PL/SQL Programming – Intermediate

Hexaware TECHNOLOGIES

Course Objectives

- Joins
- PL/SQL Fundamentals
- Exception Handling
- Cursors
- Procedures
- Functions
- Packages
- Performance Tuning

Session Objectives

- Exception Handling

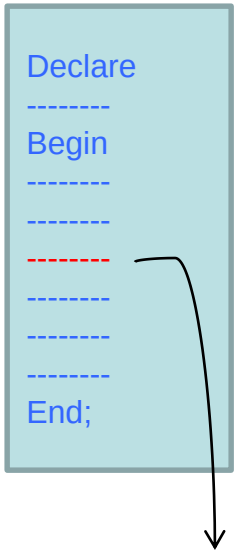
Exception Handling

Exception Handling

- **What is Exception ?**

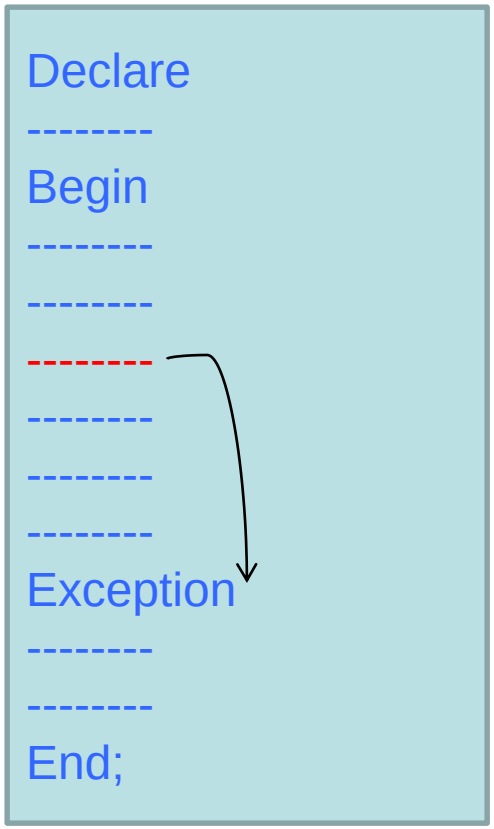
- In PL/SQL, a warning or error condition is called an *exception.*

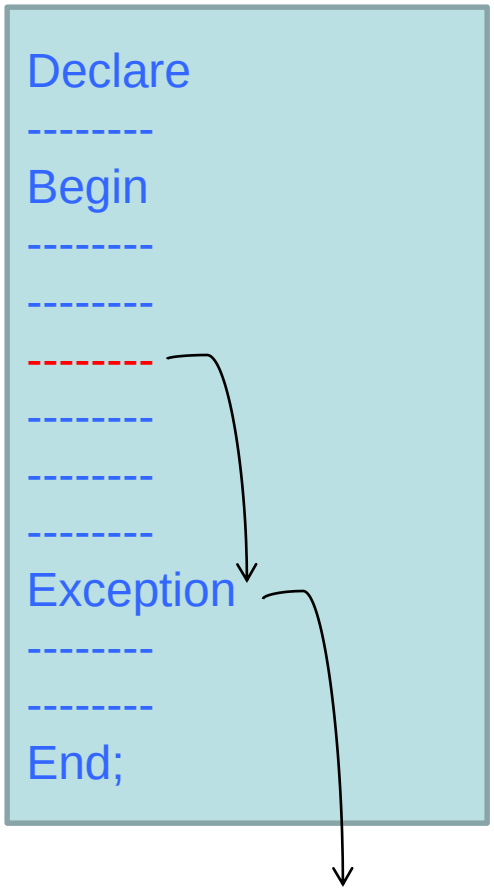
- Exceptions can be internally defined (by the run-time system) or user defined.



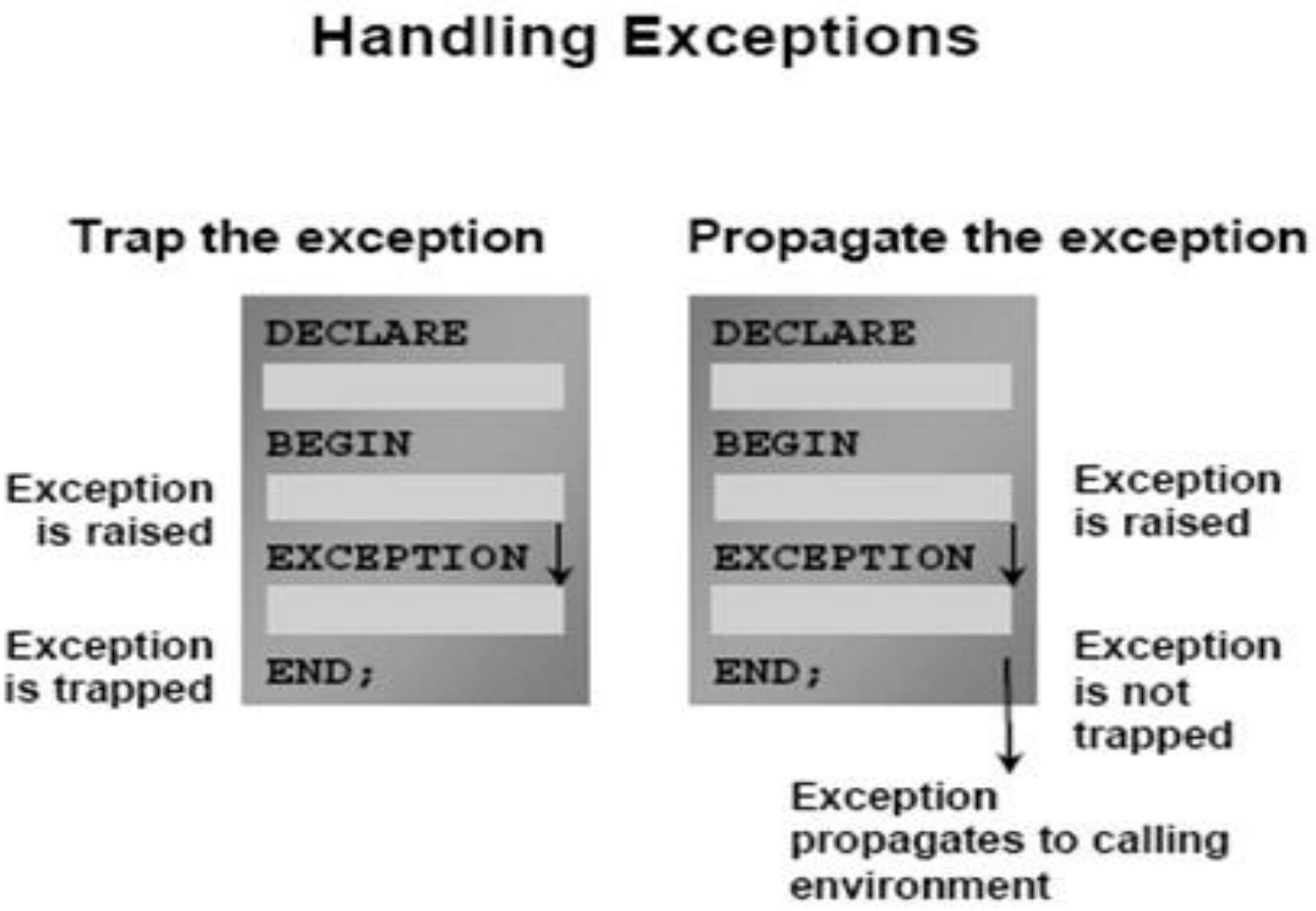
Exception Handling

- 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.



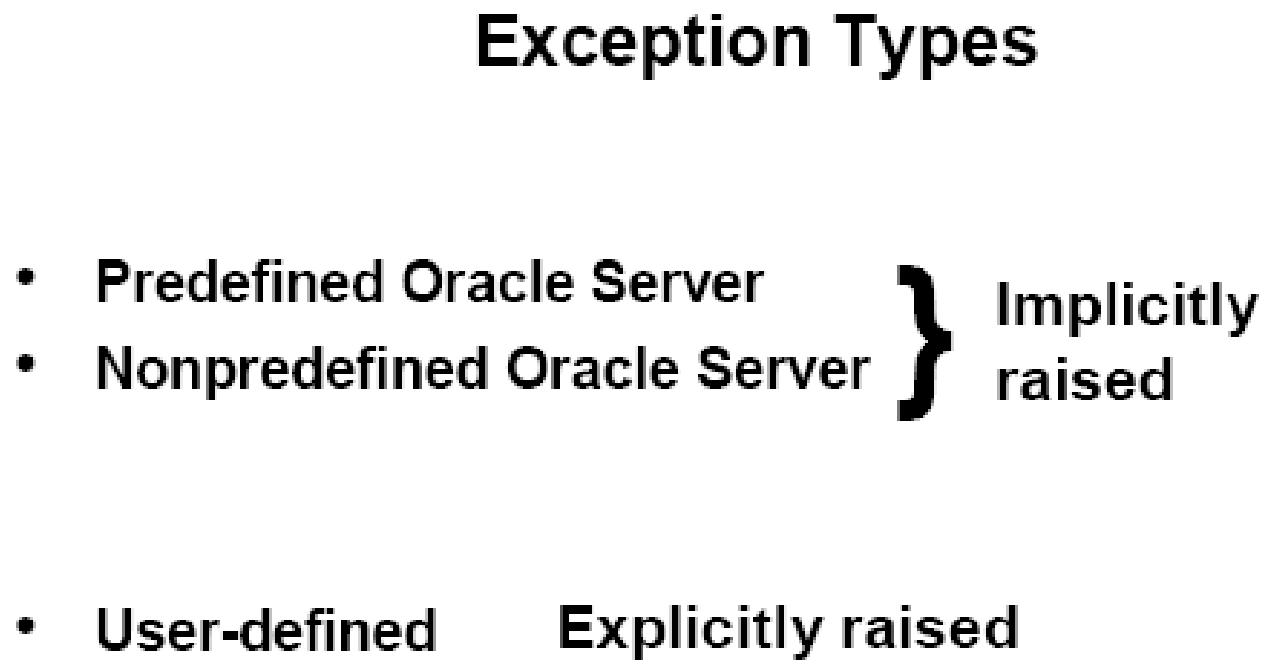


Exception Handling Contd..



Exception Handling Contd..

- **Errors can be handled by the system or by the user**
- **Exception Types**



Exception Handling Contd..

Exception Types and Their description

Exception	Description
Predefined Exceptions	Raised Implicitly whenever a PL/SQL program violates the Oracle rule or exceeds a system-dependent limit
Non-Predefined Exceptions	Declared in the declaration section corresponding to the oracle error number and raised implicitly
User defined Exceptions	Declared in the declaration section and raised explicitly using RAISE statements

Exception Handling - by the user

Trapping Exceptions

- Exceptions can be trapped by including corresponding routine in the exception-handling section of the PL/SQL block.
- Each handler consists of a WHEN clause, which specifies an exception, followed by sequence of statements to be executed when that statement is raised.

Syntax

```
EXCEPTION
WHEN exception1 or [exception2…] THEN
  statement1;
  statement2;
[WHEN exception3 or [exception4…] THEN
  statement1;
  statement2;]
[WHEN OTHERS THEN
  statement1;
  statement2;
…]
```

Predefined Exceptions

❖Trapping Predefined Oracle Errors

- Predefined exceptions can be trapped by referencing its standard name in the corresponding exception handling routine

Exception Name	Oracle Server Error No
ACCESS_INTO_NULL	ORA-06530
CASE_NOT_FOUND	ORA-06592
INVALID_NUMBER	ORA-01722
PROGRAM_ERROR	ORA-06501
TOO_MANY_ROWS	ORA-01422
ZERO_DIVIDE	ORA-01476

Predefined Exceptions

Exception Name	Oracle Server Error Number	Description
ACCESS_INTO_NULL	ORA-06530	Attempted to assign values to the attributes of an uninitialized object
CASE_NOT_FOUND	ORA-06592	None of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	ORA-06531	Attempted to apply collection methods other than EXISTS to an uninitialized nested table or varray
CURSOR_ALREADY_OPEN	ORA-06511	Attempted to open an already open cursor
DUP_VAL_ON_INDEX	ORA-00001	Attempted to insert a duplicate value
INVALID_CURSOR	ORA-01001	Illegal cursor operation occurred
INVALID_NUMBER	ORA-01722	Conversion of character string to number fails
LOGIN_DENIED	ORA-01017	Logging on to Oracle with an invalid username or password
NO_DATA_FOUND	ORA-01403	Single row SELECT returned no data
NOT_LOGGED_ON	ORA-01012	PL/SQL program issues a database call without being connected to Oracle
PROGRAM_ERROR	ORA-06501	PL/SQL has an internal problem
ROWTYPE_MISMATCH	ORA-06504	Host cursor variable and PL/SQL cursor variable involved in an assignment have

Predefined Exceptions

Exception Name	Oracle Server Error Number	Description
STORAGE_ERROR	ORA-06500	PL/SQL ran out of memory or memory is corrupted.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Referenced a nested table or varray element using an index number larger than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Referenced a nested table or varray element using an index number that is outside the legal range (–1 for example)
SYS_INVALID_ROWID	ORA-01410	The conversion of a character string into a universal ROWID fails because the character string does not represent a valid ROWID.
TIMEOUT_ON_RESOURCE	ORA-00051	Time-out occurred while Oracle is waiting for a resource.
TOO_MANY_ROWS	ORA-01422	Single-row SELECT returned more than one row.
VALUE_ERROR	ORA-06502	Arithmetic, conversion, truncation, or size-constraint error occurred.
ZERO_DIVIDE	ORA-01476	Attempted to divide by zero

Predefined Exceptions

Example

```
BEGIN
SELECT ...
SELECT ...
SELECT ...
...
EXCEPTION
WHEN NO_DATA_FOUND THEN -- catches all 'no data found' errors
…
WHEN TOO_MANY_ROWS THEN– catches all 'too many rows' errors
…
WHEN OTHERS THEN– catches all other unhandled errors
…
```

Non Predefined Exceptions

- **Trapping Non Predefined Oracle Errors**

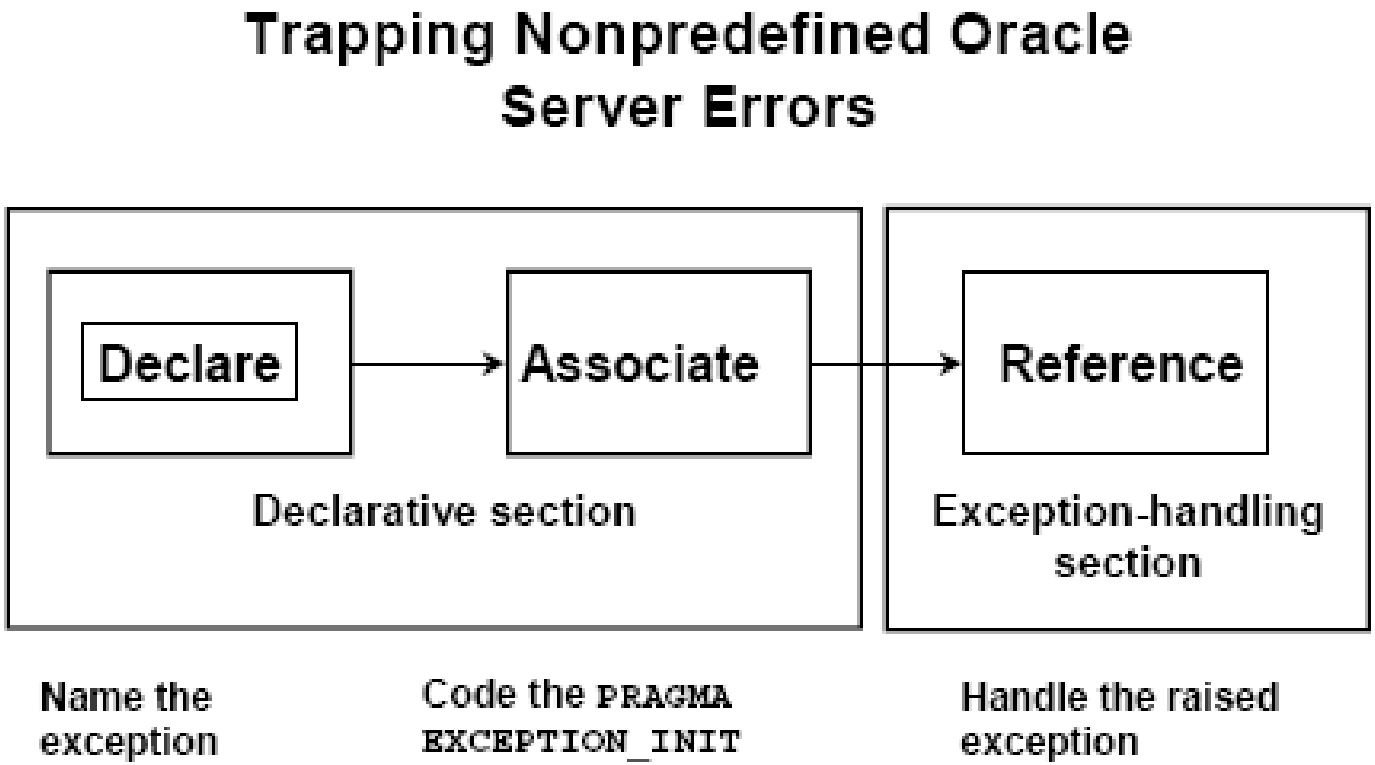
- To handle error conditions (typically ORA- messages) that have no predefined name, you must use the OTHERS handler or the pragma EXCEPTION_INIT.

- A pragma is a compiler directive that is processed at compile time, not at run time. In PL/SQL, the pragma EXCEPTION_INIT tells the compiler to associate an exception name with an Oracle error number, That lets you refer to any internal exception by name and to write a specific handler for it.

Syntax

```
Exception EXCEPTION;
PRAGMA EXCEPTION_INIT(exception,-Oracle error number);
```

Non Predefined Exceptions



Non Predefined Exceptions

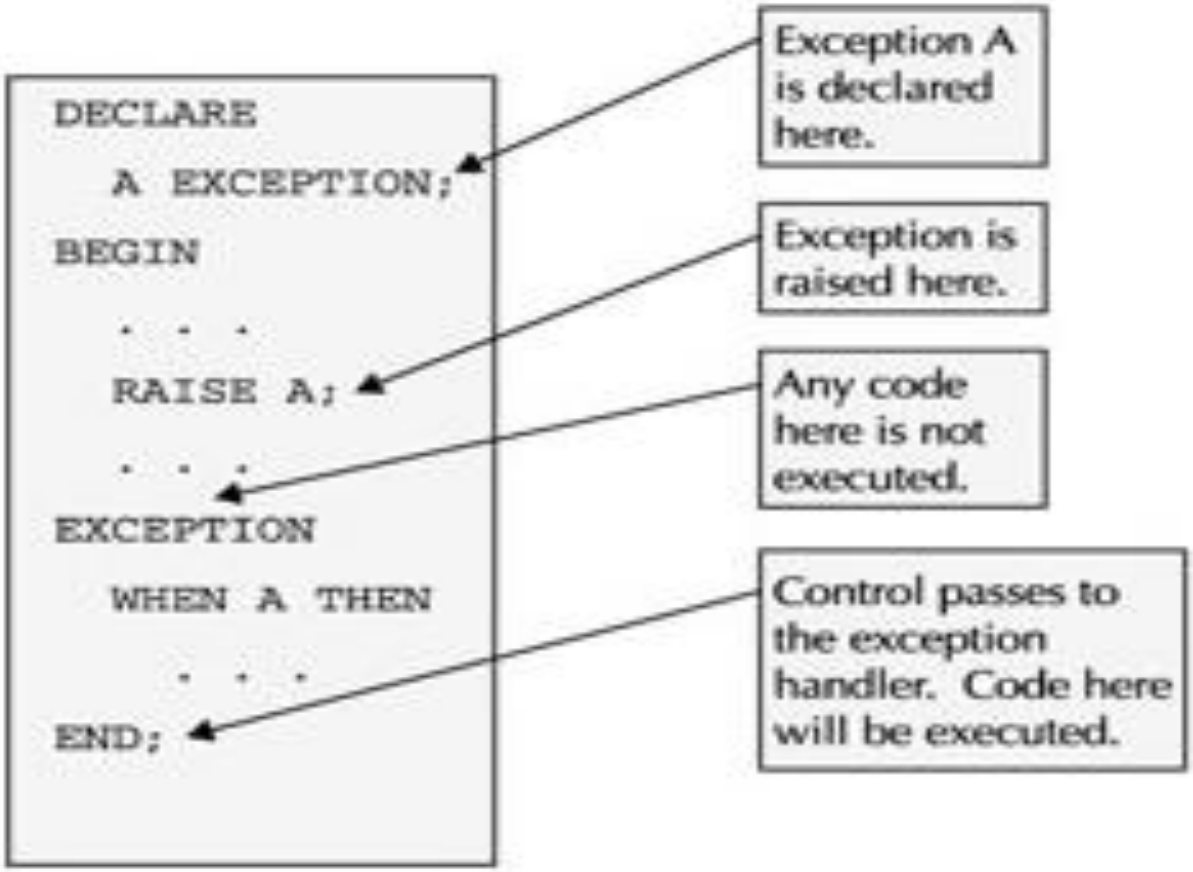
Example

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

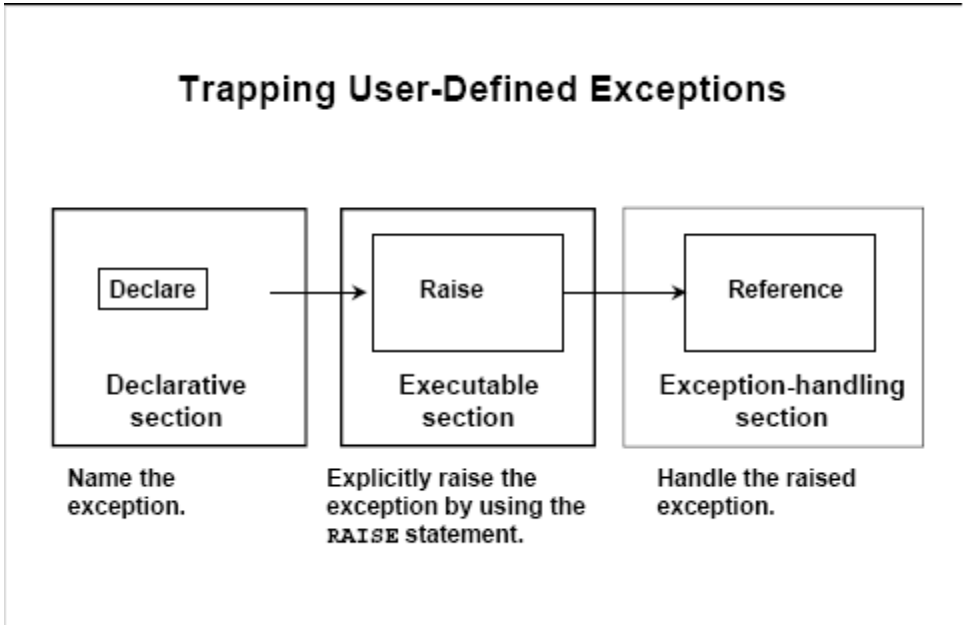
```
DEFINE p_deptno = 10
DECLARE
  e_emps_remaining EXCEPTION;
  PRAGMA EXCEPTION_INIT
    (e_emps_remaining, -2292);
BEGIN
  DELETE FROM departments
  WHERE  department_id = &p_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining  THEN
   DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
   TO_CHAR(&p_deptno)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

User Defined Exceptions

- **Trapping User-defined exceptions**
  - Declared in the declaration section.
  - Raised explicitly with RAISE statements.



User Defined Exceptions



User Defined Exceptions

Example

```
DECLARE
out_of_stock EXCEPTION;              -- DECLARE
number_on_hand NUMBER(4);
BEGIN
...
IF number_on_hand < 1 THEN
RAISE out_of_stock;                  -- RAISE
END IF;
EXCEPTION
WHEN out_of_stock THEN               -- REFERENCE
-- handle the error
END;
```

User Defined Exceptions – Reraising

```
DECLARE
    excep1 EXCEPTION;
    BEGIN
       BEGIN .
            ..
            IF (condition) THEN
              RAISE excep1;
            END IF;
       EXCEPTION
       WHEN excep1 THEN
         NULL;
         RAISE;
    EXCEPTION
     WHEN excep1 THEN
         log_error;
    END;
```

Exception raised

Exception re raised

Exception Handling Contd..

2. **Defining User-defined exceptions same as Pre-defined exceptions**

- User-defined exceptions defined of type EXCEPTION can have the same name as the predefined exceptions.
- To handle predefined and user-defined exceptions in the same block separately, prefix the predefined exception with the word STANDARD followed by a dot (.).
- A user-defined exception with the same name as a predefined PL/SQL exception overrides the predefined exception.

```
DECLARE
    DUP_VAL_ON_INDEX  EXCEPTION;
BEGIN
   ...
   ...
EXCEPTION
 WHEN DUP_VAL_ON_INDEX
 or
 STANDARD.DUP_VAL_ON_INDEX THEN
 ...
 ... END;
```

Exception Handling Contd..

Example SQLCODE Values

SQLCODE Value	Description
0	No Exception encountered
1	User-defined exception
+100	NO_DATA_FOUND exception
Negative Number	Another Oracle server error number

Exception Handling Contd..

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;
```

Exception Handling Contd..

Propagating Exceptions

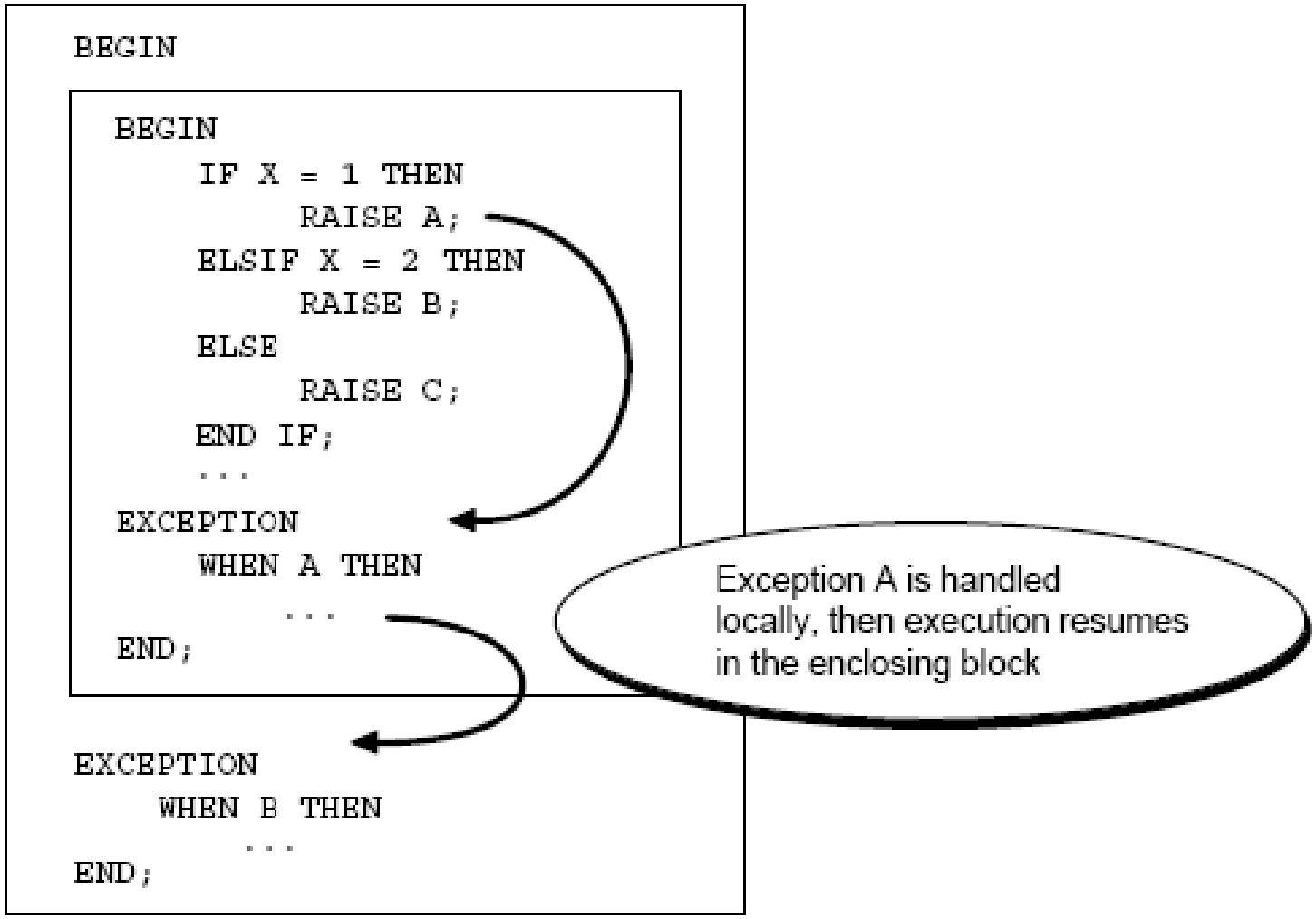
- Exceptions can occur in the declarative, the executable, or the exception section of a PL/SQL block.
- When an exception is raised, if PL/SQL cannot find a handler for it in the current block or subprogram, the exception *propagates.*
- That is, the exception reproduces itself in successive enclosing blocks until a handler is found or there are no more blocks to search.

Exceptions Raised in Executable section

- When an exception is raised in the executable section of a block, PL/SQL uses the following algorithm to determine which exception handler to invoke:
- If the current block has a handler for the exception, execute it and complete the block successfully. Control then passes to the enclosing block.

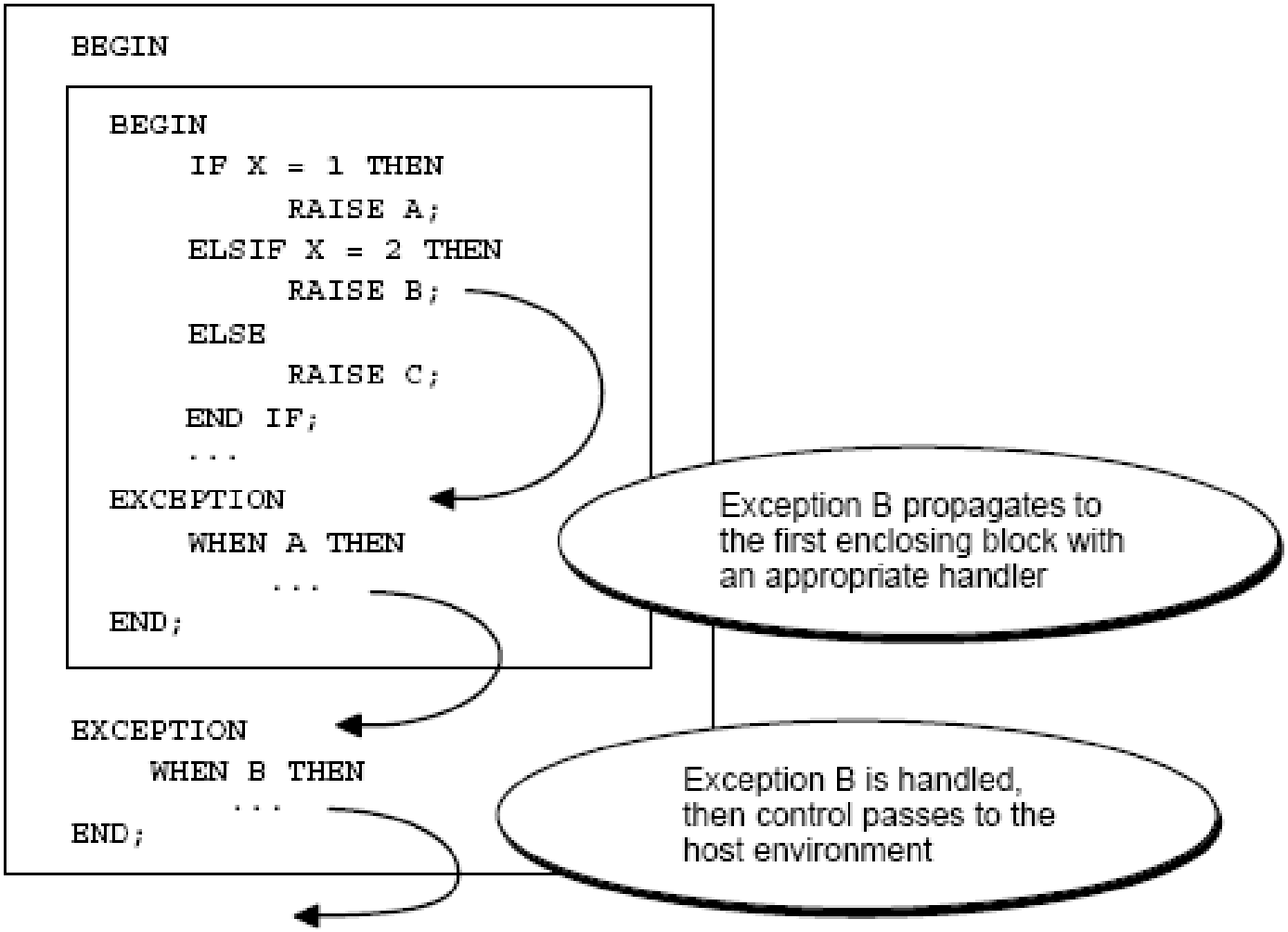
Exception Handling Contd..

- **Exception handled locally (inside the sub block)**



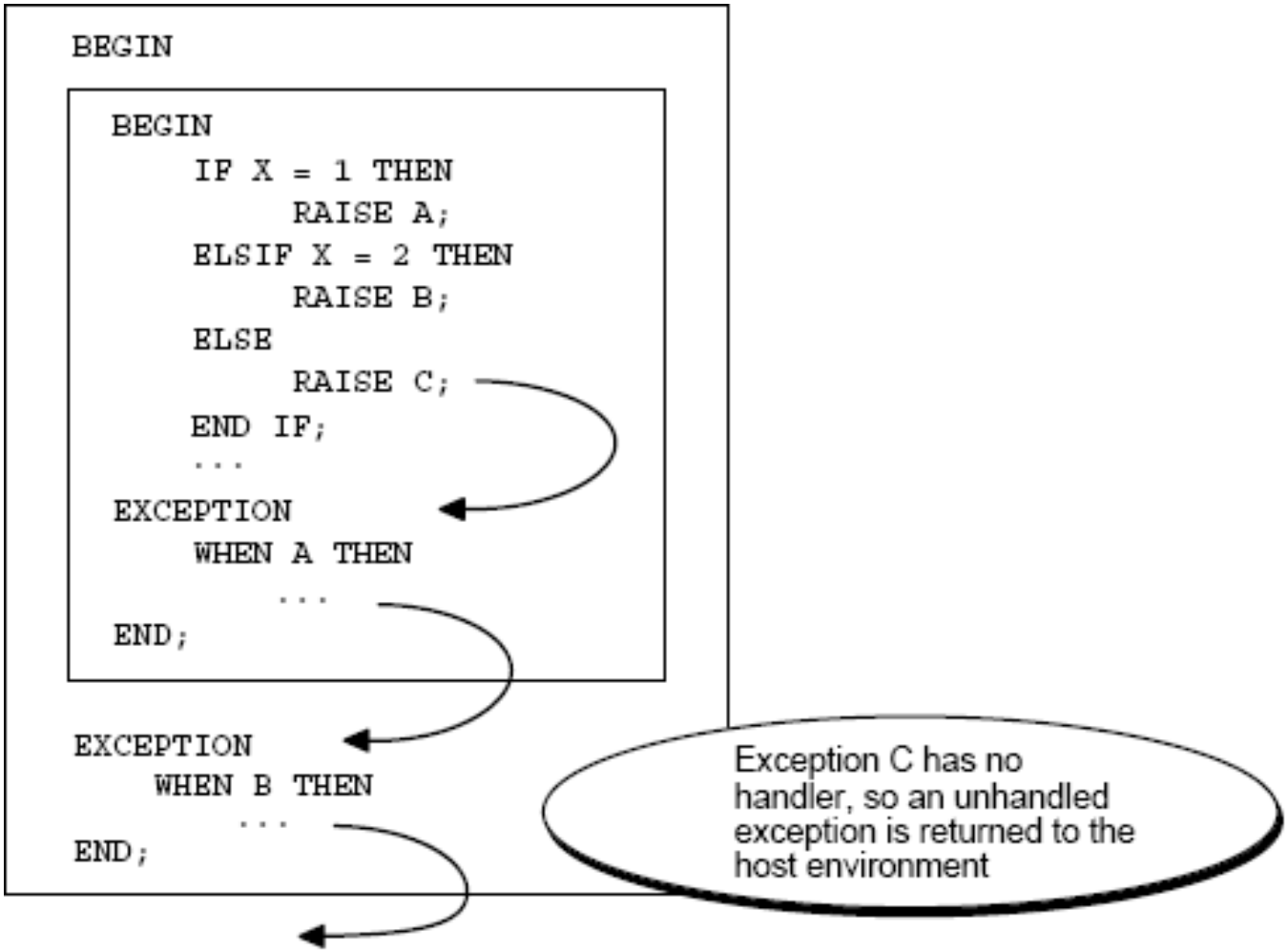
Exception Handling Contd..

- **Exception handled in the enclosing block**



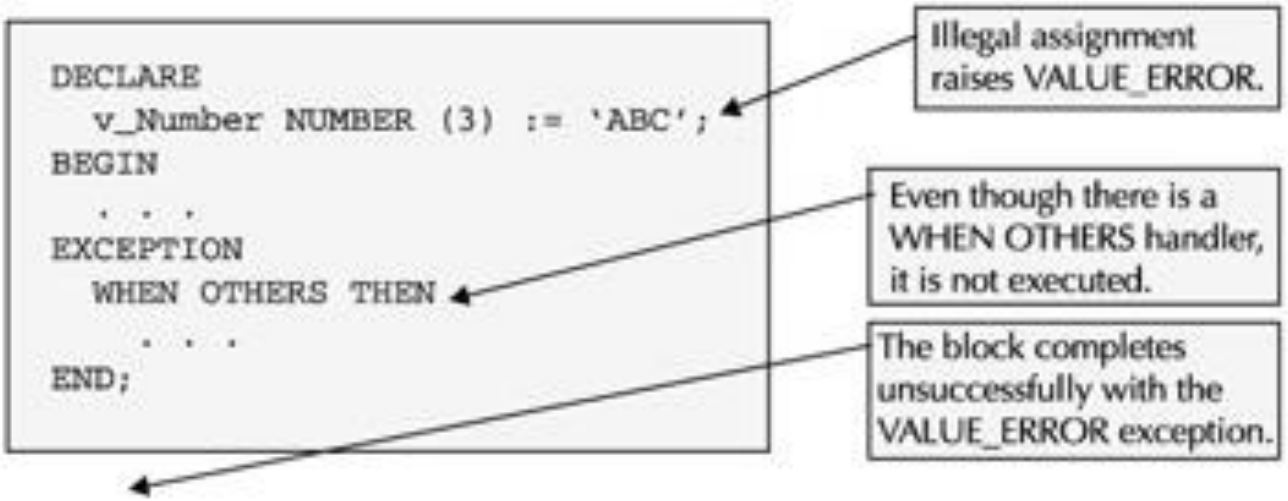
Exception Handling Contd..

- **Unhandled Exception**



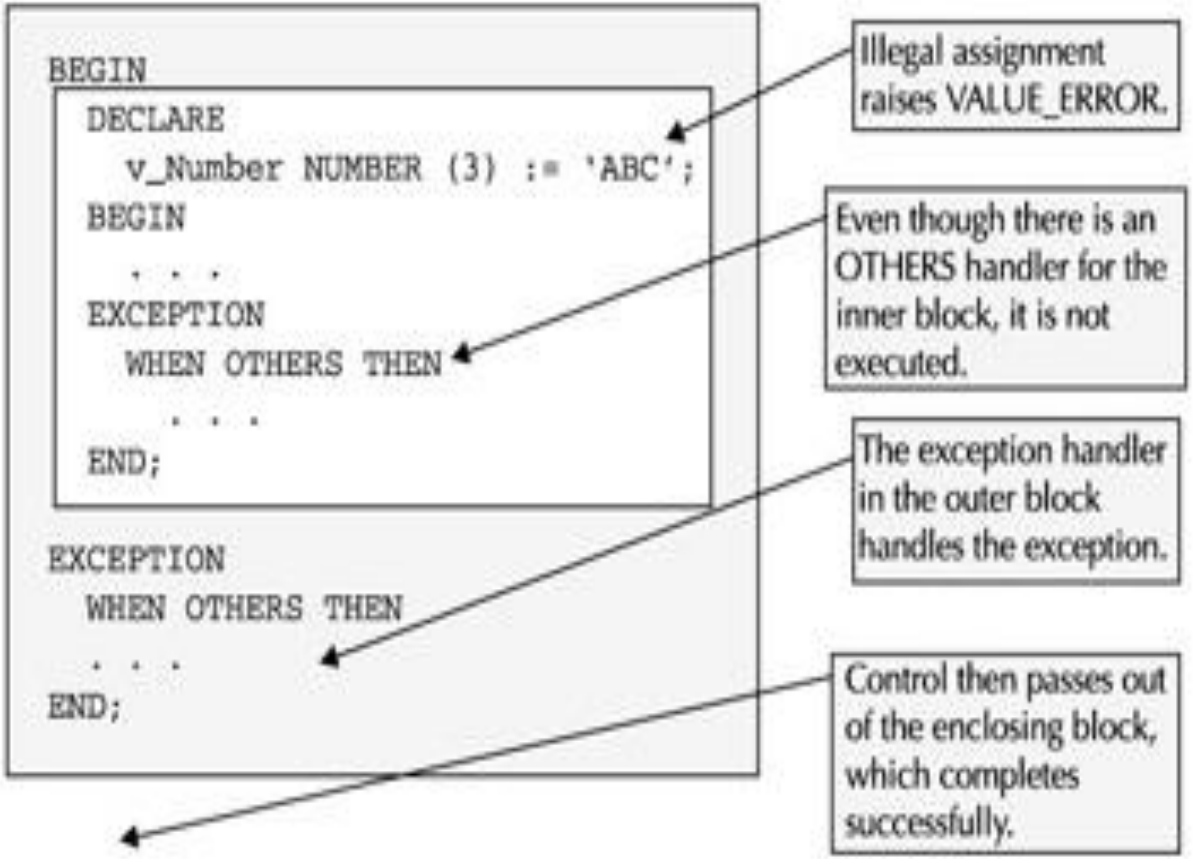
Exception Handling Contd..

Exceptions Raised in Declarative section – Example 1



Exception Handling Contd..

Exceptions Raised in Declarative section – Example 2



Exception Handling Contd..

- **Exceptions Raised in Exception section**

- Exceptions can also be raised while in an exception handler, either explicitly via the RAISE statement or implicitly via a runtime error.

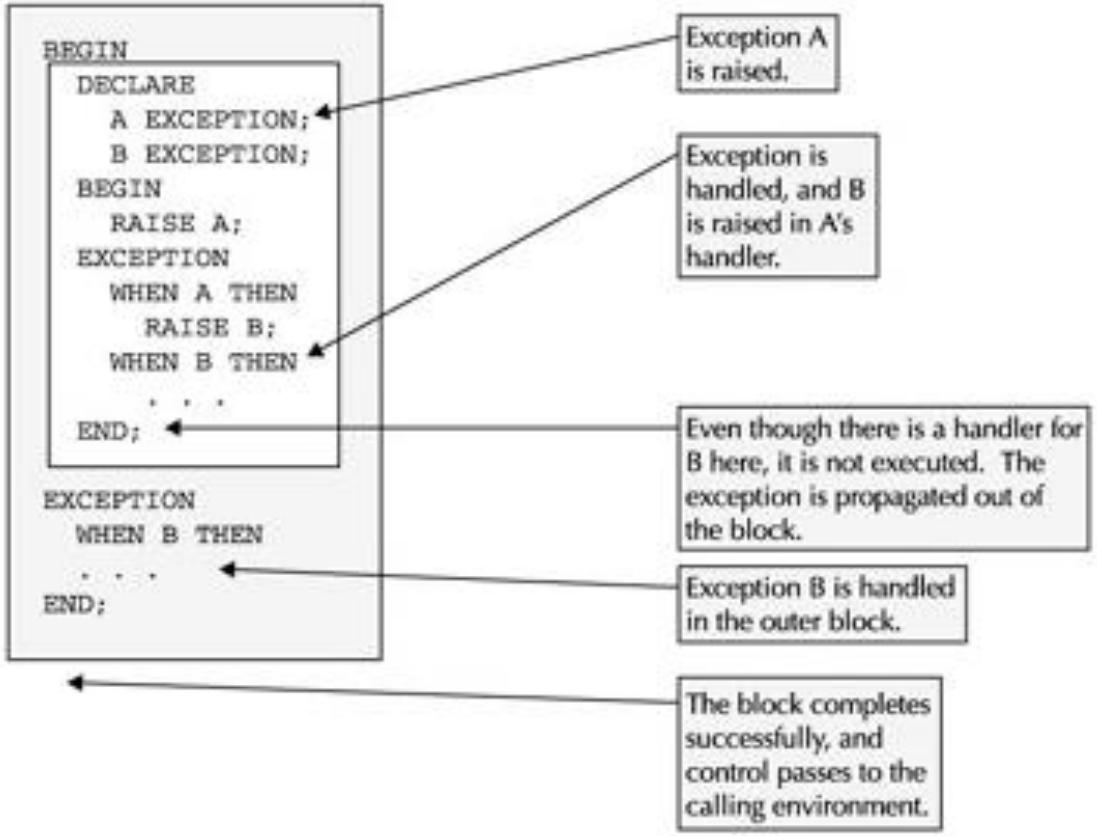
- In either case, the exception is propagated immediately to the enclosing block, like exceptions raised in the declarative section.

- This is done because only one exception at a time can be "active" in the exception section.

- As soon as one is handled, another can be raised. But there cannot be more than one exception raised simultaneously.

Exception Handling Contd..

Exceptions Raised in Exception section - Example 2



Exception Handling Contd..

- **Using RAISE_APPLICATION_ERROR procedure**

- The procedure RAISE_APPLICATION_ERROR lets you issue user-defined ORA error messages from stored subprograms.
- You can report errors to your application and avoid returning unhandled exceptions.

Syntax

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

where

error_number is a negative integer in the range -20000 .. -20999.

message is a character string up to 2048 bytes long.

If the optional third parameter is TRUE, the error is placed on the stack of previous errors. If the parameter is FALSE (the default), the error replaces all previous errors.

Exception Handling Contd..

- RAISE_APPLICATION_ERROR procedure can be used in two different places:
  - Executable section
  - Exception section
- Returns error that is consistent with other Oracle server errors. The error number and message is displayed to the user.

Exception Handling Contd..

Example ,

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;
```

Key Points to Remember

- Categorize errors in your PL/SQL code by segregating them into predefined exceptions, non-predefined Oracle errors, user-defined exceptions, and user-defined PL/SQL error messages.
- For all of these types of errors, write generic error-handling routines that separate error processing from business logic processing.

Best Practices

Best Practice	Benefits
Whenever multiple SELECT statements are used in a block, handle 'NO_DATA_FOUND' and 'TOO_MANY_ROWS' exceptions for each of the SELECT statements in the block	Indicates which SELECT statement has resulted in the exception. Debugging becomes easier
When naming user-defined exceptions, give suitable meaningful names inaccordance with the context. Although giving same name as predefined exceptions is permitted, avoid it.	Aids in better understanding and tracability.Avoids confusion.
Specify the WHEN OTHERS handler at the topmost level of your program. This handler may simply log the error and where it occurred.	Ensures no error goes undetected

Thank You