Chapter 17

EXCEPTION HANDLING

- What is an exception?
- How to handle exceptions?
- Predefined exceptions
- When NO_DATA_FOUND exception is not raised?
- User-defined exception
- Reraising an exception
- Associating an exception With An Oracle Error
- Exception propagation
- When is a PL/SQL block successful or failure?

What is an Exception?

In PL/SQL, errors and warnings are called as *exceptions*. Whenever a predefined error occurs in the program, PL/SQL raises an exception. For example, if you try to divide a number by zero then PL/SQL raises an exception called ZERO_DIVIDE and if SELECT can not find a record then PL/SQL raises exception NO_DATA_FOUND.

PL/SQL has a collection of predefined exceptions. Each exception has a name. These exceptions are automatically raised by PL/SQL whenever the corresponding error occurs.

In addition to PL/SQL predefined exceptions, user can also create his own exceptions to deal with errors in the applications. Understanding how to handle exception raised by PL/SQL is as important as understanding how to write code to achieve task. Because exception handling is an important part of any application and application is not complete without exception handling.

How to handle exceptions?

When PL/SQL raises a predefined exception, the program is aborted by displaying error message. But if program is to handle exception raised by PL/SQL then we have to use *Exception Handling* part of the block.

Exception handling part is used to specify the statements to be executed when an exception occurs. Control is transferred to exception handling part whenever an exception occurs. After the exception handler is executed, control is transferred to

next statement in the enclosing block. If there is no enclosing block then control returns to Host (from where you ran the PL/SQL block).

The following is the syntax of exception handling part.

```
WHEN exception-1 [or exception -2] ...   THEN
             statements;

[WHEN exception-3 [or exception-4] ... THEN
             statements; ]  ...

[WHEN OTHERS THEN
             statements; ]
```

exception-1, *exception-2* are exceptions that are to be handled. These exceptions are either pre-defined exceptions or user-defined exceptions.

The following example handles NO_DATA_FOUND exception. If SELECT statement doesn't retrieve any row then PL/SQL raises NO_DATA_FOUND exception, which is handled in exception handling part.

```
declare
       …
begin
        select …

exception
        when no_data_found then
             statements;
end;
```

When two or more exceptions are given with a single WHEN then the statements are executed whenever any of the specified exceptions occur.

The following exception handling part takes the same action when either NO_DATA_FOUND or TOO_MANY_ROWS exceptions occur.

```
declare
       ...
begin
       select ...

exception
        when no_data_found or too_many_rows then
               statements;
end;
```

The following snippet handles these two exceptions in different ways.

```
declare
       ...
begin
       select ...

exception
        when no_data_found then
              statements;
        when too_many_rows then
              statements;
end;
```

WHEN OTHERS is used to execute statements when an exception other than what are mentioned in exception handler has occurred.

Note: *If an exception is raised but not handled by exception handling part then PL/SQL block is terminated by displaying an error message related to the exception.*

Sample Programs

The following is an example of exception handler. This program assigns course fee of "C" to course "C++". If course "C" does not exist then it sets course fee of "C++" to average fee of all courses.

```
declare
   v_fee courses.fee%type;
begin
   select fee into v_fee
   from  courses
   where ccode  = 'c';

   update courses
    set fee = v_fee
   where  ccode='c++';

exception
   when no_data_found then
       update courses
         set fee = ( select avg(fee) from courses)
       where ccode = 'c++';
end;
/
```

If SELECT cannot find a row course code "c" then it raises NO_DATA_FOUND exception. When exception is raised, control is transferred to exception handling part and course fee of "c++" is set to average course fee of all courses. If course code "c" is found then it sets the course fee of course "c++" to the course fee of "c".

Getting information about error - SQLCODE and SQLERRM

In WHEN OTHERS section of exception handler, you can use SQLCODE and SQLERRM functions to get the error number and error message respectively. As there is no predefined exception for each of Oracle errors, you will not get a particular exception for most of the errors. However, it is possible to know the error code and error message of the most recently occurred error using these two functions. This is one way of knowing which Oracle error has exactly occurred. The other method is associating an exception with an Oracle error. Please see "Associating an exception with Oracle error" section for details.

The following example demonstrates how to use SQLCODE and SQLERRM.

```
declare
    newccode varchar2(5) := null;
begin
     update courses set ccode = newccode     where  ccode = 'c';
exception
     when dup_val_on_index then
          dbms_output.put_line('Duplicate course code');
     when others then
          dbms_output.put_line( sqlerrm);
end;
```

If you run the above program, the following output will be generated.

```
ORA-01407: cannot update ("BOOK"."COURSES"."CCODE") to NULL

PL/SQL procedure successfully completed.
```

The above output is generated by WHEN OTHERS part of exception handling part. SQLERRMS returns the error message of the most recent error. As we are trying to set CCODE, which is a not null column to NULL value, PL/SQL raises an exception. But as the error (-01407) is not associated with any predefined exception, WHEN OTHERS part of exception handling part is executed.

Note: *You cannot use `SQLCODE` or `SQLERRM` directly in a SQL statement. Instead, you must assign their values to variables then use the variables in the SQL statement.*

Predefined exceptions

PL/SQL has defined certain common errors and given names to these errors, which are called as *predefined exceptions*. Each exception has a corresponding Oracle error code. The following is the list of predefined exceptions and the corresponding Oracle error code.

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Table 1: Predefined Exceptions

The following is the description of some of the pre-defined exceptions.

CURSOR_ALREADY_OPEN	Raised if you try to open an already open cursor.
DUP_VAL_ON_INDEX	Raised if you try to store duplicate values in a database column that is constrained by a unique index.
INVALID_CURSOR	Raised if you try an illegal cursor operation.
INVALID_NUMBER	Raised in an SQL statement if the conversion of a character string to a number fails because the string does not represent a valid number.
NO_DATA_FOUND	Raised if a SELECT INTO statement returns no rows or if you reference an un-initialized row in a PL/SQL table. See the section "When NO_DATA_FOUND is not raised?".
SUBSCRIPT_BEYOND_COUNT	Raised when the program references a nested table or varray element using an index number larger than the number of elements in the collection.
TOO_MANY_ROWS	Raised if a SELECT INTO statement returns more than one row.
VALUE_ERROR	Raised if an arithmetic, conversion, truncation, or size–constraint error occurs.
ZERO_DIVIDE	Raised when your program attempts to divide a number by zero.

When NO_DATA_FOUND exception is not raised?

As NO_DATA_FOUND exception is most commonly used exception, let us have a close look at this exception. We have so far understood that NO_DATA_FOUND exception is raised by PL/SQL whenever SELECT command doesn't retrieve any rows.

In the following cases NO_DATA_FOUND exception is not raised by PL/SQL even though no row is retrieved or effected:

- When a group function is used in the SELECT statement.
- When UPDATE and DELETE commands are used.

When SELECT command uses any group function then NO_DATA_FOUND exception will be not be raised by PL/SQL although no row is retrieved. For example, if SUM function is used in SELECT no record is retrieved by the SELECT command then SUM function returns NULL value but doesn't raise NO_DATA_FOUND exception. Please see examples given below.

Note: *When COUNT function is used in SELECT and no row is retrieved then COUNT function returns 0 and not NULL value.*

The following example is used to display the average duration of C++ batches. If no C++ batch has been completed then it displays a message. Since AVG function returns NULL when no row is retrieved by SELECT, we check the return value of AVG and display error message if it is NULL.

```
declare
      v_avgdur   number(3);
begin
     -- get average duration of C++ batches
     select avg( enddate - stdate) into v_avgdur
     from  batches
     where enddate is not null and ccode = 'c++';

     /* display error if AVG return null */

     if  v_avgdur is null then
       dbms_output.put_line ('No batch of C++ has been completed');
     else
       dbms_output.put_line ('Average duration of C++ :' || v_avgdur);
     end if;
end;
/
```

We will understand how to detect whether UPDATE or DELETE command has affected any row in the table, in the next chapter.

User-defined exception

PL/SQL allows you to create exceptions of your own. These exceptions are available to the block in which they are created. Unlike a predefined exception, which is predefined and automatically raised whenever the corresponding error occurs, a user-defined error has the following steps.

Declaring userdefined exception

A userdefined exception is to be declared in the declare section of the block. The following is the syntax to declare an exception.

```
     exception-name  exception;
```

exception-name is the name of the exception to be created.

The following example declare an exception classed OUT_OF_STOCK.

```
declare
       out_of_stock   exception;
begin

       statements;
end;
```

Raising userdefined exception using RAISE command

Unlike predefined exceptions, userdefined exception is to be raised explicitly using RAISE command.

```
RAISE exception-name;
```

We have to decide when the user-defined exception has to be raised. For example, if you want to raise OUT_OF_STOCK exception when value of variable QTY is less then 10, give the following:

```
 if  qty < 10 then
     raise out_of_stock;
 end if;
```

Once a userdefined exception is raised using RAISE command, it is to be handled just like a predefined exception. So handling exception OUT_OF_STOCK is no way different from what we have seen so far. The following PL/SQL block will declare, raise and handle a user-defined exception.

```
declare
   out_of_stock   exception;    -- declare  exception
begin

  if condition then
   raise out_of_stock;   -- raise userdefined  exception
end if;
```

```
exception

  when out_of_stock then  -- handle userdefined  exception
  . . .

end;
```

Reraising an exception

RAISE command can also be used to reraise an exception so that the current exception is propagated to outer block. If a sub block executes RAISE statement without giving exception name in exception handler then the current exception is raised again.

The following example will illustrate the process of re-raising an exception.

```
declare
   out_of_stock exception;
begin
   ...
   begin  ---------- sub-block (inner block) begins
      ...
      if ... then
         raise out_of_stock;  -- raise the exception
      end if;
       .
       .
   exception
      when out_of_stock then
        -- handle the error in the sub block
        raise;  -- reraise the current exception, which is out_of_stock
      ...
   end;  ------------ sub-block ends

exception
   when out_of_stock then
      -- handle the exception (that is reraised) in outer block
         ...
end;
```

Note: *RAISE statement without exception name is valid only in exception handler.*

Associating an exception With An Oracle Error

It is possible to connect a userdefined exception with an Oracle error number so that whenever the Oracle error occurs then the user-defined exception will be raised by PL/SQL automatically.

The following example associates exception NULL_VALUE_ERROR with error number –1407, which occurs when a not null column is set to null value, using PRAGAMA EXCEPTION_INIT statement.

```
declare
  null_value_error   exception;
  pragma exception_init(no_privilege, -1407);
```

Now, whenever Oracle error -1407 occurs, NULL_VALUE_ERROR exception is raised by PL/SQL. The following example will illustrate important points related to associating an Oracle error with a user-defined exception.

```
declare
  null_value_error   exception;
  pragma exception_init(null_value_error, -1407);

  newccode varchar2(5) := null;
begin
     update courses
        set ccode = newccode
     where  ccode = 'c';

exception

 when null_value_error then
   dbms_output.put_line('trying to set null value to a not null column');
end;
/
```

Exception propagation

When an exception is raised by PL/SQL and if it not handled in the current block then the exception is propagated. That means, the exception is sent to enclosing blocks one after another from inside to outside until an error handler is found in one of the enclosing blocks or there are no more blocks to search for handlers.

When an exception is not handled in any of the enclosing blocks then it is sent to host environment.

The following figures illustrate how exceptions propagate.

In figure 1, exception A is raised by inner block. As there is an exception handler for exception A, the exception is handled there itself. After the exception is handled, control resumes with statements after inner block in outer block.

As the exception is handled in the block in which exception is raised, the exception is not propagated and control resumes with the enclosing block.

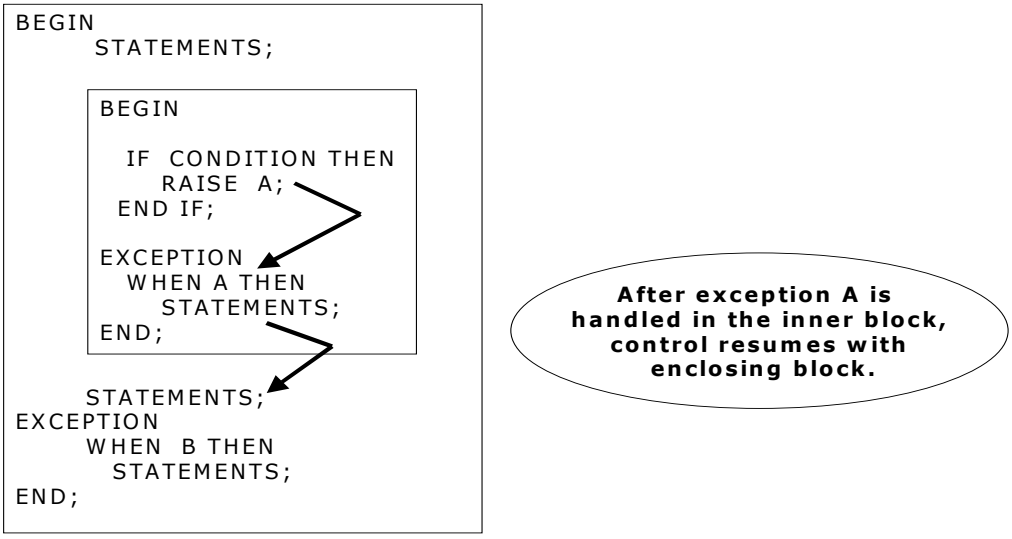

Figure 1: Exception handling.

Exception propagating to outer block

In figure 2, inner block raises exception "A" but as it is not handled in the current block (in inner block) it is propagated to first outer block. As there is an exception handler for "A" in the outer block, control is passed to it and exception is handled in the outer block.

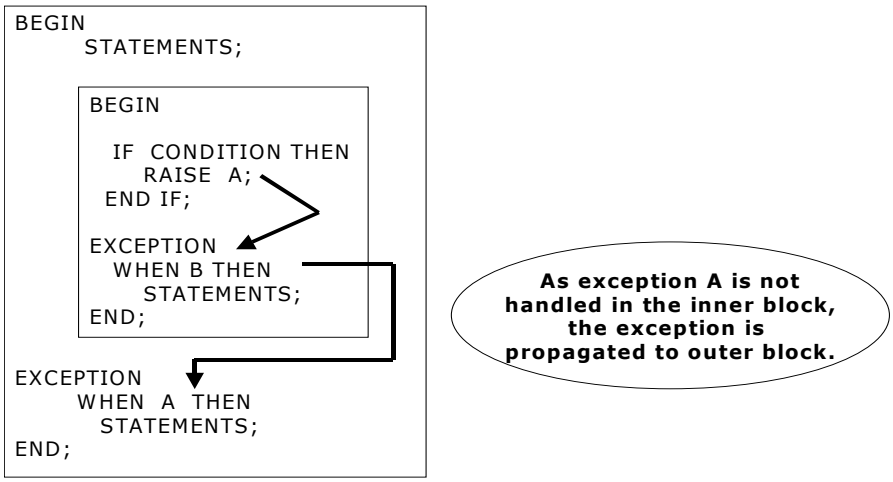

Figure 2: An exception is propagated from inner block to outer block.

Exception propagating to host

In figure 3, exception "A" is neither handled in the block in which it is raised nor handled in any of the outer blocks. As the result exception is propagated to host (the environment from where you ran the outer most block). When an exception is propagated to host the action taken by host depends on host. Examples for host are SQL* PLUS, Oracle forms, and Oracle Server.

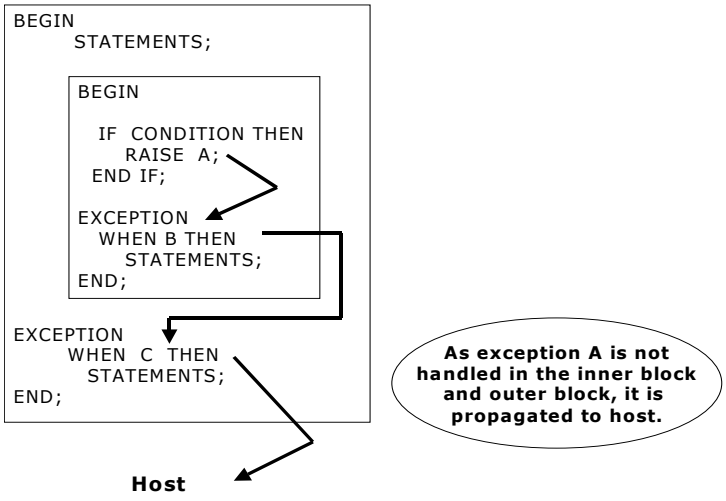

Figure 3: An exception propagating to Host.

In the next section we will see when a PL/SQL is considered to be successful or failure.

Exception raised in Declaration

When an exception is raised in the declaration of a block, the exception is immediately propagated to enclosing block and not handled in that block.

The following example illustrates how an exception that is raised in the declaration of inner block is propagated to outer block.

```
begin
   dbms_output.put_line('in outer block');
   declare
     ccode varchar2(5) := 'abcdef';
   begin
      -- some statements
      dbms_output.put_line('in inner block');
   exception
     when others  then
        dbms_output.put_line(sqlerrm);
  end;
  dbms_output.put_line(' back in outer block');
exception
  when others then
       dbms_output.put_line('Error in outer block: ' || sqlerrm);
end;
```

When you run the above block, the following output will be generated:

```
in outer block
Error in outer block: ORA-06502: PL/SQL: numeric or value error: character
string buffer too small
```

When is a PL/SQL block successful or failure?

Each PL/SQL block exits either successfully or unsuccessfully. The exit status of PL/SQL block plays a very important role at a later stage (for example in database triggers). But for now let us just understand when a PL/SQL block is considered to be successful and when is it considered to be a failure.

A PL/SQL block is successful if it:

- Exits without raising any exceptions.
- Raises an exception but the exception is handled in the block's exception handling part.

A PL/SQL block is unsuccessful if it:

- Exits with an unhandled exception. That means the executable part raises an exception (either predefined or user-defined) and it is not handled in the block's exception handler.
- Executes RAISE_APPLICATION_ERROR procedure to generate an user-defined error.

Detecting where exception is raised

It is sometimes important to know which statement has actually raised the exception. Especially when two or more SELECT statements are there in the block and if one of them has raised NO_DATA_FOUND exception, it is difficult to know which statement has actually caused the problem.

The following example will use a variable to know which SELECT statement has actually raised the exception.

```
declare
    n number(1) :=0;
begin
    select ...
    n := 1;
    select ...
    n:= 2;
    select ...

exception
    when no_data_found then
         if  n = 0 then
               ...
         elsif  n = 1 then
               ...
         else
               ...
         end if
end;
```

In the above example, variable N is set to 0 at the time of declaration. If first SELECT statement raised NO_DATA_FOUND exception then control is transferred to exception handler and the value of N will be 0. If first SELECT succeeds and second SELECT has failed then the value of N will be 1 and similarly the value of N will be 2 if second SELECT also succeeds but third SELECT fails.

In the exception handler, it is possible to know which SELECT has failed by using the value of variable N.

Summary

Errors and warnings in PL/SQL are called as exceptions. PL/SQL exceptions may be either predefined or user-defined. Predefined exceptions are those that represent a general failure. User can also define exception, in addition to predefined, and use them identical to predefined exceptions. But user-defined exceptions are to be explicitly declared and raised.

Oracle allows errors to be associated with user-defined exceptions using PRAGMA EXCEPTION_INIT statement. When an exception is raised first PL/SQL tries to handle the exception in the current block. If current block doesn't have an exception handler for the exception then exception is propagated to outer block. This propagation will go either until the exception handler is found in one of the enclosing block or until Host is reached.

Exercises

1. Look for student number 1008. If it is not found then write a suitable error message on the screen otherwise display the total amount paid by student so far.
2. _____ statement is used to re-raise an exception.
3. _____ function is used to get error message of the most recent error.
4. How do you associate an Oracle error with a user-defined error.
5. When UPDATE command could not update any rows then which of the following will happen?
    a. NO_DATA_FOUND exception occurs
    b. INVALID_UPDATE exception occurs
    c. No exception is raised
6. When an exception is not handled in the current block
    a. It results in error and terminates the block
    b. It is propagated to outer block
    c. It is ignored.