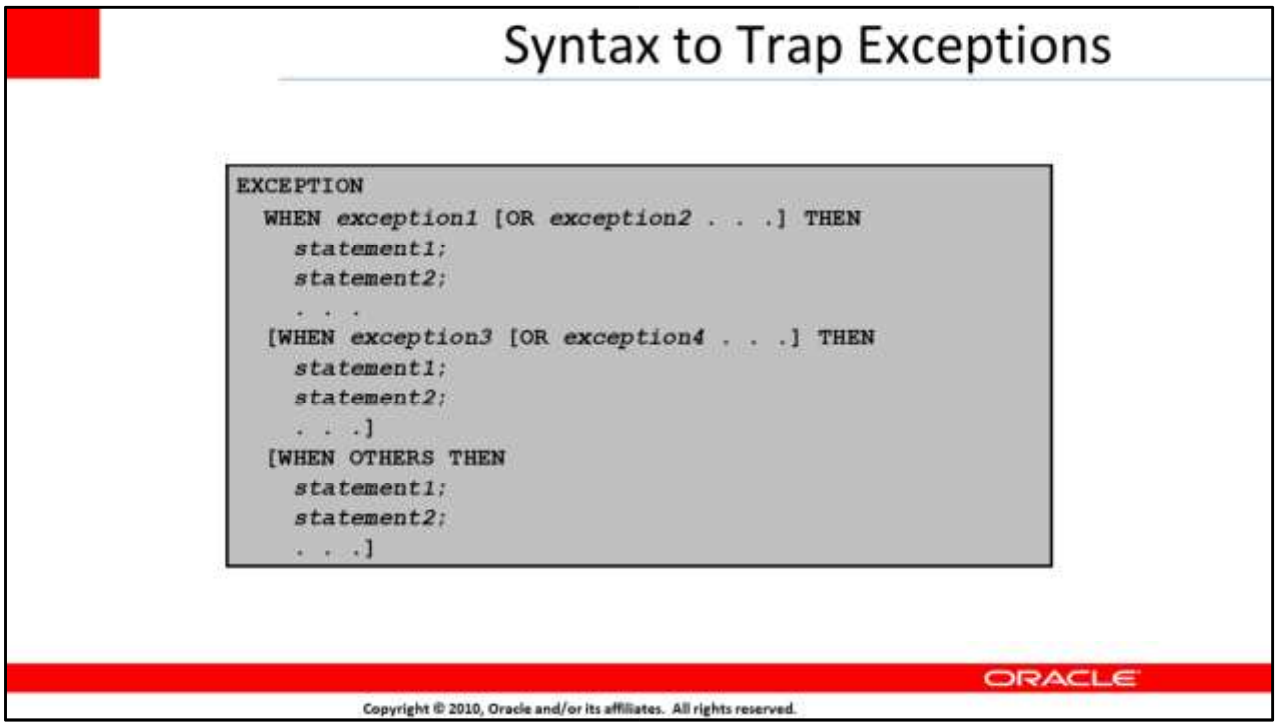


Syntax to Trap Exceptions

You can trap any error by including a corresponding handler within the exception-handling section of the PL/SQL block. Each handler consists of a WHEN clause, which specifies an exception name, followed by a sequence of statements to be executed when that exception is raised.

You can include any number of handlers within an EXCEPTION section to handle specific exceptions. However, you cannot have multiple handlers for a single exception.

Exception trapping syntax includes the following elements:

exception	Is the standard name of a predefined exception or the name of a userdefined exception declared within the declarative section
statement	Is one or more PL/SQL or SQL statements
OTHERS	Is an optional exception-handling clause that traps any exceptions that have not been explicitly handled

Exception Trapping Syntax (continued)

WHENOTHERS Exception Handler

As stated previously, the exception-handling section traps only those exceptions that are specified.

To trap any exceptions that are not specified, you use the OTHERS exception handler. This option traps any exception not yet handled. For this reason, if the OTHERS handler is used, it must be the last exception handler that is defined.

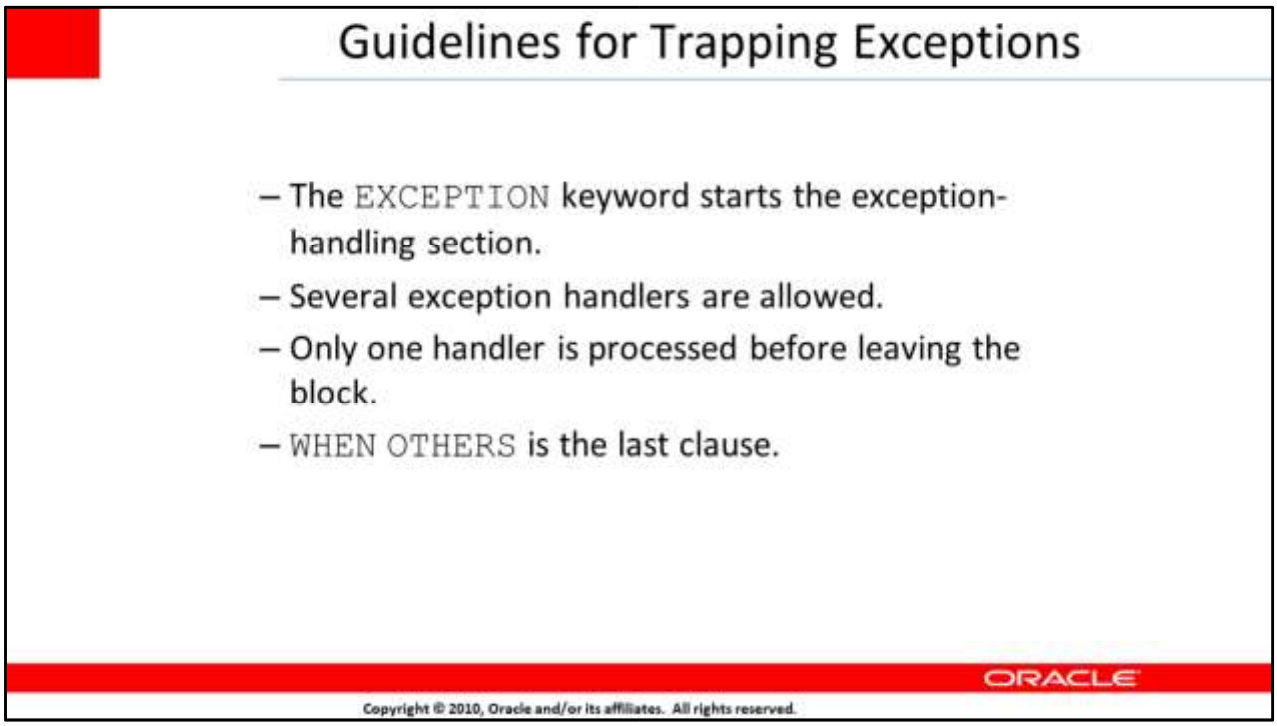
For example:

```
WHEN NO_DATA_FOUND THEN
 statement1;
 ...
WHEN TOO_MANY_ROWS THEN
 statement1;
 ...
WHEN OTHERS THEN
 statement1;
```

Example

Consider the preceding example. If the NO_DATA_FOUND exception is raised by the program, the statements in the corresponding handler are executed. If the TOO_MANY_ROWS exception is raised, the statements in the corresponding handler are executed. However, if some other exception is raised, the statements in the OTHERS exception handler are executed.

The OTHERS handler traps all the exceptions that are not already trapped. Some Oracle tools have their own predefined exceptions that you can raise to cause events in the application. The OTHERS handler also traps these exceptions.



Guidelines for Trapping Exceptions

Begin the exception-handling section of the block with the EXCEPTION keyword.

Define several exception handlers, each with its own set of actions, for the block.

When an exception occurs, PL/SQL processes only one handler before leaving the block.

Place the OTHERS clause after all other exception-handling clauses.

You can have only one OTHERS clause.

Exceptions cannot appear in assignment statements or SQL statements.

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

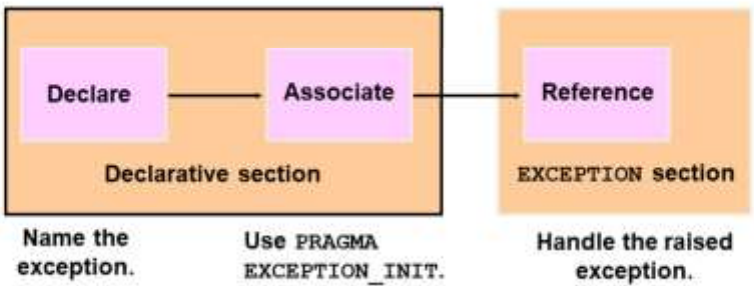
Trapping Predefined Oracle Server Errors

Trap a predefined Oracle Server error by referencing its predefined name within the corresponding exception-handling routine.

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

Note: PL/SQL declares predefined exceptions in the STANDARD package.

Trapping Non-Predefined Oracle Server Errors



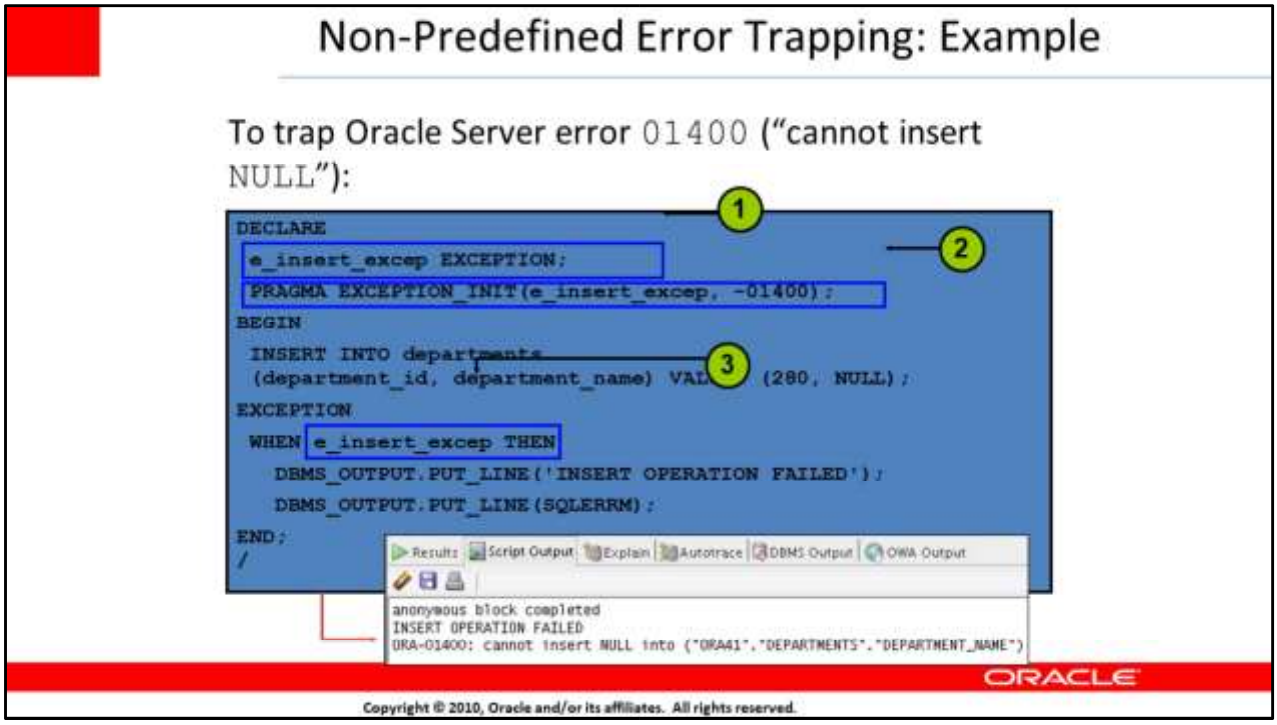
ORACLE

Trapping Non-Predefined Oracle Server Errors

Non-predefined exceptions are similar to predefined exceptions; however, they are not defined as PL/SQL exceptions in the Oracle Server. They are standard Oracle errors. You create exceptions with standard Oracle errors by using the PRAGMAEXCEPTION_INIT function. Such exceptions are called non-predefined exceptions.

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

Note: PRAGMA (also called pseudoinstructions) is the keyword that signifies that the statement is a compiler directive, which is not processed when the PL/SQL block is executed. Rather, it directs the PL/SQL compiler to interpret all occurrences of the exception name within the block as the associated Oracle Server error number.



Non-Predefined Error Trapping: Example

The example illustrates the three steps associated with trapping a nonpredefined error:

1. Declare the name of the exception in the declarative section, using the syntax: exception EXCEPTION; In the syntax, exception is the name of the

exception.

2. Associate the declared exception with the standard Oracle Server error number by using the PRAGMA EXCEPTION_INIT function. Use the following syntax:

```
PRAGMA EXCEPTION_INIT(exception, error_number);
```

In the syntax, exception is the previously declared exception and error_number is a standard Oracle Server error number.

3. Reference the declared exception within the corresponding exception-handling routine.

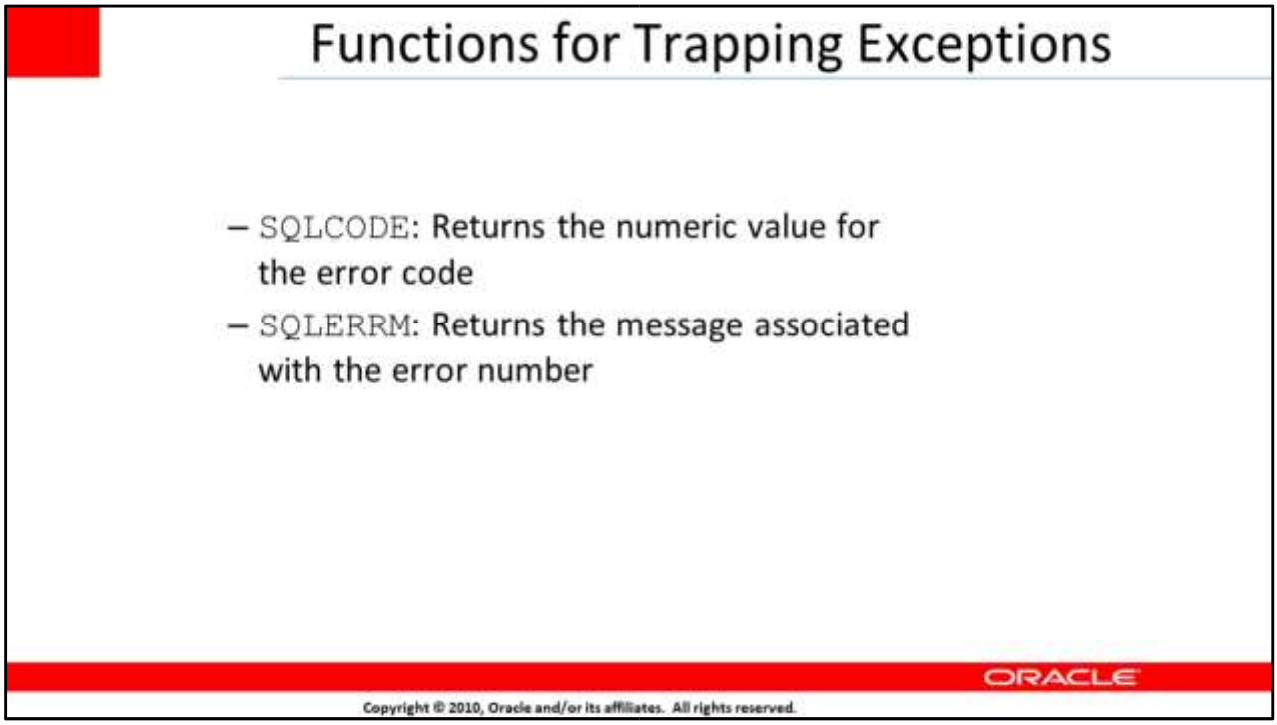
Example

The example in the slide tries to insert the NULL value for the

department_name column of the departments table. However, the operation is not successful because department_name is a NOTNULL column. Note the following line in the example:

```
DBMS_OUTPUT.PUT_LINE(SQLERRM);
```

The SQLERRM function is used to retrieve the error message. You learn more about SQLERRM in the next few slides.



Functions for Trapping Exceptions

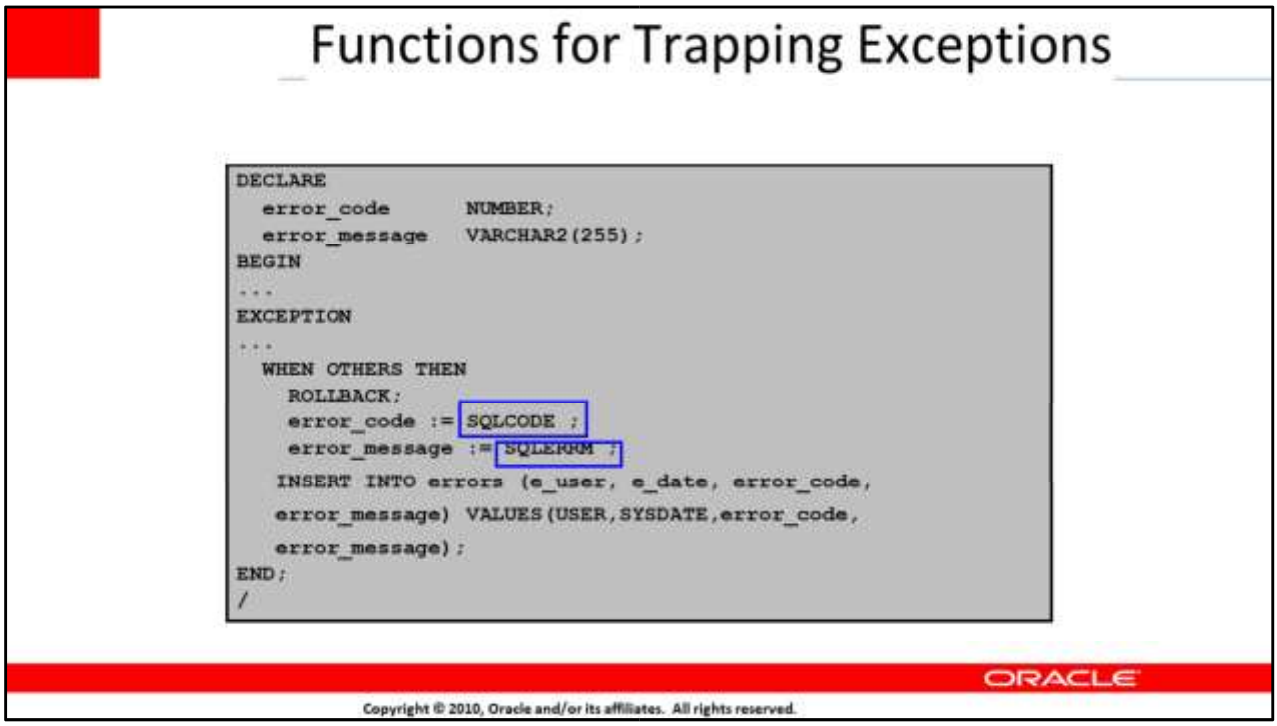
When an exception occurs, you can identify the associated error code or error message by using two functions. Based on the values of the code or the message, you can decide which subsequent actions to take.

SQLCODE returns the Oracle error number for internal exceptions. SQLERRM returns the message associated with the error number.

Function SQLCODE	Values: Examples Description
SQLCODE	Returns the numeric value for the error code (You can assign it to a NUMBER variable.)
SQLERRM	Returns character data containing the message associated with the error number

SQLCODE Value	Description
0	No exception encountered
1	User-defined exception

+100	NO_DATA_FOUND exception
negative number	Another Oracle server error number



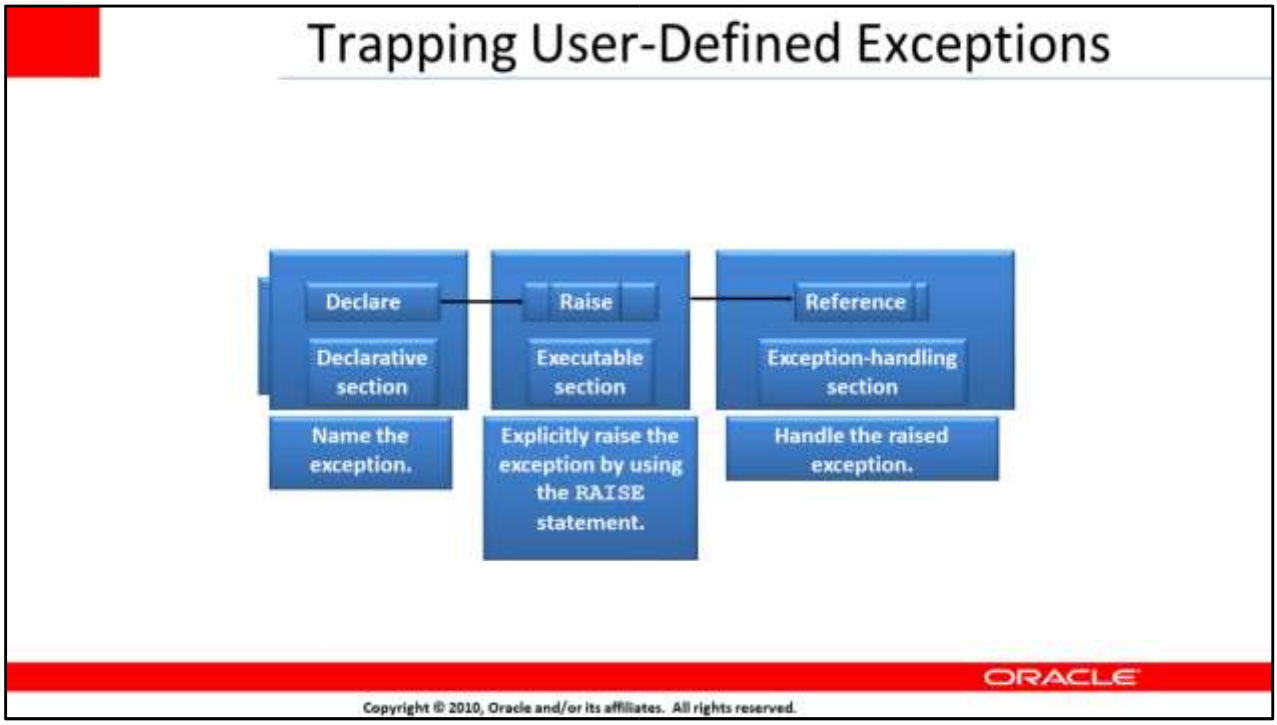
Functions for Trapping Exceptions (continued)

When an exception is trapped in the WHENOTHERS exception handler, you can use a set of generic functions to identify those errors. The example in the slide illustrates the values of SQLCODE and SQLERRM assigned to variables, and then those variables being used in a SQL statement.

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

```
DECLARE err_num NUMBER; err_msg VARCHAR2(100);
BEGIN
  ...
EXCEPTION
  ...
  WHEN OTHERS THEN
    err_num := SQLCODE;
    err_msg := SUBSTR(SQLERRM, 1, 100);
 INSERT INTO errors VALUES (err_num, err_msg);
```

```
END;
/
```



Trapping User-Defined Exceptions

PL/SQL enables you to define your own exceptions depending on the requirements of your application. For example, you may prompt the user to enter a department number.

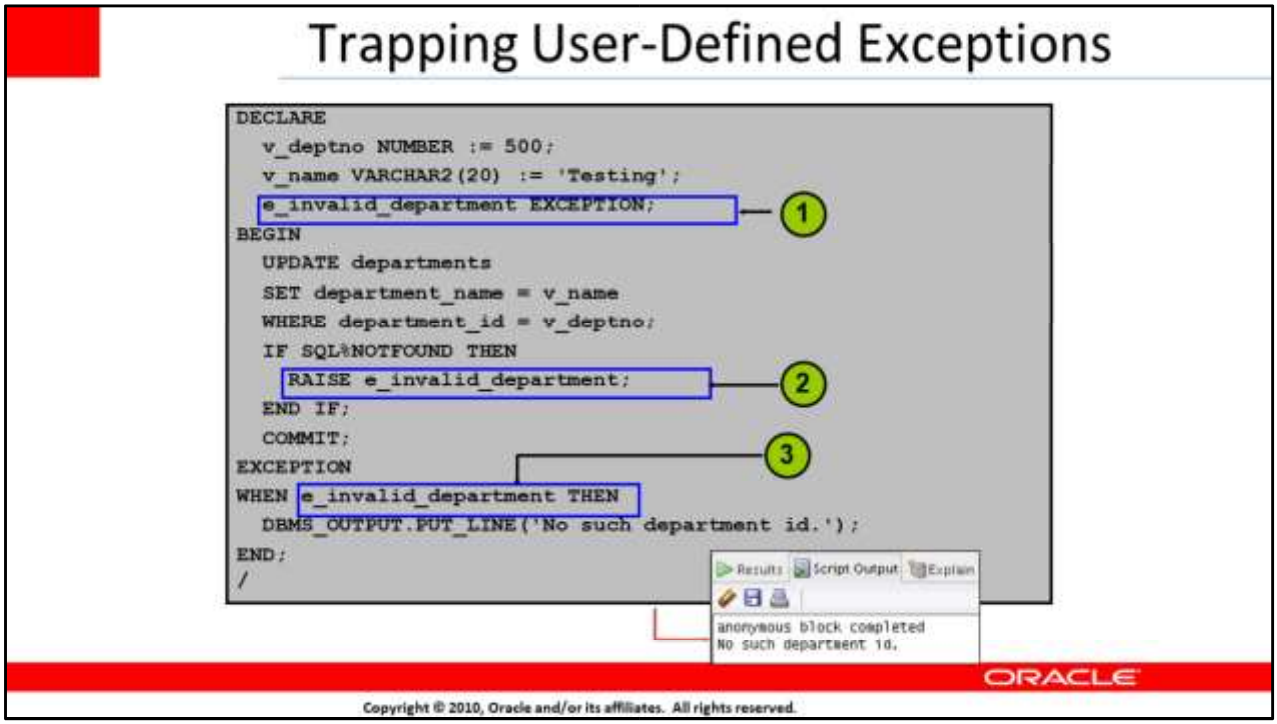
Define an exception to deal with error conditions in the input data. Check whether the department number exists. If it does not, you may have to raise the user-defined exception.

PL/SQL exceptions must be:

Declared in the declarative section of a PL/SQL block

Raised explicitly with RAISE statements

Handled in the EXCEPTION section



Trapping User-Defined Exceptions (continued)

You trap a user-defined exception by declaring it and raising it explicitly. 1. Declare the name of the user-defined exception within the declarative section.

Syntax:

exception EXCEPTION;

In the syntax, exception is the name of the exception.

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

Syntax:

RAISE exception;

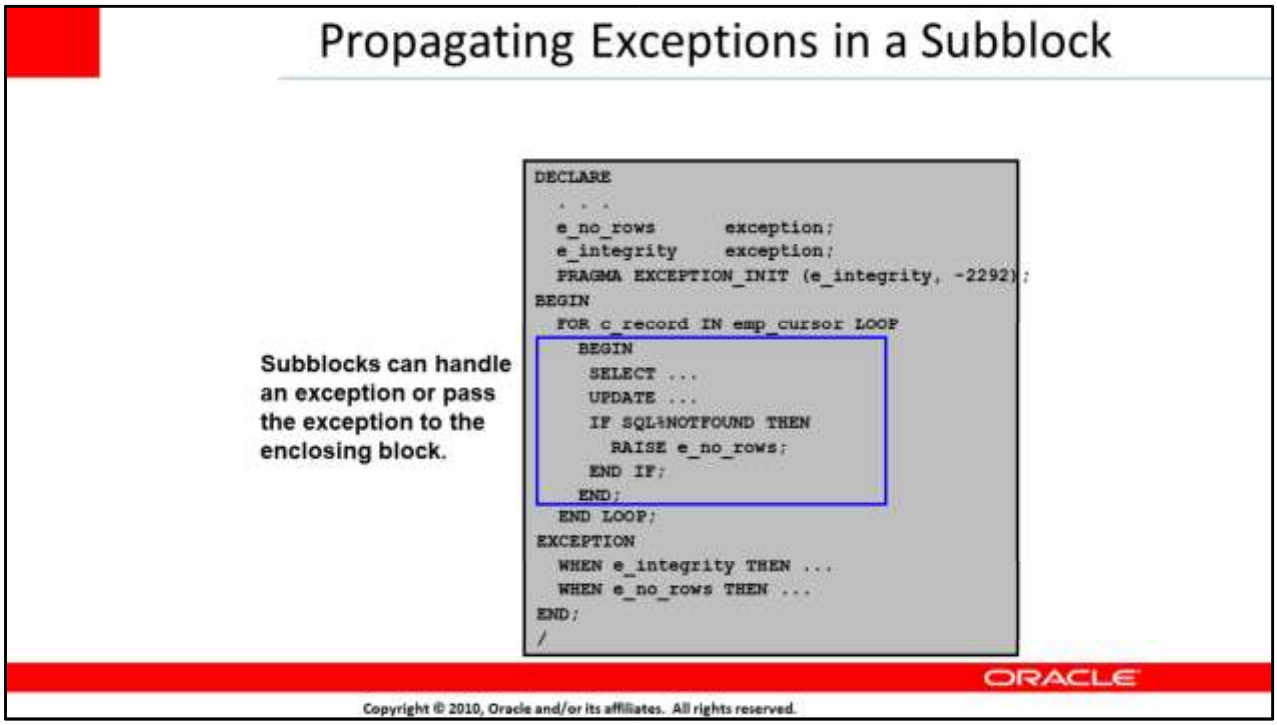
In the syntax, exception is the previously declared exception. 3. Reference the declared exception within the corresponding exception-handling routine.

Example

The block shown in the slide updates the department_name of a department. The user supplies the department number and the new name. If the supplied department number does not exist, no rows are updated in the departments table. An exception is raised

and a message is printed for the user that an invalid department number was entered.

Note: Use the RAISE statement by itself within an exception handler to raise the same exception again and propagate it back to the calling environment.



Propagating Exceptions in a Subblock

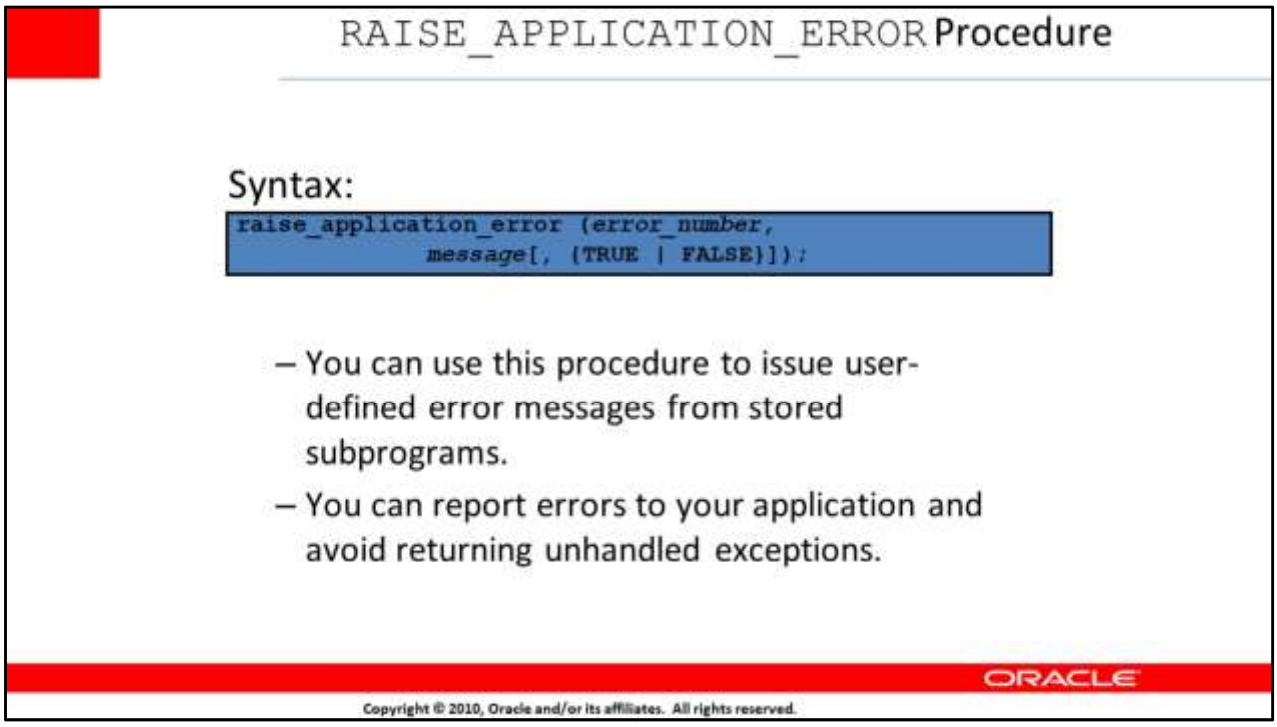
When a subblock handles an exception, it terminates normally. Control resumes in the enclosing block immediately after the subblock's END statement.

However, 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. If none of these blocks handles the exception, an unhandled exception in the host environment results.

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 to the enclosing block.

Note in the example that the exceptions (no_rows and integrity) are declared in the outer block. In the inner block, when the no_rows exception is raised, PL/SQL looks for the exception to be handled in the subblock. Because the exception is not handled in the subblock, the exception propagates to the outer block, where PL/SQL finds the handler.

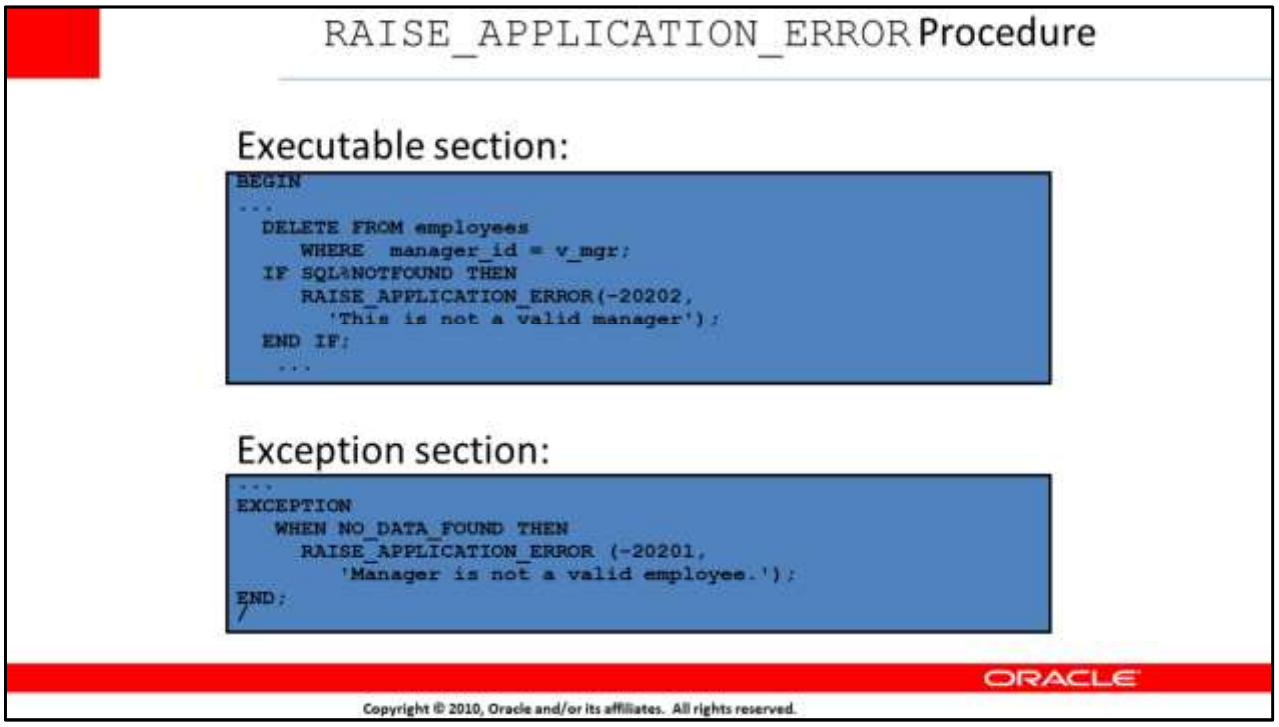


RAISE_APPLICATION_ERROR Procedure

Use the RAISE_APPLICATION_ERROR procedure to communicate a predefined exception interactively by returning a nonstandard error code and error message. With RAISE_APPLICATION_ERROR, you can report errors to your application and avoid returning unhandled exceptions. In the syntax:

error_number	Is a user-specified number for the exception between –20,000 and –20,999
message	Is the user-specified message for the exception; is a character string up to 2,048 bytes long

TRUE | FALSE Is an optional Boolean parameter (If TRUE, the error is placed on the stack of previous errors. If FALSE, which is the default, the error replaces all previous errors.)



RAISE_APPLICATION_ERROR Procedure (continued)

The RAISE_APPLICATION_ERROR procedure can be used in either the executable section or the exception section of a PL/SQL program, or both. The returned error is consistent with how the Oracle Server produces a predefined, non-predefined, or user-defined error. The error number and message are displayed to the user.

RAISE_APPLICATION_ERROR Procedure (continued)

The slide shows that the RAISE_APPLICATION_ERROR procedure can be used in both the executable and the exception sections of a PL/SQL program. Here is another example of using the RAISE_APPLICATION_ERROR procedure:

```
DECLARE e_name
 EXCEPTION;
 BEGIN
 ...
DELETE FROM employees
 WHERE  last_name = 'Higgins';
```

```
  IF SQL%NOTFOUND THEN RAISE
  e_name; END IF;
EXCEPTION
 WHEN e_name THEN
   RAISE_APPLICATION_ERROR (-20999, 'This is
not a valid last name');   ...
END;
/
```