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

Example of an Exception

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

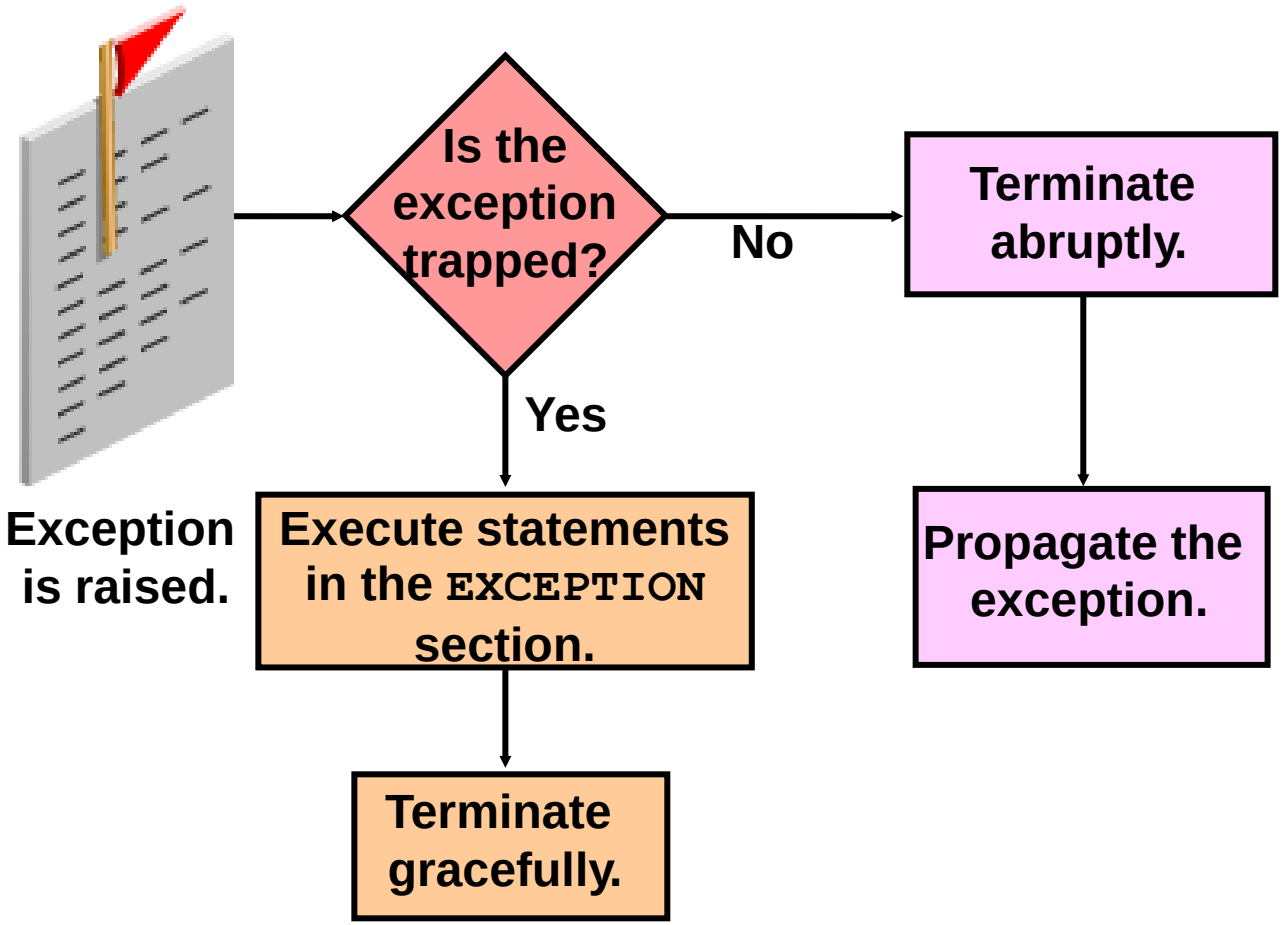
Example of an Exception

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

Handling 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

- Predefined Oracle server
- Non-predefined 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;
    . . .]
```

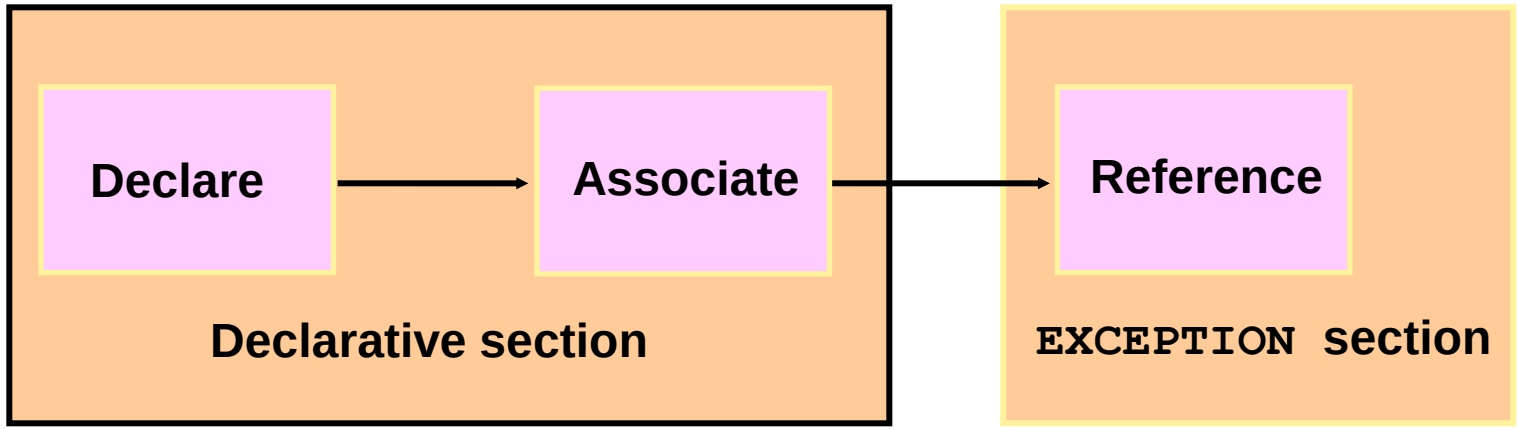
Guidelines for Trapping Exceptions

- 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 Predefined Oracle Server Errors

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

Trapping Non-Predefined Oracle Server Errors



Name the exception.

Use `PRAGMA EXCEPTION_INIT`.

Handle the raised exception.

Non-Predefined Error

- To trap Oracle server error number –01400 ("cannot insert NULL"):

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

Functions for Trapping Exceptions

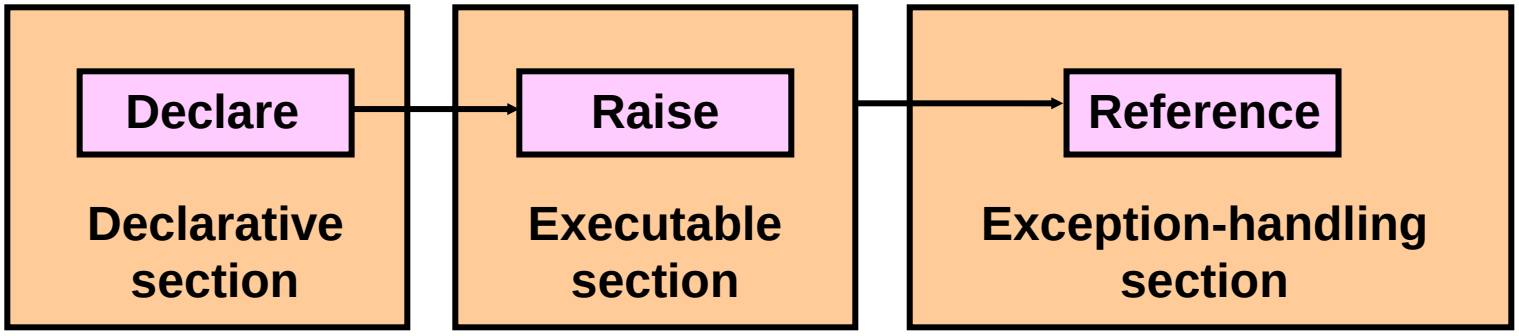
- `SQLCODE`: Returns the numeric value for the error code
- `SQLERRM`: Returns the message associated with the error number

Functions for Trapping Exceptions

- Example

```
DECLARE
  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



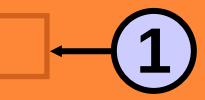
Name the exception.

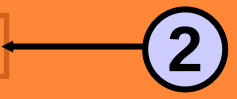
Explicitly raise the exception by using the `RAISE` statement.

Handle the raised exception.

Trapping User-Defined Exceptions

```
...
ACCEPT deptno PROMPT 'Please enter the department number:'
ACCEPT name   PROMPT 'Please enter the department name:'
DECLARE
  invalid_department EXCEPTION;
  name VARCHAR2(20):='&name';
  deptno NUMBER :=&deptno;
BEGIN
  UPDATE  departments
  SET     department_name = name
  WHERE   department_id = deptno;
  IF SQL%NOTFOUND THEN
    RAISE invalid_department;
  END IF;
  COMMIT;
EXCEPTION
  WHEN invalid_department THEN
    DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
/
```





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 an `ON-ERROR` 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 in a Subblock

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

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

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.

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 Procedure

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 how to:
  - Define PL/SQL exceptions
  - Add an `EXCEPTION` section to the PL/SQL block to deal with exceptions at run time
  - Handle different types of exceptions:
    - Predefined exceptions
    - Non-predefined exceptions
    - User-defined exceptions
  - Propagate exceptions in nested blocks and call applications

Practice 8: Overview

- This practice covers the following topics:
  - Handling named exceptions
  - Creating and invoking user-defined exceptions