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

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

Trap the exception

```
DECLARE
BEGIN
END;
```

Exception

Exception is raised

```
EXCEPTION
END;
```

Exception is trapped

Propagate the exception

```
DECLARE
BEGIN
END;
```

Exception

Exception is not trapped

```
EXCEPTION+
END;
```

Exception propagates to calling environment

Exception Types

- Predefined Oracle Server
- Non-predefined Oracle Server
- User-defined

Trapping Exceptions

Syntax

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

Implicitly raised

Explicitly raised
Trapping Exceptions Guidelines

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

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

Trapping Non-Predefined Oracle Server Errors

- Declare
- Associate
- Reference
- Declarative section
- Exception-handling section
- Name the exception
- Code the PRAGMA EXCEPTION_INIT
- Handle the raised exception

Predefined Exception

Syntax

```sql
BEGIN
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;
  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

Non-Predefined Error

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

```sql
DECLARE
    e_emps_remaining EXCEPTION;
    v_deptno%TYPE := &p_deptno;
BEGIN
    DELETE FROM dept
    WHERE deptno = v_deptno;
    COMMIT;
    EXCEPTION
      WHEN e_emps_remaining THEN
        DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
          TO_CHAR(v_deptno) || '.  Employees exist. ');
END;
```
Functions for Trapping Exceptions

- **SQLCODE**
  Returns the numeric value for the error code

- **SQLERRM**
  Returns the message associated with the error number

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;
```

Trapping User-Defined Exceptions

- **Declare**
  Declarative section

- **Raise**
  Executable section

- **Reference**
  Exception-handling section

- **Name the exception**
  Explicitly raise the exception by using the RAISE statement

- **Handle the raised exception**

Example

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

Calling Environments

<table>
<thead>
<tr>
<th>Environment</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Plus</td>
<td>Displays error number and message to screen</td>
</tr>
<tr>
<td>Procedure</td>
<td>Displays error number and message to screen</td>
</tr>
<tr>
<td>Builder</td>
<td></td>
</tr>
<tr>
<td>Oracle Developer Forms</td>
<td>Accesses error number and message in a trigger by means of the ERROR_CODE and ERROR_TEXT packaged functions</td>
</tr>
<tr>
<td>Precompiler application</td>
<td>Accesses exception number through the SOLCA data structure</td>
</tr>
<tr>
<td>An enclosing PL/SQL block</td>
<td>Traps exception in exception-handling routine of enclosing block</td>
</tr>
</tbody>
</table>

Propagating Exceptions

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

**Summary**

- Exception types:
  - Predefined Oracle Server error
  - Non-predefined Oracle Server error
  - User-defined error
- Exception trapping
- Exception handling:
  - Trap the exception within the PL/SQL block.
  - Propagate the exception.

**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