DATABASE MANAGEMENT SYSTEMS

Subject Teacher: Zartasha Baloch

EXCEPTION HANDLING

Lecture # 33

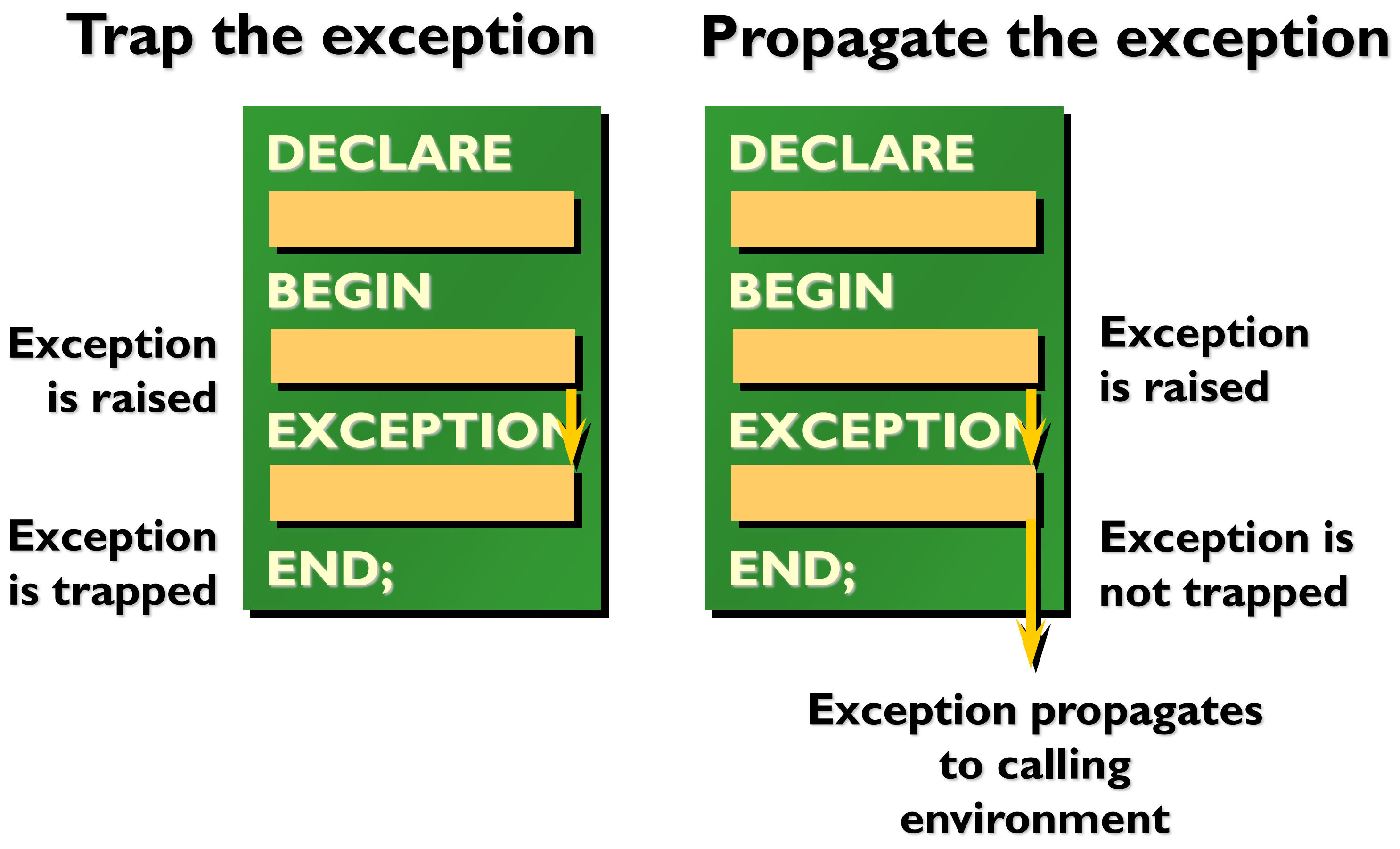
Disclaimer: The material used in this presentation to deliver the lecture i.e., definitions/text and pictures/graphs etc. does not solely belong to the author/presenter. The presenter has gathered this lecture material from various sources on web/textbooks. Following sources are especially acknowledged:

1. Connolly, Thomas M., and Carolyn E. Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.
2. *https://www.tutorialspoint.com/plsql/index.htm*
3. *https://www.oracle.com/database/technologies/appdev/plsql.html*
4. Greenberg, Nancy, and Instructor Guide PriyaNathan. "Introduction to Oracle9i: SQL." ORACLE, USA (2001).

HANDLING EXCEPTIONS WITH PL/SQL

- What is an exception?
  - 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



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;
    . . .]
```

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 EXCEPTION

- Syntax

```
BEGIN  SELECT ... COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;
  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

PREDEFINED EXCEPTION

```
DECLARE
   c_id customers.id%type := 8;
   c_name customerS.Name%type;
   c_addr customers.address%type;
BEGIN
   SELECT  name, address INTO  c_name, c_addr
   FROM customers
   WHERE id = c_id;
   DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
   DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);

EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
/
```

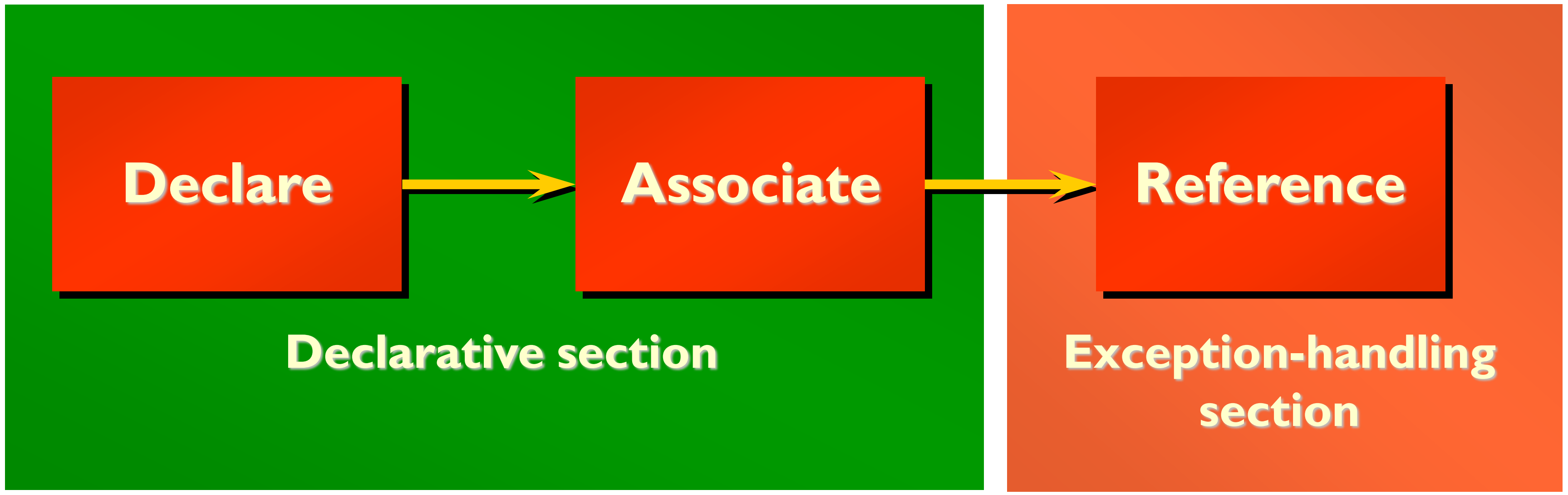
ASSOCIATING A PL/SQL EXCEPTION WITH A NUMBER: PRAGMA EXCEPTION_INIT

- 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. When you see an error stack, or sequence of error messages, the one on top is the one that you can trap and handle.
- You code the pragma EXCEPTION_INIT in the declarative part of a PL/SQL block, subprogram, or package using the syntax

```
PRAGMA EXCEPTION_INIT(exception_name, -Oracle_error_number);
```

where exception_name is the name of a previously declared exception and the number is a negative value corresponding to an ORA- error number. The pragma must appear somewhere after the exception declaration in the same declarative section

TRAPPING NON-PREDEFINED ORACLE SERVER ERRORS



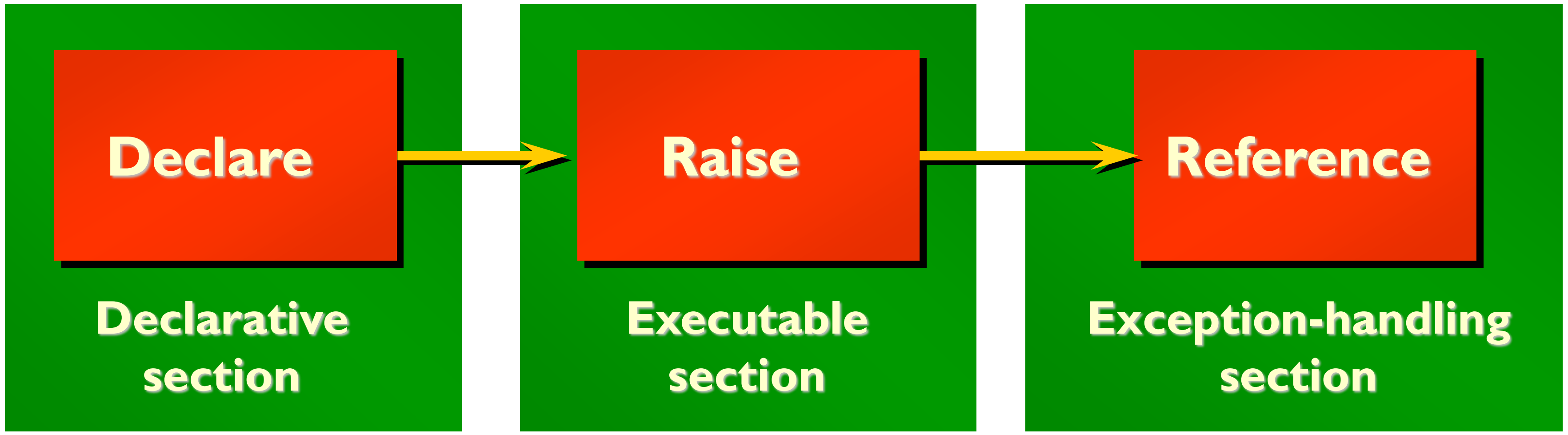
- Name the exception
- Code the PRAGMA EXCEPTION_INIT
- Handle the raised exception

NON-PREDEFINED ERROR

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

```
DECLARE
   e_emps_remaining     EXCEPTION;                       1
   PRAGMA EXCEPTION_INIT (
              e_emps_remaining, -2292);                  2
  v_deptno     dept.deptno%TYPE := &p_deptno;
BEGIN
  DELETE FROM dept
  WHERE        deptno = v_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining THEN                             3
   DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
   TO_CHAR(v_deptno) || '.  Employees exist. ');
END;
```

TRAPPING USER-DEFINED EXCEPTIONS



- Name the exception
- Explicitly raise the exception by using the RAISE statement
- Handle the raised exception

USER-DEFINED EXCEPTION

Example

```
DECLARE
  e_invalid_product  EXCEPTION;                               1
BEGIN
  UPDATE      product
  SET         descrip = '&product_description'
  WHERE       prodid = &product_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_product;                                  2
  END IF;
  COMMIT;
EXCEPTION
  WHEN  e_invalid_product THEN                                3
    DBMS_OUTPUT.PUT_LINE('Invalid product number.');
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
  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;
```

PRE-DEFINED EXCEPTIONS

Exception	Oracle Error	SQLCODE	Description
ACCESS_INTO_NULL	06530	-6530	It is raised when a null object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the WHEN clause of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when a program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.

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;
    EXCEPTION
      WHEN e_integrity THEN ...
      WHEN e_no_rows THEN ...
    END;
END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN . . .
  WHEN TOO_MANY_ROWS THEN . . .
END;
```

RAISE_APPLICATION_ERROR PROCEDURE

Syntax

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

- A procedure that lets you issue user-defined error messages from stored subprograms
- Called only from an executing stored subprogram

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