Tech Mahindra

Exception Handling

Objectives

- After completing this lesion, 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

Introduction

- In PL/SQL, errors and warnings are called as exceptions. Whenever a predefined error occurs in the program, PL/SQL raises an exception.
  - ZERO_DIVIDE, NO_DATA_FOUND
- PL/SQL has a collection of predefined exceptions.
- In addition to predefined exceptions, user can also create his own exceptions to deal with errors in the application.
- When an exception occurs (is raised) in a PL/SQL block, its execution section immediately terminates. Control is passed to the exception section.
- Every exception in PL/SQL has an error number and error message; some exceptions also have names.

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.

Exceptions - Types

- We can use 3 different types of exceptions

Exception	Description	Directions for Handling
Predefined Oracle Server error (Named system exception)	One of approximately 20 errors that occur most often in PL/SQL code	Do not declare and allow the Oracle server to raise them implicitly
No predefined Oracle Server error (Unnamed system exceptions)	Any other standard Oracle Server error	Declare within the declarative section and allow the Oracle Server to raise them implicitly.
User-defined error	A condition that the developer determines is abnormal	Declare within the declarative section, and raise explicitly.

Named System Exceptions

- System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

- **For example:** NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.
- Named system exceptions are:
    1) Not Declared explicitly,
    2) Raised implicitly when a predefined Oracle error occurs,
    3) caught by referencing the standard name within an exception-handling routine.

Exception Name	Reason	Error Number
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.	ORA-06511
INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened.	ORA-01001
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.	ORA-01403
TOO_MANY_ROWS	When you SELECT or fetch more than one row into a record or variable.	ORA-01422
ZERO_DIVIDE	When you attempt to divide a number by zero.	ORA-01476

Unnamed System Exceptions

These exception do not occur frequently. These Exceptions have a code and an associated message.

There are two ways to handle unnamed sysyem exceptions:

1. By using the WHEN OTHERS exception handler, or
2. By associating the exception code to a name and using it as a named exception.

We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION_INIT.**
EXCEPTION_INIT will associate a predefined Oracle error number to a programmer_defined exception name.

Steps to be followed to use unnamed system exceptions are

- They are raised implicitly.
- If they are not handled in WHEN Others they must be handled explicity.
- To handle the exception explicity, they must be declared using Pragma EXCEPTION_INIT as given above and handled referecing the user-defined exception name in the exception section.

The general syntax to declare unnamed system exception using EXCEPTION_INIT is:

```
DECLARE
   exception_name EXCEPTION;
   PRAGMA
   EXCEPTION_INIT (exception_name, Err_code);
BEGIN
Execution section
EXCEPTION
  WHEN exception_name THEN
     handle the exception
END;
```

User-defined Exceptions

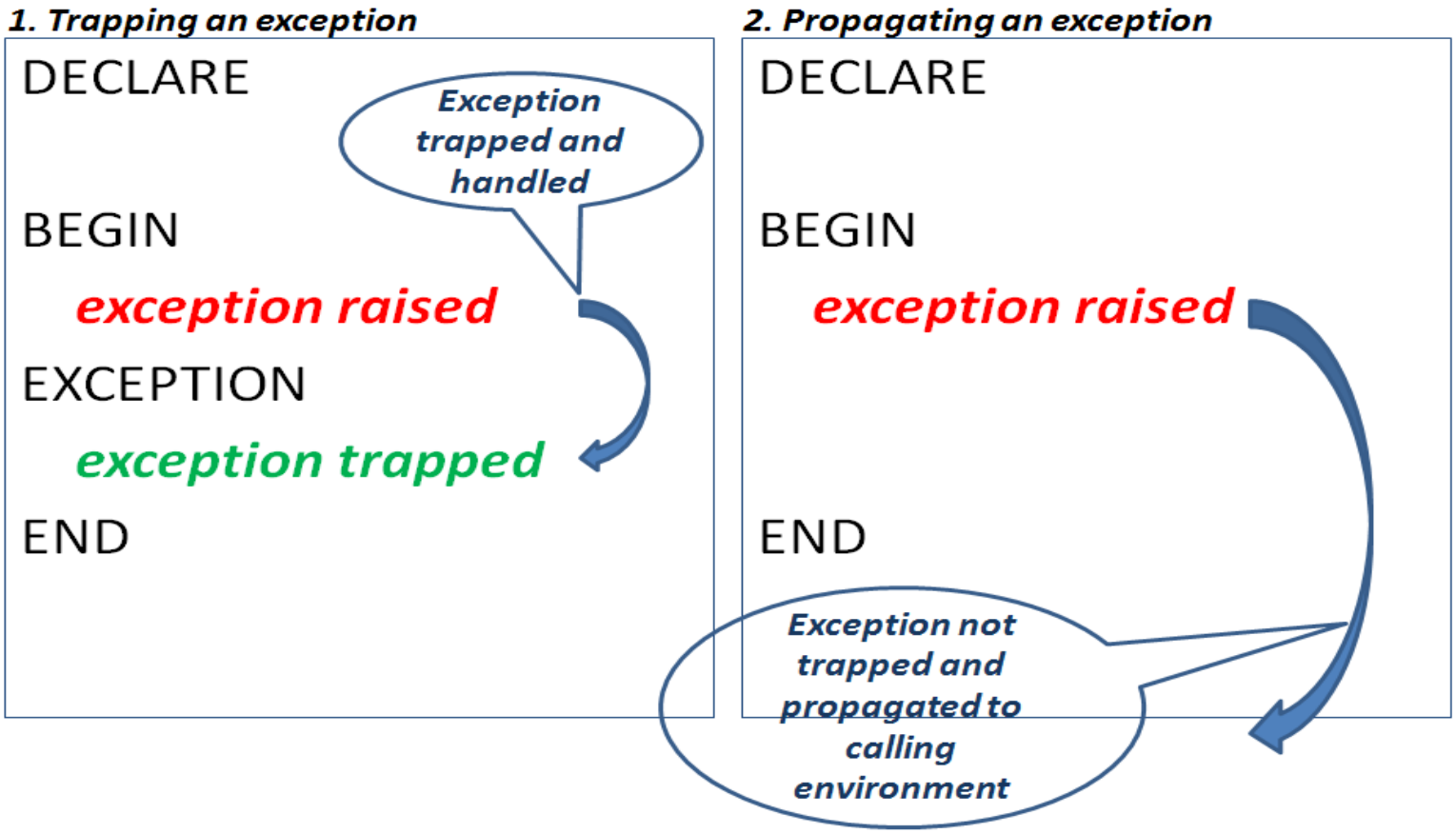
- Apart from system exceptions we can explicitly define exceptions based on business rules. These are known as user-defined exceptions.

- Steps to be followed to use user-defined exceptions:
  - They should be explicitly declared in the declaration section.
  - They should be explicitly raised in the Execution Section.
  - They should be handled by referencing the user-defined exception name in the exception section.

- **For Example:** Lets consider the product table and order items table from sql joins to explain user-defined exception.

  Lets create a business rule that if the total no of units of any particular product sold is more than 20, then it is a huge quantity and a special discount should be provided.

Handling exceptions



Trapping an Exception

- If the exception is raised in the executable section of the block, processing branches to the corresponding exception handler in the exception section of the block.

- If PL/SQL successfully handles the exception, then the exception does not propagates to the enclosing block or environment. The PL/SQL block terminates successfully.

Propagating an Exception

- If the exception is raised in the executable section of the block and there is no corresponding exception handler, the PL/SQL block terminates with failure and the exception is propagated to the calling environment.

Trapping Exceptions

Syntax:

```
EXCEPTION
WHEN   exception1  [  OR  exception2  … ] THEN
     Statement1;
     Statement2;
WHEN exception3   [   OR  exception4 …]   THEN
     Statement3;
     Statement4;
WHEN OTHERS THEN
     Statement5;
     Statement6;
```

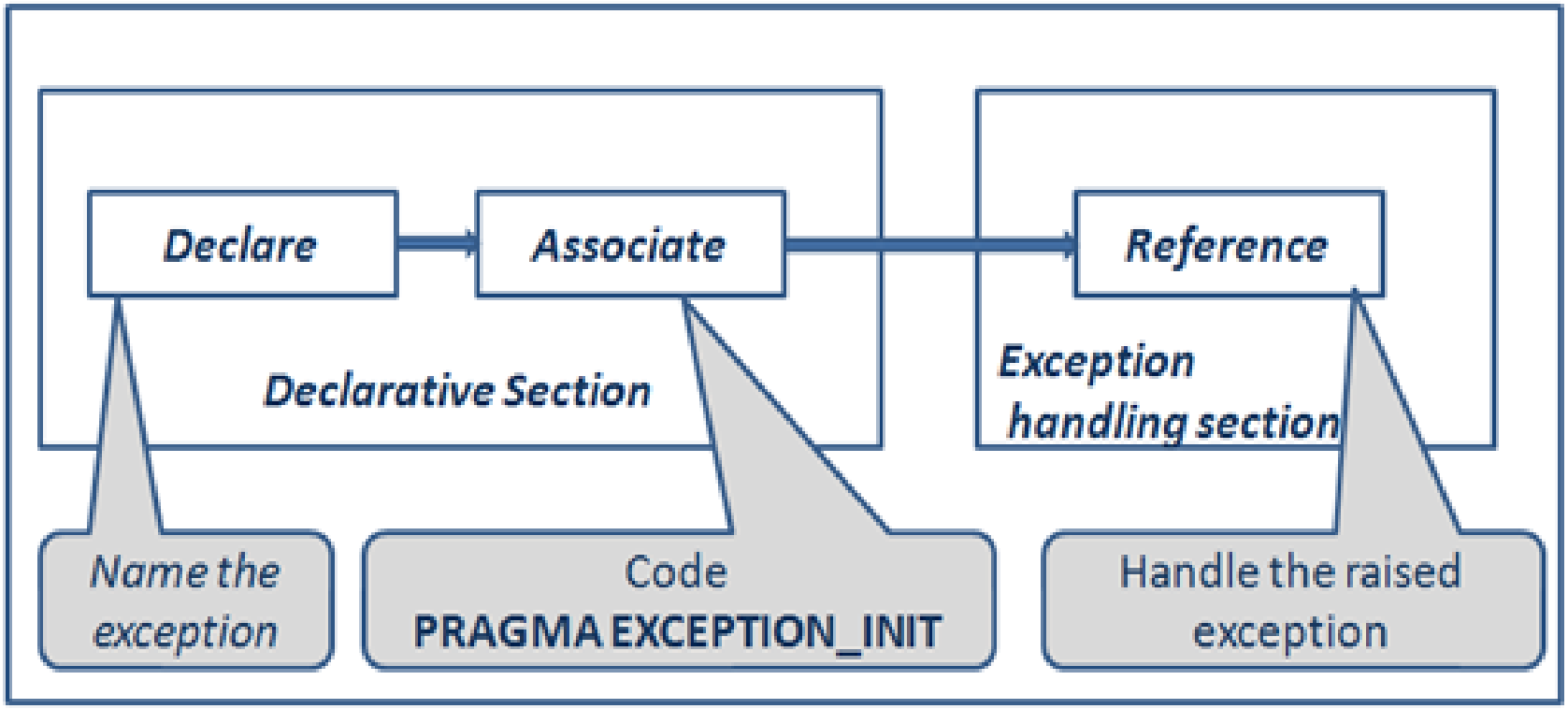
Example

```
DECLARE
vempno NUMBER;
BEGIN
SELECT empno INTO vempno FROM EMP WHERE ename = 'RAM';
EXCEPTION
WHEN NO_DATA_FOUND THEN
INSERT INTO emp (empno, ename, job, deptno) VALUES (101,'RAM',
'EXECUTIVE', 10);
END;
```

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.
- Exceptions cannot appear in assignment statements or SQL statements.
- You can have only one OTHERS clause.

Trapping Non-predefined Oracle Server Errors



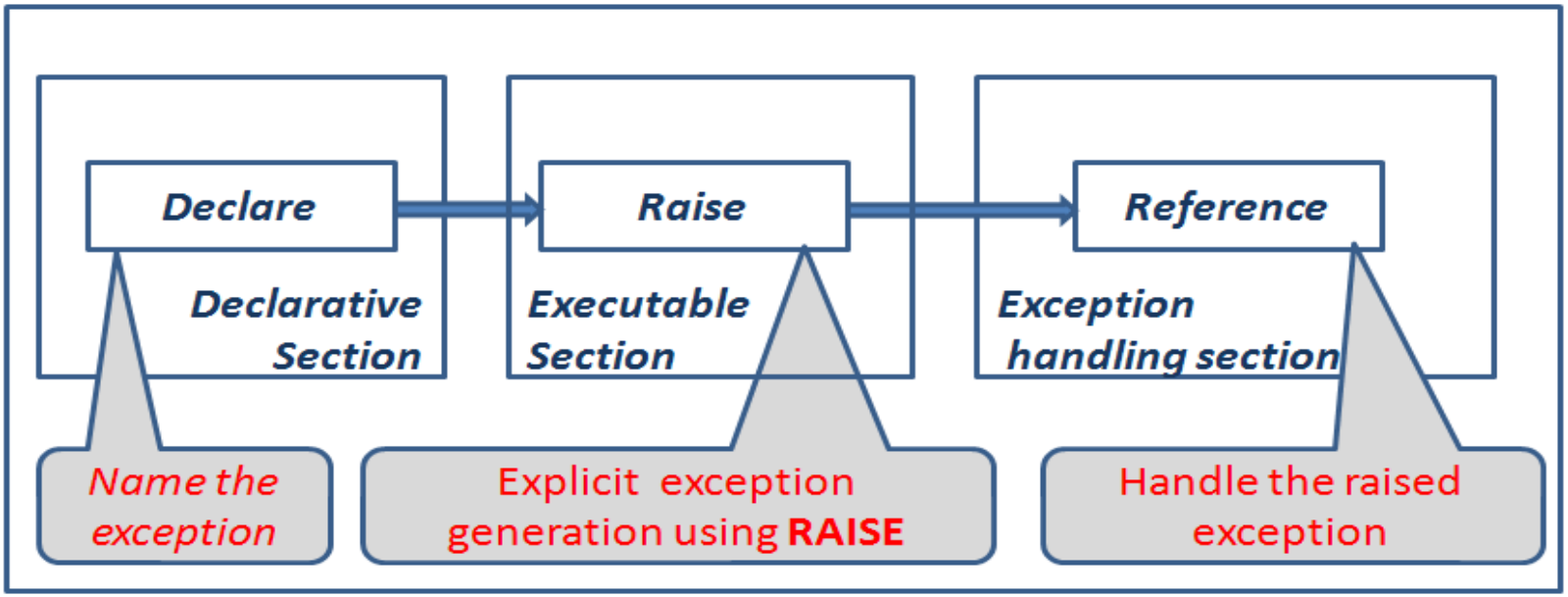
Example

```
declare
deptno_in number;
still_have_employees EXCEPTION;
PRAGMA EXCEPTION_INIT(still_have_employees,  -2292);
BEGIN
      deptno_in := &deptno;
      DELETE FROM DEPT WHERE deptno = deptno_in;
      EXCEPTION WHEN still_have_employees THEN
      DBMS_OUTPUT.PUT_LINE('Please delete employees in dept first');
      ROLLBACK;
      -- RAISE; /* Re-raise the current exception. */
END;
```

Assume that when ON DELETE CASCADE is not defined at the time of establishing the relationship with EMP, then it does not allow you to delete any row from DEPT as long as related data is there in EMP.

Above PLSQL program shows how to handle such exception in PLSQL.

Trapping User-Defined Exceptions



PL/SQL allows you to define your own exceptions. User-defined PL/SQL exceptions must be:

- Declared in the declare section of a PL/SQL block
- Raised explicitly with RAISE statements

Program

- Write a program to maintain students information with Rollno, Student_name, sub1,sub2, Total,average Where sub1,sub2 are the marks in different subjects (handle exception when subject marks are negative).

```
DECLARE
v_sturec      STUDENT%rowtype;
e_Negative    EXCEPTION;
BEGIN
-- generate rollno automatically
SELECT NVL(MAX(rollno),0) + 1 INTO v_sturec.rollno FROM student;
-- input name and marks in two subjects
v_sturec.sname  := ‘&name’;
v_sturec.sub1   := &marks1;
v_sturec.sub2   := &marks2;
IF v_sturec.sub1 < 0 or v_sturec..sub2  < 0 THEN
   RAISE e_Negative;
END IF;
v_sturec..total      := v_sturec..sub1 + v_sturec..sub2;
v_sturec..average:= v_sturec..total/2;
INSERT INTO STUDENT VALUES stu;
COMMIT;
EXCEPTION
WHEN e_negative THEN
DBMS_OUTPUT.PUT_LINE(‘ –VE MARKS’);
END;
/
```

Exercise

- Write a PL/SQL program to maintain Employee Payroll data with:

Column	datatype	Integrity Constraint
EMPNO	NUMBER(4)	PRIMARY KEY
ENAME	VARCHAR2(20)	
BASIC	NUMBER(6)	NOT NULL
DA	NUMBER(8,2)	
HRA	NUMBER(7,2)	
GROSS	NUMBER(10,2)	
PF	NUMBER(7,2)	
NET	NUMBER(10,2)	

Business Logic:

1. Generate EMPNO automatically
2. Input name and basic (handle suitable exception when basic is negative)
3. Calculate other particulars (handle exception when name is NULL)
   - DA is % on basic
   - HRA is % on basic
   - GROSS = BASIC + DA = HRA
   - PF = % ON BASIC
   - NET = GROSS – PF
4. Store the data into the table

Program

- Write a program to get the salary of an employee

```
DECLARE
   v_empno    emp.empno%TYPE;
   v_sal      emp.sal%TYPE;
BEGIN
   v_empno   :=  &empno;
   SELECT sal into v_sal FROM emp WHERE empno = v_empno;
   DBMS_OUTPUT.PUT_LINE('SALARY = ' || v_sal);
EXCEPTION
   WHEN NO_DATA_FOUND THEN                -- we are handling exception here
      DBMS_OUTPUT.PUT_LINE('Employee not found…');
END;
/
```

Please verify the above program for the following

- Check what happened when we input a 5 digit number
- Check what happens when we input employee number in single quotes
- Check what happens when we input alpha-numeric information
- Check what happens when there is a duplicate empno
- Use SQLCODE and SQLERRM Error trapping functions

Program

- Write a PL/SQL block that prints the number of employees who make plus or minus $100 of the salary value entered.
- 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.
- If there is one or more employees within that range, the message should indicate how many employees are in that salary range.
- Handle any other exception with an appropriate exception handler. The message should indicate that some other error occurred.

- Solution : In the next slide

```
VARIABLE g_message VARCHAR2(100)
SET VERIFY OFF
ACCEPT p_sal PROMPT 'Please enter the salary : '
DECLARE
V_sal              emp.sal%type      := &p_sal;
V_low_sal          emp.sal%type      := v_sal – 100;
V_high_sal         emp.sal%type      :=  v_sal + 100;
V_No_Emp           NUMBER(7);
E_no_emp_returned   EXCEPTION;
E_more_than_one_emp EXCEPTION;
BEGIN
SELECT count(enme) INTO v_no_emp FROM emp WHERE sal between v_low_sal and
v_high_sal;
IF v_no_emp = 0 THEN
RAISE e_no_emp_returned;
ELSIF v_no_emp > 0 THEN
RAISE e_more_than_one_emp;
END IF;
EXCEPTION
WHEN e_no_emp_returned THEN
:g_message := 'THERE is no employee salary between ' || v_lowsal || '  and '  ||
v_high_sal;
WHEN e_more_than_one_emp THEN
:g_message := ' There is/are ' || v_no_emp || ' employee(s) with a salary
between ' || v_low_sal || ' and ' || v_high_sal;
END;
/
SET VERIFY ON
PRINT g_message
```

Exercise

Write a program to maintain the passbook

sno	vchdate	trntype	amount	balance
1	sysdate	d	20000	20000
2	sysdate	w	5000	15000
3	sysdate	w	3000	12000
4	sysdate	d	10000	22000

- generate sno automatically
- get the opening balance for each transaction
- Input (D)eposit / (W)ithdraw for trntype
- Input amount and calculate the balance
- Exceptions
  - Amount is always a +ve number
  - For 1st transaction only Deposit(D) is allowed
  - TRNTYPE should be either D or W
  - Withdrawal amount is always within the balance