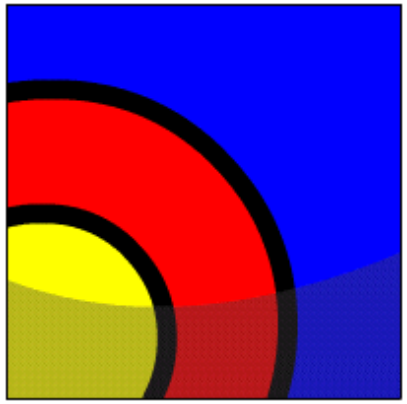
Handling Exceptions

What Will I Learn?

In this lesson, you will learn 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



Why Learn It?

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. So 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.

Tell Me / Show Me

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?

Tell Me / Show Me

Exceptions in PL/SQL

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

```
Republic of Korea

Statement processed.
```

This example works fine. But what if you entered Korea, South instead of Republic of Korea?

Tell Me / Show Me

Exceptions in PL/SQL (continued)

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

Tell Me / Show Me

Exceptions in PL/SQL (continued)

Observe that 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.

```
ORA-01403: no data found
```

This type of error in PL/SQL is called an exception. When an exception occurs, you say that the exception has been “raised.”

When an exception is raised, the rest of the execution section of the PL/SQL block is not executed.

Tell Me / Show Me

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.

Tell Me / Show Me

What Is an Exception Handler? (continued)

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!

Tell Me / Show Me

Why is Exception Handling Important? (continued)

Some reasons include:

- Protects the user from errors (frequent errors can frustrate the user and/or cause the user to quit the application)
- Protects the database from errors (data can be lost or overwritten)
- Major errors take a lot of system resources (if a mistake is made, correcting the mistake can be costly; users might frequently call the help desk for assistance with errors).
- Code is more readable because error-handling routines can be written in the same block in which the error occurred.

Tell Me / Show Me

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. The exception section begins with the keyword EXCEPTION.

```
DECLARE
  v_country_name wf_countries.country_name%TYPE := 'Korea, South';
  v_elevation    wf_countries.highest_elevation%TYPE;
BEGIN
  SELECT highest_elevation INTO v_elevation
    FROM wf_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;
```

Tell Me / Show Me

Handling Exceptions with PL/SQL (continued)

When an exception is handled, the PL/SQL program does not terminate abruptly. When the exception is raised, the control shifts to the exception section and the 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.
```

Only one exception can occur at a time. When an exception occurs, PL/SQL processes only one handler before leaving the block.

Tell Me / Show Me

Handling Exceptions with PL/SQL (continued)

```
DECLARE
  v_country_name wf_countries.country_name%TYPE := 'Korea, South';
  v_elevation    wf_countries.highest_elevation%TYPE;
BEGIN
  SELECT highest_elevation INTO v_elevation
    FROM wf_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;
```

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

Tell Me / Show Me

Handling Exceptions with PL/SQL (continued)

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
```

Tell Me / Show Me

Handling Exceptions with PL/SQL (continued)

The following code includes a handler for a predefined Oracle server error called TOO_MANY_ROWS. You will learn more about predefined server errors in the next lesson.

```
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;
```

Tell Me / Show Me

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;
    . . .]
```

Tell Me / Show Me

Trapping Exceptions (continued)

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

Tell Me / Show Me

Trapping Exceptions (continued)

In the syntax:

- *exception i*s the standard name of a predefined exception or the name of a user-defined 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
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

Tell Me / Show Me

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.

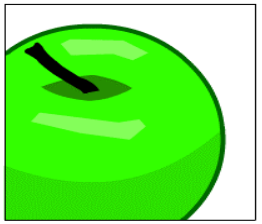
```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

Tell Me / Show Me

Guidelines for 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.
- 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.

Tell Me / Show Me



Terminology

Key terms used in this lesson include:

Exception

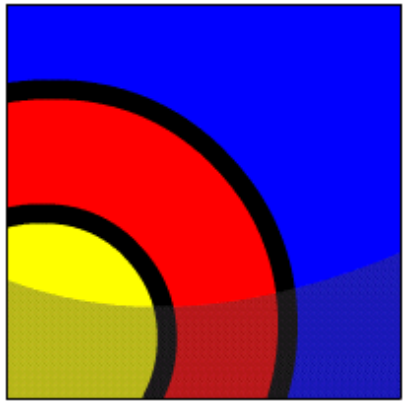
Exception handler

Trapping Oracle Server Exceptions

What Will I Learn?

In this lesson, you will learn 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
- 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

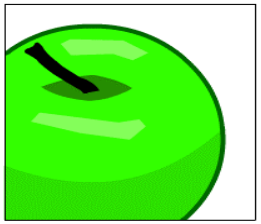


Why Learn It?

PL/SQL error handling is flexible and allows programmers to use both errors defined by the Oracle server and errors defined by the programmer.

This lesson discusses predefined and non-predefined Oracle server errors. Predefined errors are the common Oracle errors for which PL/SQL has predefined exception names. Non-predefined errors make use of the ORA error codes and messages. The syntax is different for each, but you can trap both kinds of errors in the EXCEPTION section of your PL/SQL program.

Tell Me / Show Me



Exception Types

Exception	Description	Instructions 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 (automatically).
Non-predefined Oracle server error	Any other standard Oracle server error	Declare within the declarative section and allow the Oracle Server to raise them implicitly (automatically).
User-defined error	A condition that the PL/SQL programmer decides is abnormal	Declare within the declarative section, and raise explicitly.

Tell Me / Show Me

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`.
- 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. These are explained in the next lesson.

Tell Me / Show Me

Two Types of Oracle Server Error

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. There are two types of Oracle server errors:

- Predefined Oracle server errors: Each of these errors 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`.
- Non-predefined Oracle server errors: Each of these errors has a standard Oracle error number (ORA-nnnnn) 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.

Tell Me / Show Me

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`
- For a partial list of predefined exceptions, refer to the short list available from the Student Resources in Section 0. For a complete list of predefined exceptions, see the *PL/SQL User's Guide and Reference*.

Tell Me / Show Me

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;
```

Tell Me / Show Me

Trapping Several Predefined Oracle Server Errors

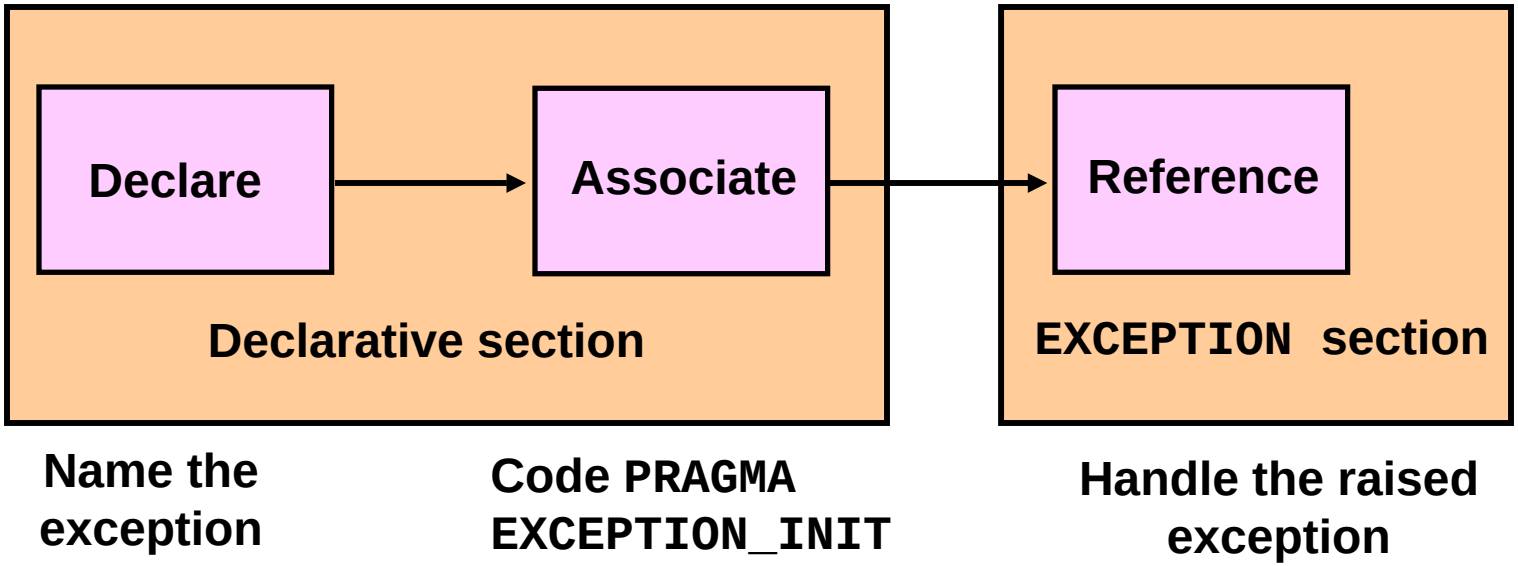
This example handles TOO_MANY_ROWS and NO_DATA_FOUND, with an OTHERS handler in case any other error occurs.

```
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;
```

Tell Me / Show Me

Trapping Non-Predefined Oracle Server Errors

Non-predefined exceptions are similar to predefined exceptions; however, they do not have predefined names in PL/SQL. They are standard Oracle server errors and have ORA- error numbers. You create your own names for them in the DECLARE section and associate these names with ORA- error numbers using the PRAGMA EXCEPTION_INIT function.



Tell Me / Show Me

Trapping Non-Predefined Oracle Server Errors (continued)

- 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 an Oracle error number.

- This allows you to refer to any Oracle Server exception by name and to write a specific handler for it.

Tell Me / Show Me

Non-Predefined Error

Examine the following example.

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

```
ORA-01400: cannot insert NULL into ("USVA_TEST_SQL01_S01"."DEPARTMENTS"."DEPARTMENT_NAME")
```

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 way to work around this problem is to declare you own name and associate it with the ORA-01400 error.

Tell Me / Show Me

Non-Predefined Error (continued)

1. Declare the name of the exception in the declarative section.

```
DECLARE
  e_insert_excep EXCEPTION;                    <-- 1
  PRAGMA EXCEPTION_INIT
      (e_insert_excep, -01400);                <-- 2
BEGIN
  INSERT INTO departments
    (department_id, department_name)
    VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep                          <-- 3
    THEN
      DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

Syntax:

exception name EXCEPTION;

where EXCEPTION is the name of the exception

Tell Me / Show Me

Non-Predefined Error (continued)

2. Associate the declared exception with the standard Oracle server error number using the PRAGMA EXCEPTION_INIT function.

```
DECLARE
  e_insert_excep EXCEPTION;                    <-- 1
  PRAGMA EXCEPTION_INIT
      (e_insert_excep, -01400);                <-- 2
BEGIN
  INSERT INTO departments
  (department_id, department_name)
    VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep                          <-- 3
    THEN
      DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

PRAGMA EXCEPTION_INIT (exception, error_number);

where exception is the previously declared exception name and error_number is a standard Oracle server error number, including the hyphen in front of it.

Tell Me / Show Me

Non-Predefined Error (continued)

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

```
DECLARE
  e_insert_excep EXCEPTION;                    <-- 1
  PRAGMA EXCEPTION_INIT                        <-- 2
      (e_insert_excep, -01400);
BEGIN
  INSERT INTO departments
  (department_id, department_name)
    VALUES (280, NULL);
EXCEPTION
  WHEN e_insert_excep                          <-- 3
    THEN
      DBMS_OUTPUT.PUT_LINE('INSERT FAILED');
END;
```

Tell Me / Show Me

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.)

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

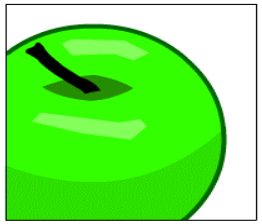
Tell Me / Show Me

Functions for Trapping Exceptions (continued)

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:

```
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;
```

Tell Me / Show Me



Terminology

Key terms used in this lesson include:

Predefined Oracle server errors

Non-predefined Oracle server errors

`PRAGMA EXCEPTION_INIT`

`SQLERRM`

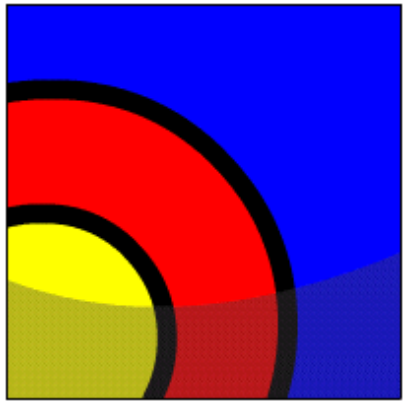
`SQLCODE`

Trapping User-Defined Exceptions

What Will I Learn?

In this lesson, you will learn 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`



Why Learn It?

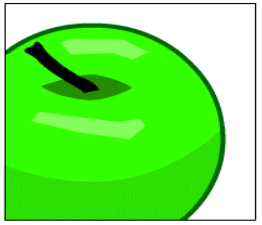
Another kind of error handled by PL/SQL is a user-defined error.

These errors are not automatically raised by the Oracle server, but are defined by the programmer and are specific to the programmer's code.

An example of a programmer-defined error is `INVALID_MANAGER_ID`.

You can define both an error code and an error message for user-defined errors.

Tell Me / Show Me



Exception Types

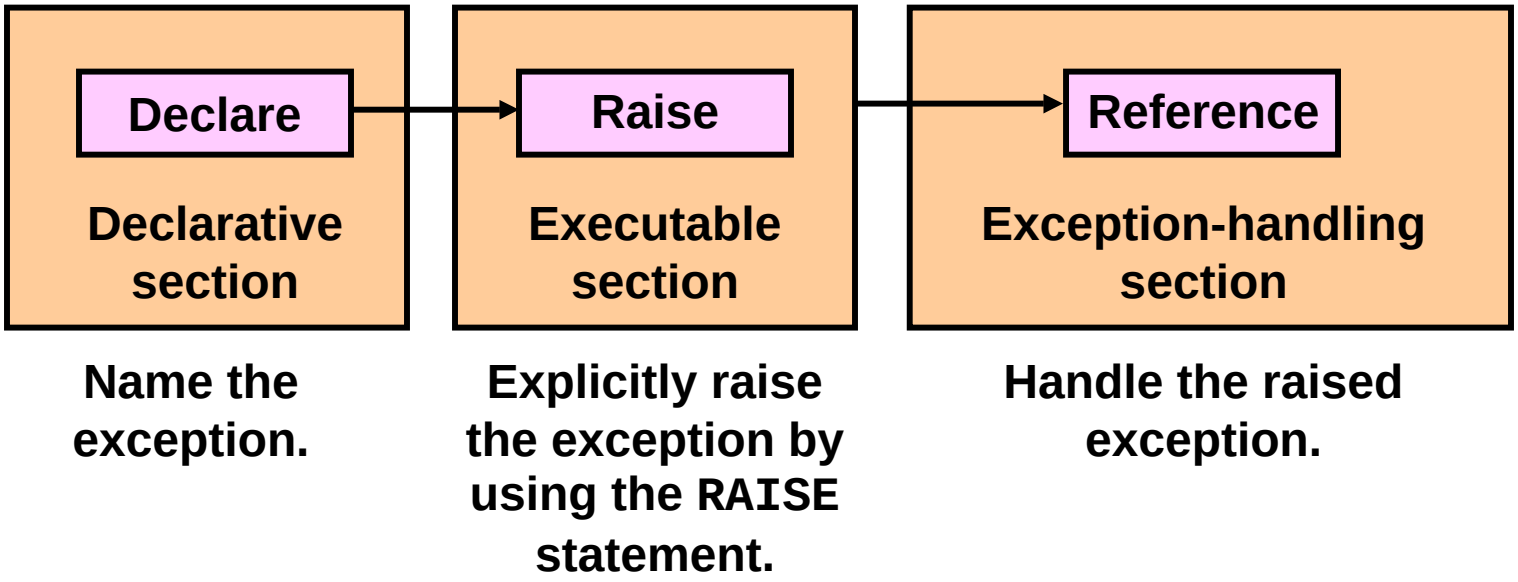
This lesson discusses user-defined errors.

Exception	Description	Instructions 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	Declare within the declarative section and allow the Oracle server to raise them implicitly.
User-defined error	A condition that the developer determines is abnormal	Declare within the declarative section, and raise explicitly.

Tell Me / Show Me

Trapping User-Defined Exceptions

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



Tell Me / Show Me

Trapping User-Defined Exceptions

One example of when you might want to create a user-defined exception is when you need to address error conditions in the input data. For example, let's assume that your program prompts the user for a department number and name so that 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;
```

What happens when the user enters an invalid department? The above code doesn't produce an Oracle error. You need to define a predefined-user error to raise an error.

Tell Me / Show Me

Trapping User-Defined Exceptions (continued)

What happens when the user enters an invalid department? The code as written doesn't produce an Oracle error. You need to define a predefined-user error to raise an error. 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;
```

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

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

Tell Me / Show Me

Trapping User-Defined Exceptions (continued)

The following is the completed code.

```
DECLARE
  e_invalid_department EXCEPTION;
  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;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

1

2

3

Tell Me / Show Me

Trapping User-Defined Exceptions (continued)

1. Declare the name of the user-defined exception within the declarative section. Syntax: *exception* EXCEPTION;

where: *exception* is the name of the exception

```
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;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department                <-- 3
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

Tell Me / Show Me

Trapping User-Defined Exceptions (continued)

2. Use the RAISE statement to raise the exception explicitly within the executable section. Syntax: RAISE *exception*; where: *exception* is the previously declared exception

```
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;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department                <-- 3
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

Tell Me / Show Me

Trapping User-Defined Exceptions (continued)

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

```
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;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department                <-- 3
    THEN DBMS_OUTPUT.PUT_LINE('No such department id.');
         ROLLBACK;
END;
```

Tell Me / Show Me

The RAISE Statement

You can use the RAISE statement to raise a named exception.
You can raise:

- An exception of your own (that is, 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;
```

- 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;
```

Tell Me / Show Me

The RAISE_APPLICATION_ERROR Procedure

You can use the RAISE_APPLICATION_ERROR procedure to return user-defined error messages from stored subprograms. The main advantage of using RAISE_APPLICATION_ERROR instead of RAISE is that RAISE_APPLICATION_ERROR allows you to associate your own error number and meaningful message with the exception. The error numbers must fall between -20000 and -20999.

Syntax:

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

Tell Me / Show Me

The RAISE_APPLICATION_ERROR Procedure (continued)

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

- *error_number* is a user-specified number for the exception between –20000 and –20999
- *message* is the user-specified message for the exception. It 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, the default, the error replaces all previous errors.)

The number range -20000 to -20999 is reserved by Oracle for programmer use, and is never used for predefined Oracle Server errors.

Tell Me / Show Me

The RAISE_APPLICATION_ERROR Procedure (continued)

You can use the RAISE_APPLICATION_ERROR in two different places:

- Executable section
- Exception section

Tell Me / Show Me

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;
```

Tell Me / Show Me

RAISE_APPLICATION_ERROR in the Exception Section

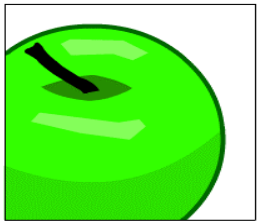
```
DECLARE
  v_mgr          PLS_INTEGER := 27;
  v_employee_id employees.employee_id%TYPE;
BEGIN
  SELECT employee_id into v_employee_id
    FROM employees
    WHERE manager_id = v_mgr;
  DBMS_OUTPUT.PUT_LINE('The employee who works for
       manager_id '||v_mgr||'  is: '||v_employee_id);
EXCEPTION
   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;
```

Tell Me / Show Me

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;
```

Tell Me / Show Me



Terminology

Key terms used in this lesson include:

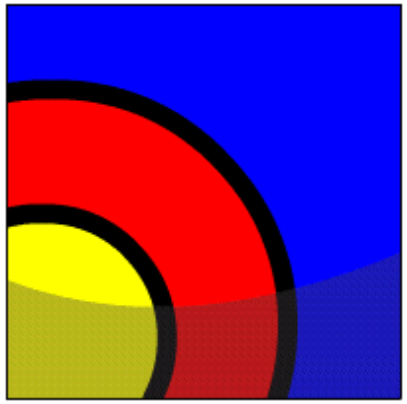
User-defined error

`RAISE`

`RAISE_APPLICATION_ERROR`

Recognizing the Scope of Variables

What Will I Learn?



In this lesson, you will learn to:

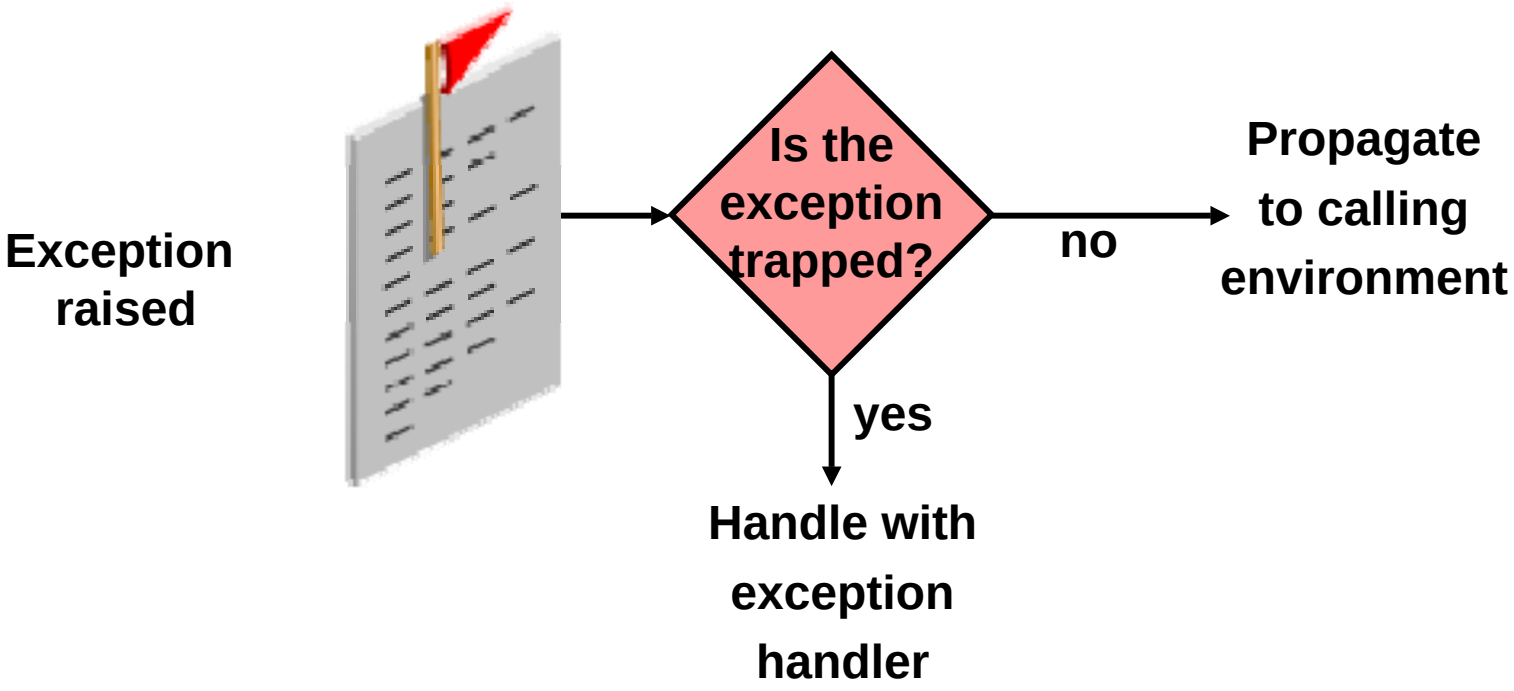
- Describe the rules for variable scope when a variable is nested in a block
- Recognize a variable-scope issue when a variable is used in nested blocks
- Qualify a variable nested in a block with a label
- Describe the scope of an exception
- Recognize an exception-scope issue when an exception is within nested blocks
- Describe the effect of exception propagation in nested blocks

Tell Me / Show Me

Exception Handling in Nested Blocks

You can deal with an exception by:

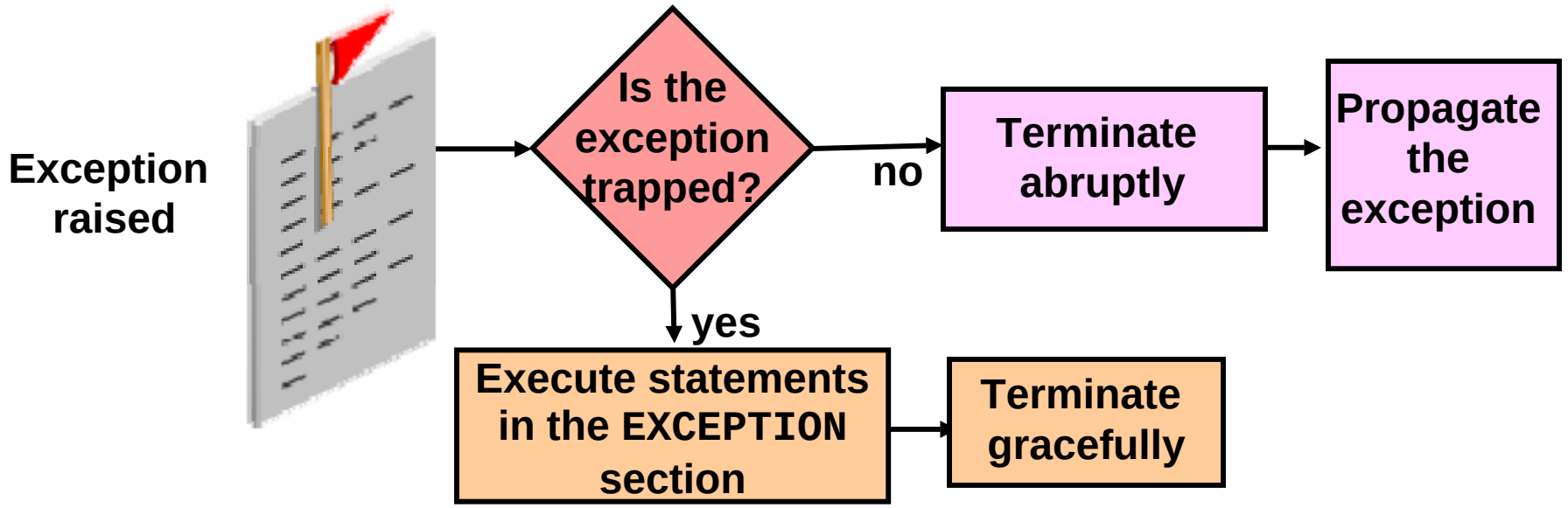
- Handling it ("trapping it") in the block in which it occurs, or
- Propagating it to the calling environment (which can be a higher-level block)



Tell Me / Show Me

Propagating Exceptions to an Outer Block

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



Tell Me / Show Me

Propagating Exceptions to an Outer Block (continued)

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.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.

```
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;
```

Tell Me / Show Me

Propagating Exceptions from a Subblock

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.

Tell Me / Show Me

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;
```

Tell Me / Show Me

Propagating User-named Exceptions from a Sub-Block

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;
```

Tell Me / Show Me

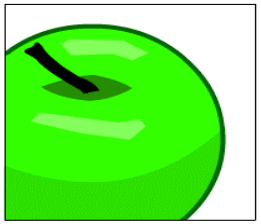
Scope of Exception Names

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.

Therefore, a user-named exception declared within an inner block cannot be referenced in the exception section of an outer block.

Tell Me / Show Me



Terminology

Key terms used in this lesson include:

Scope

Visibility

Qualifier