EXCEPTIONS

Bordoloi and

Errors

- Two types of errors can be found in a program: compilation errors and runtime errors.
- There is a special section in a PL/SQL block that handles the runtime errors.
- This section is called the *exception-handling section*, and in it, runtime errors are referred to as *exceptions*.
- The exception-handling section allows programmers to specify what actions should be taken when a specific exception occurs.

Exception Handling

- In order to handle run time errors in the program, an exception handler must be added.
- The exception-handling section has the following structure:

```
EXCEPTION
    WHEN EXCEPTION_NAME
    THEN
        ERROR-PROCESSING STATEMENTS;
```

- The exception-handling section is placed after the executable section of the block.

- The section of the example in bold letters shows the exception-handling section of the block.
- When this example is executed with values of 4 and 0 for variables v_num1 and v_num2, respectively, the following output is produced:

Enter value for sv_num1: 4
old 2: v_num1 integer := &sv_num1;
new 2: v_num1 integer := 4;
Enter value for sv_num2: 0
old 3: v_num2 integer := &sv_num2;
new 3: v_num2 integer := 0;
A number cannot be divided by zero.
PL/SQL procedure successfully completed.

Example

```
DECLARE
  v_num1 integer := &sv_num1;
  v_num2 integer := &sv_num2;
  v_result number;
BEGIN
  v_result := v_num1 / v_num2;
  DBMS_OUTPUT.PUT_LINE
    ('v_result: '| |v_result);
EXCEPTION
  WHEN ZERO_DIVIDE
  THEN
    DBMS_OUTPUT.PUT_LINE
    ('A number cannot be divided by
    zero.');
END;
```

Exception Handling

- This output shows that once an attempt to divide v_num1 by v_num2 was made, the exception-handling section of the block was executed.
- Therefore, the error message specified by the exception-handling section was displayed on the screen.
- This example illustrates several advantages of using an exception-handling section.
- You have probably noticed that the output looks cleaner. Even though the error message is still displayed on the screen, the output is more informative.
- In short, it is oriented more toward a user than a programmer.

Exception Handling

- In addition, an exception-handling section allows a program to execute to completion, instead of terminating prematurely.
- Another advantage offered by the exception-handling section is isolation of error-handling routines. In other words, all error-processing code for a specific block is located in the single section. As a result, the logic of the program becomes easier to follow and understand.
- Finally, adding an exception-handling section enables event-driven processing of errors.
- In case of a specific exception event, the exception-handling section is executed.

Exception Handling

- Just like in the example shown earlier, in case of the division by 0, the exception-handling section was executed.
- In other words, the error message specified by the DBMS_OUTPUT.PUT_LINE statement was displayed on the screen.

BUILT-IN EXCEPTIONS

- When a built-in exception occurs, it is said to be raised implicitly.
- In other words, if a program breaks an Oracle rule, the control is passed to the exception-handling section of the block.
- At this point, the error processing statements are executed.
- It is important for you to realize that after the exception-handling section of the block has executed, the block terminates.
- Control will not return to the executable section of this block.

Example

```
DECLARE
  v_student_name VARCHAR2(50);
BEGIN
 SELECT first_name||‘ ’||last_name
 INTO v_student_name
 FROM student
 WHERE student_id = 101;
   DBMS_OUTPUT.PUT_LINE
   (‘Student name is’||v_student_name);
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
     DBMS_OUTPUT.PUT_LINE
     (‘There is no such student’);
END;
```

- This example produces the following output:

  There is no such student

  PL/SQL procedure successfully completed.
- Because there is no record in the STUDENT table with student ID 101, the SELECT INTO statement does not return any rows.
- As a result, control passes to the exception-handling section of the block, and the error message “There is no such student” is displayed on the screen.
- Even though there is a DBMS_OUTPUT.PUT_LINE statement right after the SELECT statement, it will not be executed because control has been transferred to the exception-handling section.

BUILT-IN EXCEPTIONS

- Control will never return to the executable section of this block, which contains the DBMS_OUTPUT.PUT_LINE statement.
- While every Oracle runtime error has a number associated with it, it must be handled by its name in the exception-handling section.
- One of the outputs from the previous example has the following error message:

  ORA-01476: divisor is equal to zero

  where ORA-01476 stands for error number.
- This error number refers to the error named ZERO_DIVIDE.
- So, some common Oracle runtime errors are predefined in the PL/SQL as exceptions.

BUILT-IN EXCEPTIONS

- The list shown below explains some commonly used predefined exceptions and how they are raised:
- **NO_DATA_FOUND** This exception is raised when a SELECT INTO statement, which makes no calls to group functions, such as SUM or COUNT, does not return any rows.
- For example, you issue a SELECT INTO statement against STUDENT table where student ID equals 101.
- If there is no record in the STUDENT table passing this criteria (student ID equals 101), the NO_DATA_FOUND exception is raised.

TOO_MANY_ROWS

- This exception is raised when a SELECT INTO statement returns more than one row.
- By definition, a SELECT INTO can return only single row.
- If a SELECT INTO statement returns more than one row, the definition of the SELECT INTO statement is violated.
- This causes the TOO_MANY_ROWS exception to be raised.
- For example, you issue a SELECT INTO statement against the STUDENT table for a specific zip code.
- There is a big chance that this SELECT statement will return more than one row because many students can live in the same zip code area.

ZERO_DIVIDE

- This exception is raised when a division operation is performed in the program and a divisor is equal to zero.
- Previous example in the illustrates how this exception is raised.

LOGIN_DENIED

- This exception is raised when a user is trying to login on to Oracle with invalid username or password.

PROGRAM_ERROR

- This exception is raised when a PL/SQL program has an internal problem.

VALUE_ERROR

- This exception is raised when conversion or size mismatch error occurs.
- For example, you select student's last name into a variable that has been defined as VARCHAR2(5).
- If student's last name contains more than five characters, VALUE_ERROR exception is raised.

DUP_VALUE_ON_INDEX

- This exception is raised when a program tries to store a duplicate value in the column or columns that have a unique index defined on them.
- For example, you are trying to insert a record into the SECTION table for the course number “25,” section 1.
- If a record for the given course and section numbers already exists in the SECTION table, DUP_VAL_ON_INDEX exception is raised because these columns have a unique index defined on them.

HANDLING DIFFERENT EXCEPTIONS

- So far, you have seen examples of the programs able to handle a single exception only.
- For example, a PL/SQL contains an exception-handler with a single exception ZERO_DIVIDE.
- However, many times in the PL/SQL block you need to handle different exceptions.
- Moreover, often you need to specify different actions that must be taken when a particular exception is raised.

```
DECLARE
  v_student_id NUMBER := &sv_student_id;
  v_enrolled VARCHAR2(3) := 'NO';
BEGIN
   DBMS_OUTPUT.PUT_LINE
   ('Check if the student is enrolled');
SELECT 'YES'
INTO v_enrolled
FROM enrollment
 WHERE student_id = v_student_id;
    DBMS_OUTPUT.PUT_LINE
    ('The student is enrolled into one course');
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
    DBMS_OUTPUT.PUT_LINE('The student
      is not enrolled');
WHEN TOO_MANY_ROWS
  THEN
    DBMS_OUTPUT.PUT_LINE
   ('The student is enrolled into many courses');
END;
```

- This example contains two exceptions in the single exception handling section.
- The first exception, NO_DATA_FOUND, will be raised if there are no records in the ENROLLMENT table for a particular student.
- The second exception, TOO_MANY_ROWS, will be raised if a particular student is enrolled into more than one course.

OTHERS Handler

- You have seen examples of exception-handling sections that have particular exceptions, such as NO_DATA_FOUND or ZERO_DIVIDE.
- However, you cannot always predict beforehand what exception might be raised by your PL/SQL block.
- In cases like this, there is a special exception handler called OTHERS.
- All predefined Oracle errors (exceptions) can be handled with the help of the OTHERS handler.

Example

```
DECLARE
 v_instructor_id NUMBER :=
   &sv_instructor_id;
 v_instructor_name VARCHAR2(50);
BEGIN
 SELECT first_name||' '||last_name
 INTO v_instructor_name
 FROM instructor
 WHERE instructor_id =
    v_instructor_id;
  DBMS_OUTPUT.PUT_LINE
  ('Instructor name is'
    ||v_instructor_name);
EXCEPTION
 WHEN OTHERS
 THEN
   DBMS_OUTPUT.PUT_LINE('An
   error has occurred');
END;
```

- When run, this example produces the following output:

```
Enter value for sv_instructor_id: 100
old 2: v_instructor_id NUMBER :=
     &sv_instructor_id;
new 2: v_instructor_id NUMBER := 100;
An error has occurred
PL/SQL procedure successfully
     completed.
```

- This demonstrates not only the use of the OTHERS exception handler, but also a bad programming practice.
- The exception OTHERS has been raised because there is no record in the INSTRUCTOR table for instructor ID 100.

EXCEPTION SCOPE

- The scope of an exception is the portion of the block that is covered by this exception.
- Even though variables and exceptions serve different purposes, the same scope rules apply to them.

Example

```
DECLARE
      v_student_id NUMBER := &sv_student_id;
      v_name VARCHAR2(30);
      v_total NUMBER(1);
-- outer block
BEGIN
      SELECT RTRIM(first_name)||' '||RTRIM(last_name)
      INTO v_name
      FROM student
      WHERE student_id = v_student_id;
      DBMS_OUTPUT.PUT_LINE('Student name is '||v_name);
            -- inner block
            BEGIN
                  SELECT COUNT(*)
                  INTO v_total
```

Example

```
            FROM enrollment
            WHERE student_id = v_student_id;
            DBMS_OUTPUT.PUT_LINE
            ('Student is registered for '||v_total||' course(s)');
        EXCEPTION
            WHEN VALUE_ERROR OR INVALID_NUMBER
            THEN
            DBMS_OUTPUT.PUT_LINE('An error has
occurred');
        END;
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
    DBMS_OUTPUT.PUT_LINE('There is no such student');
END;
```

Bordoloi and

Example explained

- The inner block has structure similar to the outer block.
- It has a SELECT INTO statement and an exception section to handle errors.
- When VALUE_ERROR or INVALID_NUMBER error occurs in the inner block, the exception is raised.
- It is important that you realize that exceptions VALUE_ERROR and INVALID_NUMBER have been defined for the inner block only.
- Therefore, they can be raised in the inner block only.
- If one of these errors occurs in the outer block, this program will be unable to terminate successfully.

Example explained

- The exception NO_DATA_FOUND has been defined in the outer block; therefore, it is global to the inner block.
- This example will never raise the exception NO_DATA_FOUND in the inner block as it contains a group function in the SELECT statement.
- It is important to note that if you define an exception in a block, it is local to that block.
- However, it is global to any blocks enclosed by that block.
- In other words, in the case of nested blocks, any exception defined in the outer block becomes global to its inner blocks.

User Defined Exceptions

- Often in your programs you may need to handle problems that are specific to the program you write.
- For example, your program asks a user to enter a value for student_id. This value is then assigned to the variable v_student_id that is used later in the program.
- Generally, you want a positive number for an id. By mistake, the user enters a negative number.
- However, no error has occurred because student_id has been defined as a number, and the user has supplied a legitimate numeric value.
- Therefore, you may want to implement your own exception to handle this situation.

User Defined Exceptions

- This type of an exception is called a *user-defined exception* because it is defined by the programmer.
- Before the exception can be used, it must be declared.
- A user-defined exception is declared in the declarative part of a PL/SQL block as shown below:

```
DECLARE
exception_name EXCEPTION;
```

- Once an exception has been declared, the executable statements associated with this exception are specified in the exception-handling section of the block.
- The format of the exception-handling section is the same as for built-in exceptions.

Example

```
DECLARE
    e_invalid_id EXCEPTION;
BEGIN
    …
EXCEPTION
    WHEN e_invalid_id
    THEN
        DBMS_OUTPUT.PUT_LINE ('An id cannot be
negative');
END;
```

Raising Exception

A user-defined exception must be raised explicitly.

In other words, you need to specify in your program under which circumstances an exception must be raised as shown :

```
DECLARE
  exception_name EXCEPTION;
BEGIN
  …
  IF CONDITION
  THEN
     RAISE exception_name;
  ELSE
     …
  END IF;
EXCEPTION
   WHEN exception_name
   THEN
        ERROR-PROCESSING
        STATEMENTS;
END;
```

EXCEPTION PROPAGATION

- A runtime error may occur in the executable section, declaration section of the block or in the exception-handling section of the block.
- The rules that govern how exceptions are raised in these situations are referred to as *exception propagation*.

EXCEPTION PROPAGATION

- When a runtime error occurs in the executable section of the PL/SQL block, If there is an exception specified associated with a particular error, the control is passed to the exception-handling section of the block.
- Once the statements associated with the exception are executed, the control is passed to the host environment or to the enclosing block.
- If there is no exception handler for this error, the exception is propagated to the enclosing block (outer block).
- Then, the steps described above are repeated again.
- If no exception handler is found, the execution of the program halts, and the control is transferred to the host environment.

EXCEPTION PROPAGATION

- When a runtime error occurs in the declaration section of the block and if there is no outer block, the execution of the program halts, and the control is passed to the host environment.
- When a runtime error occurs in the declaration section of the PL/SQL block, the exception-handling section of this block will not be able to catch the error.
- When a runtime error occurs in the declaration section of the inner block, the exception immediately propagates to the enclosing (outer) block.

EXCEPTION PROPAGATION

- When a run time error occurs in the exception-handling section, just like in the previous case, if there is no outer block, the execution of the program halts, and the control is passed to the host environment.
- When a runtime error occurs in the exception-handling section of the PL/SQL block, the exception-handling section of this block is not able to prevent the error.
- When a runtime error occurs in the exception-handling section of the inner block, the exception immediately propagates to the enclosing block.

EXCEPTION PROPAGATION

- Only one exception can be raised in the exception-handling section of the block.
- Only after one exception has been handled, another can be raised, but two or more exceptions cannot be raised simultaneously.

Example

```
--outer block
DECLARE
    e_exception1 EXCEPTION;
    e_exception2 EXCEPTION;
BEGIN
    -- inner block
    BEGIN
        RAISE e_exception1;
    EXCEPTION
    WHEN e_exception1
    THEN
        RAISE e_exception2;
```

Example contd.

```
    WHEN e_exception2
    THEN
        DBMS_OUTPUT.PUT_LINE ('An error has occurred in the
inner'|| 'block');
    END;
EXCEPTION
WHEN e_exception2
THEN
    DBMS_OUTPUT.PUT_LINE ('An error has occurred in the
program');
END;
```

Output

An error has occurred in the program

PL/SQL procedure successfully completed.

- Here two exceptions are declared: e_exception1 and e_exception2.
- The exception e_exception1 is raised in the inner block via statement RAISE.
- In the exception-handling section of the block, the exception e_exception1 tries to raise e_exception2.
- Even though there is an exception handler for the exception e_exception2, the control is transferred to the outer block.
- This happens because only one exception can be raised in the exception-handling section of the block.

RERAISING AN EXCEPTION

- On some occasions you may want to be able to stop your program if a certain type of error occurs.
- In other words, you may want to handle an exception in the inner block and then pass it to the outer block.
- This process is called *reraising an exception*. The following example illustrates this point.

```
-- outer block
DECLARE
    e_exception EXCEPTION;
BEGIN
    -- inner block
    BEGIN
        RAISE e_exception;
    EXCEPTION
    WHEN e_exception
    THEN
        RAISE;
    END;
EXCEPTION
WHEN e_exception
THEN
DBMS_OUTPUT.PUT_LINE   ('An error has occurred');
END;
```

Bordoloi and Bock

Output

The error has occurred
PL/SQL procedure successfully completed.

The exception, e_exception, is declared in the outer block.

It is raised in the inner block.

As a result, the control is transferred to the exception handling section of the inner block.

The statement RAISE in the exception-handling section of the block causes the exception to propagate to the exception-handling section of the outer block.

RERAISING AN EXCEPTION

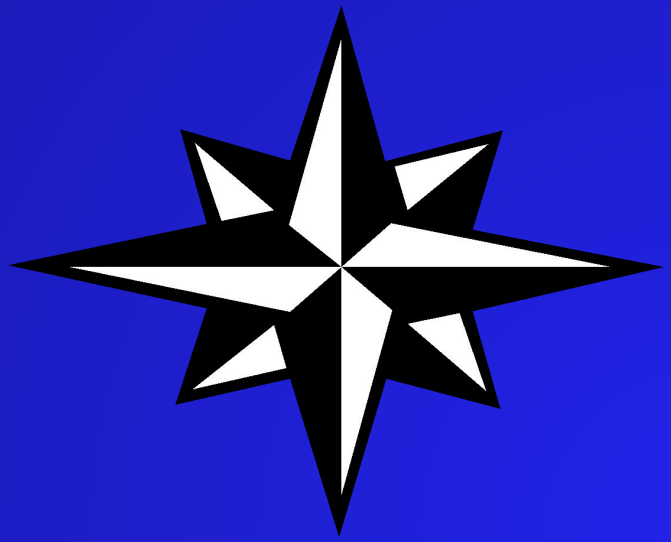
- It is important to note that when an exception is reraised in the block that is not enclosed by any other block, the program is unable to complete successfully

```
DECLARE
    e_exception EXCEPTION;
BEGIN
    RAISE e_exception;
EXCEPTION
    WHEN e_exception
    THEN
    RAISE;
END;
```

Bordoloi and

Output

```
DECLARE
*
ERROR at line 1:
ORA-06510: PL/SQL: unhandled user-defined exception
ORA-06512: at line 8
```



END

Bordoloi and
Bock