https://docs.oracle.com/database/121/LNPLS/toc.htm

Blocks

PL/SQL Block Structure

```
<< label >> (optional)
DECLARE    -- Declarative part (optional)
  -- Declarations of local types, variables, & subprograms
BEGIN      -- Executable part (required)
  -- Statements (which can use items declared in declarative part)
[EXCEPTION -- Exception-handling part (optional)
  -- Exception handlers for exceptions (errors) raised in executable part]
END;
```

Processing a Query Result Set One Row at a Time

Processing Query Result Rows One at a Time

```
BEGIN
  FOR someone IN (
    SELECT * FROM employees
    WHERE employee_id < 120
    ORDER BY employee_id
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE('First name = ' || someone.first_name ||
                         ', Last name = ' || someone.last_name);
  END LOOP;
END;
/
```

Using the %ROWTYPE Attribute

The `%ROWTYPE` attribute lets you declare a record that represents either a full or partial row of a database table or view.

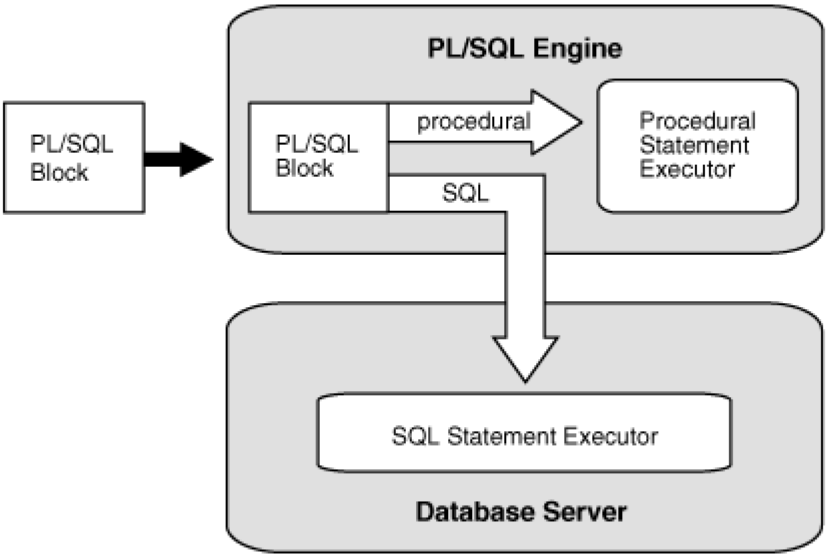
Using the %TYPE Attribute

The `%TYPE` attribute lets you declare a data item of the same data type as a previously declared variable or column (without knowing what that type is).

Architecture of PL/SQL

PL/SQL Engine

Figure 1-1 PL/SQL Engine



PL/SQL Units and Compilation Parameters

PL/SQL units are affected by PL/SQL compilation parameters (a category of database initialization parameters). Different PL/SQL units—for example, a package specification and its body—can have different compilation parameter settings.

A PL/SQL unit is one of these:

- PL/SQL anonymous block
- `FUNCTION`
- `LIBRARY`
- `PACKAGE`
- `PACKAGE BODY`
- `PROCEDURE`
- `TRIGGER`
- `TYPE`
- `TYPE BODY`

PL/SQL Language Fundamentals

Single-Line Comments

```
DECLARE
  howmany     NUMBER;
  num_tables  NUMBER;
BEGIN
  -- Begin processing
  SELECT COUNT(*) INTO howmany
  FROM USER_OBJECTS
  WHERE OBJECT_TYPE = 'TABLE'; -- Check number of tables
  num_tables := howmany;       -- Compute another value
END;
/
```

Multiline Comments

```
/*
  IF 2 + 2 = 4 THEN
    some_condition := TRUE;
  -- We expect this THEN to always be performed
  END IF;
*/
```

```
DECLARE
  some_condition  BOOLEAN;
  pi              NUMBER := 3.1415926;
  radius          NUMBER := 15;
  area            NUMBER;
BEGIN
  /* Perform some simple tests and assignments */
  IF 2 + 2 = 4 THEN
    some_condition := TRUE;
  /* We expect this THEN to always be performed */
  END IF;
  /* This line computes the area of a circle using pi,
  which is the ratio between the circumference and diameter.
  After the area is computed, the result is displayed. */
  area := pi * radius**2;
  DBMS_OUTPUT.PUT_LINE('The area is: ' || TO_CHAR(area));
END;
/
```

Declarations

Variable Declaration with NOT NULL Constraint

```
DECLARE
  acct_id INTEGER(4) NOT NULL := 9999;
  a NATURALN                  := 9999;
  b POSITIVEN                 := 9999;
  c SIMPLE_INTEGER            := 9999;
BEGIN
  NULL;
END;
/
```

Scalar Variable Declarations

```
DECLARE
  part_number       NUMBER(6);     -- SQL data type
  part_name         VARCHAR2(20);  -- SQL data type
  in_stock          BOOLEAN;       -- PL/SQL-only data type
  part_price        NUMBER(6,2);   -- SQL data type
  part_description  VARCHAR2(50);  -- SQL data type
BEGIN
  NULL;
END;
/
```

Constant Declarations

```
DECLARE
  credit_limit     CONSTANT REAL    := 5000.00;  -- SQL data type
  max_days_in_year CONSTANT INTEGER := 366;      -- SQL data type
  urban_legend     CONSTANT BOOLEAN := FALSE;    -- PL/SQL-only data type
BEGIN
  NULL;
END;
/
```

Variable and Constant Declarations with Initial Values

```
DECLARE
  hours_worked    INTEGER := 40;
  employee_count  INTEGER := 0;
  pi     CONSTANT REAL := 3.14159;
  radius          REAL := 1;
  area            REAL := (pi * radius**2);
BEGIN
  NULL;
END;
/
```

Variable Initialized to NULL by Default

```
DECLARE
  counter INTEGER;  -- initial value is NULL by default
BEGIN
  counter := counter + 1;  -- NULL + 1 is still NULL
  IF counter IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('counter is NULL.');
  END IF;
END;
/
```

Declaring Variable of Same Type as Column

```
DECLARE
  surname  employees.last_name%TYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE('surname=' || surname);
END;
/
```

Declaring Variable of Same Type as Another Variable

```
DECLARE
  name     VARCHAR(25) NOT NULL := 'Smith';
  surname  name%TYPE := 'Jones';
BEGIN
  DBMS_OUTPUT.PUT_LINE('name=' || name);
  DBMS_OUTPUT.PUT_LINE('surname=' || surname);
END;
/
```

Scope and Visibility of Identifiers

```
-- Outer block:
DECLARE
  a CHAR;  -- Scope of a (CHAR) begins
  b REAL;    -- Scope of b begins
BEGIN
  -- Visible: a (CHAR), b

  -- First sub-block:
  DECLARE
    a INTEGER;  -- Scope of a (INTEGER) begins
    c REAL;       -- Scope of c begins
  BEGIN
    -- Visible: a (INTEGER), b, c
    NULL;
  END;          -- Scopes of a (INTEGER) and c end

  -- Second sub-block:
  DECLARE
    d REAL;     -- Scope of d begins
  BEGIN
    -- Visible: a (CHAR), b, d
    NULL;
  END;          -- Scope of d ends

-- Visible: a (CHAR), b
END;            -- Scopes of a (CHAR) and b end
/
```

Qualifying Redeclared Global Identifier with Block Label

```
<<outer>>  -- label
DECLARE
  birthdate DATE := TO_DATE('09-AUG-70', 'DD-MON-YY');
BEGIN
  DECLARE
    birthdate DATE := TO_DATE('29-SEP-70', 'DD-MON-YY');
  BEGIN
    IF birthdate = outer.birthdate THEN
      DBMS_OUTPUT.PUT_LINE ('Same Birthday');
    ELSE
      DBMS_OUTPUT.PUT_LINE ('Different Birthday');
    END IF;
  END;
END;
/
```

Assigning Values to Variables

Assigning Values to Variables with Assignment Statement

```
DECLARE  -- You can assign initial values here
  wages          NUMBER;
  hours_worked   NUMBER := 40;
  hourly_salary  NUMBER := 22.50;
  bonus          NUMBER := 150;
  country        VARCHAR2(128);
  counter        NUMBER := 0;
  done           BOOLEAN;
  valid_id       BOOLEAN;
  emp_rec1       employees%ROWTYPE;
  emp_rec2       employees%ROWTYPE;
  TYPE commissions IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  comm_tab       commissions;

BEGIN  -- You can assign values here too
  wages := (hours_worked * hourly_salary) + bonus;
  country := 'France';
  country := UPPER('Canada');
  done := (counter > 100);
  valid_id := TRUE;
  emp_rec1.first_name := 'Antonio';
  emp_rec1.last_name := 'Ortiz';
  emp_rec1 := emp_rec2;
  comm_tab(5) := 20000 * 0.15;
END;
/
```

Assigning Value to Variable with SELECT INTO Statement

```
DECLARE
  bonus   NUMBER(8,2);
BEGIN
  SELECT salary * 0.10 INTO bonus
  FROM employees
  WHERE employee_id = 100;
END;

DBMS_OUTPUT.PUT_LINE('bonus = ' || TO_CHAR(bonus));
/
```

Assigning Value to BOOLEAN Variable

```
DECLARE
  done    BOOLEAN;              -- Initial value is NULL by default
  counter NUMBER := 0;
BEGIN
  done := FALSE;                -- Assign literal value
  WHILE done != TRUE            -- Compare to literal value
    LOOP
      counter := counter + 1;
      done := (counter > 500);  -- Assign value of BOOLEAN expression
    END LOOP;
END;
/
```

Expressions

Concatenation Operator

```
DECLARE
  x VARCHAR2(4) := 'suit';
  y VARCHAR2(4) := 'case';
BEGIN
  DBMS_OUTPUT.PUT_LINE (x || y);
END;
/
```

Concatenation Operator with NULL Operands

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('apple' || NULL || NULL || 'sauce');
END;
/
```

Controlling Evaluation Order with Parentheses

```
DECLARE
  a INTEGER := 1+2**2;
  b INTEGER := (1+2)**2;
BEGIN
  DBMS_OUTPUT.PUT_LINE('a = ' || TO_CHAR(a));
  DBMS_OUTPUT.PUT_LINE('b = ' || TO_CHAR(b));
END;
/
```

CHAR and VARCHAR2 Blank-Padding Difference

```
DECLARE
  first_name  CHAR(10 CHAR);
  last_name   VARCHAR2(10 CHAR);
BEGIN
  first_name := 'John ';
  last_name  := 'Chen ';

  DBMS_OUTPUT.PUT_LINE('*' || first_name || '*');
  DBMS_OUTPUT.PUT_LINE('*' || last_name || '*');
END;
/
```

PLS_INTEGER Calculation Raises Overflow Exception

```
DECLARE
  p1 PLS_INTEGER := 2147483647;
  p2 PLS_INTEGER := 1;
  n NUMBER;
BEGIN
  n := p1 + p2;
END;
/
```

Preventing Overflow

```
DECLARE
  p1 PLS_INTEGER := 2147483647;
  p2 INTEGER := 1;
  n NUMBER;
BEGIN
  n := p1 + p2;
END;
/
```

Predefined Subtypes of PLS_INTEGER Data Type

Data Type	Data Description
`NATURAL`	Nonnegative `PLS_INTEGER` value
`NATURALN`	Nonnegative `PLS_INTEGER` value with `NOT NULL` constraint
`POSITIVE`	Positive `PLS_INTEGER` value
`POSITIVEN`	Positive `PLS_INTEGER` value with `NOT NULL` constraint
`SIGNTYPE`	`PLS_INTEGER` value -1, 0, or 1 (useful for programming tri-state logic)
`SIMPLE_INTEGER`	`PLS_INTEGER` value with `NOT NULL` constraint.

IF THEN ELSIF Statement Simulates Simple CASE Statement

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := 'B';

  IF grade = 'A' THEN
    DBMS_OUTPUT.PUT_LINE('Excellent');
  ELSIF grade = 'B' THEN
    DBMS_OUTPUT.PUT_LINE('Very Good');
  ELSIF grade = 'C' THEN
    DBMS_OUTPUT.PUT_LINE('Good');
  ELSIF grade = 'D' THEN
    DBMS_OUTPUT. PUT_LINE('Fair');
  ELSIF grade = 'F' THEN
    DBMS_OUTPUT.PUT_LINE('Poor');
  ELSE
    DBMS_OUTPUT.PUT_LINE('No such grade');
  END IF;
END;
/
```

Simple CASE Statement

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := 'B';

  CASE grade
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
    WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
    ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
  END CASE;
END;
/
```

Searched CASE Statement

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := 'B';

  CASE
    WHEN grade = 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN grade = 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
    WHEN grade = 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
    WHEN grade = 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
    WHEN grade = 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
    ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
  END CASE;
END;
/
```

Basic LOOP Statement with EXIT Statement

```
DECLARE
  x NUMBER := 0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Inside loop:  x = ' || TO_CHAR(x));
    x := x + 1;
    IF x > 3 THEN
      EXIT;
    END IF;
  END LOOP;
  -- After EXIT, control resumes here
  DBMS_OUTPUT.PUT_LINE(' After loop:  x = ' || TO_CHAR(x));
END;
/
```

Basic LOOP Statement with EXIT WHEN Statement

```
DECLARE
  x NUMBER := 0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('Inside loop:  x = ' || TO_CHAR(x));
    x := x + 1;  -- prevents infinite loop
    EXIT WHEN x > 3;
  END LOOP;
  -- After EXIT statement, control resumes here
  DBMS_OUTPUT.PUT_LINE('After loop:  x = ' || TO_CHAR(x));
END;
/
```

Nested, Labeled Basic LOOP Statements with EXIT WHEN Statements

```
DECLARE
  s  PLS_INTEGER := 0;
  i  PLS_INTEGER := 0;
  j  PLS_INTEGER;
BEGIN
  <<outer_loop>>
  LOOP
    i := i + 1;
    j := 0;
    <<inner_loop>>
    LOOP
      j := j + 1;
      s := s + i * j; -- Sum several products
      EXIT inner_loop WHEN (j > 5);
      EXIT outer_loop WHEN ((i * j) > 15);
    END LOOP inner_loop;
  END LOOP outer_loop;
  DBMS_OUTPUT.PUT_LINE
    ('The sum of products equals: ' || TO_CHAR(s));
END;
/
```

Nested, Unabeled Basic LOOP Statements with EXIT WHEN Statements

```
DECLARE
  i PLS_INTEGER := 0;
  j PLS_INTEGER := 0;
BEGIN
  LOOP
    i := i + 1;
    DBMS_OUTPUT.PUT_LINE ('i = ' || i);

    LOOP
      j := j + 1;
      DBMS_OUTPUT.PUT_LINE ('j = ' || j);
      EXIT WHEN (j > 3);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE ('Exited inner loop');

    EXIT WHEN (i > 2);
  END LOOP;

  DBMS_OUTPUT.PUT_LINE ('Exited outer loop');
END;
/
```

FOR LOOP Statements

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('lower_bound < upper_bound');
  FOR i IN 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
/
```

Reverse FOR LOOP Statements

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('upper_bound > lower_bound');
  FOR i IN REVERSE 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
/
```

WHILE LOOP Statements

```
DECLARE
  done  BOOLEAN := FALSE;
BEGIN
  WHILE done LOOP
    DBMS_OUTPUT.PUT_LINE ('This line does not print.');
    done := TRUE;  -- This assignment is not made.
  END LOOP;

  WHILE NOT done LOOP
    DBMS_OUTPUT.PUT_LINE ('Hello, world!');
    done := TRUE;
  END LOOP;
END;
/
```

GOTO Statement

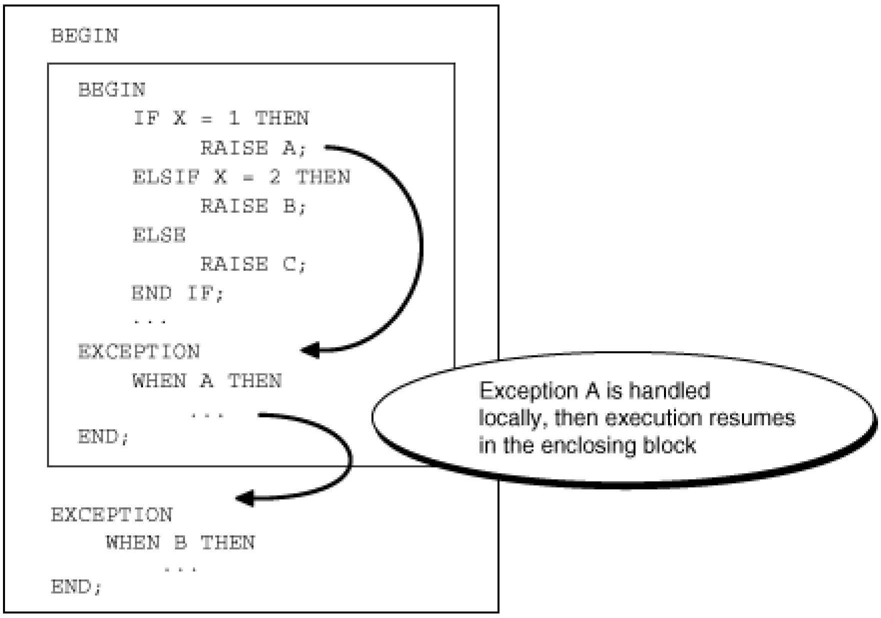
```
DECLARE
  p  VARCHAR2(30);
  n  PLS_INTEGER := 37;
BEGIN
  FOR j in 2..ROUND(SQRT(n)) LOOP
    IF n MOD j = 0 THEN
```

```
      p := ' is not a prime number';
      GOTO print_now;
    END IF;
  END LOOP;

  p := ' is a prime number';

  <<print_now>>
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);
END;
/
```

PL/SQL Error Handling

```
EXCEPTION
  WHEN ex_name_1 THEN statements_1                 -- Exception handler
  WHEN ex_name_2 OR ex_name_3 THEN statements_2  -- Exception handler
  WHEN OTHERS THEN statements_3                      -- Exception handler
END;
```

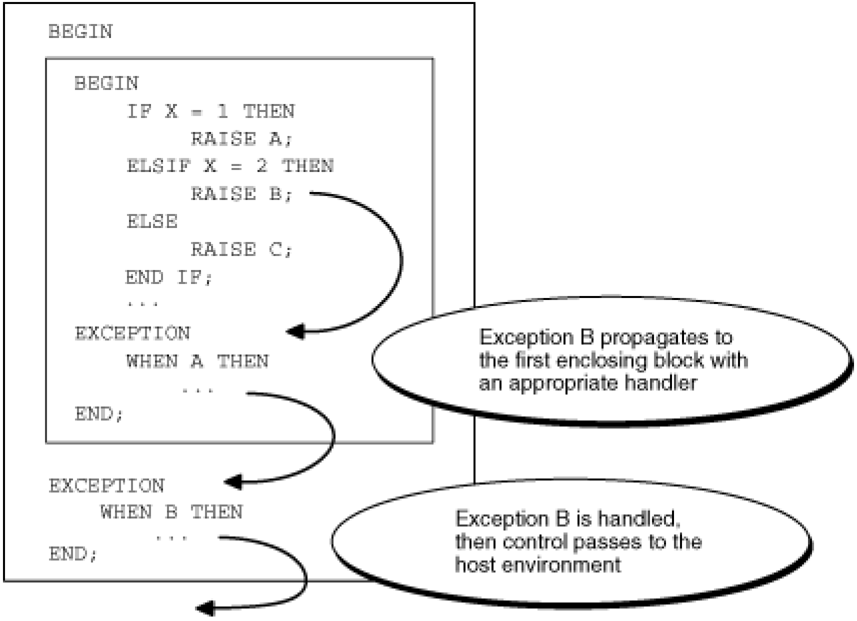
Exception Does Not Propagate



```
BEGIN
  BEGIN
     IF X = 1 THEN
          RAISE A;
     ELSIF X = 2 THEN
          RAISE B;
     ELSE
          RAISE C;
     END IF;
     ...
  EXCEPTION
     WHEN A THEN
          ...
  END;
EXCEPTION
    WHEN B THEN
       ...
END;
```

Exception A is handled locally, then execution resumes in the enclosing block

Exception Propagates from Inner Block to Outer Block

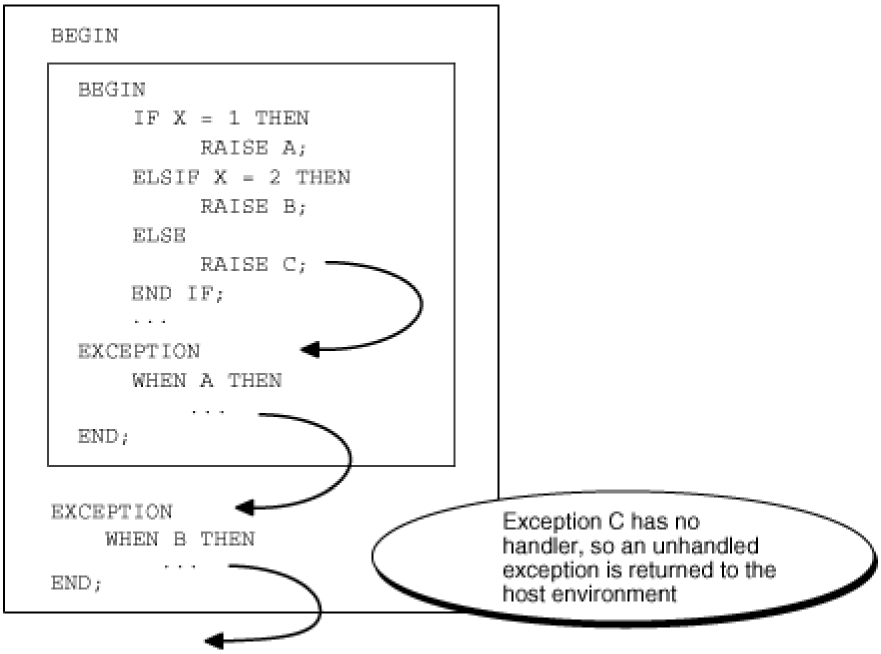


```
BEGIN
  BEGIN
     IF X = 1 THEN
          RAISE A;
     ELSIF X = 2 THEN
          RAISE B;
     ELSE
          RAISE C;
     END IF;
     ...
  EXCEPTION
     WHEN A THEN
          ...
  END;
EXCEPTION
    WHEN B THEN
       ...
END;
```

Exception B propagates to the first enclosing block with an appropriate handler

Exception B is handled, then control passes to the host environment

PL/SQL Returns Unhandled Exception Error to Host Environment



```
BEGIN
  BEGIN
      IF X = 1 THEN
           RAISE A;
      ELSIF X = 2 THEN
           RAISE B;
      ELSE
           RAISE C;
      END IF;
      ...
  EXCEPTION
      WHEN A THEN
           ...
  END;

EXCEPTION
    WHEN B THEN
       ...
END;
```

Exception C has no handler, so an unhandled exception is returned to the host environment

PL/SQL Predefined Exceptions

Exception Name	Error Code
ACCESS_INTO_NULL	-6530
CASE_NOT_FOUND	-6592
COLLECTION_IS_NULL	-6531
CURSOR_ALREADY_OPEN	-6511
DUP_VAL_ON_INDEX	-1
INVALID_CURSOR	-1001
INVALID_NUMBER	-1722
LOGIN_DENIED	-1017
NO_DATA_FOUND	+100
NO_DATA_NEEDED	-6548
NOT_LOGGED_ON	-1012
PROGRAM_ERROR	-6501
ROWTYPE_MISMATCH	-6504
SELF_IS_NULL	-30625
STORAGE_ERROR	-6500
SUBSCRIPT_BEYOND_COUNT	-6533
SUBSCRIPT_OUTSIDE_LIMIT	-6532
SYS_INVALID_ROWID	-1410
TIMEOUT_ON_RESOURCE	-51
TOO_MANY_ROWS	-1422
VALUE_ERROR	-6502
ZERO_DIVIDE	-1476

Anonymous Block Handles ZERO_DIVIDE

```
DECLARE
  stock_price   NUMBER := 9.73;
  net_earnings  NUMBER := 0;
  pe_ratio      NUMBER;
BEGIN
  pe_ratio := stock_price / net_earnings;  -- raises ZERO_DIVIDE exception
  DBMS_OUTPUT.PUT_LINE('Price/earnings ratio = ' || pe_ratio);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Company had zero earnings.');
    pe_ratio := NULL;
END;
/
```

Redeclared Predefined Identifier

```
DROP TABLE t;
CREATE TABLE t (c NUMBER);
```

```
DECLARE
  default_number NUMBER := 0;
BEGIN
  INSERT INTO t VALUES(TO_NUMBER('100.00', '9G999'));
EXCEPTION
  WHEN INVALID_NUMBER THEN
    DBMS_OUTPUT.PUT_LINE('Substituting default value for invalid number.');
    INSERT INTO t VALUES(default_number);
END;
/
```

Exception that Propagates Beyond Scope is Handled

```
BEGIN

  DECLARE
    past_due     EXCEPTION;
    PRAGMA EXCEPTION_INIT (past_due, -4910);
    due_date     DATE := trunc(SYSDATE) - 1;
    todays_date  DATE := trunc(SYSDATE);
  BEGIN
    IF due_date < todays_date THEN
      RAISE past_due;
    END IF;
  END;

EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    RAISE;
END;
/
```

Exception that Propagates Beyond Scope is Not Handled

```
BEGIN

  DECLARE
    past_due     EXCEPTION;
    due_date     DATE := trunc(SYSDATE) - 1;
    todays_date  DATE := trunc(SYSDATE);
  BEGIN
    IF due_date < todays_date THEN
      RAISE past_due;
    END IF;
  END;

END;
/
```

Reraising Exception

```
DECLARE
  salary_too_high   EXCEPTION;
  current_salary    NUMBER := 20000;
  max_salary        NUMBER := 10000;
  erroneous_salary  NUMBER;
BEGIN
  BEGIN
    IF current_salary > max_salary THEN
      RAISE salary_too_high;   -- raise exception
    END IF;
  EXCEPTION
    WHEN salary_too_high THEN  -- start handling exception
      erroneous_salary := current_salary;
      DBMS_OUTPUT.PUT_LINE('Salary ' || erroneous_salary ||' is out of range.');
      DBMS_OUTPUT.PUT_LINE ('Maximum salary is ' || max_salary || '.');
      RAISE;  -- reraise current exception (exception name is optional)
  END;

EXCEPTION
  WHEN salary_too_high THEN    -- finish handling exception
    current_salary := max_salary;

    DBMS_OUTPUT.PUT_LINE (
      'Revising salary from ' || erroneous_salary ||
      ' to ' || current_salary || '.'
    );
END;
/
```

Exception Raised in Declaration is Not Handled

```
DECLARE
  credit_limit CONSTANT NUMBER(3) := 5000;  -- Maximum value is 999
BEGIN
  NULL;
EXCEPTION
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('Exception raised in declaration.');
END;
/
```

Exception Raised in Declaration is Handled by Enclosing Block

```
BEGIN

  DECLARE
    credit_limit CONSTANT NUMBER(3) := 5000;
  BEGIN
    NULL;
  END;

EXCEPTION
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('Exception raised in declaration.');
END;
/
```

Retrieving Error Code and Error Message

```
DECLARE
  stock_price   NUMBER := 9.73;
  net_earnings  NUMBER := 0;
  pe_ratio      NUMBER;
BEGIN
  pe_ratio := stock_price / net_earnings;  -- raises ZERO_DIVIDE exception
  DBMS_OUTPUT.PUT_LINE('Price/earnings ratio = ' || pe_ratio);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Company had zero earnings.');
    pe_ratio := NULL;
    DBMS_OUTPUT.PUT_LINE('Error code '||SQLCODE||': '||SUBSTR(SQLERRM,1,64));
END;
/
```

Write errors to the table ERRORS

```
CREATE TABLE errors (
  code      NUMBER,
  message   VARCHAR2(64)
);
```

```
DECLARE
  stock_price   NUMBER := 9.73;
  net_earnings  NUMBER := 0;
  pe_ratio      NUMBER;
  v_code  NUMBER;
  v_errm  VARCHAR2(64);
BEGIN
  pe_ratio := stock_price / net_earnings;  -- raises ZERO_DIVIDE exception
  DBMS_OUTPUT.PUT_LINE('Price/earnings ratio = ' || pe_ratio);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Company had zero earnings.');
    pe_ratio := NULL;
    v_code := SQLCODE;
    v_errm := SUBSTR(SQLERRM, 1, 64);
    DBMS_OUTPUT.PUT_LINE ('Error code ' || v_code || ': ' || v_errm);
    INSERT INTO errors (code, message) VALUES (v_code, v_errm );
    COMMIT;   END;
/
```