8

Handling Exceptions

Objectives

After completing this lesson, you should be able to do the following:

- Define PL/SQL exceptions
- Recognize unhandled exceptions
- List and use different types of PL/SQL exception handlers
- Trap unanticipated errors
- Describe the effect of exception propagation in nested blocks
- Customize PL/SQL exception messages

Handling Exceptions with PL/SQL

- An exception is an identifier in PL/SQL that is raised during execution.
- How is it raised?
  - An Oracle error occurs.
  - You raise it explicitly.
- How do you handle it?
  - Trap it with a handler.
  - Propagate it to the calling environment.

Handling Exceptions

Trap the exception

Exception is raised

Exception is trapped

```
DECLARE

BEGIN

EXCEPTION

END;
```

Propagate the exception

```
DECLARE

BEGIN

EXCEPTION

END;
```

Exception is raised

Exception is not trapped

Exception propagates to calling environment

Exception Types

- Predefined Oracle Server
- Nonpredefined Oracle Server

} Implicitly raised

- User-defined — Explicitly raised

Trapping Exceptions

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

Trapping Exceptions Guidelines

- The `EXCEPTION` keyword starts exception-handling section.
- Several exception handlers are allowed.
- Only one handler is processed before leaving the block.
- `WHEN OTHERS` is the last clause.

Trapping Predefined Oracle Server Errors

- Reference the standard name in the exception-handling routine.
- Sample predefined exceptions:
  - `NO_DATA_FOUND`
  - `TOO_MANY_ROWS`
  - `INVALID_CURSOR`
  - `ZERO_DIVIDE`
  - `DUP_VAL_ON_INDEX`

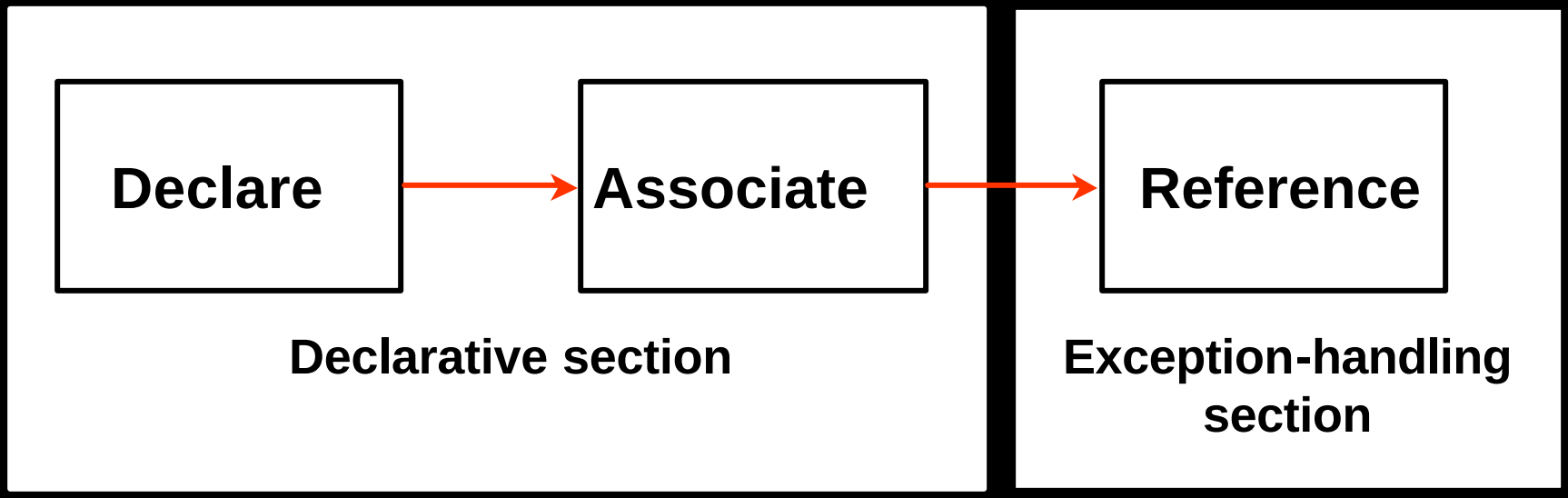
Predefined Exceptions

Syntax:

```
BEGIN
. . .
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;

  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

Trapping Nonpredefined Oracle Server Errors



Name the exception

Code the `PRAGMA EXCEPTION_INIT`

Handle the raised exception

Nonpredefined Error

Trap for Oracle server error number –2292, an integrity constraint violation.

```
DEFINE p_deptno = 10
DECLARE
  e_emps_remaining EXCEPTION;
  PRAGMA EXCEPTION_INIT
    (e_emps_remaining, -2292);
BEGIN
  DELETE FROM departments
  WHERE  department_id = &p_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining  THEN
   DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
   TO_CHAR(&p_deptno) || '.  Employees exist. ');
END;
```

1

2

3

Functions for Trapping Exceptions

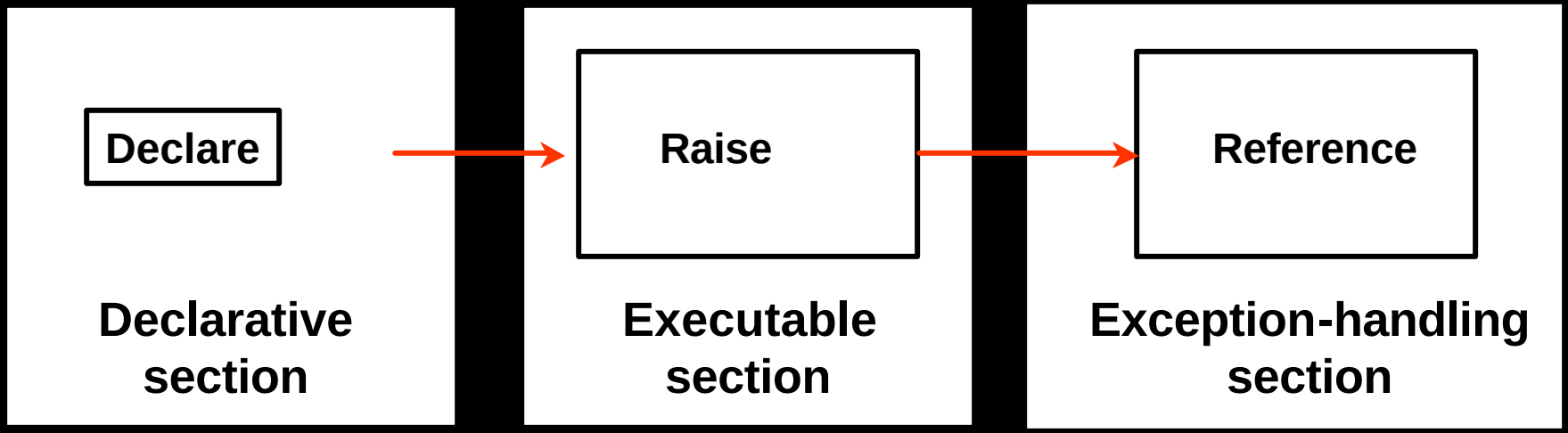
- `SQLCODE`: Returns the numeric value for the error code
- `SQLERRM`: Returns the message associated with the error number

Functions for Trapping Exceptions

Example:

```
DECLARE
  v_error_code      NUMBER;
  v_error_message   VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    v_error_code := SQLCODE ;
    v_error_message := SQLERRM ;
    INSERT INTO errors
    VALUES(v_error_code, v_error_message);
END;
```

Trapping User-Defined Exceptions



Name the exception.

Explicitly raise the exception by using the `RAISE` statement.

Handle the raised exception.

User-Defined Exceptions

Example:

```
DEFINE p_department_desc = 'Information Technology '
DEFINE P_department_number = 300
```

```
DECLARE
  e_invalid_department EXCEPTION;
BEGIN
  UPDATE      departments
  SET         department_name = '&p_department_desc'
  WHERE       department_id = &p_department_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```

1

2

3

Calling Environments

*i*SQL*Plus	Displays error number and message to screen
Procedure Builder	Displays error number and message to screen
Oracle Developer Forms	Accesses error number and message in a trigger by means of the `ERROR_CODE` and `ERROR_TEXT` packaged functions
Precompiler application	Accesses exception number through the `SQLCA` data structure
An enclosing PL/SQL block	Traps exception in exception-handling routine of enclosing block

Propagating Exceptions

Subblocks can handle an exception or pass the exception to the enclosing block.

```
DECLARE
  . . .
  e_no_rows       exception;
  e_integrity     exception;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
     SELECT ...
     UPDATE ...
     IF SQL%NOTFOUND THEN
       RAISE e_no_rows;
     END IF;
    END;
  END LOOP;
EXCEPTION
  WHEN e_integrity THEN ...
  WHEN e_no_rows THEN ...
END;
```

The RAISE_APPLICATION_ERROR Procedure

Syntax:

```
raise_application_error (error_number,
                         message[, {TRUE | FALSE}]);
```

- You can use this procedure to issue user-defined error messages from stored subprograms.
- You can report errors to your application and avoid returning unhandled exceptions.

The `RAISE_APPLICATION_ERROR` Procedure

- Used in two different places:
  - Executable section
  - Exception section
- Returns error conditions to the user in a manner consistent with other Oracle server errors

RAISE_APPLICATION_ERROR

Executable section:

```
BEGIN
...
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
       'This is not a valid manager');
  END IF;
   ...
```

Exception section:

```
...
EXCEPTION
     WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR (-20201,
         'Manager is not a valid employee.');
END;
```

Summary

In this lesson, you should have learned that:

- Exception types:
  - Predefined Oracle server error
  - Nonpredefined Oracle server error
  - User-defined error
- Exception trapping
- Exception handling:
  - Trap the exception within the PL/SQL block.
  - Propagate the exception.

ORACLE

Practice 8

1. Write a PL/SQL block to select the name of the employee with a given salary value.
   a. Use the `DEFINE` command to provide the salary.
   b. Pass the value to the PL/SQL block through a *i*SQL*Plus substitution variable. If the salary entered returns more than one row, handle the exception with an appropriate exception handler and insert into the `MESSAGES` table the message "More than one employee with a salary of *<salary>*."
   c. If the salary entered does not return any rows, handle the exception with an appropriate exception handler and insert into the `MESSAGES` table the message "No employee with a salary of *<salary>*."
   d. If the salary entered returns only one row, insert into the `MESSAGES` table the employee's name and the salary amount.
   e. Handle any other exception with an appropriate exception handler and insert into the `MESSAGES` table the message "Some other error occurred."
   f. Test the block for a variety of test cases. Display the rows from the `MESSAGES` table to check whether the PL/SQL block has executed successfully. Some sample output is shown below.

RESULTS
More than one employee with a salary of 6000
No employee with a salary of 5000
More than one employee with a salary of 7000
No employee with a salary of 2000

2. Modify the code in `p3q3.sql` to add an exception handler.
   a. Use the `DEFINE` command to provide the department ID and department location. Pass the values to the PL/SQL block through a *i*SQL*Plus substitution variables.
   b. Write an exception handler for the error to pass a message to the user that the specified department does not exist. Use a bind variable to pass the message to the user.
   c. Execute the PL/SQL block by entering a department that does not exist.

G_MESSAGE
Department 200 is an invalid department

Practice 8 (continued)

3. Write a PL/SQL block that prints the number of employees who earn plus or minus $100 of the salary value set for an *i*SQL*Plus substitution variable. Use the `DEFINE` command to provide the salary value. Pass the value to the PL/SQL block through a *i*SQL*Plus substitution variable.

   a. If there is no employee within that salary range, print a message to the user indicating that is the case. Use an exception for this case.

   b. If there are one or more employees within that range, the message should indicate how many employees are in that salary range.

   c. Handle any other exception with an appropriate exception handler. The message should indicate that some other error occurred.

```
DEFINE p_sal = 7000
DEFINE p_sal = 2500
DEFINE p_sal = 6500
```

G_MESSAGE
There is/are 4 employee(s) with a salary between 6900 and 7100

G_MESSAGE
There is/are 12 employee(s) with a salary between 2400 and 2600

G_MESSAGE
There is/are 3 employee(s) with a salary between 6400 and 6600