

Database Systems

Lab11





PL/SQL Exception Handling

Exceptions

- All error handling statements are placed in the EXCEPTION program block
- **Exception handler:** program command that provides information about an error, and suggest correction actions

Predefined Exceptions

Common errors that have been given predefined names that appear instead of error numbers

Error Code	Exception Name	Description
ORA-00001	DUP_VAL_ON_INDEX	Unique constraint violated
ORA-01001	INVALID_CURSOR	Illegal cursor operation
ORA-01403	NO_DATA_FOUND	Query returns no records
ORA-01422	TOO_MANY_ROWS	Query returns more rows than expected
ORA-01476	ZERO_DIVIDE	Division by zero
ORA-01722	INVALID_NUMBER	Invalid numeric conversion
ORA-06502	VALUE_ERROR	Error in arithmetic or numeric function operation

Exception Handler Syntax for Predefined Exceptions

```
WHEN exception1_name THEN
    exception handling statements;
WHEN exception2_name THEN
    exception handling statements;
...
WHEN OTHERS THEN
    exception handling statements;
```

Undefined Exceptions

- ❑ Less-common errors that have not been given predefined names
- ❑ ORA- error code appears
- ❑ Exception handler tests for ORA- error code and provides alternate error message

User-Defined Exceptions

- ❑ Errors that will not cause a run-time error, but will violate business rules
(i.e. they are created for logical errors)
- ❑ Programmer creates a custom error message

Example of a User-Defined Exception

Declare

```
huge_quantity      exception; -- Declaration of the exception
V_qty      number(10);
v_msg      varchar2(100);
```

Begin

```
v_qty := &V_qty; -- Here this value is requested from the user
if v_qty > 1000 then
    v_msg := 'very huge quantity';
    raise huge_quantity; -- Raising the exception huge_quantity
else
    v_msg := 'Good';
end if;
dbms_output.put_line (v_msg);
```

Exception

```
when huge_quantity then -- Handling of exception huge_quantity
    dbms_output.put_line (v_msg);
```

End;

Using an error number for a User-Defined Exception

- Oracle provides the numbers from -20000 to -20999 to User-Defined Exceptions.
- In the previous example, you can handle the huge_quantity exception using an error number.

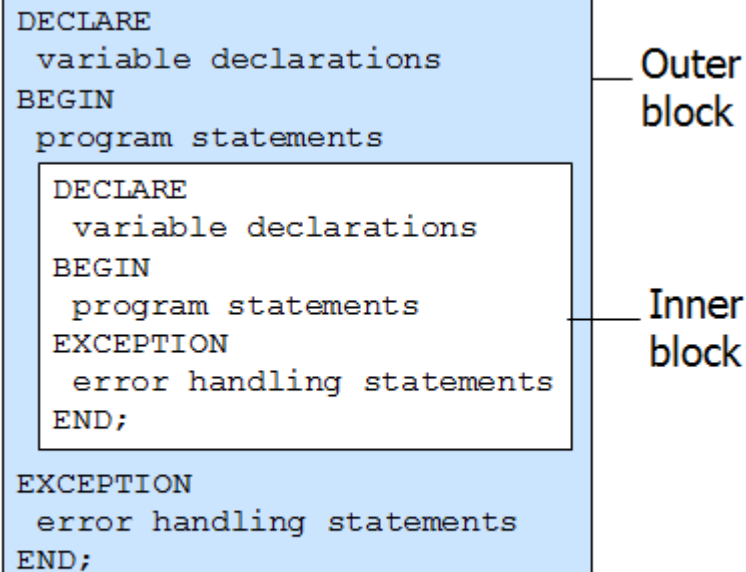
Exception

```
when huge_quantity then  
    raise_application_error (-20100, v_msg);
```

```
End;
```

Nested PL/SQL Program Blocks

- An inner program block can be nested within an outer program block



The diagram illustrates nested PL/SQL blocks. It consists of two nested rectangles. The outer rectangle is light blue and represents the 'Outer block'. It contains the following code: `DECLARE`, `variable declarations`, `BEGIN`, `program statements`, and `EXCEPTION` followed by `error handling statements` and `END;`. The inner rectangle is white with a black border and represents the 'Inner block'. It is nested within the 'BEGIN' and 'program statements' section of the outer block. It contains the following code: `DECLARE`, `variable declarations`, `BEGIN`, `program statements`, `EXCEPTION` followed by `error handling statements`, and `END;`. Labels 'Outer block' and 'Inner block' are placed to the right of their respective rectangles, with lines pointing to them.

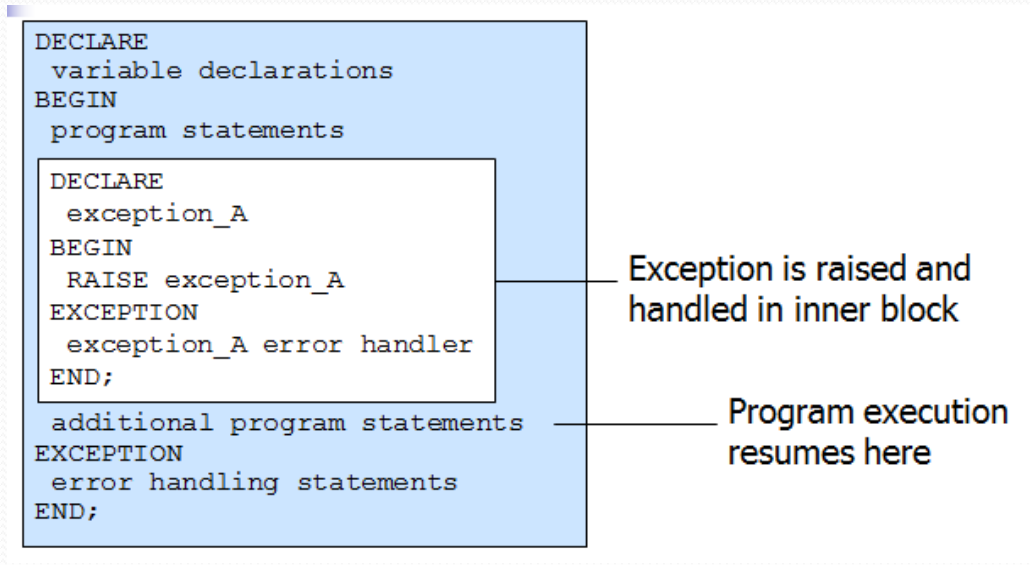
```
DECLARE
  variable declarations
BEGIN
  program statements
  DECLARE
    variable declarations
  BEGIN
    program statements
  EXCEPTION
    error handling statements
  END;
EXCEPTION
  error handling statements
END;
```

Outer block

Inner block

Exception Handling in Nest Program Blocks

- If an exception is raised and handled in an inner block, program execution resumes in the outer block



- Exceptions raised in inner blocks can be handled by exception handlers in outer blocks



Advanced PL/SQL Programs

Anonymous PL/SQL Programs

- PL/SQL blocks that we have written so far.
- Write code in text editor, execute it in SQL*Plus
- Code can be stored as text in file system
- Program cannot be called by other programs, or executed by other users
- Cannot accept or pass parameter values

Named PL/SQL Programs

- Can be created:
 - Using text editor & executed in SQL*Plus
 - Using Procedure Builder (an application installed within Oracle Developer Package).
- Can be stored:
 - As compiled objects in database
 - As source code libraries in file system
- Can be called by other programs
- Can be executed by other users

Named Program Locations

- Server-side
 - Stored in database as database objects
 - Execute on the database server
- Client-side
 - Stored in the client workstation file system
 - Execute on the client workstation

Named Program Types

- Program Units
 - Procedures
 - Functions
- Packages
- Triggers

Program Units

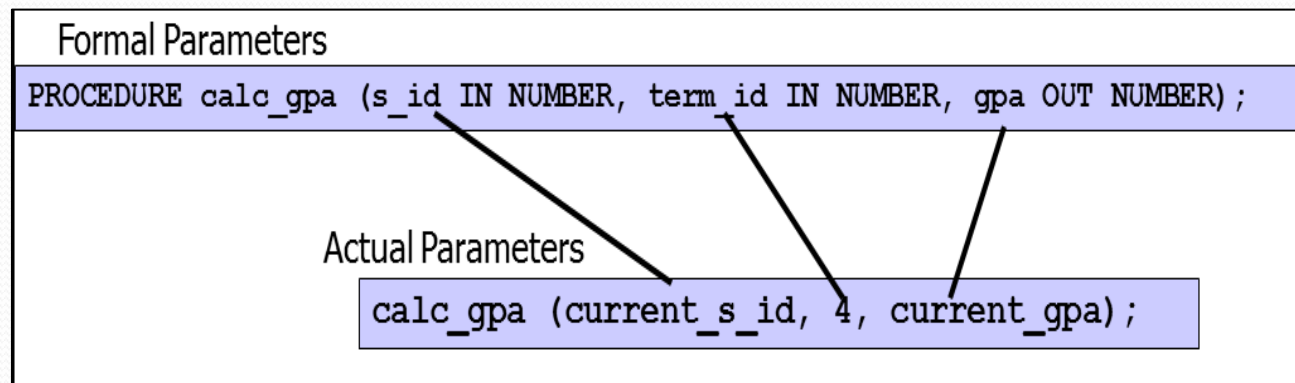
- Procedures
 - Can receive and pass multiple parameter values
 - Can call other program units
- Functions
 - Like procedures, except they return a single value

Parameters

- ❑ Variables used to pass data values in/out of program units
- ❑ Declared in the procedure/function header
- ❑ Parameter values are passed when the procedure/function is called from the calling program

Parameter Types

- **Formal parameters:** declared in procedure header
- **Actual parameters:** values placed in parameter list when procedure is called
- Values correspond based on order



Parameter Modes

- IN
 - Incoming values, read-only (default)
- OUT
 - Outgoing values, write-only
- IN OUT
 - Can be both incoming and outgoing

Procedures

□ Creating a procedure

```
CREATE OR REPLACE PROCEDURE procedure_name
    (parameter1 mode datatype,
     parameter2 mode datatype, ...
    ) IS | AS
    local variable declarations
BEGIN
    program statements
EXCEPTION
    exception handlers
END;
```

header

body

Procedures

- Executing a Procedure (in SQLPlus)

```
EXECUTE procedure_name  
      (parameter1_value, parameter2_value, ...);
```

- Calling a Procedure from another Procedure or Function

```
procedure_name  
      (parameter1_value, parameter2_value, ...);
```

Procedures

- **Dropping a Procedure**

```
DROP PROCEDURE proc_name
```

Functions

```
CREATE OR REPLACE FUNCTION function_name
  (parameter1 mode datatype,
   parameter2 mode datatype, ...
  )
RETURN function_return_data_type
IS local variable declarations
BEGIN
  program statements
  RETURN return_value;
EXCEPTION
  exception handlers
  RETURN EXCEPTION_NOTICE;
END;
```

header

body

Functions

- **Function Syntax Details**

- RETURN command in header specifies data type of value the function will return
- RETURN command in body specifies actual value returned by function

Functions

□ Calling a Function

- Can be called from either named or anonymous PL/SQL blocks
- Can be called within SQL queries
- **Note:** return_value should be a declared variable.

```
return_value :=  
    function_name(parameter1_value,  
    parameter2_value, ...);
```

Example1

- Create a procedure that prints all employees for a given department number.

```
CREATE OR REPLACE PROCEDURE Get_emp_names (V_dno IN NUMBER)
IS
    Emp_name    VARCHAR2(30);
    CURSOR c1 IS
        SELECT fname FROM employee
        WHERE dno = v_dno;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO Emp_name;
        EXIT WHEN c1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(Emp_name);
    END LOOP;
    CLOSE c1;
END;
```

Example1

□ Execution

```
SQL> set serveroutput on;  
SQL> exec Get_emp_names (5);  
James  
Franklin  
JOHN  
  
PL/SQL procedure successfully completed.  
  
SQL>
```

Example2

- Assuming that the salary field in table employee stores the annual salary of an employee, create a function that returns the monthly salary of a given employee.

```
CREATE OR REPLACE FUNCTION Mon_Sal (v_ssn employee.ssn%type)
RETURN NUMBER
IS
    Monthly_sal NUMBER(10,2);
BEGIN
    SELECT round (salary/12)
    INTO Monthly_sal
    FROM Employee
    WHERE ssn = v_ssn;
    RETURN (Monthly_sal);
END;
```

Example2

□ Execution

```
SQL> select Mon_Sal (<'123456789'>
      2  from dual;

MON_SAL<'123456789'>
-----
                2500
```