**PL/SQL Error Handling**

(Notes #5)

Topics

1. Overview of Exception Handling
2. Internal Defined Exceptions (Ora-n errors, without name) (page 2)
3. Predefined Exceptions (given name) (page 3)
4. User defined Exceptions and Raise statements (page 5)
5. Naming Internally defined Exceptions (page 8)
6. Retrieving Error Code and Error Message (page 9)
7. Transaction Rollback (page 13)
8. Exception Propagation (page 14)

Appendix, reraise (page 17)

1. **Overview of Exception Handling**

There are two kinds of errors in PL/SQL. One is compilation error, another is run-time error, also called exception.

A program unit will be undergoing a parsing process called compilation. During this process, the compiler will check the program for any syntax (keywords, statements, ...)

or semantics (such as if table is available for the user) failure. If anything fails, the compilation of the unit fails, you will see some appropriate compilation error message.

Run time errors arise from design faults, coding mistakes, hardware failures, and many other sources. Although you cannot anticipate all possible errors, you can plan to handle certain kinds of errors meaningful to your PL/SQL program.

In PL/SQL, warnings, error conditions are called exceptions.

There are two kinds of error conditions:

internally defined system errors and

user-defined application program error.

Exceptions are processed in a specific section of a program: exception handler.

When an error is encountered in a program, an exception is raised, which means normal execution stops, and control transfers to an exception handler.

Internal exceptions are raised implicitly (automatically) by the runtime system.

User-defined exceptions must be raised explicitly by **RAISE** statements, which can also

raise predefined exceptions.

Each block can have an exception section, which can have one or more exception handlers.

* The Exception section consists of handlers for all the exceptions.
* An exception handler contains the code that is executed when the error associated with the exception occurs, and the exception is raised.

After an exception handler runs, the current block stops executing and the control transfers to the next statement in the enclosing block - if any.

If there is no enclosing block, control returns to the host environment.

BEGIN

......

EXCEPTION

WHEN exception\_name\_1 THEN

statements\_1a; [statements\_1b; ] ...

-- exception handler, statements\_1 can be one statement or more

WHEN exception\_name\_2 OR exception\_name\_3 THEN

statements\_2a; [statements\_2b; ] ...

WHEN OTHERS THEN

statements\_3a; [statements\_3b; ] ...

/\* The WHEN OTHERS clause is used to trap all remaining exceptions that

have not been handled by your named system exceptions and named

programmer-defined exceptions. \*/

[raise;] -- reraise the exception, see appendix.

END;

**Example 1**. WHEN OTHERS THEN

DECLARE

emp\_name emp.ename%TYPE;

BEGIN

SELECT ename

INTO emp\_name

FROM emp;

-- Where empno = 7654;

DBMS\_OUTPUT.PUT\_LINE ('employee\_name = ' || emp\_name);

EXCEPTION

**WHEN OTHERS THEN**

DBMS\_OUTPUT.PUT\_LINE ('Exception raised.');

End;

/

Result:

Exception raised.

PL/SQL procedure successfully completed.

-- Even there is error, the program itself is “completed”,

-- Without the Exception, then the program will not run, system will give the

error report -

01422. 00000 - "exact fetch returns more than requested number of rows"

1. **Internally Defined Exceptions** (ORA-n errors)

PL/SQL provides a set of (thousands) internally defined exceptions that used to process internal Oracle system error.

The runtime system raises internally defined exceptions implicitly (automatically).

An internally defined exception always has an error code, but does not have name unless PL/SQL gives it one or the user gives it one. In section 5 of this note,

“Naming Internally defined Exceptions”, we will introduce the method,

PRAGMA EXCEPTION\_INIT (*exception\_name*, *error\_code*) ).

Every system error has an associated number.

for example: ORA-27102 (out of memory).

Oracle Database Error Messages provides more details.

<https://docs.oracle.com/database/121/ERRMG/title.htm>

For example:

**ORA-00000: normal, successful completion**

**Cause:** Normal exit.

**Action:** None

**ORA-00001: unique constraint (*string*.*string*) violated**

**Cause:** An UPDATE or INSERT statement attempted to insert a duplicate key. For Trusted Oracle configured in DBMS MAC mode, you may see this message if a duplicate entry exists at a different level.

**Action:** Either remove the unique restriction or do not insert the key.

**ORA-00060: deadlock detected while waiting for resource**

**Cause:** Transactions deadlocked one another while waiting for resources.

**Action:** Look at the trace file to see the transactions and resources involved. Retry if necessary.

There are two SQLCODE values that do not represent a SQL error:

SQL Error Codes 0 and 100

Error Code Description

0 Successful Completion

100 No (more) data -- it indicates that the SQL operation was successful, but found no data to act upon.

1. **Predefined Exceptions,** page 11-11 PL/SQL manual

(also called Named System Exceptions)

Among the thousands internally defined exceptions, PL/SQL only has given a few (those most common errors) names. In Manual of PL/SQL, table 11-3 on page 11-11 lists these PL/SQL Predefined (or Named system) Exceptions.

PL/SQL declares these predefined exceptions globally in the package STANDARD. The runtime system raises predefined exceptions implicitly (automatically). Because predefined exceptions have names, you can write exception handlers specifically for them. Below are some samples:

Exception Name Error Code (SQLCODE)

- - - - - - - - - - - - - - -

ACCESS\_INTO\_NULL (ORA-6530) -6530

DUP\_VAL\_ON\_INDEX (ORA-00001) -1

CURSOR\_ALREADY\_OPEN(ORA-6511) -6511

VALUE\_ERROR (ORA-6502) -6502

INVALID\_CURSOR(ORA-01001) -1001

NO\_DATA\_FOUND (ORA-01403) +100

INVALID\_NUMBER (ORA-01722) -1722

TOO\_MANY\_ROWS(ORA-01422) -1422

ZERO\_DIVIDE(ORA-01476) -1476

Oracle system error occurs (during the run time) when an Oracle system rule is violated, such as when there is a duplicate index value, an invalid cursor, etc.

Example:

TOO\_MANY\_ROWS, (ORA-01422),

which is raised when a single row select statement returns more than one row.

**Example 1B**. TOO\_MANY\_ROWS

DECLARE

emp\_name emp.ename%TYPE;

BEGIN

SELECT ename

INTO emp\_name

FROM emp ;

-- Where empno = 7654 ;

DBMS\_OUTPUT.PUT\_LINE('employee\_name = ' || emp\_name);

EXCEPTION

**WHEN** TOO\_MANY\_ROWS **THEN**

**DBMS\_OUTPUT.PUT\_LINE('Exception raised, too many rows.');**

End;

/

RESULT

Exception raised, too many rows.

PL/SQL procedure successfully completed.

\*\* replace the exception with an incorrect one, say

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE ('Company had zero earnings.');

End;

then run the program, you will get error message.

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

**Example 2**. ZERO\_DIVIDE

DECLARE

stock\_price NUMBER := 9.73;

net\_earnings NUMBER := 0;

pe\_ratio NUMBER;

BEGIN

pe\_ratio := stock\_price / net\_earnings;

-- raises ZERO\_DIVIDE exception

DBMS\_OUTPUT.PUT\_LINE ('Price/earnings ratio = ' || pe\_ratio);

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE ('Company had zero earnings.');

pe\_ratio := NULL;

END;

/

Result:

Company had zero earnings.

PL/SQL procedure successfully completed.

Using case statement,

DECLARE

stock\_price NUMBER := 9.73;

net\_earnings NUMBER := 2; -- change to := 0, run again

pe\_ratio NUMBER;

BEGIN

pe\_ratio :=

CASE net\_earnings

WHEN 0 THEN NULL

ELSE stock\_price / net\_earnings

END;

DBMS\_OUTPUT.PUT\_LINE ('pe\_ratio is ' || NVL (pe\_ratio, 0));

END;

/

1. **User defined Exceptions and Raise statements**

User defined Exceptions must be given names, declared in the Declare section.

Syntax:

DECLARE

user\_defined\_exception\_name1 EXCEPTION;

user\_defined\_exception\_name2 EXCEPTION;

Scope: they are local to this block, global to the subprogram.

If a user defined EXCEPTION has the same name of predefined exception, the

user defined has the precedence. It is called **Redeclared** Predefined Exceptions, Oracle recommends against redeclaring predefined exceptions.

To raise an exception explicitly, use either the RAISE statement or

RAISE\_APPLICATION\_ERROR procedure.

# Raise: user defined EXCEPTIONS must be explicitly "raised", when an error occurs.

The syntax

RAISE user\_defined\_exception\_name1;

Thus, the control passes to the WHEN exception\_name statement,

Exception Handler: -- most times, they are at the end of the block or loop

EXCEPTION

WHEN user\_defined\_exception\_name\_1 THEN

statements\_1;

WHEN user\_defined\_exception\_name\_2 THEN

statements\_2;

WHEN **OTHERS** THEN

statements\_3;

END;

/

Without "OTHERS", some exception could be "unhandled", it will propagate to

the enclosing block. until to find a right defined exception or pass the control to

the host environment.

**Example 3.** no database table is involved in this sample

(For reraise, you may refer appendix Example 3b at the end of this note)

DECLARE

**salary\_too\_high EXCEPTION;**

current\_salary NUMBER := 20000; -- change this number, then run again

max\_salary NUMBER := 10000;

erroneous\_salary NUMBER;

BEGIN

IF current\_salary > max\_salary THEN

**RAISE** salary\_too\_high; -- raise exception

ELSE

DBMS\_OUTPUT.PUT\_LINE ('Salary ' || current\_salary ||' is fine.');

END IF;

EXCEPTION

**WHEN salary\_too\_high THEN** -- start handling exception

erroneous\_salary := current\_salary;

DBMS\_OUTPUT.PUT\_LINE ('Salary ' || erroneous\_salary ||

' is out of range.');

DBMS\_OUTPUT.PUT\_LINE ('Maximum salary is ' || max\_salary || '.');

current\_salary := max\_salary - 1;

DBMS\_OUTPUT.PUT\_LINE ( 'Revising salary from ' || erroneous\_salary || ' to ' || current\_salary || '.');

**WHEN OTHERS THEN**

DBMS\_OUTPUT.PUT\_LINE ( 'This is exception of OTHERS.');

END;

/

Result:

Salary 20000 is out of range.

Maximum salary is 10000.

Revising salary from 20000 to 9999.

**RAISE\_APPLICATION\_ERROR Procedure**

Oracle provides the RAISE\_APPLICATION\_ERROR procedure (defined in the

DBMS\_STANDARD package) to raise application-specific errors in your application. Its advantage over “raise” is that you can associate an error message with the exception.

This procedure returns the error code and error message of a user-defined exception.

RAISE\_APPLICATION\_ERROR (

error\_number, message [, {TRUE | FALSE}]);

You must have assigned *error\_code* to the user-defined exception with the

EXCEPTION\_INIT pragma. The syntax is:

PRAGMA EXCEPTION\_INIT (*exception\_name*, *error\_code*)

The *error\_code* is an integer in the range between -20,000 and -20,999, and the *message* is a character string of at most 2048 bytes.

If the optional third parameter (named as “keeperrostack”) is TRUE, the error will be added on the stack of previous errors. If the parameter is FALSE (the default), the error will *replace* the previous errors.

RAISE\_APPLICATION\_ERROR is part of package DBMS\_STANDARD, and as with package STANDARD, you do not need to qualify references to it.

**Example 4a. using Raise**

DECLARE

Past\_Due EXCEPTION; -- declare exception

Due\_Date DATE := '06-JAN-2020';

Today DATE := '08-JAN-2020'; -- Change the date, run again

-- or using: Today DATE DEFAULT SYSDATE;

BEGIN

IF TODAY > Due\_Date THEN

**RAISE** Past\_due;

END IF;

DBMS\_OUTPUT.PUT\_LINE ('Account is fine.');

EXCEPTION

WHEN Past\_Due THEN -- handle exception

DBMS\_OUTPUT.PUT\_LINE ('Account past due.');

END;

/

OUTPUT

Account past due.

**Example 4b. Same program, using** RAISE\_APPLICATION\_ERROR

DECLARE

Past\_Due EXCEPTION; -- declare exception with name Past\_due

**PRAGMA EXCEPTION\_INIT** (Past\_Due, **-20000**); -- assign error code to exception

Due\_Date DATE := '06-JAN-2020';

Today DATE := '08-JAN-2020';

BEGIN

IF TODAY > Due\_Date THEN

**RAISE\_APPLICATION\_ERROR** (-20000, 'Account past due.');

-- instead of "RAISE”

END IF;

DBMS\_OUTPUT.PUT\_LINE ('Account is fine.');

EXCEPTION

WHEN Past\_Due THEN -- handle exception

DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(SQLERRM(-20000)));

END;

/

OUTPUT

ORA-20000: Account past due.

1. **Naming Internally Defined Exception**

(Programmer names the Unnamed System Exception)

If you know that your database operations might raise specific **internally defined**

**exceptions** that do not have names, then give them names so that you can write

exception handlers specifically for them. Otherwise, you can handle them only with

**OTHERS** exception handlers.

To give a name to an internally defined exception, do the following in the declarative

part of the appropriate block.

1. Declare the name.

DECLARE

*exception\_name* EXCEPTION;

2. Associate the name with the error code of the internally defined exception.

The syntax is:

**PRAGMA EXCEPTION\_INIT** (*exception\_name*, *error\_code*)

EXCEPTION\_INIT is a compile-time command or Pragma used to associate a name with an internal error code. EXCEPTION\_INIT instructs the compiler to associate an identifier, declared as an EXCEPTION, with a specific error number. Once you have made that association, you can then raise that exception by name and write an explicit WHEN handler that traps the error.

The EXCEPTION\_INIT pragma can appear only in the same declarative part as its

associated exception, anywhere after the exception declaration.

**Example 5. (**in manual, example 11-5 Naming Internally Defined Exception**)**

ORA-00060: deadlock detected while waiting for resource -- internal defined, it does not have a system defined exception name. It is an Unnamed System Exception.

DECLARE

deadlock\_detected EXCEPTION; -- now, user gives it a name

PRAGMA EXCEPTION\_INIT (deadlock\_detected, **-60**);

-- This Pragma must be in the same declarative section

BEGIN

Null;

EXCEPTION

WHEN deadlock\_detected THEN

DBMS\_OUTPUT.PUT\_LINE ('deadlock detected.');

END;

/

The EXCEPTION\_INIT pragma also can associates a user-defined exception name with an error code. Refer example on page 11 on this note.

Table 11-2 Exception categories, page 11-5 PL/SQL manual

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Definer** | **Has error**  **Code** | **Has Name** | **Raised**  **implicitly** | **Raised**  **explicitly** |
| Internally  defined | Runtime  System | Always | No (original)  - - - - - -  Via Pragma | Yes | No (original)  - - - - - -  If with name |
| Predefined | Runtime  System | Always | Always (given by system) | Yes | possible |
| User-defined | User | No (original)  - - - - - -  Via Pragma | Always  (declared by user) | No | Always |

**Note:** Do not mix the Exception name with Error Messages.

1. **Retrieving Error Code and Error Message (of System Exception)**

Oracle provides the built-in functions for user to identify, analyze and respond to the errors that occur in your program.

**SQLCODE** - Returns the numeric value of the error code of the most recently raised exception in your block. If there is no error, SQLCODE returns 0.

**SQLERRM** - Returns the message associated with the error number

If you do not pass an error code, the SQLERRM returns the error message associated with the value returned by the SQLCODE.

If SQLCODE is 0, SQLERRM returns: ora-0000; normal, successful completion

If SQLCODE is 1, SQLERRM returns the string: User-defined Exception

You may use these functions to retrieve the related Error code and error message.

Note: A SQL statement cannot invoke SQLCODE/SQLERRM function.

BEGIN

DBMS\_OUTPUT.PUT\_LINE (SQLERRM (-1403));

END;

result:

ORA-01403: no data found -- same output for SQLERRM (100));

We may try:

DBMS\_OUTPUT.PUT\_LINE ('mesg of code 1: ' || SQLERRM ( 1));

DBMS\_OUTPUT.PUT\_LINE ('mesg of code -1: ' || SQLERRM ( -1));

DBMS\_OUTPUT.PUT\_LINE ('mesg for + 100: ' || SQLERRM ( +100));

DBMS\_OUTPUT.PUT\_LINE ('mesg for -1403: ' || SQLERRM ( -1403));

DBMS\_OUTPUT.PUT\_LINE ('current SQLCODE value is: ' || SQLCODE);

output

mesg of code 1: User-Defined Exception

mesg of code -1: ORA-00001: unique constraint (.) violated

mesg for + 100: ORA-01403: no data found

mesg for -1403: ORA-01403: no data found

current SQLCODE value is: 0

**Example 6.**

DECLARE

error\_code NUMBER;

error\_message VARCHAR2(255);

dept\_name departments.department\_name%TYPE;

BEGIN

SELECT department\_name

INTO dept\_name

FROM departments;

/\* Where manager\_ID = 200; \*/

DBMS\_OUTPUT.PUT\_LINE('For the department with manager ID 200,

its name is ' || dept\_name);

EXCEPTION

WHEN OTHERS THEN

error\_code := SQLCODE;

-- assign SQLCODE value to local variable

error\_message := SQLERRM;

DBMS\_OUTPUT.PUT\_LINE ('This Exception Error Code = ' ||

TO\_CHAR(error\_code) || '; Error Message: ' ||

error\_message);

END;

/

Result:

This Exception Error Code = -1422; Error Message: ORA-01422: exact fetch returns more than requested number of rows

Or changed to:

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE ('This Exception Error Code = '

|| SQLCODE || '; Error Message: ' || SQLERRM);

An SQL statement cannot invoke SQLCODE or SQLERRM. To use their values in a SQL statement, assign them to local variables first.

**Example 7.** Display SQLCODE and SQLERRM values (example in manual 11-23)

DROP TABLE errors;

CREATE TABLE errors (

code NUMBER,

message VARCHAR2(64)

);

Declare

name EMPLOYEES.LAST\_NAME%TYPE;

v\_code NUMBER;

v\_errm VARCHAR2(64);

BEGIN

SELECT last\_name INTO name

FROM EMPLOYEES

WHERE EMPLOYEE\_ID = -1; -- the ID is wrong,

EXCEPTION

WHEN OTHERS THEN

v\_code := SQLCODE;

v\_errm := SUBSTR(SQLERRM, 1, 64); -- returns substring from 1st of length 64

DBMS\_OUTPUT.PUT\_LINE

('Error code ' || v\_code || ': ' || v\_errm);

**INSERT** INTO errors (code, message)

VALUES (v\_code, v\_errm);

-- A SQL statement cannot invoke SQLCODE or SQLERRM

END;

/

Result

Error code 100: ORA-01403: no data found

PL/SQL procedure successfully completed.

SELECT \* FROM errors;

CODE MESSAGE

- - - - - - - - - - - - - - - - - - -- - -

100 ORA-01403: no data found

**Example 8a.**

All user-defined exceptions have an **error code of 1** and the error message “**user-defined exception**” unless you use the EXCEPTION\_INIT pragma, (refer page 7 on this notes).

DECLARE

**e\_too\_many\_employees** EXCEPTION;

v\_current\_employees NUMBER(3);

v\_max\_employees NUMBER(3);

v\_Department\_ID employees.department\_id%TYPE := 50;

v\_error\_code NUMBER;

v\_error\_message VARCHAR2(255);

BEGIN

v\_max\_employees := 40;

SELECT count (\*)

INTO v\_current\_employees

FROM employees

WHERE department\_id = v\_department\_id;

IF v\_current\_employees > v\_max\_employees THEN

RAISE **e\_too\_many\_employees**;

END IF;

EXCEPTION

WHEN **e\_too\_many\_employees** THEN

v\_error\_code := SQLCODE;

v\_error\_message := SQLERRM;

DBMS\_OUTPUT.PUT\_LINE('The department with its ID = ' || v\_department\_id || ' has ' || v\_current\_employees || ' employees, it exceeds the limit of ' || v\_max\_employees || '. ');

DBMS\_OUTPUT.PUT\_LINE('User Defined Exception: ' ||

TO\_CHAR(v\_error\_code) || ' ' || v\_error\_message);

END;

/

RESULT

The department with its ID = 50 has 45 employees, it exceeds the limit of 40.

User Defined Exception: 1 User-Defined Exception

**Example 8b. (related to Example 8)**

**(**use the EXCEPTION\_INIT pragma associating a user-defined exception name with an error code, allowed values between -20,000 to -20,999)

DECLARE

e\_too\_many\_employees EXCEPTION;

**pragma exception\_init** (e\_too\_many\_employees, **-20005**);

v\_current\_employees NUMBER(3);

v\_max\_employees NUMBER(3);

v\_error\_code NUMBER;

v\_error\_message VARCHAR2(255);

BEGIN

v\_max\_employees := 40;

SELECT count (\*)

INTO v\_current\_employees

FROM employees

WHERE department\_id = 50;

IF v\_current\_employees > v\_max\_employees THEN

**RAISE\_APPLICATION\_ERROR**

(-20005, 'Too many employees for department\_id = 50.');

END IF;

EXCEPTION

WHEN e\_too\_many\_employees THEN

v\_error\_code := SQLCODE;

v\_error\_message := SQLERRM;

DBMS\_OUTPUT.PUT\_LINE('My exception: ' ||

TO\_CHAR(v\_error\_code) || ' ' || v\_error\_message);

END;

/

result

My exception: -20005 ORA-20005: Too many employees for department\_id = 50.

1. **Transaction Rollback**

When an exception is raised in a PL/SQL block, it does not rollback your current transaction, even if the block itself issued an INSERT, UPDATE, or DELETE. You must issue your own ROLLBACK statement if you want to clean up your transaction due to the exception.

**Example 9.**

SET SERVEROUTPUT ON;

-- SELECT \* FROM departments; (27 rows selected)

DECLARE

v\_1 NUMBER;

v\_2 NUMBER := 0;

BEGIN

INSERT INTO departments (Department\_ID, Department\_name,

Manager\_ID, Location\_ID) VALUES

(300, 'To\_Be\_Deleted', null, 1700);

v\_1 := 5 / v\_2; -- setup an error, divide by zero

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE ('Processing . . .');

-- ROLLBACK;

END;

/

Output

PL/SQL procedure successfully completed.

Run select \* again, 28 rows selected.

If you uncomment the Rollback, then 27 selected.

If your exception goes unhandled (propagates out of the outermost block), however, most host environments will then **force an automatic, unqualified rollback** of any outstanding changes in your session.

**Example 9b.**

DECLARE

v\_1 NUMBER;

v\_2 NUMBER := 0;

BEGIN

INSERT INTO departments VALUES (301, 'TESTING', null, 1700);

v\_1 := 5 / v\_2;

END;

/

Run select statement before and after executing this sample, in both cases, 27 rows are selected.

1. **Exception Propagation**

If an exception is raised in a (sub) block that has no exception handler for it, then the

exception **propagates** to the outer block. That is, the exception reproduces itself in successive enclosing blocks until either a block has a handler for it or there is no enclosing block.

When Exceptions occur in the declarative or exception section of a PL/SQL block, the Exception handler cannot catch up it as it occurs in execute section (normal case),

**Figure 11-1** in PL/SQL manual page 11-20

BEGIN

BEGIN

IF X = 1 THEN

**RAISE A**;

ELSIF X = 2 THEN

RAISE B;

ELSE

RAISE C;

END IF;

...

EXCEPTION

WHEN **A** THEN

...

END;

EXCEPTION

WHEN B THEN

END;

/

1. For exception A, “RAISE A” transfers control to “EXCEPTION ... WHEN A THEN”; The exception is then handled. Done. Sub Block completed, control transfers to next statement of outer block, that is “Exception ... WHEN B THEN”.
2. For Exception B, “RAISE B” is not handled in inner (sub) block, Exception propagates to outer block “EXCEPTION WHEN B THEN”. It is then handled in the outer block. Control transfers to host environment.
3. For Exception C, “RAISE C” is not handled in inner (sub) block, Exception propagates to outer block “EXCEPTION, WHEN B THEN”; It is not handled in the outer block neither; so, PL/SQL returns an unhandled exception error to host environment. Shown as the diagram below.



**8.1) Exception Raised in the Declarative Section**

Exceptions raised in declarations will not be handled by the EXCEPTION in the current (inner) block (not “BEGIN” yet). An exception raised in a declaration propagates immediately to the enclosing block (or the invoker, or host environment)

**Example 10a. (exception not handled)**

DECLARE

credit\_limit CONSTANT NUMBER(3) := 5000; -- Maximum value is 999

BEGIN

NULL;

EXCEPTION

WHEN VALUE\_ERROR THEN

DBMS\_OUTPUT.PUT\_LINE('Exception raised in declaration.');

END;

/

**Example 10b. (handled by the outer block)**

BEGIN

DECLARE

credit\_limit CONSTANT NUMBER(3) := 5000;

-- Maximum value is 999

BEGIN

NULL;

END;

EXCEPTION

WHEN OTHERS THEN -- cope with unexpected

DBMS\_OUTPUT.PUT\_LINE ('Exception raised in declaration.');

END;

/

Result:

Exception raised in declaration.

**8.2) Exception Raised in the Exception Section**

An exception raised in an exception section propagates immediately to the enclosing block (or to the invoker, or host environment if there is no enclosing block).

**Example 11.** Exception Raised in Exception Handler is handled by Enclosing Block,

manual 11-22

DECLARE

i INTEGER;

i\_is\_one EXCEPTION;

**BEGIN**

**BEGIN**

i := 3; -- assign value 3

LOOP

DBMS\_OUTPUT.PUT\_LINE ('Current value of i is: '||i||'.');

IF i = 1 THEN

RAISE i\_is\_one;

ELSE

DBMS\_OUTPUT.PUT\_LINE ('Reciprocal of '|| i ||

' is '|| TO\_CHAR(1/i, '99.99'));

END IF;

i := i - 1;

DBMS\_OUTPUT.PUT\_LINE ('Last line of loop');

DBMS\_OUTPUT.PUT\_LINE ('After loop, i is: '||i||'.');

END LOOP;

**EXCEPTION**

WHEN i\_is\_one THEN

DBMS\_OUTPUT.PUT\_LINE('1 is its own reciprocal.');

DBMS\_OUTPUT.PUT\_LINE('Reciprocal of ' || TO\_CHAR(i-1) ||

' is ' || TO\_CHAR**(1/(i-1))**);

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Inside Exception Error:');

DBMS\_OUTPUT.PUT\_LINE(1/i || ' is undefined');

**END;**

**EXCEPTION**

**WHEN ZERO\_DIVIDE THEN -- handles exception raised in exception handler**

DBMS\_OUTPUT.PUT\_LINE('Outside Exception Error:');

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

**END;**

/

Result:

Current value of i is: 3.

Reciprocal of 3 is .33

Last line of loop

After loop, i is: 2.

Current value of i is: 2.

Reciprocal of 2 is .50

Last line of loop

After loop, i is: 1.

Current value of i is: 1.

1 is its own reciprocal.

Outside Exception Error:

1/0 is undefined

The processing:

Starting, i = 3; no error, back to loop;

i = 2 ; no error, back to loop;

i = 1; if i = 1 TRUE, so RAISE exception i\_is\_one;

Control transfers to Exception inside that sub block;

Program executes the EXCEPTIOIN WHEN 1\_is\_one,

the second DBMS\_OUTPUT.PUT\_LINE has TO\_CHAR(1/(i-1)) expression,

this equals TO\_CHAR(1/0) - this is error of Zero\_Divive,

This is an exception in the EXCEPTION section, thus control transfers to the

enclosing block (outer block) EXCEPTION, WHEN ZERO\_DIVIDE (not the Exception

in the sub block).

**Appendix, Reraise Exception. refer to manual Example 11-12**

The example 3 in this note is a simplified version of example 11-12.

In this sample, you have nested blocks. The program handles the exception starting from the inner block and finishing in the outer block. The outer block declares the exception, so the exception name exists in both blocks, and each block has an exception handler specifically for that exception. The inner block raises the exception, and its exception handler does the initial handling and then reraises the exception, passing it to the outer block for further handling.

DECLARE

**salary\_too\_high EXCEPTION;**

current\_salary NUMBER := 20000;

max\_salary NUMBER := 10000;

erroneous\_salary NUMBER;

BEGIN

BEGIN

IF current\_salary > max\_salary THEN

**RAISE salary\_too\_high; -- raise exception**

END IF;

EXCEPTION

**WHEN salary\_too\_high THEN -- start handling exception**

erroneous\_salary := current\_salary;

DBMS\_OUTPUT.PUT\_LINE('Salary ' || erroneous\_salary ||

' is out of range.');

DBMS\_OUTPUT.PUT\_LINE ('Maximum salary is ' ||

max\_salary || '.');

**RAISE; -- reraise current exception (exception name is optional)**

END;

EXCEPTION

**WHEN salary\_too\_high THEN -- finish handling exception**

current\_salary := max\_salary - 1 ;

DBMS\_OUTPUT.PUT\_LINE ('Revising salary from ' ||

erroneous\_salary || ' to ' || current\_salary || '.' );

END;

Result:

Salary 20000 is out of range.

Maximum salary is 10000.

Revising salary from 20000 to 9999.

Note:

For the above sample, if there is no outer block (the enclosing block) after the “reraise”, then the exception propagation is ended.

If that is “raise” is in a function/procedure, then the exception propagates to the invoker program exception handler. refer to note5\_reraise.docx