

UNIT 3

PL/SQL Exception

(5 Hours) — Database Programming (CACS453)

1. Introduction to Exceptions

An exception is a runtime error or warning condition that disrupts the normal flow of a PL/SQL block's execution. Rather than letting the program terminate abruptly and uncontrollably, PL/SQL provides a dedicated `EXCEPTION` section within a block where such error conditions can be trapped and handled gracefully — allowing the program to log the problem, clean up, and either continue or fail predictably.

1.1 Categories of Exceptions

Category	Description	Example
Predefined Oracle Exception	Raised automatically by the runtime for common error conditions; has a standard, already-declared name	<code>NO_DATA_FOUND</code> , <code>ZERO_DIVIDE</code>
Non-Predefined (Named) Oracle Exception	An internal Oracle error that has an error number but no predefined name, until associated with one via <code>PRAGMA EXCEPTION_INIT</code>	<code>ORA-02292</code> (integrity constraint violated)
User-Defined Exception	Declared explicitly by the programmer for an application-specific error condition	<code>insufficient_funds</code> , <code>invalid_grade</code>

1.2 Common Predefined Exceptions

Exception Name	Raised When
<code>NO_DATA_FOUND</code>	A <code>SELECT INTO</code> returns no rows
<code>TOO_MANY_ROWS</code>	A <code>SELECT INTO</code> returns more than one row
<code>ZERO_DIVIDE</code>	An attempt is made to divide a number by zero
<code>VALUE_ERROR</code>	An arithmetic, conversion, truncation, or size-constraint error occurs
<code>INVALID_NUMBER</code>	A string fails to convert successfully to a number
<code>DUP_VAL_ON_INDEX</code>	An <code>INSERT/UPDATE</code> attempts to store a duplicate value in a unique index
<code>CURSOR_ALREADY_OPEN</code>	An <code>OPEN</code> is issued on a cursor that is already open

Exception Name	Raised When
OTHERS	A catch-all handler matching any exception not caught by an earlier WHEN clause

Handling a Predefined Exception — Example

DECLARE

v_salary employees.salary%TYPE;

BEGIN

SELECT salary INTO v_salary

FROM employees WHERE emp_id = 9999;

EXCEPTION

WHEN NO_DATA_FOUND THEN

DBMS_OUTPUT.PUT_LINE('No employee found with that ID.');

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);

END;

/

Note: SQLCODE and SQLERRM return the numeric error code and error message of the exception currently being handled — most useful inside a WHEN OTHERS handler where the specific exception is not otherwise known.

2. Exception Scope

Like variables, exceptions declared in a PL/SQL block have a scope — the region of code within which they are visible and can be referenced. An exception declared in the DECLARE section of a block is local to that block (and any blocks nested inside it), and is undefined outside it.

2.1 Local vs Global Exceptions

- An exception declared in an outer block is visible to, and can be raised or handled within, any inner (nested) block — it behaves as a global exception relative to the inner block.
- An exception declared in an inner block is local to that block only. It cannot be referenced by name from the enclosing outer block, since it goes out of scope once the inner block completes.
- If an inner block declares an exception with the same name as one in an outer block, the inner declaration takes precedence within the inner block — the outer exception is

temporarily hidden there.

Exception Scope — Example

```
DECLARE
    ex_outer EXCEPTION; -- visible to inner block too
BEGIN
    DECLARE
        ex_inner EXCEPTION; -- local to inner block only
    BEGIN
        RAISE ex_outer; -- legal: outer exception is in scope
    END;
EXCEPTION
    WHEN ex_outer THEN
        DBMS_OUTPUT.PUT_LINE('Caught ex_outer in outer block');
END;
/
```

Note: Referencing `ex_inner` from the outer block's `EXCEPTION` section would raise a PLS-00201 'identifier must be declared' compile-time error, since `ex_inner` has already gone out of scope by that point.

3. User-Defined Exceptions

Oracle's predefined exceptions cover only generic runtime errors; they cannot express business-rule violations specific to an application (e.g. "insufficient balance" or "invalid grade"). PL/SQL therefore allows exceptions to be declared explicitly by the programmer, given a meaningful name, raised deliberately with `RAISE`, and handled in the usual `EXCEPTION` section.

3.1 Declaring and Raising a User-Defined Exception

A user-defined exception is declared in the `DECLARE` section using the `EXCEPTION` keyword, exactly like a variable declaration, but with no datatype:

Declaration Syntax

```
DECLARE
    exception_name EXCEPTION;
```

It is then raised explicitly, typically inside an IF statement that tests the application's business rule:

User-Defined Exception — Complete Example

```
DECLARE
    insufficient_balance EXCEPTION;
    v_balance NUMBER := 500;
    v_withdrawal NUMBER := 800;
BEGIN
    IF v_withdrawal > v_balance THEN
        RAISE insufficient_balance;
    END IF;
    v_balance := v_balance - v_withdrawal;
EXCEPTION
    WHEN insufficient_balance THEN
        DBMS_OUTPUT.PUT_LINE('Error: withdrawal exceeds available balance.');
```

END;

/

Output

Error: withdrawal exceeds available balance.

Note: Unlike predefined exceptions, a user-defined exception is never raised automatically by the PL/SQL runtime — it must always be raised explicitly with a RAISE statement somewhere in the executable section.

4. Exception Propagation

When an exception is raised inside a block and there is no matching WHEN clause in that block's own EXCEPTION section, the exception propagates — it is passed up to the next enclosing block. This continues outward, block by block, until a handler is found or the outermost block is reached, at which point the exception propagates out to the calling environment (e.g. SQL*Plus) and the block terminates with an unhandled error.

4.1 Propagation in Nested Blocks

- If an exception is raised in the executable section of an inner block and is handled there, the outer block continues normally after the inner block finishes.

- If it is not handled in the inner block, it propagates to the executable section of the immediately enclosing block, where it is treated as though raised at the point the inner block was called.
- If an exception is raised within an exception handler itself, it immediately propagates to the enclosing block — a handler cannot catch an exception raised inside itself or a later handler in the same EXCEPTION section.

Propagation to an Outer Block — Example

```
BEGIN
  DECLARE
    v_result NUMBER;
  BEGIN
    v_result := 10 / 0; -- raises ZERO_DIVIDE, not handled here
  END;
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Caught in outer block: division by zero.');
```

END;

/

Note: Because the inner block has no EXCEPTION section at all, ZERO_DIVIDE propagates immediately to the outer block, where it is caught.

4.2 Propagation Out of a Sub-program

If a procedure or function raises an exception that it does not handle, the exception propagates to the block or sub-program that called it, exactly as with nested anonymous blocks. If it is never handled anywhere along the call chain, it ultimately propagates to the host environment (SQL*Plus, an application, etc.), which reports an unhandled exception error.

5. Advanced Exception Concepts

5.1 RAISE_APPLICATION_ERROR

RAISE_APPLICATION_ERROR is a built-in procedure that lets a PL/SQL program raise a custom error with its own error number and message, and return it to the calling environment in the same way a native Oracle error would be returned. This is the standard way to surface user-defined exceptions with meaningful, application-specific error text outside the PL/SQL

block — for example, back to a calling application or SQL*Plus session.

Syntax

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

error_number must be between -20000 and -20999

message a custom string up to 2048 characters

RAISE_APPLICATION_ERROR — Example

```
DECLARE
```

```
insufficient_balance EXCEPTION;
```

```
v_balance NUMBER := 500;
```

```
v_withdrawal NUMBER := 800;
```

```
BEGIN
```

```
IF v_withdrawal > v_balance THEN
```

```
    RAISE_APPLICATION_ERROR(  
        -20001, 'Withdrawal amount exceeds available balance.');
```

```
END IF;
```

```
END;
```

```
/
```

Result (reported to SQL*Plus)

ORA-20001: Withdrawal amount exceeds available balance.

ORA-06512: at line 6

Note: Reserved error numbers -20000 to -20999 are set aside by Oracle specifically for application-defined errors, so they never collide with genuine internal Oracle error codes.

The optional third parameter controls how the error is added to the error stack: TRUE appends the new error to any errors already raised, while FALSE (the default) replaces the stack with just this error.

5.2 PRAGMA EXCEPTION_INIT

Many internal Oracle errors (identified only by an ORA- error number) have no predefined exception name, which makes them awkward to trap with a specific WHEN clause — normally they could only be caught generically via WHEN OTHERS. The PRAGMA EXCEPTION_INIT compiler directive solves this by associating a programmer-declared exception name with a specific Oracle error number, so it can then be referenced by name in a

WHEN clause just like a predefined exception.

Syntax

```
PRAGMA EXCEPTION_INIT(exception_name, error_number);
```

PRAGMA EXCEPTION_INIT — Example

```
DECLARE
    fk_violation EXCEPTION;
    PRAGMA EXCEPTION_INIT(fk_violation, -2292);
BEGIN
    DELETE FROM departments WHERE department_id = 10;
EXCEPTION
    WHEN fk_violation THEN
        DBMS_OUTPUT.PUT_LINE(
            'Cannot delete: employees still reference this department. ');
END;
/
```

Note: The PRAGMA is a compiler instruction, not an executable statement — it takes effect at compile time, associating the name permanently with that error number for the scope of the block in which it is declared.

5.3 RAISE_APPLICATION_ERROR vs EXCEPTION_INIT

Aspect	RAISE_APPLICATION_ERROR	PRAGMA EXCEPTION_INIT
Purpose	Raises a brand-new, custom application error	Names an existing Oracle internal error so it can be caught specifically
Direction	Sends an error out to the calling environment	Brings an unnamed error in, to be handled inside PL/SQL
Error Number Source	Chosen by the programmer (-20000 to -20999)	An existing Oracle error number (e.g. -2292, -1400)
Typical Use	Enforcing business rules with a clear message	Catching a specific internal error (e.g. constraint violation) by name

Key Takeaways

- ✓ Exceptions fall into three categories: predefined Oracle exceptions, non-predefined (named) Oracle exceptions, and user-defined exceptions.
- ✓ An exception's scope follows normal PL/SQL block-nesting rules: outer exceptions are visible to inner blocks, but inner exceptions are local and go out of scope once the inner block ends.
- ✓ User-defined exceptions are declared with the `EXCEPTION` keyword and must always be raised explicitly with `RAISE` — they are never triggered automatically.
- ✓ An unhandled exception propagates outward through nested blocks and sub-programs until a matching handler is found, or it reaches the calling environment.
- ✓ `RAISE_APPLICATION_ERROR` sends a custom error (number -20000 to -20999 and message) out to the calling environment.
- ✓ `PRAGMA EXCEPTION_INIT` associates a declared exception name with an existing Oracle error number, allowing it to be caught by name instead of via `WHEN OTHERS`.

— *End of Unit 3* —