never used for predefined Oracle server errors
- message is the user-specified message for the exception
- It is a character string up to 2,048 bytes long

```
RAISE_APPLICATION_ERROR (error_number,
            message[, {TRUE | FALSE}]);
```

ORACLE Academy

The RAISE_APPLICATION_ERROR Syntax

- TRUE | FALSE is an optional Boolean parameter
- If TRUE, the error is placed on the stack of previous errors
- If FALSE—the default—the error replaces all previous errors

```
RAISE_APPLICATION_ERROR (error_number,
             message[, {TRUE | FALSE}]);
```

ORACLE Academy

The RAISE_APPLICATION_ERROR Usage

- You can use the RAISE_APPLICATION_ERROR in two different places:
  - Executable section
  - Exception section

RAISE_APPLICATION_ERROR in the Executable Section

- When called, the RAISE_APPLICATION_ERROR procedure displays the error number and message to the user
- This process is consistent with other Oracle server errors

```
DECLARE
  v_mgr       PLS_INTEGER := 123;
BEGIN
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
  'This is not a valid manager');
  END IF;
END;
```

ORACLE Academy

Note that an error raised by RAISE_APPLICATION_ERROR is an unhandled exception which is propagated back to the calling environment. The whole idea is to allow the calling application to display a business-meaningful error message to the user. If the exception was handled successfully within the PL/SQL block, the application would not see the error at all.

RAISE_APPLICATION_ERROR in the Exception Section

```
DECLARE
  v_mgr       PLS_INTEGER := 27;
  v_employee_id     employees.employee_id%TYPE;
BEGIN
  SELECT employee_id INTO v_employee_id
    FROM employees
    WHERE manager_id = v_mgr;
  DBMS_OUTPUT.PUT_LINE('Employee #' || v_employee_id ||
      ' works for manager #' ||  v_mgr || '.');
EXCEPTION
   WHEN NO_DATA_FOUND THEN
     RAISE_APPLICATION_ERROR(-20201,
   'This manager has no employees');
   WHEN TOO_MANY_ROWS THEN
     RAISE_APPLICATION_ERROR(-20202,
   'Too many employees were found.');
END;
```

ORACLE Academy

Imagine an office worker trying to create a report and he/she receives an error message. Which of these messages would be more helpful?

ORA-01403: no data found

ORA-20201: This manager has no employees

Using the RAISE_APPLICATION_ERROR with a User-Defined Exception

```
DECLARE
  e_name          EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_name, -20999);
  v_last_name     employees.last_name%TYPE := 'Silly Name';

BEGIN
  DELETE FROM employees
  WHERE last_name = v_last_name;

IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20999, 'Invalid last name');
  ELSE
    DBMS_OUTPUT.PUT_LINE(v_last_name ||' deleted');
  END IF;
EXCEPTION
  WHEN e_name THEN
    DBMS_OUTPUT.PUT_LINE('Valid last names are: ');
    FOR c1 IN (SELECT DISTINCT last_name FROM employees)
 LOOP
        DBMS_OUTPUT.PUT_LINE(c1.last_name);
 END LOOP;
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error deleting from employees');
END;
```

ORACLE Academy

PLSQL 7-3
Trapping User-Defined Exceptions

In this example, when an invalid last name is entered, the user-defined exception e_name is raised using its error number -20999. Because the PRAGMA EXCEPTION_INIT was used to define the error name e_name and its error code –20999, when the error code is raised, the exception handler for e_name is run, showing the user (or the calling environment) a list of valid last names.

Terminology

- Key terms used in this lesson included:
  - RAISE
  - RAISE_APPLICATION_ERROR
  - User-defined error

ORACLE Academy

- RAISE - Use this statement to raise a named exception.
- RAISE_APPLICATION_ERROR - A procedure used to return user-defined error messages from stored subprograms.
- User-Defined error - These errors are not automatically raised by the Oracle Server, but are defined by the programmer and are specific to the programmer's code.

Summary

- In this lesson, you should have learned how to:
  - Write PL/SQL code to name a user-defined exception
  - Write PL/SQL code to raise an exception
  - Write PL/SQL code to handle a raised exception
  - Write PL/SQL code to use RAISE_APPLICATION_ERROR

ORACLE Academy

ORACLE

Academy

ORACLE

Academy

Database Programming with PL/SQL

7-4

Recognizing the Scope of Exceptions

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Describe the scope of an exception
  - Recognize an exception-scope issue when an exception is within nested blocks
  - Describe the effect of exception propagation in nested blocks

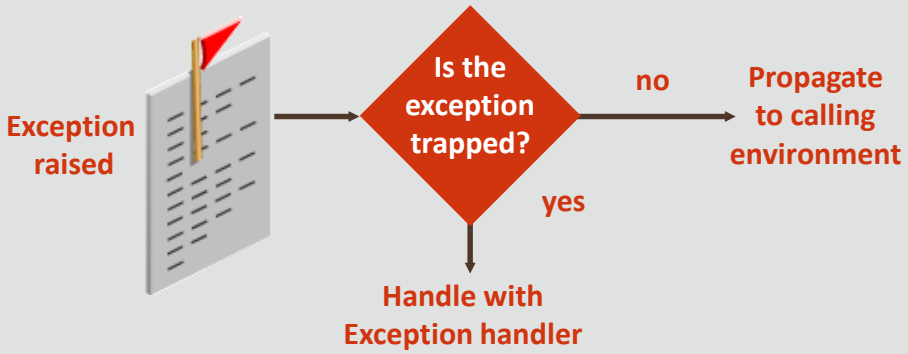
ORACLE Academy

Purpose

- You learned about nested blocks and scope of variables in an earlier lesson
- An exception is a PL/SQL variable; therefore, it follows the same scoping and visibility rules as any other kind of variable
- To handle exceptions correctly, you must understand the scope and visibility of exception variables
- This is particularly important when using nested blocks

Exception Handling in Nested Blocks

- You can deal with an exception by:
  - Handling it ("trapping it") in the block in which it occurs, or
  - Propagating it to the calling environment (which can be a higher-level block)



ORACLE Academy

PLSQL 7-4
Recognizing the Scope of Exceptions

5

A PL/SQL block nested within another PL/SQL block may be called a nested block, an enclosed block, a child block, or a sub-block.

A PL/SQL block that calls another PL/SQL block, anonymous or named, may be referred to as either the enclosing block or the parent block.

Handling Exceptions in an Inner Block

- In this example, an error occurs during the execution of the inner block
- The inner block's EXCEPTION section deals with the exception successfully, and PL/SQL considers that this exception is now finished
- The outer block resumes execution as normal

ORACLE Academy

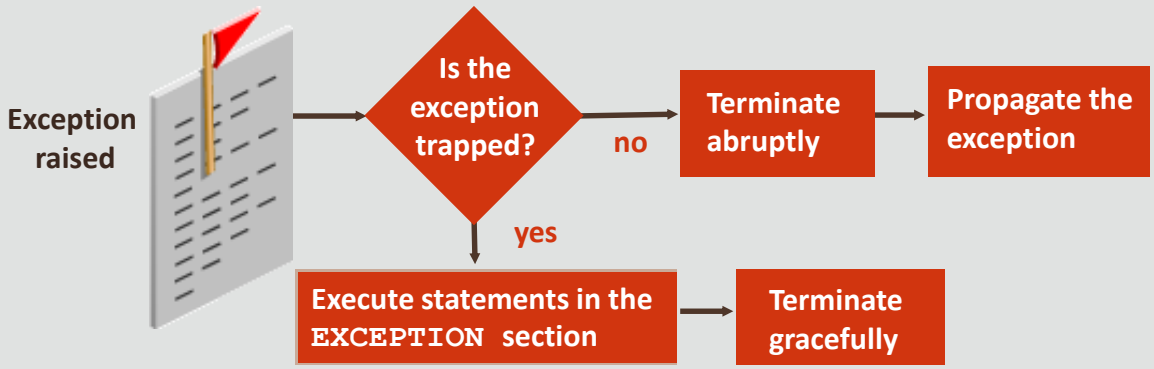
Handling Exceptions in an Inner Block

```
BEGIN  -- outer block
  ...
  BEGIN -- inner block
   ...  -- exception_name occurs here
   ...
  EXCEPTION
    WHEN exception_name THEN -- handled here
    ...
  END; -- inner block terminates successfully
  ...  -- outer block continues execution
END;
```

ORACLE Academy

Propagating Exceptions to an Outer Block

- If the exception is raised in the executable section of the inner block and no corresponding exception handler exists, the PL/SQL block terminates with failure and the exception is propagated to an enclosing block



ORACLE Academy

The general advantage of a nested block is that you create a scope for all the declared objects and executable statements in that block. You can use this scope to improve your control over activity in your program. This also means that if an error occurs in a block, then it is just in this block execution is halted, once the error has been handled locally, program control continues outside that block.

Propagating Exceptions to an Outer Block

- In this example, an exception occurs during the execution of the inner block
- The inner block's EXCEPTION section does not deal with the exception

ORACLE Academy

Propagating Exceptions to an Outer Block

```
DECLARE     -- outer block
  e_no_rows     EXCEPTION;
BEGIN
  BEGIN     -- inner block
    IF ... THEN RAISE e_no_rows; -- exception occurs here
    ...
  END;    -- Inner block terminates unsuccessfully
  ...     -- Remaining code in outer block's executable
  ...     -- section is skipped
  EXCEPTION
    WHEN e_no_rows THEN – outer block handles the exception
    ...
END;
```

Propagating Exceptions to an Outer Block

- The inner block terminates unsuccessfully and PL/SQL passes (propagates) the exception to the outer block
- The outer block's EXCEPTION section successfully handles the exception

ORACLE Academy

Propagating Exceptions to an Outer Block

```
DECLARE    -- outer block
  e_no_rows     EXCEPTION;
BEGIN
  BEGIN    -- inner block
    IF ... THEN RAISE e_no_rows; -- exception occurs here
    ...
  END;   -- Inner block terminates unsuccessfully
  ...    -- Remaining code in outer block's executable
  ...    -- section is skipped
  EXCEPTION
    WHEN e_no_rows THEN – outer block handles the exception
    ...
END;
```

ORACLE Academy

Propagating Exceptions from a Sub-Block

- If a PL/SQL raises an exception and the current block does not have a handler for that exception, the exception propagates to successive enclosing blocks until it finds a handler
- When the exception propagates to an enclosing block, the remaining executable actions in that block are bypassed
- One advantage of this behavior is that you can enclose statements that require their own exclusive error handling in their own block, while leaving more general exception handling (for example WHEN OTHERS) to the enclosing block
- The next slide shows an example of this

Propagating Predefined Oracle Server Exceptions from a Sub-Block

- Employee_id 999 does not exist
- What is displayed when this code is executed?

```
DECLARE
  v_last_name    employees.last_name%TYPE;
BEGIN
  BEGIN
    SELECT last_name INTO v_last_name
    FROM employees WHERE employee_id = 999;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

ORACLE Academy

PLSQL 7-4
Recognizing the Scope of Exceptions

Answer : Since employee_id 999 does not exist, the SELECT statement within the inner block results in an error and reverts to the local exception handling section. Because the error is not handled there, the error propagates to the calling environment. In this case, the calling environment is the outer block. The WHEN OTHERS exception handler gets executed and the output is “Message 4.”

Propagating User-named Exceptions: Ex. 1

- What happens when this code is executed?

```
BEGIN
  DECLARE
    e_myexcep    EXCEPTION;
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN e_myexcep THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

ORACLE Academy

Answer: An error, E_MYEXCEP is not declared in the outer block.

Scope of Exception Names

- Predefined Oracle server exceptions, such as NO_DATA_FOUND, TOO_MANY_ROWS, and OTHERS are not declared by the programmer
- They can be raised in any block and handled in any block
- User-named exceptions (non-predefined Oracle server exceptions and user-defined exceptions) are declared by the programmer as variables of type EXCEPTION
- They follow the same scoping rules as other variables

ORACLE Academy

Scope of Exception Names

- Therefore, a user-named exception declared within an inner block cannot be referenced in the exception section of an outer block
- To avoid this, always declare user-named exceptions in the outermost block

ORACLE Academy

Propagating User-named Exceptions: Ex. 2

- Now what happens when this code is executed?

```
DECLARE
  e_myexcep    EXCEPTION;
BEGIN
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN e_myexcep THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

ORACLE Academy

Answer: "Message 4" is displayed.

Propagating User-named Exceptions: Ex. 3

- What happens when this code is executed?

```
DECLARE
  e_myexcep    EXCEPTION;
BEGIN
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```

ORACLE Academy

Answer: since the "User-Defined Exception" is not handled by the code, it is returned to the calling environment (in our case APEX).

Propagating Unhandled Exceptions to the Calling Environment

- If a raised exception is not handled in a block, the block is exited with the exception still raised
- If there is an enclosing block for the current block, the exception is passed on to that block
- The enclosing block then becomes the current block
- If a handler for the raised exception is not found, the process repeats
- If no handler is found in any block, the calling environment, for example Application Express, must then try to handle the exception

ORACLE Academy

Propagating Unhandled Exceptions to the Calling Environment

- Because Application Express is Oracle software and therefore understands PL/SQL exceptions, Application Express will display an error message
- But other applications cannot always do this, and may fail with unexpected errors
- To avoid this, always handle exceptions within PL/SQL
- One way to guarantee this is to always include a WHEN OTHERS handler in the outermost block

ORACLE Academy

Terminology

- Key terms used in this lesson included:
  - Exception scope
  - Exception visibility
  - Propagation of exceptions

ORACLE Academy

- Exception Scope – The portion of a program in which the exception is declared and is accessible.
- Exception Visibility – The portion of the program where the exception can be accessed without using a qualifier.
- Propagation of exceptions – The inner block terminates unsuccessfully, and PL/SQL passes the exception to the outer block.

Summary

- In this lesson, you should have learned how to:
  - Describe the scope of an exception
  - Recognize an exception-scope issue when an exception is within nested blocks
  - Describe the effect of exception propagation in nested blocks

ORACLE Academy

ORACLE

Academy