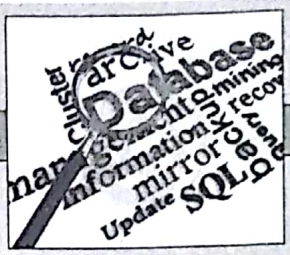
Chapter 31



ERROR HANDLING

CHAPTER OBJECTIVES

In this chapter you will learn:

- Concept of Handling of Errors
- Trapping and Propagating of an Exception
- Advantages of Exceptions
- Exception Types like Implicitly raised exceptions and Explicitly raised exceptions
- Trapping Predefined Oracle Server errors
- Trapping of Non-Predefined Oracle Server errors Or User defined System
- Trapping of User defined Exceptions
- Examples of Error Trapping Functions
- Special cases of Exceptions Raised in Declarations and Exceptions Raised in Handlers

31.1 OVERVIEW OF ERROR HANDLING

There are a numbers of reasons due to which run time errors may be raised during the execution of a PL/SQL block. Although you cannot anticipate all possible errors, you can plan to handle certain kinds of errors meaningful to your PL/SQL program. With PL/SQL, a mechanism called exception handling lets you "bulletproof" your program so that it can continue operating in the presence of errors.

When an error occurs, an exception is raised; normal execution is stopped and control is transferred to an exception-handling section of PL/SQL program. A specific section can be defined in the program to handle exceptions. Separate subroutines called exception handlers can be created to perform all exception processing. Once an exception is raised and control is transferred to the exception part of a program, it cannot return to the execution part of the program.

31.2 HANDLING OF ERRORS

Errors are handles in two ways, one is to trap the error and other is to propagate to the calling environment.

31.2.1 Trapping an exception

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, then the exception does not propagate to the enclosing block or environment. The PL/SQL block terminates successfully as shown in figure 31.1.

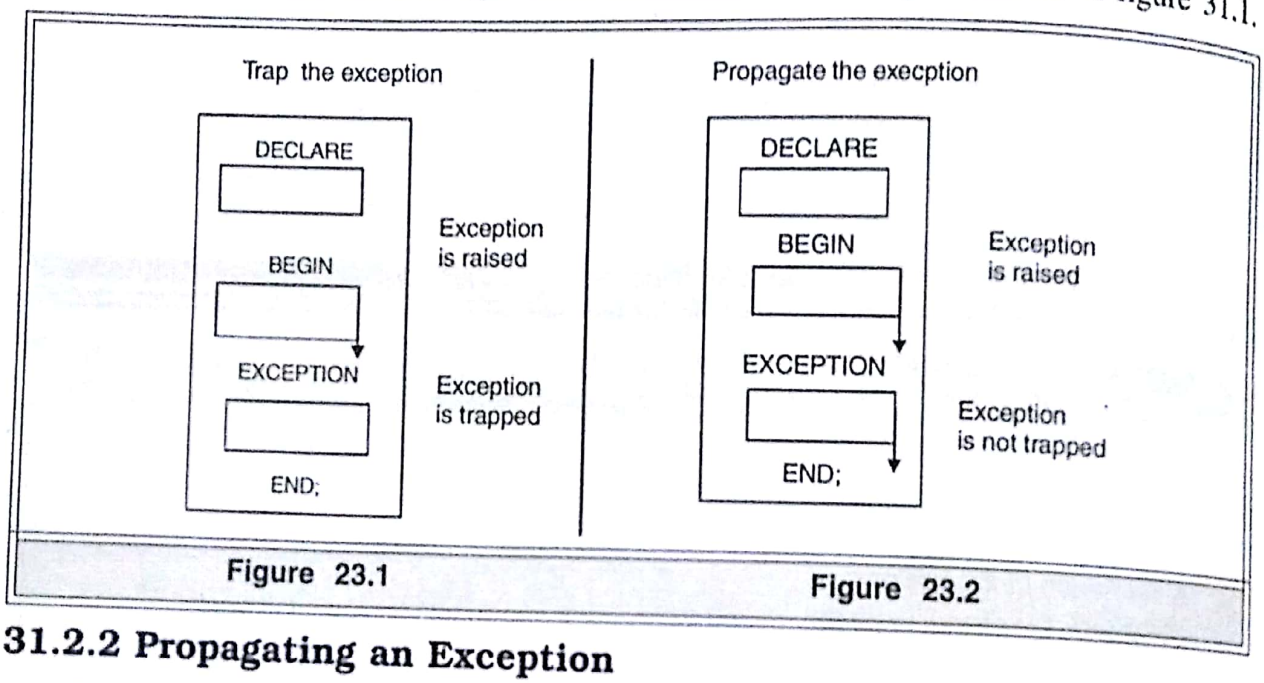


Figure 23.1 Figure 23.2

31.2.2 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 the calling environment as shown in figure 31.2.

31.3 ADVANTAGES OF EXCEPTIONS

Using exceptions for error handling has several advantages. These are:

With exception handling no need to check errors after each SQL command. So, errors processing is not mixed with normal processing. Without exception handling, every time you issue a command, you must check for execution errors, as follows:

```
BEGIN
    SELECT ...
        — check for 'no data found' error
    SELECT ...
        — check for 'no data found' error
    SELECT ...
        — check for 'no data found' error
```

In this case error processing is not clearly separated from normal processing. With exceptions, you can handle errors conveniently without the need to code multiple checks, as follows:

```
BEGIN
   SELECT ...
   SELECT ...
   SELECT ...
   ...
EXCEPTION
   WHEN NO_DATA_FOUND THEN  — catches all 'no data found' errors
END;
```

So, exception provides the mechanism to handle multiple errors, without intermixing the error checks with normal code.

- Exceptions improve readability by letting you isolate error-handling routines.
- Exceptions also improve reliability. You need not worry about checking for an error at every point it might occur. Just add an exception handler to your PL/SQL block. If the exception is ever raised in that block (or any sub-block), you can be sure it will be handled.
- All error conditions can be captured with OTHERS clause in the exception handler, which provides a mechanism to process all expected errors as well as those unforeseen.

31.4 EXCEPTION TYPES

There are two types of exceptions:

- Implicitly raised exceptions
- Explicitly raised exceptions

31.4.1 Implicitly raised exceptions

PL/SQL provides a predefined set of exceptions that are implicitly raised (automatically raised) by the system at run time in case an error is encountered in PL/SQL.

There are two types of implicitly raised exceptions:

- Predefined Oracle Server
- Non-predefined Oracle Server

31.4.2 Explicitly raised exceptions

User can also define their own set of user-defined exceptions and explicitly raise them on encountering an error condition.

Exception	Description	Directions for Handling
Predefined Oracle Server error (Implicitly raised).	Non-predefined Oracle Server error (Implicitly raised).	User-defined error (Explicitly raised).
One of approximately 20 errors that occur most often in PL/SQL code.	Any other standard Oracle Server error.	A condition that the developer determines is abnormal.
Do not declare and allow the Oracle Server to raise them implicitly.	Declare within the declarative section and allow the Oracle Server to raise them implicitly.	Declare within the declarative section and raise explicitly.

31.5 TRAPPING PREDEFINED ORACLE SERVER ERRORS

PL/SQL has a set of exceptions that are predefined for the common Oracle system errors. Predefined exceptions are used to detect and handle Oracle system errors that occur internally at program run time.

An internal exception is raised implicitly whenever your PL/SQL program violates an Oracle rule or exceeds a system-dependent limit. Every Oracle error has a number, but exceptions must be handled by name. So, PL/SQL predefines some common Oracle errors as exceptions. For example, PL/SQL raises the predefined exception NO_DATA_FOUND if a SELECT INTO statement returns no rows.

Syntax:

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

In the syntax

Exception	It is the standard name of a predefined exception or the name of a user defined exception declared within the declarative section.
Statement	It is one ore more PL/SQL or SQL statements.
Others	It is an optional exception-handling clause that traps unspecified exceptions.

WHEN OTHERS Exception Handler

The exception-handling section traps only those exceptions specified; any other exceptions are not trapped unless you use the OTHERS exception handler. This traps any exception not yet handled. For this reason, OTHERS is the last exception handler defined. The OTHERS handler traps all exceptions not already trapped.

Some Guidelines to Trap Exception

- Begin the exception-handling section of the block with the keyword EXCEPTION.
- 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 at most one OTHERS clause.
- Exceptions cannot appear in assignment statements or SQL statements.

List of Some Predefined Exceptions

Exception	Error	Raised if
CURSOR_ALREADY_OPEN	ORA-06511	An attempt is made to OPEN an already open cursor. A Cursor must be CLOSE before it can reopen it.
DUP_VAL_ON_INDEX	ORA-00001	Attempt is made to INSERT or UPDATE duplicate values in a UNIQUE database column.
INVALID_CURSOR	ORA_01001	An illegal cursor operation such as closing an unopened cursor.
INVALID_NUMBER	ORA-01722	The conversion of a character string to a number fails in a SQL statement.
LOGIN_DENIED	ORA_01017	An attempt is made to log on to Oracle with an invalid username/password.
NO_DATA_FOUND	ORA-01403	A SELECT INTO returns no rows, or you refer to an uninitialized row in a PL/SQL table.
NOT_LOGGED_ON	ORA-01012	PL/SQL program issues a database call without being logged on to Oracle.
PROGRAM_ERROR	ORA-06501	PL/SQL has an internal problem such as exiting a function that has no RETURN statement.
STORAGE_ERROR	ORA-06500	PL/SQL runs out of memory or memory is corrupted.
TOO_MANY_ROWS	ORA-01422	A SELECT INTO returns more than one row.
VALUE_ERROR	ORA-06502	The conversion of a character string to a number fails in a procedural statement, or an arithmetic, conversion, truncation, or constraint error occurs.
ZERO_DIVIDE	ORA-01476	An attempt is made to divide a number by zero.

Example 31.1 ***To show use of inbuilt exceptions.***

```
DECLARE
    item_code NUMBER;
BEGIN
    SELECT code INTO item_code From try
    WHERE code=5;
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('TOO MANY ROWS');
WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO DATA FOUND');
END;
```

31.6 TRAPPING OF NON-PREDEFINED ORACLE SERVER ERRORS OR USER DEFINED SYSTEM

PL/SQL provides a limited set of predefined exceptions for the internal oracle system errors, but it provides a mechanism to declare user-defined exceptions to handle the other unnamed Oracle System errors. An exception variable can be defined and linked to the error using the PRAGMA EXCEPTION_INIT statement in a block's declarative section.

Syntax:

PRAGMA EXCEPTION_INIT (exception_name, error_number);

PRAGMA is the keyword that signifies that the statement is a compiler directive, which is processed at compile time and not at run time. Rather, it directs the PL/SQL compiler to associate an exception name with an Oracle system error number.

Steps to trapping a Non-predefined Oracle Server Exception

1. Declare the name for the exception within the declarative section.

   Syntax

   exception EXCEPTION;

   Here, exception is the name of the exception.

2. Associate the declared exception with the standard Oracle Server error number using the PRAGMA EXCEPTION_INIT statement.

   Syntax

   PRAGMA EXCEPTION_INIT(exception, error_number);

   Here: exception is the previously declared exception.

   error_number is a standard Oracle Server error number.

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

Example 31.2 ***To show use of User Defined System Exceptions***

```
DECLARE
    large_value EXCEPTION;
    PRAGMA EXCEPTION_INIT(LARGE_VALUE,-01438);
```

```
BEGIN
    INSERT INTO try VALUES(8,'STEEL TABLE CHAIR',12);
EXCEPTION
    WHEN large_value THEN
    DBMS_OUTPUT.PUT_LINE('VALUE TOO BIG');
END;
```

Description: The insert statement tries to insert CODE, DESCRIPTION and QTY into the table TRY. If any of the values specified is larger than column width user-defined exception LARGE VALUE will be invoked. (01438 is the error number for the error 'inserted value too large for column').

31.7 TRAPPING USER DEFINED EXCEPTIONS

User defined exceptions are used to handle error conditions specific to a application program. For example in case of Library system the date of issue of a book cannot be more than date of return, if that happens then an error must be raised explicitly by the application programmer and that facility is developed with the help of User defined exceptions. Unlike predefined exceptions, user-defined exceptions must be declared and must be raised explicitly by RAISE statements.

Steps to Trap user-defined exceptions

1. Declare the name for the user-defined exception within the declarative section.

   Syntax

   exception_name EXCEPTION;

   Here: exception_name is the name of the exception.

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

   Syntax

   RAISE exception_name;

   Here: exception_name is the previously declared exception.

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

Declaring User-Defined Exceptions

Exceptions can be declared only in the declarative part of a PL/SQL block, subprogram, or package. You declare an exception by introducing its name, followed by the keyword EXCEPTION.

Syntax

<exception_name> EXCEPTION;

In the following example, you declare an exception named past_due:

```
DECLARE
past_due EXCEPTION;
```

Exception and variable declarations are similar. But remember, an exception is an error condition, not a data item. Unlike variables, exceptions cannot appear in assignment statements or SQL statements. However, the same scope rules apply to variables and exceptions.

Raising User-Defined Exceptions

User-defined exceptions must be explicitly raised when associated error condition is detected. During program execution checking can be done for error conditions specified as EXCEPTION in the DECLARE section of PL/SQL program. If an error condition is encountered, the defined exception is explicitly raised by calling the RAISE statement. RAISE statement for a given exception can be placed anywhere within the scope of that exception.

Example 31.3. ***In the following example, user defined exception is raised when an item becomes out of stock.***

```
DECLARE
    out_of_stock EXCEPTION;
    number_on_hand NUMBER(4);
BEGIN
    .....
    .....
IF number_on_hand < 1 THEN
    RAISE out_of_stock;
    END IF;
    .....
EXCEPTION
        WHEN out_of_stock THEN
    — handle the error
END;
```

We can also raise a predefined exception explicitly as shown below.

```
DECLARE
        acct_type INTEGER;
    ...
BEGIN
    .....
    .....
IF acct_type NOT IN (1, 2, 3) THEN
    RAISE INVALID_NUMBER;  — raise predefined exception
END IF;
    .....
EXCEPTION
    WHEN INVALID_NUMBER THEN
    ROLLBACK;
    .....
END;
```

31.8 ERROR TRAPPING FUNCTIONS

In an exception handler, you can use the functions SQLCODE and SQLERRM to find out which error occurred and to get the associated error message.

Function	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

For internal exceptions, SQLCODE returns the number of the Oracle error. The number that SQLCODE returns is negative unless the Oracle error is no data found, in which case SQLCODE returns +100. SQLERRM returns the corresponding error message. The message begins with the Oracle error code.

For user-defined exceptions, SQLCODE returns +1 and SQLERRM returns the message: User-Defined Exception

If no exception has been raised, SQLCODE returns zero and SQLERRM returns the message: ORA-0000: normal, successful completion

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

The maximum length of an Oracle error message is 512 characters. SQLCODE or SQLERRM cannot be directly used in a SQL statement. Instead, their values must be assigned to local variables, which can be used in the SQL statement.

Example 31.4 : *To show use of Error trapping functions.*

```
DECLARE
    err_num NUMBER;
    err_msg CHAR(100);
BEGIN
    .....
EXCEPTION
    .....
    WHEN OTHERS THEN
    err_num:=SQLCODE;
    err_msg:=SUBSTR(SQLERRM,1,100); /* returns first 100 characters from
    mesassge*/
    INSERT INTO errors VALUES (err_num, err_msg);
END;
```

The maximum length of an Oracle error message is 512 characters,so truncate the value of SQLERRM to a known length before attempting to write it to a variable.

31.9 HOW EXCEPTIONS PROPAGATE ?

When an exception is raised, if PL/SQL cannot find a handler for it in the current block or subprogram, the exception propagates. That is, the exception reproduces itself in successive

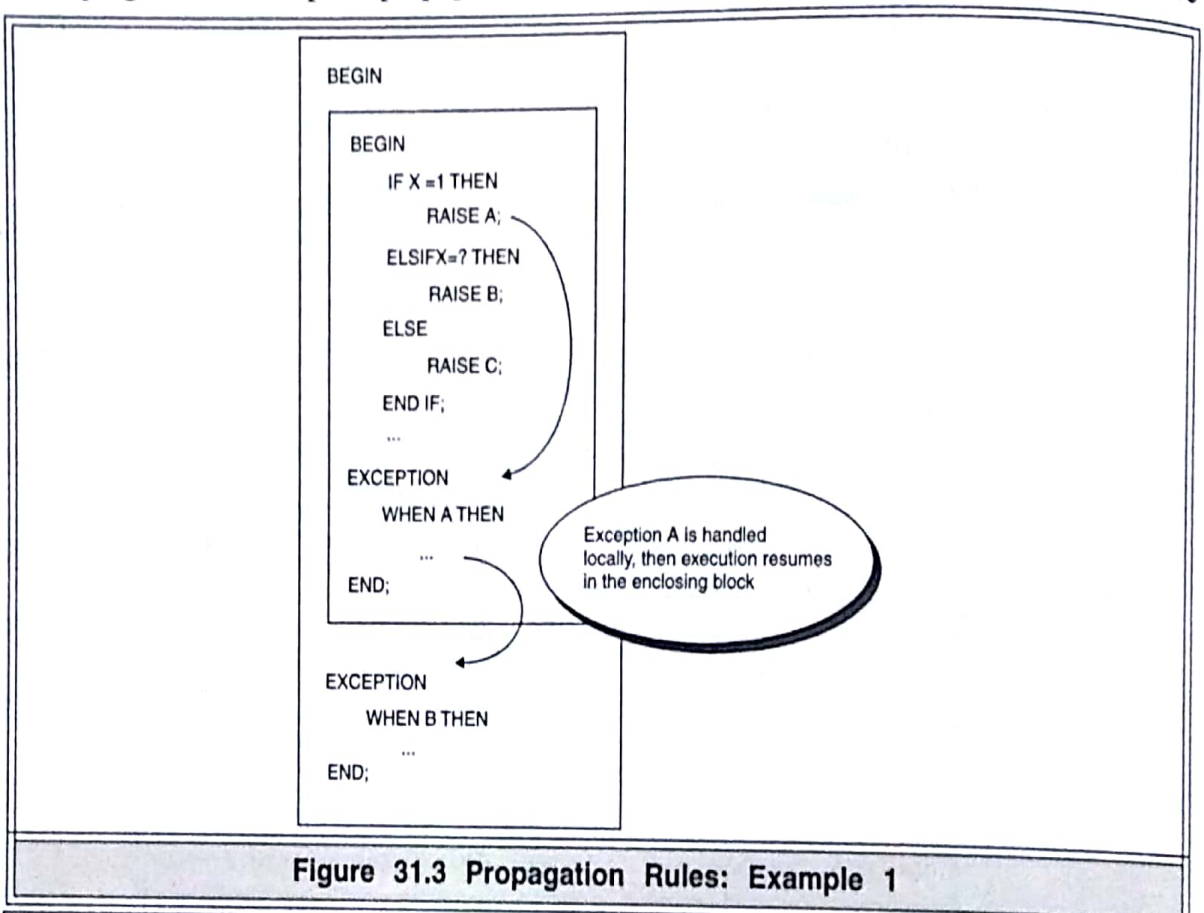


Figure 31.3 Propagation Rules: Example 1

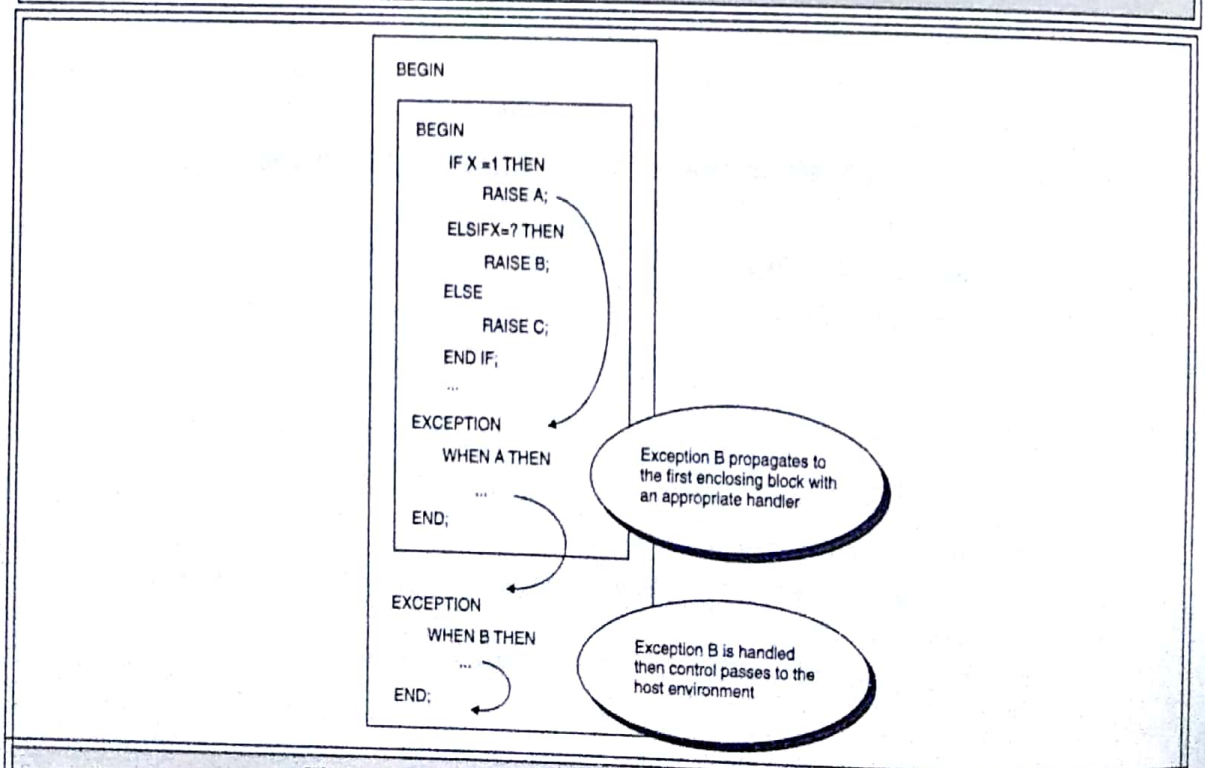


Figure 31.4 Propagation Rules: Example 2

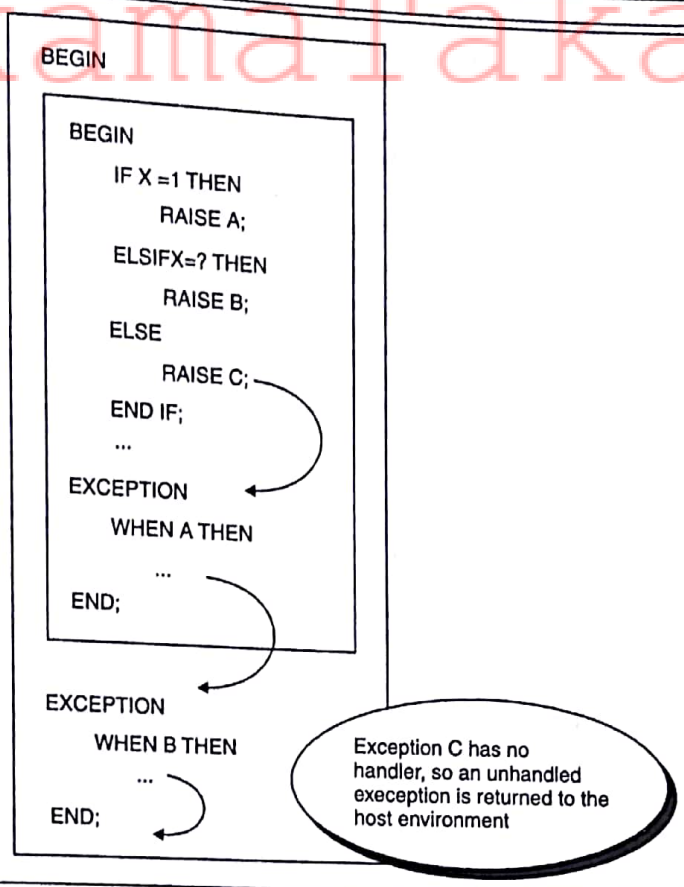


Figure 31.5 Propagation Rules: Example 3

enclosing blocks until a handler is found or there are no more blocks to search. In the latter case, PL/SQL returns an unhandled exception error to the host environment. Different cases of propagation of exceptions are illustrated in figure 21.3, 20.4 and 20.5.

An exception can propagate beyond its scope, that is, beyond the block in which it was declared. Consider the following example:

```
BEGIN
      DECLARE – – sub-block begins
           past_due EXCEPTION;
      BEGIN
           IF ... THEN
                RAISE past_due;
           END IF;
      END; – – sub-block ends
EXCEPTION
      WHEN OTHERS THEN
       ROLLBACK;
END;
```

Description: Because the block in which it was declared has no handler for the exception named *past_due*, it propagates to the enclosing block. But, according to the scope rules, enclosing blocks cannot reference exceptions declared in a sub-block. So, only an OTHERS handler can catch the exception.

31.10 RERAISING OF AN EXCEPTION

Sometimes, we want to *reraise* an exception, that is, handle it locally, then pass it to an enclosing block. For example, we might want to roll back a transaction in the current block, then log the error in an enclosing block. To reraise an exception, simply place a RAISE statement in the local handler, as shown in the following example:

```
DECLARE
    out_of_balance   EXCEPTION;
BEGIN
    ...
    BEGIN  - - sub-block begins
        IF ... THEN
            RAISE out_of_balance;  - - raise the exception
        END IF;
    EXCEPTION
        WHEN out_of_balance THEN
            - - handle the error
            RAISE;  - - reraise the current exception
    END; - - sub-block ends
EXCEPTION
        WHEN out_of_balance THEN
        - - handle the error differently
END;
```

Omitting the exception name in a RAISE statement-allowed only in an exception handler-reraises the current exception.

31.11 HANDLING RAISED EXCEPTIONS

When an exception is raised, normal execution of your PL/SQL block or subprogram stops and control transfers to its exception-handling part, which is formatted as follows:

```
EXCEPTION
    WHEN exception_name1 THEN  — handler
        sequence_of_statements1
    WHEN exception_name2 THEN  — another handler
        sequence_of_statements2
        ...
    WHEN OTHERS THEN           — optional handler
        sequence_of_statements3
```

To catch raised exceptions, you must write exception handlers. Each handler consists of a WHEN clause, which specifies an exception, followed by a sequence of statements to be executed when that exception is raised. These statements complete execution of the block or subprogram; control does not return to where the exception was raised. In other words, you cannot resume processing where you left off.

The optional OTHERS exception handler, which is always the last handler in a block or subprogram, acts as the handler for all exceptions not named specifically. Thus, a block or subprogram can have only one OTHERS handler.

As the following example shows, use of the OTHERS handler guarantees that *no* exception will go unhandled:

```
EXCEPTION
      WHEN ... THEN
            -- handle the error
      WHEN ... THEN
            -- handle the error
      WHEN OTHERS THEN
            -- handle all other errors
END;
```

31.12 MULTIPLE EXCEPTIONS EXECUTE SAME STATEMENTS

If you want two or more exceptions to execute the same sequence of statements, list the exception names in the WHEN clause, separating them by the keyword OR, as follows:

```
EXCEPTION
   WHEN over_limit OR under_limit OR VALUE_ERROR THEN
   -- handle the error
```

If any of the exceptions in the list is raised, the associated sequence of statements is executed. The keyword OTHERS cannot appear in the list of exception names; it must appear by itself.

You can have any number of exception handlers, and each handler can associate a list of exceptions with a sequence of statements. However, an exception name can appear only once in the exception-handling part of a PL/SQL block or subprogram.

31.13 EXCEPTIONS RAISED IN DECLARATIONS

Exceptions can be raised in declarations by faulty initialization expressions. For example, the following declaration raises an exception because the constant *limit* cannot store numbers larger than 999:

```
DECLARE
   limit CONSTANT NUMBER(3) := 5000;  -- raises an exception
BEGIN
         ...
EXCEPTION
         WHEN OTHERS THEN ...  -- cannot catch the exception
END;
```

Handlers in the current block cannot catch the raised exception because an exception raised in a declaration propagates immediately to the enclosing block.

Solution of the problem:

```
BEGIN
    DECLARE
        limit CONSTANT NUMBER(3) := 5000;   - - raises an exception
    BEGIN
            ...
    EXCEPTION
            WHEN OTHERS THEN ...   - - cannot catch the exception
    END;
    EXCEPTION
        WHEN OTHERS THEN ...    - - exception is handled
    END;
```

31.14 EXCEPTIONS RAISED IN HANDLERS

Only one exception at a time can be active in the exception-handling part of a block or subprogram. So, an exception raised inside a handler propagates immediately to the enclosing block, which is searched to find a handler for the newly raised exception. From there on, the exception propagates normally. Consider the following example:

```
EXCEPTION
    WHEN INVALID_NUMBER THEN
        INSERT INTO ...  — might raise DUP_VAL_ON_INDEX
    WHEN DUP_VAL_ON_INDEX THEN ...  — cannot catch the exception.
```

In order to handle the exception raised in the handler following code is used:

```
BEGIN
    BEGIN
        ..............
        ..............
        - - invalid number exception is raised
        ..............
    EXCEPTION
    WHEN INVALID_NUMBER THEN
        INSERT INTO ...  - - might raise DUP_VAL_ON_INDEX
    WHEN DUP_VAL_ON_INDEX THEN ...  - - cannot catch the exception
    END;
EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
    ..... - - exception is handled
END;
```

31.15 BRANCHING TO OR FROM AN EXCEPTION HANDLER

A GOTO statement cannot branch to an exception handler; nor can it branch from an exception handler into the current block. For example, the following GOTO statement is illegal:

```
DECLARE
   pe_ratio NUMBER(3,1);
BEGIN
        DELETE FROM stats WHERE symbol = 'XYZ';
         SELECT price / NVL(earnings, 0) INTO pe_ratio FROM stocks
         WHERE symbol = 'XYZ';
   <<my_label>>
         INSERT INTO stats (symbol, ratio) VALUES ('XYZ', pe_ratio);
EXCEPTION
        WHEN ZERO_DIVIDE THEN
                  pe_ratio := 0;
          GOTO my_label;  — illegal branch into current block
END;
```

However, a GOTO statement can branch from an exception handler into an enclosing block.

31.16 UNHANDLED EXCEPTIONS

Remember, if it cannot find a handler for a raised exception, PL/SQL returns an unhandled exception error to the host environment, which determines the outcome. For example, in the Oracle Precompilers environment, any database changes made by a failed SQL statement or PL/SQL block are rolled back. You can avoid unhandled exceptions by coding an OTHERS handler at the topmost level of every PL/SQL block and subprogram.

Flash Back

When an error occurs, an exception is raised; normal execution is stopped and control is transferred to an exception-handling section of PL/SQL program. A specific section can be defined in the program to handle exceptions. Errors are handled in two ways, one is to trap the error and other is to propagate to the calling environment.

Advantages of Exceptions are: With exception handling no need to check errors after each SQL command, Exceptions improve readability and it also improves reliability. All error conditions can be captured with OTHERS clause in the exception handler.

There are two types of exceptions: Implicitly raised exceptions and Explicitly raised exceptions. There are two types of implicitly raised exceptions: Predefined Oracle Server and Non-predefined Oracle Server.

Exception	Description	Directions for Handling
Predefined Oracle Server error	One of approximately 20 errors that occur most often in PL/SQL code.	Do not declare and allow the Oracle Server to raise them 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.

User can also define their own set of user-defined exceptions and explicitly raise them on encountering an error condition.

PL/SQL provides a limited set of predefined exceptions for the internal oracle system errors, but it provides a mechanism to declare user-defined exceptions to handle the other unnamed Oracle System errors. An exception variable can be defined and linked to the error using the PRAGMA EXCEPTION_INIT statement in a block's declarative section.

User defined exceptions are used to handle error conditions specific to a application program. For example in case of Library system the date of issue of a book cannot be more than date of return, if that happens then an error must be raised explicitly by the application programmer and that facility is developed with the help of User defined exceptions. Unlike predefined exceptions, user-defined exceptions must be declared and must be raised explicitly by RAISE statements. Error Trapping Functions are:

Function	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

Exceptions can be raised in declarations by faulty initialization expressions. Handlers in the current block cannot catch the raised exception because an exception raised in a declaration propagates immediately to the enclosing Only one exception at a time can be active in the exception-handling part of a block or subprogram. So, an exception raised inside a handler propagates immediately to the enclosing block, which is searched to find a handler for the newly raised exception.

REVIEW QUESTIONS

1. What is the importance of error handling? How errors are handled in PL/SQL?
2. What are advantages of Exceptions?
3. What are different types of Exceptions? Explain each with examples.
4. What are implicitly raised exceptions? Name the some predefined exceptions and their importance?
5. How we can trap the Non-Predefined Oracle Server errors? Explain with suitable examples.
6. How we can trap the user defined Exceptions
7. What are Error Trapping Functions? Explain its importance.
8. How we can handle the exceptions raised in declarations?
9. How we can handle the exceptions raised in handlers?

HANDS ON SESSION

1. What are the following ways to handle errors
   (a) Trap (b) Propagate (c) Both a & b (d) None
2. Which of the following are implicitly raised exceptions
   (a) Predefined Oracle Server (b) Post Defined Oracle Server
   (c) a only (c) Both a & b

3. CURSOR_ALREADY_OPEN exceptions are used to attempt
   (a) OPEN an already open cursor (b) OPEN a new cursor
   (c) a & b (c) Depends
4. SQLCODE is used for returning the numeric value
   (a) Error Code (b) Char Code (c) ASCII Code (d) None
5. If the value of SQLCODE returns zero means
   (a) No exception (b) User defined exception
   (c) NO_DATA_FOUND exception (d) Server Error
6. Exception can be declared only in the
   (a) BEGIN (b) Declarative
   (c) Anywhere in the PL/SQL Block (d) None
7. PRAGMA means the statement runs at
   (a) Run time (b) Compile time (c) a & b (d) None
8. What is PRAGMA exception_init used for?
   (a) To declare an Exception.
   (b) To associate an Exception name with an exception number.
   (c) To handle an Exception.
   (d) To associate a function with an exception number.
9. What command do you use to induce an eror condition ?
   (a) raiseerror (b) raise_exception (c) raise (d) Exception
10. Can the RAISE_APPLICATION_ERROR procedure append the message to the existing error stack?
    (a) Yes, this is the default behavior. (b) Yes, but this is not the default behavior.
    (c) No, this Procedure always replaces the error stack.
11. What causes a TOO_MANY_ROWS exception ?
    (a) your FETCH buffer is too small.
    (b) A SELECT INTO returned more than one row.
    (c) You performed a Cartesian Product. (d) Your array insert buffer is too small.
12. What does the following code do ?
    Declare
        Invalid_action Exception;
    Begin ——-
    (a) Associate the invalid_action exception.
    (b) Raises an invalid_action exception.
    (c) Handles an invalid_action exception.
    (d) Declares an exception called invalid_action.
13. An exception that is raised in an exception section passes control to where ?
    (a) The operating System.
    (b) The executable Section in the enclosing block.

(c) The exception section in the enclosing block

(d) The WHEN OTHERS clause in the current exception section.

14. Which of the following types of exceptions cannot be handled in an exception section?

(a) Syntax errors (b) Database errors

(c) Data type mismatch (d) Divide by zero errors

15. What function can you use to retrieve the error number for the current exception ?

(a) ERRNUM (b) SQLCODE (c) SQLNUM

(d) you can only get the exception name, not the numeric code, for an exception.

16. What function can you use to retrieve the text associated with the current exception.

(a) SQLERRM (b) SQLWRRMESG (c) ERRORTEXT (d) ERRMSG

17. What clause do you need to include in your exception section to handle any exception not previously specified.

(a) ELSE (b) WHEN ANYTHING ELSE

(c) WHEN ANY OTHER (d) WHEN OTHERS

18. Which line will cause a compile error (a syntax error) ?

1. declare
2. 54_resource_busy Exception;
3. begin
4. raise zero_divide;
5. Exception
6. when others then

(a) Line 2 (b) Line 4 (c) Line 6 (d) No compile error

19. An oracle defined exception is raised

(a) By the PL/SQL raise statement. (b) In the PL/SQL Exception handler.

(c) Automatically by Oracle. (d) By the others.

20. Which statement most accurately describes the result of not creating an exception handler for the raised exception ?

(a) The Program will continue without raising the exception.

(b) There will be memory leak.

(c) Control will pass to the PL/SQL block caller's Exception handler.

(d) The program will return a %notfound error.

Solution Keys

1. (c)	2. (a)	3. (a)	4. (a)	5. (a)	6. (b)	7. (b)	8. (b)	9. (c)
10. (b)	11. (b)	12. (d)	13. (c)	14. (a)	15. (b)	16. (a)	17. (d)	18. (a)
19. (c)	20. (c)							

●●●