**Handling Errors in PL/SQL**

In **PL/SQL**, an error condition is called an exception. **Exceptions** can be internally defined (by the runtime system) or user-defined. Examples of internally defined exceptions include division by zero and out of memory.  
When an error occurs, an exception is raised. That is normal execution stops and control transfers to the exception-handling part of your PL/SQL block or subprogram. Internal exceptions are raised implicitly (automatically) by the run-time system. User-defined exceptions must be raised explicitly by **RAISE statements**, which can also raise predefined exceptions.  
To handle raised exceptions, you write separate routines called exception handlers. After an exception handler runs, the current block stops executing and the enclosing block resumes with the next statement. If there is no enclosing block, control returns to the host environment.  
**Example: Runtime Error Handling**

DECLARE

stock\_price NUMBER := 9.73;

net\_earnings NUMBER := 0;

pe\_ratio NUMBER;

BEGIN

-- Calculation might cause division-by-zero error.

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

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

EXCEPTION -- exception handlers begin

-- Only one of the WHEN blocks is executed.

WHEN ZERO\_DIVIDE THEN -- handles 'division by zero' error

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

pe\_ratio := NULL;

WHEN OTHERS THEN -- handles all other errors

DBMS\_OUTPUT.PUT\_LINE('Some other kind of error occurred.');

pe\_ratio := NULL;

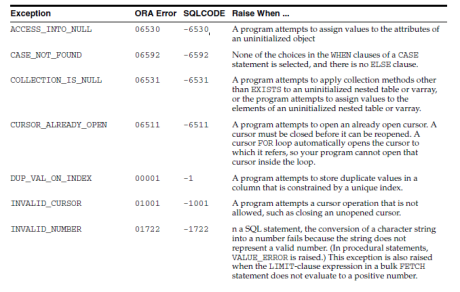
END; -- exception handlers and block end here

/

**Guidelines for Avoiding and Handling PL/SQL Errors and Exceptions**

* Add exception handlers whenever there is any possibility of an error occurring.
* Add error-checking code whenever you can predict that an error might occur if your code gets bad input data.
* Make your programs robust enough to work even if the database is not in the state you expect.
* Handle named exceptions whenever possible, instead of using WHEN OTHERS in exception handlers.
* 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.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/Exceptions.png)***Go for this in-depth job-oriented***[***PL/SQL Training***](https://intellipaat.com/pl-sql-online-training/)***Course now!***

**Defining Your Own PL/SQL Exceptions**  
PL/SQL lets you define exceptions of your own. Unlike predefined exceptions, user-defined exceptions must be declared and must be raised explicitly by RAISE statements.

**Declaring PL/SQL 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. In the following example, you declare an exception named past\_due:

DECLARE

past\_due EXCEPTION;

**Scope Rules for PL/SQL Exceptions**  
Exceptions declared in a block are considered local to that block and global to all its sub-blocks. Because a block can reference only local or global exceptions, enclosing blocks cannot reference exceptions declared in a sub-block.  
If you redeclare a global exception in a sub-block, the local declaration prevails. The sub-block cannot reference the global exception unless the exception is declared in a labeled block and you qualify its name with the block label:

block\_label.exception\_name

**Defining Your Own Error Messages: Procedure RAISE\_APPLICATION\_ERROR**  
The procedure RAISE\_APPLICATION\_ERROR lets you issue user-defined ORA- error messages from stored subprograms. That way, you can report errors to your application and avoid returning unhandled exceptions.  
To call RAISE\_APPLICATION\_ERROR, use the syntax-

raise\_application\_error(

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

***Come to Intellipaat’s***[***PL/SQL Community***](https://intellipaat.com/community/tag/plsql)***if you have more queries on PL/SQL!***

**Raising Exceptions with the RAISE Statement**  
PL/SQL blocks and subprograms should raise an exception only when an error makes it undesirable or impossible to finish processing. You can place RAISE statements for a given exception anywhere within the scope of that exception.  
**Example: Using RAISE to Force a User-Defined Exception**

DECLARE

out\_of\_stock EXCEPTION;

number\_on\_hand NUMBER := 0;

BEGIN

IF number\_on\_hand < 1 THEN

RAISE out\_of\_stock; -- raise an exception that we defined

END IF;

EXCEPTION

WHEN out\_of\_stock THEN

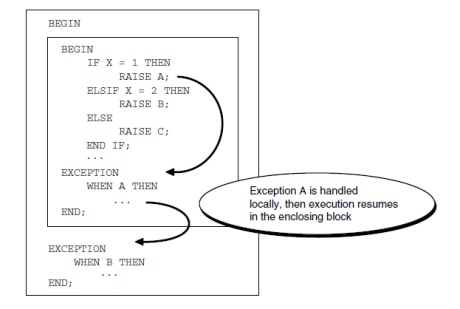
-- handle the error

DBMS\_OUTPUT.PUT\_LINE('Encountered out-of-stock error.');

END;

/

**How PL/SQL 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 enclosing blocks until a handler is found or there are no more blocks to search. If no handler is found, PL/SQL returns an unhandled exception error to the host environment.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/propagation-rules.png)

**Handling Raised PL/SQL 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 exception1 THEN -- handler for exception1

sequence\_of\_statements1

WHEN exception2 THEN -- another handler for exception2

sequence\_of\_statements2

...

WHEN OTHERS THEN -- optional handler for all other errors

sequence\_of\_statements3

END;

**Overview of PL/SQL Compile-Time Warnings**  
To make your programs more robust and avoid problems at run time, you can turn on checking for certain warning conditions. These conditions are not serious enough to produce an error and keep you from compiling a subprogram. They might point out something in the subprogram that produces an undefined result or might create a performance problem.  
To work with PL/SQL warning messages, you use the PLSQL\_WARNINGS initialization parameter, the DBMS\_WARNING package, and the USER/DBA/ALL\_PLSQL\_OBJECT\_SETTINGS views.

***Check out the top***[***PL/SQL Interview Questions***](https://intellipaat.com/blog/interview-question/pl-sql-interview-questions/)***to learn what is expected from PL/SQL professionals!***  
  
**PL/SQL Warning Categories**  
PL/SQL warning messages are divided into categories so that you can suppress or display groups of similar warnings during compilation. The categories are:

* **SEVERE**: Messages for conditions that might cause unexpected behavior or wrong results, such as aliasing problems with parameters.
* **PERFORMANCE**: Messages for conditions that might cause performance problems, such as passing a VARCHAR2 value to a NUMBER column in an [INSERT](https://intellipaat.com/blog/tutorial/sql-tutorial/insert-statement/) statement.
* **INFORMATIONAL:** Messages for conditions that do not have an effect on performance or correctness, but that you might want to change to make the code more maintainable, such as unreachable code that can never be executed.