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

What Is an Exception?

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is :' ||v_lname);
END;
```

Script Output

Task completed in 0.012 seconds

```
Error starting at line : 1 in command -
DECLARE
  v_lname VARCHAR(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name = 'John';
  DBMS_OUTPUT.PUT_LINE( ' John''s last name is ' ||v_lname);
END;
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
```

Handling an Exception: Example

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is :' ||v_lname);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
  DBMS_OUTPUT.PUT_LINE (' Your select statement retrieved
   multiple rows. Consider using a cursor.');
END;
/
```

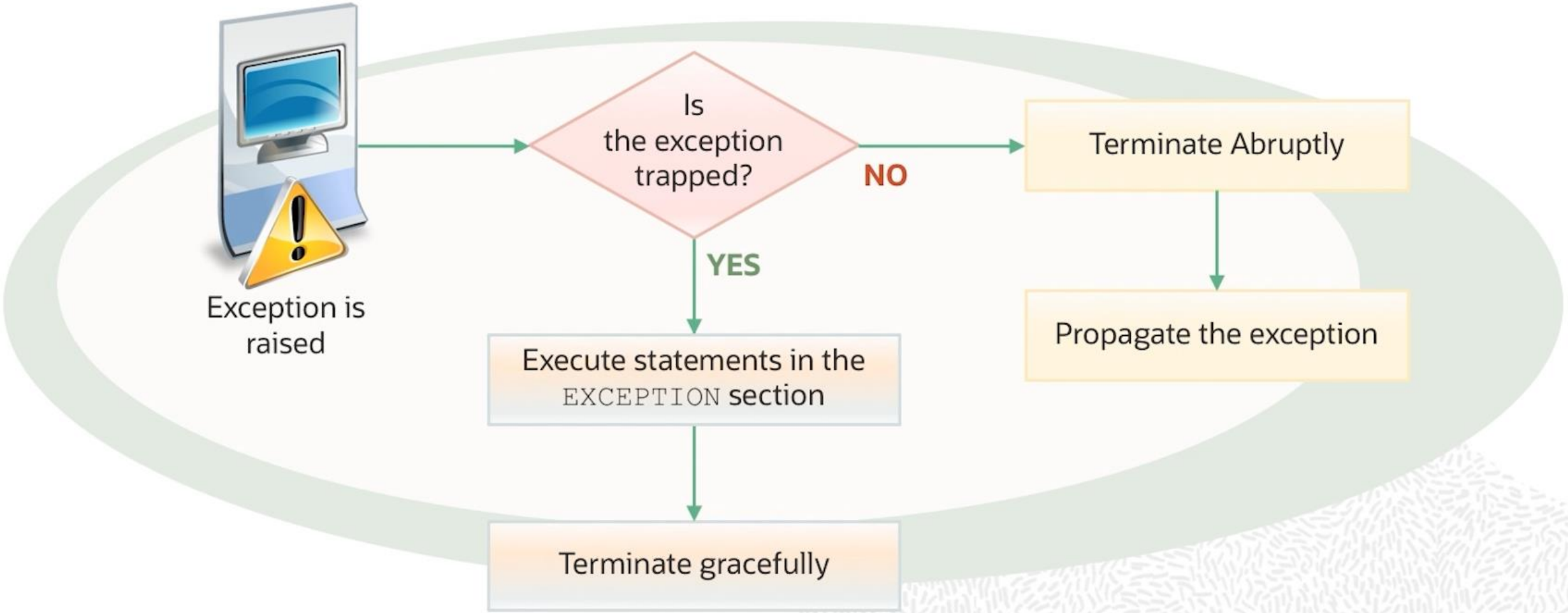
```
PL/SQL procedure successfully completed.

Your select statement retrieved multiple rows.
  Consider using a cursor
```

Understanding Exceptions with PL/SQL

- An exception is a PL/SQL error that is raised during program execution.
- An exception can be raised:
  - Implicitly by the Oracle Server
  - Explicitly by the program
- An exception can be handled:
  - By trapping it with a handler
  - By propagating it to the calling environment

Handling Exceptions



Exception Types

- Internally defined
- Predefined

Implicitly raised

- User-defined

Explicitly raised

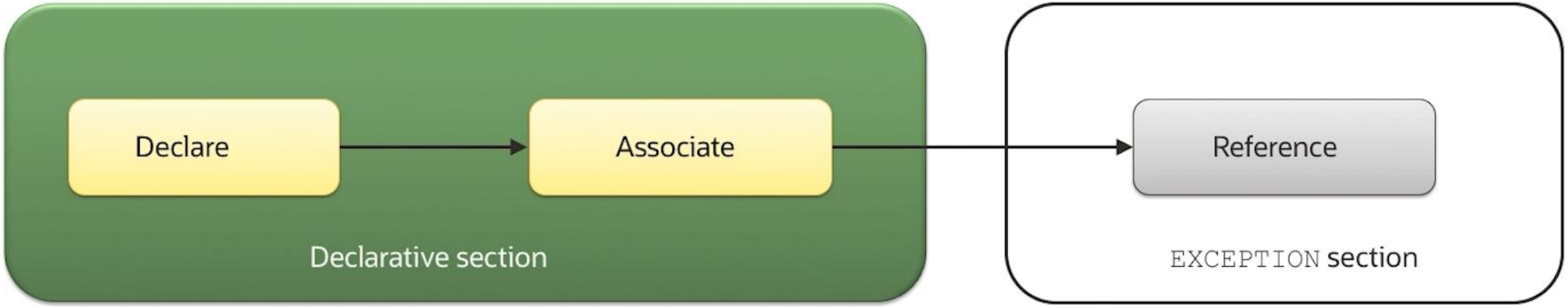
Syntax to Trap Exceptions

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

Guidelines for Trapping Exceptions

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

Trapping Internally Predefined Exceptions



Name the exception.

Use PRAGMA EXCEPTION_INIT.

Handle the raised exception.

Internally Defined Exception Trapping: Example

To trap Oracle Server error `01400` ("cannot insert `NULL`"):

```
DECLARE
 e_insert_excep EXCEPTION;
 PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
 INSERT INTO departments
 (department_id, department_name) VALUES (280, NULL);
EXCEPTION
 WHEN e_insert_excep THEN
   DBMS_OUTPUT.PUT_LINE('INSERT OPERATION FAILED');
   END;
/
```

1

2

3

Script Output

Task completed in 0.004 seconds

```
PL/SQL procedure successfully completed.

Insert operation failed
```

Trapping Predefined Exceptions

- Reference the predefined name in the `EXCEPTION` block.
- Some sample predefined exceptions are:
  - `NO_DATA_FOUND`
  - `TOO_MANY_ROWS`
  - `INVALID_CURSOR`
  - `ZERO_DIVIDE`
  - `DUP_VAL_ON_INDEX`

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
      statement1;
       ...
     WHEN TOO_MANY_ROWS THEN
     statement1;
      ...
     WHEN OTHERS THEN
      statement1;
```

Functions for Trapping Exceptions

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

```
1  declare
2  e_oops exception;
3  PRAGMA exception_init(e_oops, -01400);
4  begin
5  insert into departments
6  values(900,null, null, null);
7  exception
8  when e_oops then
9  dbms_output.put_line( 'No nulls for deptname');
10 dbms_output.put_line(SQLCODE );
11 dbms_output.put_line(SQLERRM );
12 end;
```

Script Output

Task completed in 0.039 seconds

```
PL/SQL procedure successfully completed.

No nulls for deptname
-1400


PL/SQL procedure successfully completed.

No nulls for deptname
-1400
ORA-01400: cannot insert NULL into ("ORA41"."DEPARTMENTS"."DEPARTMENT_NAME")


PL/SQL procedure successfully completed.
```

Functions for Trapping Exceptions

```
DECLARE
  error_code       NUMBER;
  error_message    VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    error_code := SQLCODE ;
    error_message := SQLERRM ;
   INSERT INTO errors (e_user, e_date, error_code,
   error_message) VALUES(USER,SYSDATE,error_code,
   error_message);
END;
/
```

Trapping User-Defined Exceptions

Declare
- Name the exception in the `DECLARE` section.

Raise
- Define the condition when the exception must be raised in executable section.

Reference
- Handle the exception.

Propagating Exceptions in a Sub-Block

Sub-blocks 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

- Is used in two different places:
  - Executable section
  - Exception section
- Returns error conditions to the user in a manner that is consistent with other Oracle Server errors

The RAISE_APPLICATION_ERROR Procedure

```
DECLARE
  v_deptno NUMBER := 500;
  v_name VARCHAR2(20) := 'Testing';
  e_invalid_department EXCEPTION;
BEGIN
  UPDATE departments
  SET department_name = v_name
  WHERE department_id = v_deptno;
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20202, 'Department number does not exist');
  END IF;
END;
```

Script Output

Task completed in 0.009 seconds

```
  SET department_name = v_name
  WHERE department_id = v_dept_no;
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20202,'Department number does not exist');
  END IF;
END;
Error report -
ORA-20202: Department number does not exist
ORA-06512: at line 10
```

Quiz

You can trap any error by including a corresponding handler within the exception-handling section of the PL/SQL block.

a. True
b. False