

# 9

## Handling Exceptions

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to do the following:

- Define PL/SQL exceptions
- Recognize unhandled exceptions
- List and use different types of PL/SQL exception handlers
- Trap unanticipated errors
- Describe the effect of exception propagation in nested blocks
- Customize PL/SQL exception messages

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

You 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.

So far it has been assumed that the code works satisfactorily if you take care of compile-time errors. However, the code may cause some unanticipated errors at run time. In this lesson, you learn how to deal with such errors in the PL/SQL block.

# Agenda

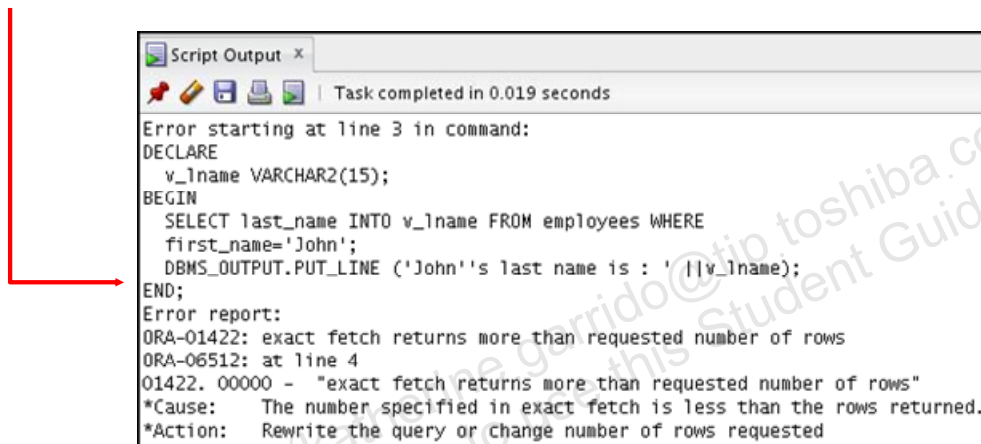
- Understanding PL/SQL exceptions
- Trapping exceptions

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

# What Is an Exception?

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John's last name is : ' || v_lname);
END;
```



Script Output x  
Task completed in 0.019 seconds

Error starting at line 3 in command:  
DECLARE  
 v\_lname VARCHAR2(15);  
BEGIN  
 SELECT last\_name INTO v\_lname FROM employees WHERE  
 first\_name='John';  
 DBMS\_OUTPUT.PUT\_LINE ('John's last name is : ' || v\_lname);  
END;  
Error report:  
ORA-01422: exact fetch returns more than requested number of rows  
ORA-06512: at line 4  
01422. 00000 - "exact fetch returns more than requested number of rows"  
\*Cause: The number specified in exact fetch is less than the rows returned.  
\*Action: Rewrite the query or change number of rows requested

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Consider the example shown in the slide. There are no syntax errors in the code, which means that you must be able to successfully execute the anonymous block. The `SELECT` statement in the block retrieves the last name of John.

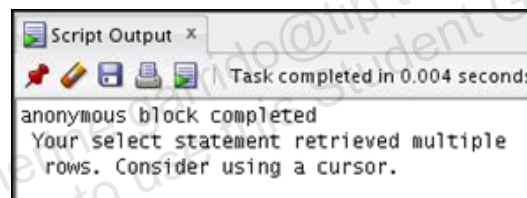
However, you see the following error report when you execute the code:

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
01422. 00000 - "exact fetch returns more than requested number of rows"
*Cause: The number specified in exact fetch is less than the rows returned.
*Action: Rewrite the query or change number of rows requested
```

The code does not work as expected. You expected the `SELECT` statement to retrieve only one row; however, it retrieves multiple rows. Such errors that occur at run time are called *exceptions*. When an exception occurs, the PL/SQL block is terminated. You can handle such exceptions in your PL/SQL block.

# Handling the Exception: An Example

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John's last name 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

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

You have previously learned how to write PL/SQL blocks with a declarative section (beginning with the `DECLARE` keyword) and an executable section (beginning and ending with the `BEGIN` and `END` keywords, respectively).

For exception handling, you include another optional section called the *exception section*.

- This section begins with the `EXCEPTION` keyword.
- If present, this must be the last section in a PL/SQL block.

## Example

In the example in the slide, the code from the previous slide is rewritten to handle the exception that occurred. The output of the code is shown in the slide as well.

By adding the `EXCEPTION` section of the code, the PL/SQL program does not terminate abruptly. When the exception is raised, the control shifts to the exception section and all the statements in the exception section are executed. The PL/SQL block terminates with normal, successful completion

# Understanding Exceptions with PL/SQL

- An exception is a PL/SQL error that is raised during program execution.
- An exception can be raised:
  - Implicitly by the Oracle Server
  - Explicitly by the program
- An exception can be handled:
  - By trapping it with a handler
  - By propagating it to the calling environment

ORACLE

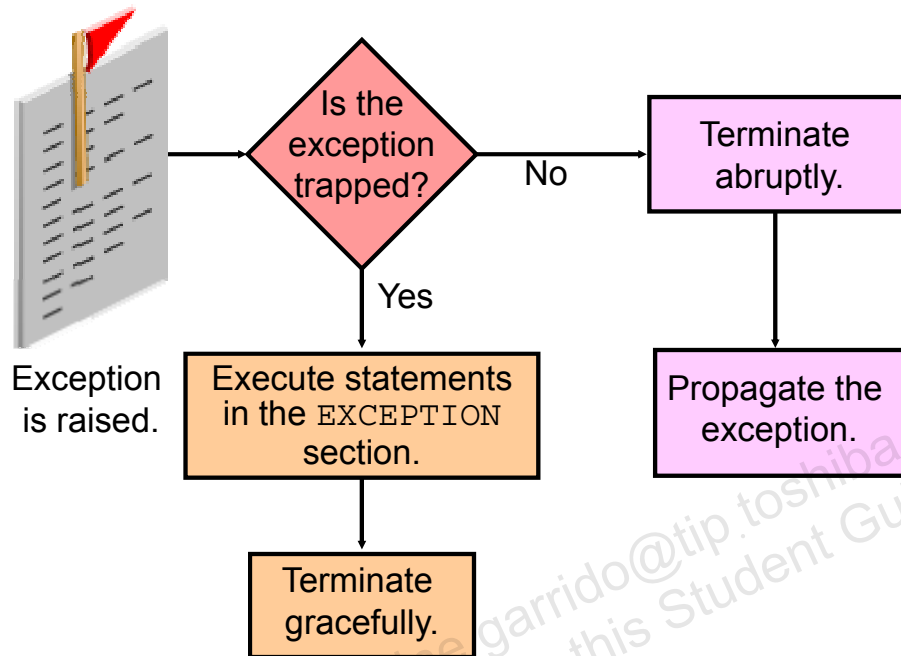
Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

An exception is an error in PL/SQL that is raised during the execution of a block. 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.

## Two Methods for Raising an Exception

- An Oracle error occurs and the associated exception is raised automatically. For example, if the `ORA-01403` error occurs when no rows are retrieved from the database in a `SELECT` statement, PL/SQL raises the `NO_DATA_FOUND` exception. These errors are converted into predefined exceptions.
- Depending on the business functionality your program implements, you may have to explicitly raise an exception. You raise an exception explicitly by issuing the `RAISE` statement in the block. The raised exception may be either user-defined or predefined. There are also some non-predefined Oracle errors. These errors are any standard Oracle errors that are not predefined. You can explicitly declare exceptions and associate them with the non-predefined Oracle errors.

# Handling Exceptions



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

## Trapping an Exception

Include an `EXCEPTION` section in your PL/SQL program to trap exceptions. If the exception is raised in the executable section of the block, processing branches to the corresponding exception handler in the exception section of the block. If PL/SQL successfully handles the exception, the exception does not propagate to the enclosing block or to the calling environment. The PL/SQL block terminates successfully.

## Propagating an Exception

If the exception is raised in the executable section of the block and there is no corresponding exception handler, the PL/SQL block terminates with failure and the exception is propagated to an enclosing block or to the calling environment. The calling environment can be any application (such as SQL\*Plus that invokes the PL/SQL program).

# Exception Types

- Predefined Oracle Server
  - Non-predefined Oracle Server
- } Implicitly raised
- User-defined
- Explicitly raised

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

There are three types of exceptions.

Exception	Description	Directions for Handling
Predefined Oracle Server error	One of approximately 20 errors that occur most often in PL/SQL code	You need not declare these exceptions. They are predefined by the Oracle server and are raised implicitly.
Non-predefined Oracle Server error	Any other standard Oracle Server error	You need to declare these within the declarative section; the Oracle server raises the error implicitly, and you can catch the error in the exception handler.
User-defined error	A condition that the developer determines is abnormal	You need to declare in the declarative section and raise explicitly.

**Note:** Some application tools with client-side PL/SQL (such as Oracle Developer Forms) have their own exceptions.



# Agenda

- Understanding PL/SQL exceptions
- Trapping exceptions

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

# Syntax to Trap Exceptions

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

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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:

<i>exception</i>	Is the standard name of a predefined exception or the name of a user-defined exception declared within the declarative section
<i>statement</i>	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

## WHEN OTHERS 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

- The `EXCEPTION` keyword starts the exception-handling section.
- Several exception handlers are allowed.
- Only one handler is processed before leaving the block.
- `WHEN OTHERS` is the last clause.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

- 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

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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

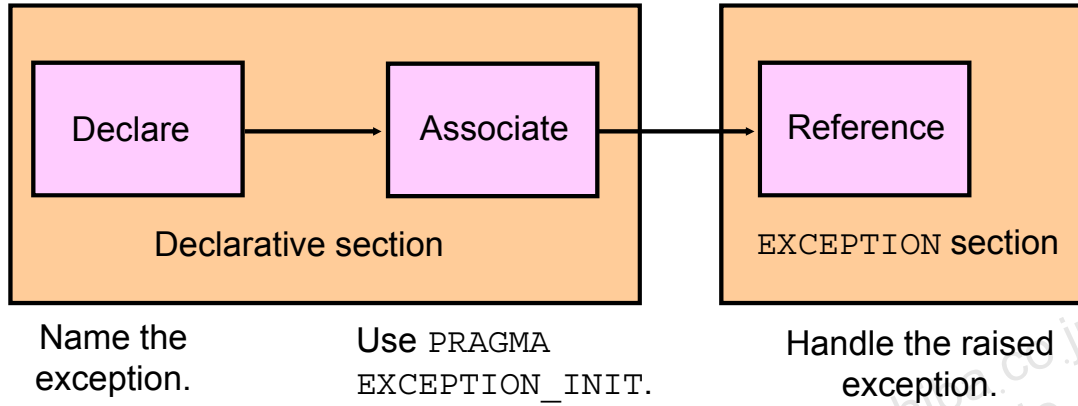
For a complete list of predefined exceptions, see the *PL/SQL User's Guide and Reference*.

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

Exception Name	Oracle Server Error Number	Description
ACCESS_INTO_NULL	ORA-06530	Attempted to assign values to the attributes of an uninitialized object
CASE_NOT_FOUND	ORA-06592	None of the choices in the WHEN clauses of a CASE statement are selected, and there is no ELSE clause.
COLLECTION_IS_NULL	ORA-06531	Attempted to apply collection methods other than EXISTS to an uninitialized nested table or VARRAY
CURSOR_ALREADY_OPEN	ORA-06511	Attempted to open an already open cursor
DUP_VAL_ON_INDEX	ORA-00001	Attempted to insert a duplicate value
INVALID_CURSOR	ORA-01001	Illegal cursor operation occurred.
INVALID_NUMBER	ORA-01722	Conversion of character string to number failed.
LOGIN_DENIED	ORA-01017	Logging on to the Oracle server with an invalid username or password
NO_DATA_FOUND	ORA-01403	Single row SELECT returned no data.
NOT_LOGGED_ON	ORA-01012	The PL/SQL program issues a database call without being connected to the Oracle server.
PROGRAM_ERROR	ORA-06501	PL/SQL has an internal problem.
ROWTYPE_MISMATCH	ORA-06504	The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types.

Exception Name	Oracle Server Error Number	Description
STORAGE_ERROR	ORA-06500	PL/SQL ran out of memory, or memory is corrupted.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Referenced a nested table or VARRAY element by using an index number larger than the number of elements in the collection
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Referenced a nested table or VARRAY element by using an index number that is outside the legal range (for example, -1)
SYS_INVALID_ROWID	ORA-01410	The conversion of a character string into a universal ROWID fails because the character string does not represent a valid ROWID.
TIMEOUT_ON_RESOURCE	ORA-00051	Time-out occurred while the Oracle server was waiting for a resource.
TOO_MANY_ROWS	ORA-01422	Single-row SELECT returned multiple rows.
VALUE_ERROR	ORA-06502	Arithmetic, conversion, truncation, or size-constraint error occurred.
ZERO_DIVIDE	ORA-01476	Attempted to divide by zero

# Trapping Non-Predefined Oracle Server Errors



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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 `PRAGMA EXCEPTION_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

To trap Oracle Server error 01400 (“cannot insert NULL”):

```

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 OPERATION FAILED');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
    
```

Script Output x

Task completed in 0.164 seconds

anonymous block completed  
 INSERT OPERATION FAILED  
 ORA-01400: cannot insert NULL into ("ORA41"."DEPARTMENTS"."DEPARTMENT\_NAME")

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

The example illustrates the three steps associated with trapping a non-predefined 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 NOT NULL 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

- **SQLCODE:** Returns the numeric value for the error code
- **SQLERRM:** Returns the message associated with the error number

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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	Description
SQLCODE	Returns the numeric value for the error code (You can assign it to a <b>NUMBER</b> variable.)
SQLERRM	Returns character data containing the message associated with the error number

### SQLCODE Values: Examples

SQLCODE Value	Description
0	No exception encountered
1	User-defined exception
+100	<b>NO_DATA_FOUND</b> exception
<i>negative number</i>	Another Oracle server error number

# Functions for Trapping Exceptions

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

ORACLE

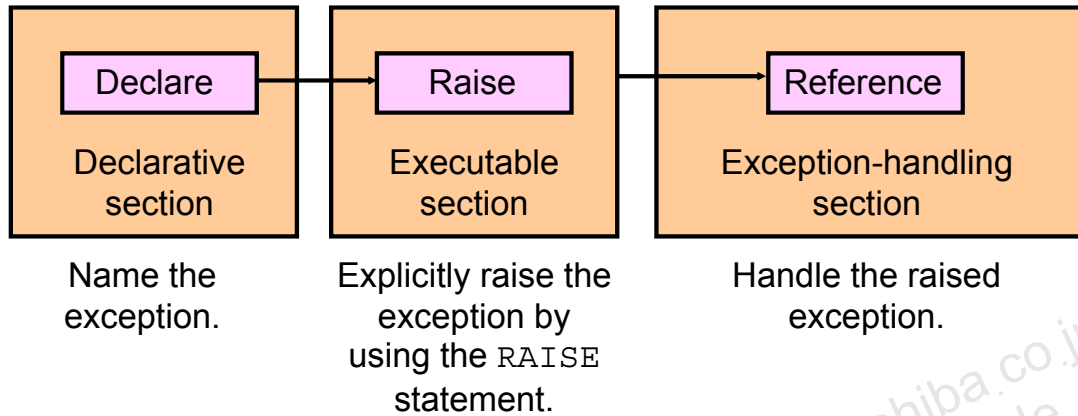
Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

When an exception is trapped in the `WHEN OTHERS` 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



ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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

```
DECLARE
  v_deptno NUMBER := 500;
  v_name VARCHAR2(20) := 'Testing';
  e_invalid_department EXCEPTION;
BEGIN
  UPDATE departments
  SET department_name = v_name
  WHERE department_id = v_deptno;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
```

Script Output x  
Task comp  
anonymous block completed  
No such department id.

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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

Subblocks can handle an exception or pass the exception to the enclosing block.

```
DECLARE
    . . .
    e_no_rows      exception;
    e_integrity     exception;
    PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
    FOR c_record IN emp_cursor LOOP
        BEGIN
            SELECT ...
            UPDATE ...
            IF SQL%NOTFOUND THEN
                RAISE e_no_rows;
            END IF;
        END;
    END LOOP;
EXCEPTION
    WHEN e_integrity THEN ...
    WHEN e_no_rows THEN ...
END;
/
```

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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

Syntax:

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

- You can use this procedure to issue user-defined error messages from stored subprograms.
- You can report errors to your application and avoid returning unhandled exceptions.

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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:

<i>error_number</i>	Is a user-specified number for the exception between –20,000 and –20,999
<i>message</i>	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

- Is used in two different places:
  - Executable section
  - Exception section
- Returns error conditions to the user in a manner consistent with other Oracle Server errors

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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

## Executable section:

```
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;
...
```

## Exception section:

```
...
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR (-20201,
      'Manager is not a valid employee. ');
END;
```

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

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;
/
```

## Quiz

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

- a. True
- b. False

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

### Answer: a

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.

# Summary

In this lesson, you should have learned to:

- Define PL/SQL exceptions
- Add an `EXCEPTION` section to the PL/SQL block to deal with exceptions at run time
- Handle different types of exceptions:
  - Predefined exceptions
  - Non-predefined exceptions
  - User-defined exceptions
- Propagate exceptions in nested blocks and call applications

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learned how to deal with different types of exceptions. In PL/SQL, a warning or error condition at run time is called an exception. Predefined exceptions are error conditions that are defined by the Oracle Server. Non-predefined exceptions can be any standard Oracle Server errors. User-defined exceptions are exceptions specific to your application. The `PRAGMA EXCEPTION_INIT` function can be used to associate a declared exception name with an Oracle Server error.

You can define exceptions of your own in the declarative section of any PL/SQL block. For example, you can define an exception named `INSUFFICIENT_FUNDS` to flag overdrawn bank accounts.

When an error occurs, an exception is raised. Normal execution stops and transfers control to the exception-handling section of your PL/SQL block. Internal exceptions are raised implicitly (automatically) by the run-time system; however, user-defined exceptions must be raised explicitly. To handle raised exceptions, you write separate routines called exception handlers.

## Practice 9: Overview

This practice covers the following topics:

- Creating and invoking user-defined exceptions
- Handling named Oracle Server exceptions

ORACLE

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

In these practices, you create exception handlers for a predefined exception and a standard Oracle Server exception.