**Exceptions**

Exceptions are used to handle run-time errors in your PL/SQL code. Earlier, you saw the following PL/SQL example that contains an EXCEPTION block:

DECLARE

v\_width INTEGER;

v\_height INTEGER := 2;

v\_area INTEGER := 6;

BEGIN

-- set the width equal to the area divided by the height

v\_width := v\_area / v\_height;

DBMS\_OUTPUT.PUT\_LINE('v\_width = ' || v\_width);

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Division by zero');

END;

The ZERO\_DIVIDE exception is raised when an attempt is made to divide a number by zero. The following example attempts to divide 1 by 0 in the BEGIN block and therefore raises the ZERO\_ DIVIDE exception:

BEGIN

DBMS\_OUTPUT.PUT\_LINE(1 / 0);

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Division by zero');

END;

O.P : Division by zero

BEGIN

DBMS\_OUTPUT.PUT\_LINE(1 / 0);

END;

\* ERROR at line 1: ORA-01476: divisor is equal to zero ORA-06512: at line 2

**The DUP\_VAL\_ON\_INDEX exception** is raised when an attempt is made to store duplicate values in a column that is constrained by a unique index.

The following example attempts to insert a row in the customers table with a customer\_id of 1; this causes DUP\_VAL\_ON\_INDEX to be raised, because the customers table already contains a row with a customer\_id of 1:

BEGIN

INSERT INTO customers ( customer\_id, first\_name, last\_name ) VALUES ( 1, 'Greg', 'Green' );

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Duplicate value on an index');

END;

/

O.P : Duplicate value on an index

**The INVALID\_NUMBER exception** is raised when an attempt is made to convert an invalid character string into a number. The following example attempts to convert the string 123X to

a number that is used in an INSERT, which causes INVALID\_NUMBER to be raised because 123X is not a valid number:

BEGIN

INSERT INTO customers ( customer\_id, first\_name, last\_name ) VALUES ( '123X', 'Greg', 'Green' );

EXCEPTION

WHEN INVALID\_NUMBER THEN

DBMS\_OUTPUT.PUT\_LINE('Conversion of string to number failed');

END;

/

O.P: Conversion of string to number failed

**You can use the OTHERS exception** to handle all exceptions, as shown here:

BEGIN

DBMS\_OUTPUT.PUT\_LINE(1 / 0);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An exception occurred');

END;

/

O.P: An exception occurred

**This is Wrong Order**

SQL> BEGIN

DBMS\_OUTPUT.PUT\_LINE(1 / 0);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An exception occurred');

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Division by zero');

END;

/

**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.

**Trapping Non-Predefined Oracle Server Errors**

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

**User Named Exception PRAGMA EXCEPTION\_INIT, RAISE\_APPLICATION\_ERROR**

PL/SQL user named exception. you can define your own error message and error number using Pragma EXCEPTION\_INIT or RAISE\_APPLICATION\_ERROR function.

### PL/SQL pragma EXCEPTION\_INIT

pragma EXCEPTION\_INIT : Pragma is a keyword directive to execute proceed at compile time. pragma EXCEPTION\_INIT function take this two argument,

1. exception\_name
2. error\_number

You can define pragrma EXCEPTION\_INIT in DECLARE BLOCK on your program.

PRAGMA EXCEPTION\_INIT(exception\_name, -error\_number);

exception\_name and error\_number define on yourself, where exception\_name is character string up to 2048 bytes suppot and error\_number is a negative integer range from -20000 to -20999.

DECLARE

myex EXCEPTION;

PRAGMA EXCEPTION\_INIT(myex,-20015);

n NUMBER := &n;

BEGIN

FOR i IN 1..n LOOP

dbms\_output.put.line(i);

IF i=n THEN

RAISE myex;

END IF;

END LOOP;

EXCEPTION

WHEN myex THEN

dbms\_output.put.line('loop finish');

END;

#### **Example Result**

n number &n= 5  
1  
2  
3  
4  
5  
loop finish  
  
PL/SQL procedure successfully operation.

### PL/SQL RAISE\_APPLICATION\_ERROR

In PL/SQL RAISE\_APPLICATION\_ERROR function use to assign exception name and exception error code. Define RAISE\_APPLICATION\_ERROR function syntax,

raise\_application\_error(error\_number, error\_message);

DECLARE

myex EXCEPTION;

n NUMBER := &n;

BEGIN

FOR i IN 1..n LOOP

dbms\_output.put.line(i);

IF i=n THEN

RAISE myex;

END IF;

END LOOP;

EXCEPTION

WHEN myex THEN

RAISE\_APPLICATION\_ERROR(-20015, 'loop finish');

END;

#### **Example Result**

n number &n= 5  
1  
2  
3  
4  
5  
ORA-20015: loop finish  
  
PL/SQL procedure successfully operation.

When RAISE\_APPLICATION\_ERROR execute it's return error message and error code looking same as oracle built-in error.