Bài thực hành số 3

PL/SQL

❖ *Tóm tắt nội dung:*

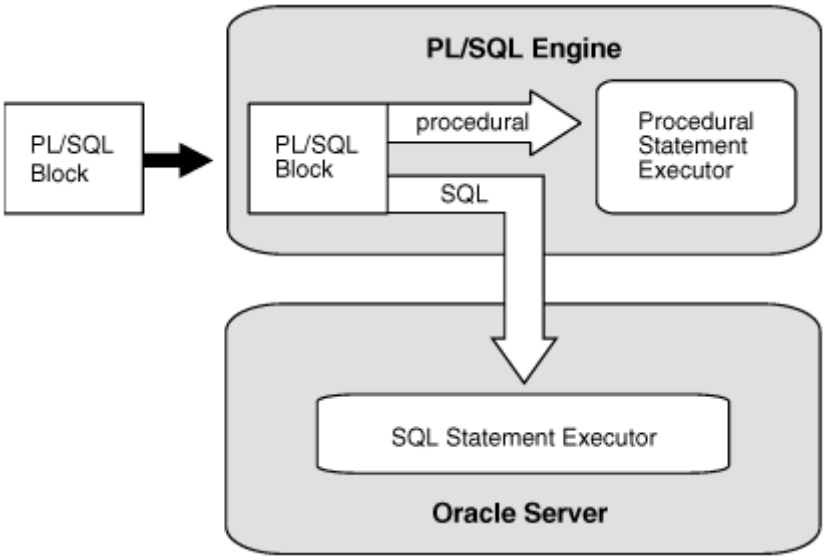
- Khái niệm PL/SQL
- Các vấn đề liên quan đến kiểu dữ liệu trong PL/SQL
- Hằng và Biến
- Cấu trúc khối PL/SQL
- Các câu lệnh điều khiển
- Xử lý ngoại lệ
- Procedure và Function
- Cursor
- Trigger

I. PL/SQL là gì ?

PL/SQL (PL : Procedural Language – Ngôn ngữ Thủ tục) là một mở rộng của SQL, kết hợp vào trong đó rất nhiều đặc tính của các ngôn ngữ lập trình gần đây. Nó cho phép các thao tác dữ liệu và các câu lệnh query SQL bao gồm các đoạn mã có cấu trúc khối và tính thủ tục (block-structure and procedural unit of code), làm cho PL/SQL thành một ngôn ngữ xử lý giao dịch mạnh mẽ.

II. Các lệnh SQL trong PL/SQL

- PL/SQL cung cấp một số câu lệnh thủ tục cho việc thao tác và kiểm tra dữ liệu, thường không cần phải dính dáng với các lệnh SQL. Dù vậy, khi cần lấy thống tin từ CSDL hoặc thay đổi trên CSDL thì nên dùng SQL.
- PL/SQL hỗ trợ tốt cho đa số các lệnh DML và các lệnh điều khiển giao dịch trong SQL. Ngoài ra, các câu lệnh SELECT có thể dùng để gán các giá trị query từ 1 hàng trong bảng cho các biến.



- Một số điểm lưu ý:
  - ✓ Một khối PL/SQL không phải là một đơn vị giao dịch (transaction unit) – các lệnh COMMIT và ROLLBACK là độc lập với các khối nhưng có thể nằm trong nó.
  - ✓ Mỗi câu lệnh SQL cần phải kết thúc bởi dấu chấm phẩy.
  - ✓ Câu lệnh SELECT có thể dùng để gán các giá trị query từ 1 hàng trong bảng cho các biến.
  - ✓ Các câu lệnh SELECT mà không trả lại **đúng một hàng** sẽ gây ra một lỗi cần phải giải quyết (thường là phải dùng phương pháp xử lý ngoại lệ hoặc cursor).
  - ✓ Các lệnh DDL không dùng được trong PL/SQL. Ví dụ:
    - Tất cả các lệnh bắt đầu bằng ALTER, CREATE, DROP, FLASHBACK
    - Các lệnh quản lý quyền: GRANT, REVOKE
    - Các lệnh audit: AUDIT, NOAUDIT

      (và còn nhiều lệnh khác)
  - ✓ Các lệnh DML có thể xử lý nhiều hàng (multiple rows).

III. Kiểu dữ liệu

- PL/SQL hỗ trợ rất nhiều kiểu dữ liệu để có thể khai báo các biến và các hằng. Có thể gán một giá trị ban đầu cho các biến khi khai báo biến và có thể thay đổi các giá trị của chúng thông qua các phát biểu gán về sau trong khối. Các hằng là các danh hiệu (identifier) lưu giữ một giá trị cố định và giá trị này phải được gán cho hằng khi hằng được khai báo.
- Các kiểu dữ liệu:

- ✓ Dữ liệu số: **NUMBER**

  Ví dụ: NUMBER(7,2)

  Nghĩa là có 7 ký số trong đó có 2 ký số sau dấu thập phân. Nếu ta không khai báo độ chính xác là 2 như câu lệnh trên thì độ chính xác mặc định là 38 ký số.
- ✓ Dữ liệu luận lí: **BOOLEAN**
- ✓ Dữ liệu ngày tháng: **DATE**
- ✓ Dữ liệu chuỗi:

VARCHAR2	Lưu trữ các dữ liệu ký tự có chiều dài thay đổi. Chiều dài mặc định là 1 ký tự. Chiều dài tối đa là 32767. Ví dụ: `VARCHAR2(30)`
CHAR	• PL/SQL Version 1: giống như VARCHAR2 nhưng chiều dài tối đa là 255. • PL/SQL Version 2: chuỗi các ký tự chiều dài cố định dài tối đa là 32767 byte. Khi so sánh hai chuỗi với nhau thì các ký tự trống sẽ được thêm vào. • *Chú ý:* Khi so sánh 2 chuỗi CHAR trong PL/SQL Version 1 thì hai chuỗi này không được thêm vào các ký tự trống, ví dụ một biến kiểu CHAR chứa 'FRED' thì khác với một biến kiểu CHAR chứa 'FRED '.

IV. Khai báo biến và hằng

1. Khai báo các biến

- Các biến PL/SQL có thể được khai báo và có thể được gán một giá trị ban đầu trong phần DECLARE của khối. Các biến khác được tham khảo đến trong phần khai báo thì chúng phải được khai báo ở trong một phát biểu trước đó.
- *Cú pháp:*

  identifier datatype [(precision, scale)] [NOT NULL] [:= expression];

  trong đó

  identifier - tên biến

  datatype - kiểu dữ liệu của biến

  precision - chiều dài của biến (số ký số của phần nguyên và phần thập phân)

scale - số số lẻ (số ký số của phần thập phân)

- Nếu không gán giá trị ban đầu cho biến thì biến sẽ chứa giá trị NULL cho đến khi gán giá trị mới. Ràng buộc NOT NULL không được dùng trong trường hợp này.

  Ví dụ:

```
v_count        NUMBER NOT NULL := 0;
v_saraly       NUMBER(7,2);
v_annsal       NUMBER(9,2) := month_sal * 12;
-- month_sal phải tồn tại trước
postcost       CHAR(7);
surname        VARCHAR2(25) := 'Skywalker';
v_message      VARCHAR2(80) := 'Data is wrong !';
married        BOOLEAN := FALSE;
today          DATE := SYSDATE;
```

- Không nên đặt tên của biến trùng tên với các tên cột của bảng được dùng trong khối. Nếu các biến trong các phát biểu SQL có cùng tên với tên cột thì Oracle xem tên này là tên cột (mà không phải là tên biến).

  Ví dụ:

```
DECLARE
  bonus  NUMBER(8,2);
  emp_id NUMBER(6) := 100;
BEGIN
  SELECT salary * 0.10 INTO bonus FROM employees
  WHERE employee_id = emp_id;
END;
```

2. Khai báo hằng

- *Cú pháp:*

  identifier CONSTANT datatype [(precision,scale)] := expression;

  Ví dụ:

```
pi    CONSTANT   NUMBER(9,5) := 3.14159;
vat   CONSTANT   NUMBER(4,2) := 17.5;
```

- Chú ý:
  - ✓ Dùng từ khóa %TYPE để khai báo cùng kiểu với cột được chỉ định trong 1 table hoặc view.

  Ví dụ: biến product_type sẽ có cùng kiểu với cột price của bảng products:

```
product_price  products.price%TYPE;
```

  - ✓ Dùng từ khoá %ROWTYPE để khai báo kiểu record đại diện cho 1 hàng trong 1 table hoặc view. Các trường trong record sẽ có cùng tên và cùng kiểu dữ liệu với các cột trong table/view đó.

  Ví dụ:

```
emprec employees_temp%ROWTYPE;
```

V. Các biến kết hợp của SQL*Plus

- SQL*Plus hỗ trợ biến kết hợp (bind variable). Đây là các biến dùng để gửi các giá trị vào trong hay ra ngoài một khối PL/SQL.
- *Cú pháp:*

VARIABLE variable_name [NUMBER | CHAR | CHAR(n) | VARCHAR2 | VARCHAR2(n)]

Ví dụ :

```
VARIABLE deptnum NUMBER;
/*Chúng có thể dùng trong các khối PL/SQL với dấu 2 chấm phía
trước */
BEGIN
 SELECT DEPTNO
   INTO     :deptnum
   FROM     DEPT
   WHERE    DNAME = 'ACCOUNTING';
 INSERT  INTO RESULTS VALUES( :deptnum);
END;
```

Trong ví dụ trên, giá trị DEPTNO được lấy ra và gán cho biến deptnum. Sau đó được ghi vào bảng RESULTS. Sau khi chạy xong khối PL/SQL, bạn có thể hiển thị giá trị của một biến kết hợp bằng lệnh PRINT :

```
SQL> PRINT deptnum
DEPTNUM
--------------------
10
```

VI. Hàm chuyển đổi kiểu

- TO_CHAR
- TO_DATE
- TO_NUMBER

Ví dụ :

```
v_message   VARCHAR2(80) := 'SCOTT earns '
                            || TO_CHAR (month_sal * 12);
```

VII. Độ ưu tiên của toán tử

	Toán tử	Tác vụ
Đầu tiên	**, NOT	Toán tử mũ, phủ định luận lý
	+, -	Đồng nhất, dấu âm
	*, /	Nhân, chia
	+, -, ||	Cộng, trừ, nối chuỗi
	=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	So sánh
	AND	Giao
Cuối cùng	OR	Hội

VIII. Cấu trúc khối

- *Cú pháp:*

[DECLARE

declaration_statements **]**

BEGIN

executable_statements

[EXCEPTION

exception_handling_statements **]**

END;

- DECLARE và EXCEPTION là phần tự chọn, có vài khối không có 2 phần này.

Ví dụ: (ví dụ trong command line)

```
SQL> DECLARE
  2  x NUMBER(7,2);
  3  BEGIN
  4     SELECT sal INTO x FROM emp WHERE empno=123;
  5        IF x<300 THEN
  6              UPDATE emp SET sal=3000
  7              WHERE empno=123;
  8        END IF;
  9  END;
 10  .
```

- Đóng buffer với dấu chấm (.)
- Để chạy PL/SQL trong buffer, gõ lệnh RUN hoặc dấu gạch chéo (/) tại dấu nhắc. Nếu khối được thi hành xong, không có một lỗi không được kiểm soát nào thì chỉ một thông báo được xuất ra :

```
'PL/SQL procedure successfully completed'
```

- Nội dung của buffer có thể soạn thảo theo cách thông thường hay lưu xuống file bằng lệnh SAVE của SQL*Plus.

IX. Lệnh rẽ nhánh

- *Cú pháp:*

IF condition THEN actions

[ELSIF condition THEN actions]

[ELSE actions]

END IF;

trong đó '**actions**' là một hay nhiều câu lệnh PL/SQL hay SQL, mỗi câu kết thúc bởi dấu chấm phẩy. Các 'action' này có thể chứa các câu lệnh IF khác lồng nhau.

Ví dụ:

```
IF count > 0 THEN
  message := 'count is positive';
  IF area > 0 THEN
    message := 'count and area are positive';
  END IF;
ELSIF count = 0 THEN
  message := 'count is zero';
ELSE
  message := 'count is negative';
END IF;
```

X. Vòng lặp

1. Vòng lặp cơ bản

- *Cú pháp:*

  LOOP

  statements

  END LOOP;

- Mỗi lần dòng chương trình gặp phải END LOOP thì quyền điều khiển trả về tại LOOP. Vòng lặp không điều khiển này sẽ lặp mãi mãi nếu trong thân của nó không có các lệnh nhảy ra khỏi nó.
- Một vòng lặp có thể kết thúc từ bên trong nếu dùng câu lệnh EXIT. EXIT cho phép điều khiển chuyển cho câu lệnh kế tiếp ngay sau END LOOP và kết thúc vòng lặp ngay lập tức.

  Cú pháp :

  EXIT [loop-label] [WHEN condition];

- EXIT có thể là một tác vụ nằm trong câu lệnh IF hoặc đứng một mình trong vòng lặp. Khi đứng một mình thì mệnh đề WHEN có thể dùng để kết thúc có điều kiện.

Ví dụ 1:

```
LOOP
      counter := counter + 1;
      INSERT INTO numbered_rows VALUES (counter);
      …
      IF counter = 10 THEN
            COMMIT;
            EXIT;
      END IF;
END LOOP;
```

Ví dụ 2 :

```
LOOP
      …
      EXIT WHEN total_sals = 60000;
      …
END LOOP;
```

- Cách ngắt vòng lặp khác là rẽ nhánh đến một nhãn ra ngoài vòng lặp, đó là dùng lệnh GOTO. Nhưng đây không phải cách viết có cấu trúc.

2. Vòng lặp WHILE

- *Cú pháp :*

  WHILE *condition* LOOP

  statements

  END LOOP;

- Điều kiện (condition) được tính toán tại điểm bắt đầu của vòng lặp và vòng lặp sẽ kết thúc nếu điều kiện này là FALSE. Nếu điều kiện này FALSE ngay tại lúc bắt đầu vào đến vòng lặp thì vòng lặp không xảy ra.

  Ví dụ:

```
counter := 0;
WHILE counter < 6 LOOP
      counter := counter + 1;
END LOOP;
```

3. Vòng lặp FOR

- *Cú pháp:*

  FOR loop_variable IN [REVERSE] lower_bound..upper_bound

  LOOP

  statements

  END LOOP;

Ví dụ:

```
FOR count2 IN 1..5 LOOP
      DBMS_OUTPUT.PUT_LINE(count2);
END LOOP;
```

4. Điều khiển các vòng lặp lồng nhau

- Thông thường, vòng lặp trong kết thúc thì không kết thúc vòng lặp ngoài (ngoại trừ có lỗi). Dù vậy, các vòng lặp có thể gán nhãn và có thể kết thúc vòng lặp ngoài bằng lệnh EXIT.
- Các nhãn trong PL/SQL được định nghĩa như sau :

  << label-name >>

Ví dụ :

```
<<main>> LOOP
            …
            LOOP
            …
            --thoát cả 2 vòng lặp
            EXIT main WHEN total_done='YES';
            --thoát khỏi vòng lặp trong
            EXIT WHEN innder_done='YES';
                  …
            END LOOP;
END LOOP main;
```

- Ngoài ra nhãn còn dùng để định danh vòng lặp khi chúng có cấu trúc lồng nhau.

Ví dụ:

```
<<block1>> DECLARE
var1 NUMBER;
BEGIN
      <<block2>> DECLARE
      var1 NUMBER := 400;
      BEGIN
            -- biến var1 của khối block1 được tăng lên 1
            block1.var1 := block1.var1 + 1;
      END block2;
END block1;
```

XI. Xử lý ngoại lệ

1. Giới thiệu về Exception

- Các Exception là các danh định trong PL/SQL mà có thể gặp phải trong khi thực thi một khối dẫn đến thân chính của các tác vụ sẽ bị kết thúc. Một khối luôn luôn kết thúc khi gặp một exception, nhưng có thể chỉ ra một exception handler để thi hành tác vụ cuối cùng trước khi khối bị kết thúc. Nếu exception được kiểm soát (handled) thì exception sẽ không truyền ra ngoài khối hay ra môi trường. Hai nhóm chính của exception là :
  - ✓ Predefined: đã được định nghĩa trước bởi PL/SQL và dính với các mã lỗi xác định.
  - ✓ User-defined: khai báo trong khối, chỉ thường dùng khi có nhu cầu cụ thể với chúng, ngoài ra có thể gắn chúng với các mã lỗi cần thiết.
- Trong bài này, chúng ta sẽ tập trung vào các exception đã định nghĩa trước:

Tên Exception	Lỗi Oracle
DUP_VAL_ON_INDEX	-1
INVALID_CURSOR	-1001
INVALID_NUMBER	-1722
LOGIN_DINIED	-1017
NO_DATA_FOUND	-1403 (ANSI +100)
NOT_LOGGED_ON	-1012

PROGRAM_ERROR	-6501
STORAGE_ERROR	-6500
TIMEOUT_ON_RESOURCE	-51
TOO_MANY_ROWS	-1422
VALUES_ERROR	-6502
ZERO_DIVIDE	-1476
CURSOR_ALREADY_OPEN	-6511
TRANSACTION_BACKED_OUT	-61

2. Bộ kiểm soát lỗi

- Nếu một exception xảy ra, quyền điều khiển sẽ chuyển cho phần EXCEPTION trong khối mà nó xảy ra. Nếu exception đó không kiểm soát được trong phần này hoặc là không có phần này thì khối sẽ kết thúc với exception unhandled và có thể tác động đến môi trường ngoài.

  Ví dụ:

```
BEGIN
      INSERT INTO dept (deptno, dname)
            VALUES (50, 'CLEANING');
      INSERT INTO dept (deptno, dname)
            VALUES (50, 'TRANING');
      -- Exception DUP_VAL_ON_INDEX xảy ra tại đây
END;
-- Khối sẽ kết thúc với exception unhandled ORA-00001
```

- Để bẫy các sự kiện này và chặn các exception, có thể định nghĩa các exception handler trong phần EXCEPTION.

  Cú pháp:

  WHEN exceptionn-identifier THEN actions;

  Ví dụ :

```
DECLARE
      v_ename     emp.ename%TYPE;
```

```
        v_job       emp.job%TYPE;
BEGIN
        SELECT      ename, job
        INTO        v_name, v_job
        FROM        emp
        WHERE hiredate BETWEEN '01/01/92' AND '31/12/92';
EXCEPTION
        WHEN no_data_found THEN
                INSERT INTO error_tab VALUES ('Nobody in 92');
        WHEN too_many_rows THEN
                INSERT  INTO  error_tab  VALUES  ('More  than  one
                person in 92');
END;
```

- Bộ kiểm soát lỗi 'WHEN OTHERS': có thể dùng định nghĩa này để chặn tất cả các exception còn lại ngoài các exception đã định nghĩa trong phần EXCEPTION. Phần này được đặt cuối cùng trong phần EXCEPTION.

  Ví dụ:

```
BEGIN
        SAVEPOINT  so_far_so_good;
        INSERT INTO statistics_tab VALUES (18, 25, 91);
EXCEPTION
        WHEN dup_val_on_index THEN
                ROLLBACK TO so_far_so_good;
        WHEN OTHERS THEN
                INSERT INTO error_tab
                VALUES ('Error during block');
END;
```

3. Các hàm dùng trong bẫy lỗi

- Khi một exception xảy ra, ta có thể xác định mã lỗi và câu chú của nó. PL/SQL cung cấp 2 hàm:

SQLCODE	Trả về mã lỗi của exception đó. Nếu dùng nó ngoài phần EXCEPTION thì mã trả ra là 0.
SQLERRM	Trả về toàn bộ câu chú lỗi (error message) và có cả mã lỗi.

Ví dụ:

```
DECLARE
      error_message    CHAR (100);
      error_code       NUMBER;
BEGIN
      …
EXCEPTION
      WHEN OTHERS THEN
            error_message := SUBSTR (SQLERRM, 1, 100);
            error_code := SQLCODE;
            INSERT INTO error
            VALUES (error_message, error_code);
END;
```

XII. Procedure

- *Cú pháp:*

  CREATE [OR REPLACE] PROCEDURE *procedure_name*
  [(*parameter_name* [IN | OUT | IN OUT] *datatype*)]
  {IS | AS}
  BEGIN
  procedure_body
  END *procedure_name;*

- **Chú ý:** Datatype là kiểu của tham số, ở đây chỉ khai báo kiểu chứ ko khai báo chiều dài của tham số. Ví dụ không được khai báo tham số là VARCHAR2(10) mà phải khai báo là VARCHAR2.

Ví dụ:

```
CREATE OR REPLACE PROCEDURE update_product_price(
```

```
  p_product_id IN products.product_id%TYPE,
  p_factor IN NUMBER)
AS
  v_product_count INTEGER;
BEGIN
     SELECT COUNT(*)
     INTO v_product_count
     FROM products
     WHERE product_id = p_product_id;
     IF v_product_count = 1 THEN
          UPDATE products
          SET price = price * p_factor
          WHERE product_id = p_product_id;
          COMMIT;
     END IF;
EXCEPTION
     WHEN OTHERS THEN ROLLBACK;
END update_product_price;
```

- Vì procedure cần phải gọi trong khối PL/SQL, nên nếu muốn chạy nó từ dấu nhắc SQL*Plus ta dùng lệnh EXECUTE hoặc lồng nó trong cặp BEGIN-END.

  Ví dụ :

```
SQL> EXECUTE update_product_price(1, 1.5);
```

  Hay có thể

```
SQL> BEGIN
2     update_product_price(1, 1.5);
3     END;
```

XIII. Function

- *Cú pháp:*

  CREATE [OR REPLACE] FUNCTION *function_name*

  [(*parameter_name* [IN | OUT | IN OUT] *datatype*)]

  RETURN *datatype*

{IS | AS}

BEGIN

function_body

END *function_name*;

Ví dụ:

```
create or replace function get_dname( y number)
return varchar2
is
     m char(14);
begin
     select dname
     into m
     from dept
     where deptno=y;
     if SQL%notfound then
          m:='Khong thay';
     end if;
     return (rtrim(m));
end;
```

- Để gọi function ta gọi trực tiếp hoặc thông qua các phép gán.

Ví dụ 1:

```
SQL>  select * from dept where dname=get_dname(10);

   DEPTNO  DNAME          LOC
---------  -------------  -------------
       10  ACCOUNTING     NEW YORK
```

Ví dụ 2:

```
SQL> select get_dname (20) from dual;

GET_DNAME (20)
--------------------------------------------------------
RESEARCH
```

XIV. Cursor

1. Định nghĩa

- Oracle dùng các vùng làm việc gọi là 'các vùng SQL dùng riêng' (private SQL areas) để thi hành các câu lệnh SQL và lưu trữ thông tin của quá trình. Một cursor là một cấu trúc PL/SQL cho phép định danh các vùng này và truy cập đến các thông tin lưu trong nó. Có 2 kiểu cursor :

Implicit Cursors	Được mô tả bởi PL/SQL là ẩn dành cho tất cả các câu lệnh DML và cho các query trả ra đơn hàng (ví dụ lệnh SELECT dùng trực tiếp trong khối).
Explicit Cursors	Mô tả rõ ràng với các danh định trong khối và được thao tác bằng các câu lệnh đặc trưng trong các tác vụ của khối. Các cursor hiện chỉ dành cho các query và cho phép nhiều hàng được xử lý từ query.

2. Explicit cursor có thể điều khiển qua 4 kiểu tác vụ riêng lẻ sau :

DECLARE	Định tên của cursor và cấu trúc của query thực thi trong nó. Tại thời điểm này, query sẽ được phân tích (các cột, bảng, …) nhưng chưa thi hành.
OPEN	Thi hành query ràng buộc các biến có tham khảo đến. Các hàng trả về bởi query gọi là 'active set' và sẵn sàng cho việc lấy dữ liệu.
FETCH	Lấy dữ liệu từ hàng hiện tại vào các biến. Hàng hiện tại là hàng mà cursor đang chỉ đến. Mỗi một lần FETCH, cursor di chuyển con trỏ đến hàng kế tiếp trên active set, như vậy mỗi một lệnh FETCH sẽ truy cập đến các hàng khác nhau trong query.
CLOSE	Hủy bỏ tập các hàng đang làm việc được sinh ra bởi lệnh OPEN cuối cùng của cursor. Có thể OPEN lại được và như vậy sẽ có tập hàng làm việc mới hơn.

3. Khai báo

- *Cú pháp:*

  CURSOR indentifier [(parameter details)] IS query-expression;

  Ví dụ:

```
DECLARE
  CURSOR c1 IS
      SELECT last_name, salary, hire_date, job_id
      FROM employees
      WHERE employee_id = 120;
/* khai báo biến record để đại diện một hàng được fetch từ
bảng employees */
      employee_rec c1%ROWTYPE;
BEGIN
-- mở cursor một cách tường minh
-- sử dụng cursor này để fetch dữ liệu đổ vào employee_rec
      OPEN c1;
      FETCH c1 INTO employee_rec;
      DBMS_OUTPUT.PUT_LINE('Employee name: '
                           || employee_rec.last_name);
END;
```

4. Các thuộc tính của explicit cursor (Explicit Cursor Attributes)

- Giống như các implicit cursor, có 4 thuộc tính để biết các thông tin về cursor. Khi dùng, thì phải để tên cursor trước các thuộc tính này.

%FOUND	Có giá trị TRUE nếu lệnh FETCH gần nhất từ cursorlấy được 1 hàng từ active set, ngược lại sẽ là FALSE
%NOTFOUND	Ngược với %FOUND
%ROWCOUNT	Trả về số hàng đã FETCH được từ active set tính đến hiện tại
%ISOPEN	TRUE nếu cursor đang mở, FALSE nếu cursor đã đóng hoặc chưa được mở trong khối

Ví dụ 1:

```
IF c1%ISOPEN THEN
      FETCH c1 INTO v_ename, v_sal, v_hiredate;
ELSE
      OPEN c1;
END IF;
```

Ví dụ 2:

```
LOOP
      FETCH c1 INTO v_ename, v_sal, v_hiredate;
      EXIT WHEN c1%ROWCOUNT > 10;
END LOOP;
```

5. Điều khiển các việc lấy nhiều dữ liệu từ các explicit cursor

- Thường thì khi muốn xử lý nhiều hàng từ explicit cursor thì dùng một vòng lặp với lệnh FETCH tại mỗi bước lặp. Nếu quá trình tiếp tục thì tất cả các hàng trong active set sẽ được xử lý. Khi một lệnh FETCH không thành công xảy ra, thuộc tính %NOTFOUND sẽ là TRUE. Mặc dù vậy, nếu dùng lệnh FETCH kế tiếp thì sẽ xảy ra lỗi :

```
ORA-1002:  Fetch out of sequence
```

- Lỗi này sẽ kết thúc khối thường là một *unhandled exception*. Vì thế cần thiết phải kiểm tra sự thành công của mỗi lần FETCH trước khi tiếp tục tham khảo cursor.

Ví dụ :

```
OPEN cursor_1;
LOOP
      FETCH cursor_1 INTO a, b, c, d;
      EXIT WHEN cursor_1%NOTFOUND;
      -- xử lý hàng hiện tại ở đây
END LOOP;
```

6. Mệnh đề FOR UPDATE OF

Ví dụ :

```
DECLARE
  CURSOR c1 IS
      SELECT empno, sal, hiredate, rowid
      FROM emp WHERE depno=20 AND job='ANALYST'
      FOR UPDATE OF sal;
  emp_record c1%ROWTYPE;
BEGIN
      OPEN c1;
      …
      FETCH c1 INTO emp_record;
      …
      IF emp_record.sal < 2000 THEN …
      …
END;
```

- Ví dụ trên dùng FOR UPDATE trong query của cursor. Nghĩa là các hàng trả về bởi query sẽ được khóa không cho ai khác truy xuất vào khi OPEN được dùng. Khi bỏ khóa tại cuối giao dịch, chúng ta không cần COMMIT.

7. Mệnh đề WHERE CURRENT OF

- Khi tham khảo 'current row' từ một explicit cursor, các lệnh SQL có thể dùng mệnh đề WHERE CURRENT OF. Nó cho phép cập nhật hay xóa bỏ tại hàng hiện tại.

Ví dụ :

```
FETCH c1 INTO emp_record;
IF emp_record.ename = 'KING' THEN
      DELETE FROM emp WHERE CURRENT OF c1;
```

XV. Triggers

- Một Database Trigger được tạo và lưu trữ trong PL/SQL block tương ứng với table. Nó được tự động gọi đến khi có sự truy nhập đến table tương ứng với các hành động định nghĩa.
- *Cú pháp*:

CREATE [OR REPLACE] TRIGGER *trigger_name*

BEFORE | AFTER

[UPDATE (OF *column*)] | [DELETE] | [INSERT] ON TABLE

(FOR EACH ROW (WHEN *condition*))

BEGIN

PL/SQL block

END *trigger_name*;

Ví dụ:

✓ Tạo bảng:

```
CREATE TABLE product_price_audit
   (product_id INTEGER
   CONSTRAINT price_audit_fk_products
   REFERENCES products(product_id),
   old_price NUMBER(5, 2),
   new_price NUMBER(5, 2));
```

✓ Tạo Trigger

```
CREATE OR REPLACE TRIGGER before_product_price_update
BEFORE UPDATE OF price
ON products
FOR EACH ROW WHEN (new.price < old.price * 0.75)
BEGIN
   dbms_output.put_line('product_id = ' || :old.product_id);
   dbms_output.put_line('Old price = ' || :old.price);
   dbms_output.put_line('New price = ' || :new.price);
   dbms_output.put_line('The price reduction is more than
25%');
   -- insert row into the product_price_audit table
   INSERT INTO product_price_audit ( product_id, old_price,
   new_price)
   VALUES (:old.product_id, :old.price, :new.price);
END before_product_price_update;
```

✓ Firing a Trigger: để thấy được output từ một trigger, bạn cần phải chạy câu lệnh:

```
SET SERVEROUTPUT ON
UPDATE products
```

```
SET price = price * .7
WHERE product_id IN (5, 10);

product_id = 10
Old price = 15.99
New price = 11.19
The price reduction is more than 25%
product_id = 5
Old price = 49.99
New price = 34.99
The price reduction is more than 25%
2 rows updated.
```

✓ Disable and Enable Trigger

Có thể cấm một trigger hoạt động và ngược lại bằng câu lệnh ALTER TRIGGER.

```
ALTER TRIGGER before_product_price_update DISABLE;
ALTER TRIGGER before_product_price_update ENABLE;
```