Ngôn ngữ thủ tục PL/SQL

Ngôn ngữ thủ tục PL/SQL

1. Giới thiệu PL/SQL.
2. Khối lệnh trong PL/SQL (block),
3. Khai báo biến và hằng số, các kiểu dữ liệu
4. Các lệnh điều kiện (IF, CASE), rẻ nhánh (GOTO), lệnh lặp (while…loop, for…loop)
5. Xử lý lỗi (Exception) trong Oracle
6. Cursors: định nghĩa, phân loại cursor: tường minh và tiềm ẩn, cách sử dụng
7. Function, Procedure, Trigger, Package

1. Giới thiệu PL/SQL (1)
- (Procedural Language/Structure Query Language)

- Ngôn ngữ thủ tục của Oracle, dùng để xây dựng các ứng dụng.
- PL/SQL là sự kết hợp giữa SQL và các cấu trúc điều khiển, các thủ tục (function), thao tác con trỏ (cursor), xử lý ngoại lệ (exception) và các lệnh giao tác.
- PL/SQL cho phép sử dụng tất cả lệnh thao tác dữ liệu gồm INSERT, DELETE, UPDATE và SELECT, COMMIT, ROLLBACK, SAVEPOINT, cấu trúc điều khiển như vòng lặp (for, while, loop), rẽ nhánh (if),...mà với SQL chúng ta không làm được.

1. Giới thiệu PL/SQL (2)
- (Procedural Language/Structure Query Language)

- PL/SQL thêm chức năng vào các công cụ không thủ tục như SQL*Forms và SQL*Report.
- Các lệnh PL/SQL được chia thành nhiều khối lệnh hợp lý (Block), các khối lệnh lồng nhau. Các biến có thể khai báo nội tại (local) bên trong block và điều khiển báo lỗi (exception) được xử lý trong block nơi lỗi phát sinh.
- Một block bao gồm ba phần: phần khai báo là nơi để khai báo biến, phần thi hành lệnh và phần xử lý các ngoại lệ (điều kiện lỗi hoặc cảnh báo).
- Khai báo biến trong PROCEDURE hay FUNCTION: nếu là Block ngoài cùng (đầu tiên) của PROCEDURE, FUNCTION thì không dùng từ khóa DECLARE (Ngược lại với TRIGGER, Block ngoài cùng (đầu tiên) phải có DECLARE)

2. Cấu trúc PL/SQL

```
DECLARE    /*Phần Khai báo biến Block 1*/          --- Block 1
        Các khai báo biến của Block 1 (Declarations)
BEGIN
        Các câu lệnh thực hiện (Executable Statements)
    DECLARE    /*Phần Khai báo biến Block 2*/      --- Block 2
            Các khai báo biến của Block 2 (Declarations)
    BEGIN
            Các câu lệnh thực hiện (Executable Statements)
    EXCEPTION
            Các xử lý ngoại lệ (Exception Handlers)
            /*làm gì nếu lỗi xuất hiện bên trong Block 2*/
    END;                                           --- End Block 2
EXCEPTION
        Các xử lý ngoại lệ (Exception Handlers)
END;                                               --- End Block 1
```

3. Khai báo biến và hằng (1)

- Khai báo biến:

  mucluong NUMBER(5);

- Khai báo hằng:

  heso CONSTANT NUMBER(3,2) := 1.86;

- Với các kiểu dữ liệu trong Oracle như NUMBER, CHAR, VARCHAR2, DATE, LONG,…hoặc PL/SQL cho phép như BOOLEAN.

Ghi chú: Ký hiệu := được sử dụng như là toán tử gán.

3. Khai báo biến và hằng (2)

- Gán biến và biểu thức:

  biến := biểu thức;

  Ví dụ:

  ```
  x:=UPPER('Nguyen');
  y:=100;
  mucluong:= mucluong + mucluong*10/100;
  ```

  Ví dụ:

  ```
  kq BOOLEAN;//ko có column kiểu boolean, chỉ có kiểu dữ liệu Boolean trong PL/SQL
  kq:= mucluong>3500000;
  ```

- Độ ưu tiên của toán tử: ** (phép lũy thừa), NOT, *, /, +, -, || (phép nối chuỗi), =, !=, <>, <=, >=, IS NULL, LIKE, BETWEEN, IN, AND, OR.

3. Khai báo biến và hằng (3)

(Các thuộc tính %TYPE và %ROWTYPE)

1. Thuộc tính %TYPE

- Dùng để khai báo một biến mà nó tham chiếu đến một cột trong cơ sở dữ liệu. (Có cấu trúc như một cột trong Table).

  Ví dụ: khai báo biến v_Manv có cùng kiểu dữ liệu với cột Manv trong bảng NHANVIEN

  v_Manv NHANVIEN.Manv%TYPE

- Khai báo có điểm thuận lợi là: kiểu dữ liệu chính xác của biến v_Manv không cần được biết, nếu định nghĩa của cột Manv trong bảng NHANVIEN bị thay đổi thì kiểu dữ liệu của biến v_Manv thay đổi tương ứng.

3. Khai báo biến và hằng (4)

Ví dụ thuộc tính %TYPE

```
declare
    x emp.empno%type;
    y emp.ename%type;
begin
  select empno, ename into x,y from emp where empno='7369';
  dbms_output.put_line('Ma nv:' || x || ' - Ho ten nhan vien:' || y);
end;
```

Chạy lệnh SET SERVEROUTPUT ON trong SQL*Plus trước. Lúc đó lệnh DBMS_OUTPUT.PUT_LINE…mới có hiệu lực in text "……." ra màn hình

3. Khai báo biến và hằng (5)

(Các thuộc tính %TYPE và %ROWTYPE)

2. Thuộc tính %ROWTYPE

- Dùng để khai báo một biến mà nó tham chiếu đến một dòng trong cơ sở dữ liệu (Có cấu trúc như một dòng trong Table).

  Ví dụ: khai báo biến v_nv có kiểu dữ liệu là một dòng trong bảng NHANVIEN

  v_nv NHANVIEN%ROWTYPE

- Khi truy xuất đến từng cột ta sử dụng giống như một bảng dữ liệu (trong trường hợp này chỉ gồm 1 record) tham chiếu đến một cột.

  Cú pháp: Tên-biến.Tên-cột VD: v_nv.HoTen

3. Khai báo biến và hằng (6)

Ví dụ thuộc tính %ROWTYPE

```
declare
    z       emp%rowtype;
begin
    select * into z from emp where empno='7369';
    dbms_output.put_line('Ma nv:' || z.empno || ' - Ho ten nhan
    vien:' || z.ename);
end;
```

4. Các cấu trúc (lệnh) điều khiển (1)

1. Lệnh rẽ nhánh If

- Cú pháp 1:

```
IF <điều kiện 1> THEN
        khối lệnh 1;
ELSE
        IF <điều kiện 2> THEN
                khối lệnh 2;
        ELSE
                …..;
        END IF;
END IF;
```

4. Các cấu trúc (lệnh) điều khiển (2)

- Cú pháp 2:

```
IF <điều kiện 1> THEN
        khối lệnh 1;
ELSIF <điều kiện2> THEN
        khối lệnh 2;
ELSIF <điều kiện 3> THEN
        khối lệnh 3;
ELSIF <điều kiện n> THEN
        khối lệnh n;
END IF;
```

4. Các cấu trúc (lệnh) điều khiển (3)

- Ví dụ cú pháp 1:

```
IF n=1 THEN
        ngay :='Sunday';
ELSE
        IF n=2 THEN
                ngay :='Monday';
        End If;
END IF;
```

4. Các cấu trúc (lệnh) điều khiển (4)

- Ví dụ cú pháp 2:

```
IF n=1 THEN
        ngay :='Sunday';
ELSIF n=2 THEN
        ngay :='Monday';
ELSIF n=3 THEN
        ngay :='Tuesday';
ELSIF n=4 THEN
        ngay :='Wedsday';
ELSIF n=5 THEN
        ngay :='Thursday';
END IF;
```

4. Các cấu trúc (lệnh) điều khiển (5)

2. Lệnh rẽ nhánh CASE

- Cú pháp lệnh CASE:

```
CASE  [ expression ]
    WHEN condition_1 THEN result_1
    WHEN condition_2 THEN result_2
    ...
    WHEN condition_n THEN result_n
    ELSE result
END
```

4. Các cấu trúc (lệnh) điều khiển (6)

- ví dụ: lệnh CASE có expression (ví dụ: owner, TH1) và CASE không có expression (TH2) => đưa đk vào sau mệnh đề WHEN [expression=đk]

```
select    table_name, TABLESPACE_NAME, CASE owner
        WHEN 'SYS'                THEN      'The owner is SYS'
        WHEN 'SYSTEM'             THEN      'The owner is SYSTEM'
        ELSE 'The owner is another value'
      END     as Owner
      from all_tables;
```

Hoặc viết cách khác

```
select    table_name, TABLESPACE_NAME, CASE
        WHEN owner = 'SYS'                THEN      'The owner is SYS'
        WHEN owner = 'SYSTEM'             THEN      'The owner is SYSTEM'
        ELSE 'The owner is another value'
        END     as Owner
from all_tables;
```

4. Các cấu trúc (lệnh) điều khiển (6)

Ví dụ lệnh CASE không có expression, khi có 2 đk trở lên khó biểu diễn expression => đưa đk vào sau mệnh đề WHEN [expression=đk1] and/or [expression=đk2]….

```
select supplier_id, CASE  WHEN supplier_name = 'IBM' and supplier_type = 'Hardware' THEN
    'North office‘
                          WHEN supplier_name = 'IBM' and supplier_type = 'Software' THEN
    'South office'
    END
    from suppliers;
```

4. Các cấu trúc (lệnh) điều khiển (7)

3. Lệnh lặp LOOP

- Cú pháp:

```
LOOP
        <khối lệnh>
IF <thỏa điều kiện dừng> THEN
        ....
        EXIT;
END IF;
END LOOP;
```

4. Các cấu trúc (lệnh) điều khiển (8)

Ví dụ:

```
declare
z  number :=1; /*khởi tạo biến z*/
BEGIN
        LOOP
        z :=z+3; /*tính biểu thức lặp*/
        IF  (z>=100)  THEN    /*nếu  thỏa  điều  kiện
  thoát khỏi vòng lặp*/
                exit;
        End IF;
        END LOOP;
END;
```

4. Các cấu trúc (lệnh) điều khiển (9)

4. Lệnh lặp FOR…LOOP

- Cú pháp 1:

```
FOR biến-chạy IN [REVERSE] giá-trị-khởi-tạo .. giá-trị-kết-thúc
                LOOP
                        <khối lệnh>
                END LOOP;
```

- Cú pháp 2 (xử lý cho câu lệnh Select):

```
FOR biến-chạy IN (câu lệnh select)
                LOOP
                <khối lệnh> (… biến-chạy.têncột1, biến-chạy.têncột2 …)
                END LOOP;
```

4. Các cấu trúc (lệnh) điều khiển (10)

- Ví dụ - cú pháp 1:

```
declare
z  number:=1; /*khởi tạo biến z*/
i   number;
BEGIN
        FOR i IN 1 .. 10
        LOOP
                z :=z+3; /*tính biểu thức lặp*/
        END LOOP;
END;
```

4. Các cấu trúc (lệnh) điều khiển (11)

Ví dụ - cú pháp 2:

```
for z in (select scott.emp.empno, scott.emp.ename from scott.emp)
loop
    Dbms_output.put_line (z.empno || '---' || z.ename);
end loop;
```

Ví dụ khác (con trỏ):

```
for m in têncursor
    loop
  dbms_output.put_line (m.têncột1 || '-----' ||m.têncột2);
    End loop
```

4. Các cấu trúc (lệnh) điều khiển (12)

5. Lệnh lặp WHILE...LOOP

- Cú pháp:

```
WHILE <điều kiện đúng>
LOOP
        <khối lệnh>
END LOOP;
```

(Phân biệt 3 lệnh Loop …End Loop và For…Loop, và While…Loop: đều sử dụng lệnh Loop)

4. Các cấu trúc (lệnh) điều khiển (13)

- Ví dụ:

```
declare
z  number:=1; /*khởi tạo biến z*/
i   number:=1; /*khởi tạo biến i*/
BEGIN
        WHILE (i<=10)
              LOOP
                    i:=i+1;
                    z :=z+3; /*tính biểu thức lặp*/
               END LOOP;
END;
```

4. Các cấu trúc (lệnh) điều khiển (14)

6. Lệnh điều khiển lặp CONTINUE (only supported in Oracle 11g), EXIT…

a) Ví dụ: lệnh CONTINUE (lệnh EXIT xem lệnh Loop ở các silde trước)

```
DECLARE
        x NUMBER := 0;
BEGIN
    LOOP -- After CONTINUE statement, control resumes here
        DBMS_OUTPUT.PUT_LINE ('Inside loop: x = ' ||TO_CHAR(x));
        x := x + 1;
        IF x < 3 THEN
                CONTINUE;
        END IF;
        DBMS_OUTPUT.PUT_LINE ('Inside loop, after CONTINUE: x = ' ||
TO_CHAR(x));
        EXIT WHEN x = 5;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE (' After loop: x = ' || TO_CHAR(x));
END;
```

4. Các cấu trúc (lệnh) điều khiển (15)

Kết quả sau khi chạy đoạn lệnh trên:

Inside loop: x = 0

Inside loop: x = 1

Inside loop: x = 2

Inside loop, after CONTINUE: x = 3

Inside loop: x = 3 Inside loop, after CONTINUE: x = 4

Inside loop: x = 4 Inside loop, after CONTINUE: x = 5 After loop: x = 5

4. Các cấu trúc (lệnh) điều khiển (16)

6. Lệnh điều khiển lặp CONTINUE (only supported in Oracle 11g)…

b) Ví dụ: lệnh CONTINUE WHEN

```
DECLARE
        x NUMBER := 0;
BEGIN
    LOOP -- After CONTINUE statement, control resumes here
        DBMS_OUTPUT.PUT_LINE ('Inside loop: x = ' || TO_CHAR(x));
        x := x + 1;
        CONTINUE WHEN x < 3;
        DBMS_OUTPUT.PUT_LINE('Inside loop, after CONTINUE: x = '
|| TO_CHAR(x));
        EXIT WHEN x = 5;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE (' After loop: x = ' || TO_CHAR(x));
END;
```

4. Các cấu trúc (lệnh) điều khiển (17)

Kết quả sau khi chạy đoạn lệnh trên:

Inside loop: x = 0

Inside loop: x = 1

Inside loop: x = 2 Inside loop, after CONTINUE: x = 3

Inside loop: x = 3 Inside loop, after CONTINUE: x = 4

Inside loop: x = 4 Inside loop, after CONTINUE: x = 5 After loop: x = 5

4. Các cấu trúc (lệnh) điều khiển (18)

(Sử dụng tên nhãn và lệnh GOTO)

7. Sử dụng tên nhãn

- Một tên nhãn được dùng để đặt tên cho một khối lệnh PL/SQL hoặc các câu lệnh bên trong khối.
- Tên nhãn được định nghĩa bằng cách sử dụng dấu móc nhọn <<tên-nhãn>>.
- Tên nhãn thường được sử dụng trong lệnh GOTO để chuyển điều khiển đến khối lệnh thực hiện trong nhãn.

4. Các cấu trúc (lệnh) điều khiển (19)

(Sử dụng tên nhãn và lệnh GOTO)

Lệnh GOTO

- Câu lệnh GOTO rẽ nhánh không điều kiện đến một nhãn. Khi thực hiện, câu lệnh GOTO thay đổi luồng điều khiển trong một khối để chuyển đến thực hiện lệnh nằm trong nhãn.

- GOTO không được phép trong một số trường hợp:
  - Từ một xử lý ngoại lệ vào trong khối hiện hành.
  - Nhảy ra ngoài chương trình con.

4. Các cấu trúc (lệnh) điều khiển (20)

- **Ví dụ:**

```
BEGIN
        <<outer_block>>
        declare
                <khai báo biến>
        Begin
                <khối lệnh 1>
                GOTO   inner_block
                <khối lệnh 2>
                <<inner_block>>
                declare
                        <khai báo biến>
                Begin
                        <khối lệnh 3>
                End;    /*End của <<inner_block>>*/
        End;    /*End của <<outer_block>>*/
END;
```

4*. Các cấu trúc (lệnh) điều khiển (21)

- **Ví dụ 1:**

```
Create Function Test_Block (m number) return number
As
        x number;
begin
                x:=m;
                if x=5 then
                  GOTO BlockB ;
                  lenh 1;
                  lenh 2;
                  ……
                else
                  GOTO BlockC ;
                end if;
                <<BlockB>>
```

4. Các cấu trúc (lệnh) điều khiển (22)

```
            declare
                    y number;
            Begin
                    y:=100;
                    return 5;
            End;    /*End cua <<BlockB>>*/
            <<BlockC>>
                    return 0;
    end;
```

- Sau khi định nghĩa Function Test_Block trong SQL*Plus, chạy lệnh sau để thấy kết quả xử lý của hàm trong 2 trường hợp khác nhau.

```
        Select Test_Block (5) from Dual;  /*Dual là table tạm*/
Hoặc    Select Test_Block (10) from Dual;
```

4. Các cấu trúc (lệnh) điều khiển (23)

- **Ví dụ 2:**

```
Create Function Test_Block2 (m number) return number  As
begin
        <<BlockA>>
        declare
                x number;
        Begin
                x:=m;
                if x=5 then
                  GOTO BlockB ;
                else
                  GOTO BlockC ;
                end if;
                <<BlockB>>
                        goto BlockD;
                <<BlockC>>
                        return 0;
                <<BlockD>>
                        return 5;
        End;
end;
```

5. Xử lý ngoại lệ (EXCEPTION) (1)

- **Khi một lỗi phát sinh, một ngoại lệ được đưa ra, việc thực hiện chương trình bình thường được dừng lại và điều khiển được chuyển tới khối PL/SQL chứa phần xử lý ngoại lệ.**

- **Những ngoại lệ bên trong được sinh ra một cách tiềm ẩn (không tường minh, implicit), trái lại những ngoại lệ do người dùng định nghĩa được sinh ra một cách tường minh (explicit) bằng cách sử dụng câu lệnh RAISE.**

- **VD: Nếu chia một số cho zero, một ngoại lệ do Oracle định nghĩa trước (ví dụ: ZERO_DIVIDE) sẽ tự động sinh ra.**

5. Xử lý ngoại lệ (EXCEPTION) (2)

(Các ngoại lệ do Oracle định nghĩa)

Ngoại lệ	Điều kiện khi ngoại lệ xảy ra
CURSOR_ALREADY_OPEN	Mở một cursor, mà cursor đó đã ở trạng thái đang mở.
DUP_VAL_ON_INDEX	Khi có thao tác INSERT hoặc UPDATE vi phạm ràng buộc UNIQUE.
INVALID_CURSOR	Mở một cursor chưa tạo hoặc đóng một cursor mà nó chưa được mở.
INVALID_NUMBER	Lỗi chuyển kiểu dữ liệu từ string sang kiểu number.
LOGIN_DENIED	Đăng nhập sai username/password.
NO_DATA_FOUND	Câu lệnh SELECT INTO không trả về dòng nào.
NOT_LOGGED_ON	Một chương trình PL/SQL cần thao tác đến Cơ sở dữ liệu Oracle nhưng lại chưa đăng nhập vào Cơ sở dữ liệu.
PROGRAM_ERROR	Một số lỗi chương trình, ví dụ một hàm (function) không chứa mệnh đề RETURN để trả về giá trị.
STORAGE_ERROR	Lỗi bộ nhớ
TIMEOUT_ON_RESOURCE	Lỗi timeout xảy ra khi Oracle đang chờ tài nguyên.
TOO_MANY_ROWS	Câu lệnh SELECT INTO trả về nhiều hơn một dòng.
VALUE_ERROR	Lỗi chuyển kiểu dữ liệu hoặc thao tác vi phạm ràng buộc toàn vẹn.
ZERO_DIVIDE	Lỗi chia một số cho zero.
OTHERS	Lỗi khác (không xác định)

5*. Xử lý ngoại lệ (EXCEPTION) (3)

(Ví dụ ngoại lệ NO_DATA_FOUND do Oracle định nghĩa)

```
DECLARE
        sHT SinhVien.Hoten%TYPE;
        StudentId number;
BEGIN
      StudentId := &abc;
      select Hoten into sHT from SinhVien where masv = StudentId;
      dbms_output.put_line(sHT);
EXCEPTION
        WHEN NO_DATA_FOUND then
        dbms_output.put_line ('khong co sv nay');
        dbms_output.put_line ('ban co the them sinh vien moi');
         insert into SinhVien values(StudentID, '&HoTen', '&DiaChi'
   , &….);
        WHEN OTHERS then
        dbms_output.put_line ('Loi khong xac dinh');
END;
```

5. Xử lý ngoại lệ (EXCEPTION) (4)

(Ví dụ ngoại lệ ZERO_DIVIDE do Oracle định nghĩa)

- Ví dụ:

```
Create Function Test_Exception (so number) return number
As
        x   number(4,2);
Begin
        x:=100/so;
        return 1;
EXCEPTION
    WHEN ZERO_DIVIDE then   /*loi do Oracle dinh nghia*/
        return 0; /*Loi chia zero*/
END;
```

- **Chạy lệnh sau trong SQL*Plus để thấy kết quả xử lý 2 trường hợp khác nhau:**

  Select Test_Exception (5) from Dual; /*Dual là table tạm*/

  Hoặc Select Test_Exception (0) from Dual;

5. Xử lý ngoại lệ (EXCEPTION) (5)

(Ngoại lệ do người dùng định nghĩa)

Định nghĩa ngoại lệ:

```
DECLARE
/*nếu là Block ngoài cùng của Function hoặc Procedure thì không dùng Declare*/
        ten_loi_ngoai_le  EXCEPTION;     /* Bước 1: khai báo lỗi */
        ....
BEGIN
        ...
        IF <điều kiện lỗi> then
            RAISE   ten_loi_ngoai_le; /* Bước 2: gọi hay là bật ngoại lệ(lỗi) */
        END IF   ;
EXCEPTION
    WHEN ten_loi_ngoai_le then       /* Bước 3: xử lý lỗi */
        .....
    WHEN OTHERS then
        ....
END;
```

5. Xử lý ngoại lệ (EXCEPTION) (6)

(Ngoại lệ do người dùng định nghĩa)

Ví dụ: (kết quả trả về 1 nếu trùng mã số,trả về 2: bình thường)

```
Create Function KiemTraTrungMSSV (maso number) return number  As
        trung_ma_so  EXCEPTION;   /*khai bao 1 ngoai le ten "trung_ma_so"*/
BEGIN
        IF maso=100 then
                RAISE  trung_ma_so; /*day la ngoai le tuong minh ➔ bat ngoai
    le bang tu khoa RAISE*/
        ELSE
          return 2;
        END IF   ;
EXCEPTION
    WHEN trung_ma_so then
        return 1; /*da co ma so nay roi*/
    WHEN OTHERS then   /*sử dụng từ khóa OTHERS cho các lỗi khác past_due,
    việc sử dụng OTHERS đảm bảo không có ngoại lệ nào sẽ không được xử lý*/
        return 0; /*loi phat sinh*/
END;
```

6. Sử dụng con trỏ trong PL/SQL (1)

(Giới thiệu Cursor)

- **Con trỏ (cursor) là một đối tượng liên kết với một tập dữ liệu và cho phép người lập trình làm việc với từng dòng của tập dữ liệu đó.**
- **Để xử lý một câu SQL, PL/SQL mở một vùng làm việc có tên là vùng ngữ cảnh (context area). PL/SQL sử dụng vùng này để thi hành câu SQL và chứa kết quả trả về. Vùng ngữ cảnh đó là phạm vi hoạt động của con trỏ.**
- **Có hai loại con trỏ: con trỏ được khai báo tường minh (explicit cursor) và con trỏ không được khai báo tường minh (hay còn gọi là con trỏ tiềm ẩn (implicit cursor)).**

6. Sử dụng con trỏ trong PL/SQL (2)

(Giới thiệu Cursor)

- **Con trỏ tường minh: Là con trỏ được đặt tên bởi người sử dụng (câu SELECT được đặt tên).**

  Ví dụ: CURSOR c_nv IS SELECT empno,sal FROM EMP

- **Con trỏ tiềm ẩn: một lệnh SQL được xử lý bởi Oracle và không được đặt tên bởi người sử dụng. Các lệnh SQL được thực hiện trong một con trỏ tiềm ẩn bao gồm UPDATE, INSERT, DELETE.**

  Ví dụ:

  Khối lệnh

  ....

  Insert into EMP (empno, sal) values (7240,1000)

  ...

6. Sử dụng con trỏ trong PL/SQL (3)

Khai báo cursor (Khai báo con trỏ)

- **Cú pháp:**

  CURSOR tên-cursor IS câu-lệnh-SELECT;

- **Trong đó, câu lệnh SELECT phải chỉ ra các cột cụ thể cần lấy cho con trỏ này.**
- **Phần khai báo này phải được đặt trong vùng khai báo biến (trước BEGIN của khối (Block)).**
- **Trong ngôn ngữ thủ tục PLSQL, để xử lý dữ liệu lưu trong cơ sở dữ liệu, đầu tiên dữ liệu cần được ghi vào các biến. Giá trị trong biến có thể được thao tác. Dữ liệu các bảng không thể được tham khảo trực tiếp.**

6. Sử dụng con trỏ trong PL/SQL (4)

Khai báo cursor (Khai báo con trỏ)

- **Ví dụ: EMP.Ename sẽ không cho truy cập vào dữ liệu có trong cột Ename của bảng EMP.**
- **Thay vào đó, câu lệnh SELECT…INTO cho phép ta nhận và lưu dữ liệu trong biến. Cú pháp như sau:**

  SELECT <danhsáchcột> INTO <danhsáchbiến>

  FROM <tên-bảng> [WHERE <condition>;]
- **Tiếp theo, các thao tác diễn ra trên các biến có trong danh sách và làm một hành động cập nhật lại (nếu có) vào cơ sở dữ liệu bằng lệnh UPDATE.**
- **SELECT…INTO thường được sử dụng cho các con trỏ tiềm ẩn.**
- **Với con trỏ tường minh thường dùng phương thức Fetch để lấy giá trị của các cột dữ liệu vào các biến.**

6. Sử dụng con trỏ trong PL/SQL (5)

Cursor

Ví dụ:

```
Create Procedure Con_tro as
        x   EMP.empno%TYPE;
        y   EMP.sal%TYPE;
        cursor cs is select empno, sal from emp; /*con trỏ tường minh*/
        begin
       open cs;
       loop
                  fetch cs into x, y;
                  exit when cs%notfound;
                              ....
        end loop;
        select deptno into d_no from DEPT; /*con trỏ tiềm ẩn*/
        ....
        end;
```

6. Sử dụng con trỏ trong PL/SQL (6)

Đặc điểm cursor

Một số đặc điểm của con trỏ:

- Tên của con trỏ không được khai báo định danh, chỉ dùng khi tham chiếu đến câu truy vấn.
- Không được gán giá trị cho tên con trỏ và không được sử dụng tên con trỏ như là một biểu thức.
- Con trỏ tường minh có thể có tham số.
- Có thể khởi tạo giá trị mặc định cho tham số của con trỏ.
- Giá trị tham số của con trỏ chỉ có nghĩa khi con trỏ đã được mở (OPENed).

6. Sử dụng con trỏ trong PL/SQL (7)

(Thao tác Cursor: Open, Fetch, Close)

- Thao tác trên con trỏ: khai báo CURSOR, OPEN, FETCH, CLOSE

Cú pháp:

CURSOR tên-cursor is câu-lệnh-SQL; /*Khai báo con trỏ*/
OPEN tên-cursor; /*Mở con trỏ thi hành câu truy vấn*/
FETCH tên-cursor INTO biến1, biến2, …, biếnn;

hoặc

FETCH tên-cursor INTO biếncókiểurecord;
/*Lệnh FETCH dùng để gọi một dòng trong tập dữ liệu của con trỏ, có thể được lặp để gọi tất cả các dòng của con trỏ*/.

CLOSE tên-cursor /*đóng con trỏ, giải phóng khỏi bộ nhớ*/

6. Sử dụng con trỏ trong PL/SQL (8)

(Thuộc tính con trỏ tường minh)

Mọi con trỏ khai báo tường minh đều có bốn thuộc tính: %NOTFOUND, %FOUND, %ROWCOUNT, %ISOPEN. Các thuộc tính này được thêm vào sau phần tên của con trỏ.

1.Thuộc tính %NOTFOUND (đi kèm lệnh Fetch)

Mang giá trị TRUE hoặc FALSE. %NOTFOUND bằng TRUE khi đã fetch đến dòng cuối cùng của con trỏ, ngược lại, bằng FALSE khi lệnh fetch trả về ít nhất một dòng hoặc chưa fetch đến dòng cuối cùng.

Ví dụ:

```
OPEN cur_first;
LOOP
        FETCH cur_first INTO v_empno,v_sal;
        EXIT WHEN cur_first%NOTFOUND;
END LOOP;
```

6. Sử dụng con trỏ trong PL/SQL (9)

(Thuộc tính con trỏ tường minh)

2.Thuộc tính %FOUND (đi kèm lệnh Fetch)

Ngược với thuộc tính NOTFOUND.

Ví dụ:

```
OPEN cur_first;
LOOP
        FETCH cur_first INTO v_empno,v_sal;
        IF cur_first%FOUND THEN
                .............
        ELSE
                CLOSE cur_first;
                EXIT;
        END IF;
END LOOP;
```

6. Sử dụng con trỏ trong PL/SQL (10)

(Thuộc tính con trỏ tường minh)

3.Thuộc tính %ROWCOUNT

Trả về số dòng của con trỏ đã được fetch.

Ví dụ:

```
LOOP
        FETCH cur_first INTO v_empno,v_sal;
            ..........
        IF cur_first%ROWCOUNT =1000 THEN
            EXIT;
        END IF;
        .......
END LOOP;
```

6. Sử dụng con trỏ trong PL/SQL (11)

(Thuộc tính con trỏ tường minh)

4.Thuộc tính %ISOPEN

Trả về giá trị TRUE nếu con trỏ ở trạng thái mở và giá trị FALSE nếu con trỏ đã được đóng.

Ví dụ:

```
IF cur_first%ISOPEN THEN
        FETCH cur_first INTO v_empno,v_sal;
ELSE
        CLOSE cur_first;
END IF;
```

6. Sử dụng con trỏ trong PL/SQL (12)

(Con trỏ (Cursor) có tham số)

5.Con trỏ có tham số:

Một con trỏ có thể nhận tham số là tham trị. Các tham số không được dùng để trả về giá trị cho cursor.

Ví dụ:

```
CURSOR cur_first (v_eno  EMP.empno%TYPE) IS
SELECT empno, sal
FROM EMP
WHERE empno= v_eno;
```

Trong đó, v_eno là tham số của con trỏ. Khi thao tác với con trỏ có tham số thì ta phải gọi tên con trỏ kèm theo giá trị của tham số.

Khi open cursor ta phải truyền vào giá trị của tham số:

```
open cur_first(22);
```

6. Sử dụng con trỏ trong PL/SQL (13)

(Thuộc tính con trỏ tiềm ẩn)

Có các thuộc tính: SQL%NOTFOUND, SQL%FOUND, SQL%ROWCOUNT. Thuộc tính SQL%IsOpen luôn là False

Lệnh OPEN, CLOSE, FETCH không được dùng cho con trỏ tiềm ẩn nhưng những thuộc tính của con trỏ vẫn được áp dụng trong vùng ngữ cảnh. Trước khi thi hành câu SQL,các thuộc tính của con trỏ tiềm ẩn có giá trị NULL.

Ví dụ: thuộc tính %NOTFOUND

```
SET SERVEROUTPUT ON
DELETE FROM emp WHERE empno='222';
IF SQL%NOTFOUND THEN
   DBMS_OUTPUT.PUT_LINE ('Ko co nhan vien 222');
END IF;
```

6*. Sử dụng con trỏ trong PL/SQL (14)

(Thuộc tính con trỏ tiềm ẩn)

Ví dụ thuộc tính %FOUND

```
SELECT empno into v_eno  FROM EMP WHERE empno=7788;
IF SQL%FOUND THEN
      DELETE FROM EMP WHERE empno=7788;
END IF;
```

Thuộc tính %ROWCOUNT

Trả về số dòng tác động bởi câu lệnh INSERT, UPDATE, DELETE, SELECT.

Ví dụ: (chạy lệnh SET SERVEROUTPUT ON trong SQL*Plus trước khi chạy đoạn lệnh bên dưới. Lúc đó lệnh DBMS_OUTPUT.PUT_LINE…mới có hiệu lực in text "Lương mới" ra màn hình)

```
      UPDATE EMP SET SAL=5000 WHERE empno=7788;
      IF SQL%ROWCOUNT >0 THEN
            DBMS_OUTPUT.PUT_LINE ('Luong moi');
      END IF;
```

6*. Sử dụng con trỏ trong PL/SQL (15)

Ví dụ Procedure sử dụng con trỏ tường minh

```
Create Procedure Hien_Thi_Muc_Luong as
 x EMP.empno%TYPE;
 y EMP.sal%TYPE;
 cursor nv is select empno, sal from emp;
begin
  open nv;
  DBMS_OUTPUT.Put ('Ma nhan vien');
  DBMS_OUTPUT.Put ('           ');
  DBMS_OUTPUT.Put ('Muc luong');
  loop
     DBMS_OUTPUT.new_line;
     fetch nv into x, y;
      exit when nv%NotFound; /* if nv%NotFound  then exit; */
     DBMS_OUTPUT.Put (x);
     DBMS_OUTPUT.Put ('           ');
     DBMS_OUTPUT.Put (y);
  end loop;
  DBMS_OUTPUT.new_line;
  DBMS_OUTPUT.Put ('So records:');
  DBMS_OUTPUT.Put (nv%ROWCOUNT);
  DBMS_OUTPUT.new_line;
  Close nv;
end;
```

6**. Sử dụng con trỏ trong PL/SQL (16)

Ví dụ trên sử dụng lệnh while

```
Create Procedure Hien_Thi_Muc_Luong as
 x EMP.empno%TYPE;
 y EMP.sal%TYPE;
 cursor nv is select empno, sal from emp;
begin
  open nv;
 DBMS_OUTPUT.Put (‘So thu tu');
  DBMS_OUTPUT.Put ('Ma nhan vien');
  DBMS_OUTPUT.Put ('          ');
  DBMS_OUTPUT.Put ('Muc luong');
  while nv%Found
  loop
     DBMS_OUTPUT.new_line;
     fetch nv into x, y;
     DBMS_OUTPUT.Put (nv%rowcount);
     DBMS_OUTPUT.Put (x);
     DBMS_OUTPUT.Put ('          ');
     DBMS_OUTPUT.Put (y);
  end loop;
  DBMS_OUTPUT.new_line;
  DBMS_OUTPUT.Put ('So records:‘ || nv%ROWCOUNT );
end;
```

6**. Sử dụng con trỏ trong PL/SQL (17)

Ví dụ trên sử dụng lệnh for

```
Create Procedure Hien_Thi_Muc_Luong as
stt   number;
begin
  DBMS_OUTPUT.Put (‘So thu tu');
  DBMS_OUTPUT.Put ('Ma nhan vien');
  DBMS_OUTPUT.Put ('            ');
  DBMS_OUTPUT.Put ('Muc luong');
  stt:=0;
  for x in (select empno, sal from emp)
  loop
      stt:=stt+1;
      DBMS_OUTPUT.Put_line (stt ||'. ' || x.empno || '__' || x.sal);
  end loop;
  DBMS_OUTPUT.new_line;
  DBMS_OUTPUT.Put ('So records:' || stt );
end;
```

6*. Sử dụng con trỏ trong PL/SQL (18)

Ví dụ Procedure sử dụng con trỏ tiềm ẩn

```
Create Procedure Tang_Luong  As
    old_luong Float;
    new_luong Float;
Begin
    select sal into old_luong from emp where empno='7788';
    if SQL%FOUND then
          new_luong:=old_luong+old_luong*10/100;
          update emp set sal=new_luong where empno='7788';
          if SQL%ROWCOUNT<>0 then
            DBMS_OUTPUT.PUT_LINE('Luong nhan vien 7788
    duoc tang 10%');
          end if;
    end if;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Khong  tim thay nhan vien 7788');
END;
```

Ghi chú: Chạy 2 trường hợp để thấy kết quả, TH 1 như trên và TH 2 trong Procedure sửa lại empno thành 7789 thì khi chạy sẽ cho ra EXCEPTION vì ko có nhân viên này

6*. Sử dụng con trỏ trong PL/SQL (19)

Ví dụ Block sử dụng con trỏ tiềm ẩn

```
CREATE TABLE course (id NUMBER, name VARCHAR2(100));
CREATE TABLE student (id NUMBER, name VARCHAR2(100));
INSERT INTO student VALUES (1, 'Nguyên Văn A');
INSERT INTO course VALUES (1, 'Toán');
INSERT INTO course VALUES (2, 'Văn');
COMMIT;

DECLARE
  v_id NUMBER;
BEGIN
    DELETE FROM student WHERE id = 200;
    FOR i in 1..2 LOOP
      DELETE FROM course WHERE idCourse = i;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE( TO_CHAR(SQL%ROWCOUNT));
END;
```

Kết quả xuất ra là? (%rowcount -> số dòng bị tác động bởi câu SQL gần nhất)

6*. Sử dụng con trỏ trong PL/SQL (20)

Ví dụ: con trỏ tiềm ẩn + EXCEPTION sẵn có của Oracle

```
Create Procedure Kiem_Tra As
        p_manv nhanvien.manv%TYPE;
        p_hoten nhanvien.hoten%TYPE;
BEGIN
   select manv, hoten into p_manv, p_hoten from nhanvien;
EXCEPTION
 when Too_many_rows then
   DBMS_OUTPUT.Put_line('Tra ve nhieu records');
 when OTHERS then
    DBMS_OUTPUT.Put_line('Loi khong xac dinh');
END;
```

Ghi chú: Chạy 2 trường hợp để thấy kết quả, TH 1 tạo một bảng nhanvien chỉ gồm 2 cột manv và hoten, nhập từ 2 nhân viên trở lên. TH 2 xóa hết các nhân viên, chỉ chừa lại 1 người. (Không nên sử dụng bảng EMP trong Procedure này vì: việc nhập dữ liệu lại sẽ khó khăn khi đã xóa).

7. Khai báo hàm, thủ tục và ràng buộc (1)

(Function, Procedure, Trigger)

1. Khai báo Hàm (Function)

- **Hàm là một chương trình con có trả về giá trị. Hàm và thủ tục giống nhau, chỉ khác nhau ở chỗ hàm thì có mệnh đề RETURN.**
- **Cú pháp:**

```
CREATE [OR REPLACE] FUNCTION  tên-hàm
[(argument1 [, argument2,...])] RETURN datatype
IS
        [khai báo biến]
BEGIN
        <khối lệnh>
[EXCEPTION <xử lý ngoại lệ>]
END; /*kết thúc hàm*/
```

7. Khai báo hàm, thủ tục và ràng buộc (2)

(Function, Procedure, Trigger)

- **Datatype có thể là Number, Char hoặc Varchar2,….**
- **Từ khóa OR REPLACE để tự động xóa và tạo mới hàm nếu tên hàm đó đã tồn tại.**
  - **Ví dụ:**

    CREATE OR REPLACE FUNCTION Hien_Thi_Ngay (m number) RETURN VARCHAR2 IS ….
- **Không được dùng Varchar2(n) trong trị trả về (RETURN) lẫn trong đối số truyền vào (argument), kiểu dữ liệu trong đối số truyền vào và trong trị trả về phải là không ràng buộc n. (Khai báo hợp lệ là: Varchar2)**
- **Argument được xác định bởi:**

  tên-đối-số-truyền-vào [IN | OUT | IN OUT] kiểu-dữ-liệu [{ := | DEFAULT value}]

7. Khai báo hàm, thủ tục và ràng buộc (3)

(Function, Procedure, Trigger)

- **Ví dụ:**

```
CREATE FUNCTION Hien_Thi_Ngay (n NUMBER) RETURN CHAR
AS
    ngay CHAR(15);
BEGIN
     IF n =1 THEN
            ngay :='Sunday';
     ELSIF n =2 THEN
            ngay :='Monday';
     ELSIF n =3 THEN
            ngay :='Tuesday';
     ELSIF n =4 THEN
            ngay :='Wednesday';
     ELSIF n =5 THEN
            ngay :='Thursday';
     ELSIF n =6 THEN
            ngay :='Friday';
     ELSIF n =7 THEN
            ngay :='Saturday';
     END IF;
     RETURN ngay;
END;
```

7. Function – tham số IN

- **Ví dụ:**

Chạy block sau và cho nhận xét Function Hien_Thi_Ngay ở slide trước

```
declare
    x char(30);
    y number;
begin
    dbms_output.put_line('Su dung tham so in trong function');
    y:=4;
    x:=HIEN_THI_NGAY(y);
    dbms_output.put_line(y);
    dbms_output.put_line(x);
end;
```

7. Function – tham số OUT

- **Ví dụ: thay Function Hien_Thi_Ngay ở slide trước bằng một hàm khác (Hien_Thi_Ngay1) với tham số tô đỏ**

CREATE FUNCTION Hien_Thi_Ngay1 (n OUT NUMBER) RETURN CHAR AS

……

Chạy block sau và cho nhận xét so với Function Hien_Thi_Ngay ở slide trước

```
declare
    x char(30);
    y number;
begin
    dbms_output.put_line('Su dung tham so out trong function');
    y:=4;
    x:=HIEN_THI_NGAY1(y);
    dbms_output.put_line(y);
    dbms_output.put_line(x);
end;
```

7. Function – tham số OUT (tiếp tục)

- **Ví dụ: thay ví dụ Function Hien_Thi_Ngay bằng một hàm khác (Hien_Thi_Ngay2) với tham số tô đỏ và bổ sung đoạn code sau:**

```
ELSE
        n :=8; /*các trường hợp khác gán n=8*/
        ngay :='Khong xac dinh';
```

CREATE FUNCTION Hien_Thi_Ngay2 (n OUT NUMBER) RETURN CHAR AS…..

Chạy block sau và cho nhận xét so với Function Hien_Thi_Ngay1 ở slide trước

```
declare
    x char(30);
    y number;
begin
    dbms_output.put_line('Su dung tham so out trong function');
    y:=4;
    x:=HIEN_THI_NGAY2(y);
    dbms_output.put_line(y);
    dbms_output.put_line(x);
end;
```

7. Function – tham số IN OUT

- **Ví dụ: thay ví dụ Function Hien_Thi_Ngay bằng một hàm khác (Hien_Thi_Ngay3) với tham số tô đỏ và bổ sung đoạn code sau vào Trường hợp ELSIF n=4 then:**

  n :=100; /*gán n=100*/

CREATE FUNCTION Hien_Thi_Ngay2 (n IN OUT NUMBER) RETURN CHAR AS…..

Chạy block sau và cho nhận xét so với Function Hien_Thi_Ngay1 ở slide trước

```
declare
    x char(30);
    y number;
begin
    dbms_output.put_line('Su dung tham so in out trong function');
    y:=4;
    x:=HIEN_THI_NGAY3(y);
    dbms_output.put_line(y);
    dbms_output.put_line(x);
end;
```

7. Khai báo hàm, thủ tục và ràng buộc (4)

(Function, Procedure, Trigger)

- **Gọi hàm trong PL/SQL:**

- Đầu tiên khai báo biến có kiểu dữ liệu trùng với kiểu dữ liệu trị trả về của một hàm. Thực hiện lệnh sau:

- Ví dụ:

```
Declare
x   CHAR(20);
BEGIN
x:=Hien_Thi_Ngay(3);
/*Tổng quát:    biến:=Tên-hàm(danh sách đối số);*/
....
END;
```

- **Lệnh xóa hàm:** **DROP FUNCTION** **tên-hàm;**

7. Khai báo hàm, thủ tục và ràng buộc (5)

(Function, Procedure, Trigger)

2. Khai báo Thủ tục (Procedure)

- **Thủ tục là một chương trình con để thực hiện một hành động cụ thể nào đó. Hàm và thủ tục giống nhau, khác nhau ở chỗ hàm thì có mệnh đề RETURN.**
- **Cú pháp:**

```
CREATE [OR REPLACE] PROCEDURE  tên-thủ tục
[(parameter1 [, parameter2,...])]  IS
        [khai báo biến]
BEGIN
        <khối lệnh>
[EXCEPTION <xử lý ngoại lệ>]
END; /*kết thúc thủ tục*/
```

7. Khai báo hàm, thủ tục và ràng buộc (6)

(Function, Procedure, Trigger)

- **Từ khóa OR REPLACE để tự động xóa và tạo mới thủ tuc nếu tên thủ tục đó đã tồn tại.**
  - **Ví dụ:**

    CREATE OR REPLACE Hien_Thi_Ngay (m number) IS

    ....
- **Không được dùng Varchar2(n) trong đối số truyền vào (argument), kiểu dữ liệu trong đối số truyền vào phải là không ràng buộc n. (Khai báo hợp lệ là: Varchar2)**

- **Argument được xác định bởi:**

  tên-tham-số-truyền-vào [IN | OUT | IN OUT] kiểu-dữ-liệu [{ := | DEFAULT value}]

7*. Khai báo hàm, thủ tục và ràng buộc (7)

(Function, Procedure, Trigger)

- **Ví dụ:**

```
CREATE PROCEDURE Hien_Thi_Ngay (n NUMBER) IS
ngay CHAR(15);
BEGIN
    IF n =1 THEN
        ngay :='Sunday';
    ELSIF n =2 THEN
        ngay :='Monday';
    ELSIF n =3 THEN
        ngay :='Tuesday';
    ELSIF n =4 THEN
        ngay :='Wednesday';
    ELSIF n =5 THEN
        ngay :='Thursday';
    ELSIF n =6 THEN
        ngay :='Friday';
    ELSIF n =7 THEN
        ngay :='Saturday';
    END IF;
END;
/* tương tự chạy Procedure với các tham số OUT, IN OUT*/
```

7. Khai báo hàm, thủ tục và ràng buộc (8)

(Function, Procedure, Trigger)

- **Gọi thủ tục trong PL/SQL:**

- Ví dụ:

```
Declare
    ....
BEGIN
    Hien_Thi_Ngay(3);
    /*Tổng quát:   Tên-hàm(danh sách tham số);*/
    ....
END;
```

Lệnh xem code của Procedure

select * from user_source where name = '<proc name>' order by line;

- **Gọi thủ tục từ SQL*Plus:**

```
SQL> EXECUTE Hien_Thi_Ngay(6)
```

- **Cú pháp xóa thủ tục: DROP PROCEDURE tên-thủ-tục;**

7*. Khai báo hàm, thủ tục và ràng buộc (9)

(Function, Procedure, Trigger)

```
Create Procedure Insert_EMP   (v_EMPNO in varchar2, v_ENAME in varchar2,
v_HIREDATE in date, v_MGR in varchar2, v_SAL in varchar2)
As
       emp_cnt int;
Begin
      select count(*) into emp_cnt from EMP where EMPNO = v_EMPNO;
      if ( emp_cnt=1) then
                DBMS_Output.Put_line('Trung khoa chinh');/*tru`ng khoa chinh */
      else
                savepoint Point_1;
                insert into EMP (EMPNO, ENAME,HIREDATE, MGR, SAL) values (v_EMPNO,
     v_ENAME,v_HIREDATE,v_MGR, v_SAL) ;
                if SQL%ROWCOUNT = 0 then
                     DBMS_Output.Put_line('Xay ra loi giao tac');        /*loi khac*/
                     ROLLBACK to savepoint Point_1;
                else
                      DBMS_Output.Put_line('Them nhan vien thanh cong') ;
                      COMMIT ;
                end if;
      end if;
end;
```

Ghi chú: Chạy 2 lệnh sau trong SQL*Plus để thấy kết quả xử lý của 2 trường hợp này.

SQL> Exec Insert_EMP (`7788', `Nguyen Van A','2 Feb 2006', '7788',1000) ; ➔ trùng khóa chính 7788

SQL> Exec Insert_EMP (`7789', `Nguyen Van A','2 Feb 2006', '7788',1000) ; ➔ thêm mới nv 7789

7*.Ví dụ: Procedure sử dụng tham số IN, OUT

```
CREATE PROCEDURE P_Ngay (n IN NUMBER,m OUT NUMBER) IS
ngay CHAR(15);
BEGIN
    IF n =1 THEN
        ngay :='Sunday';
    ELSIF n =2 THEN
        ngay :='Monday';
    ELSIF n =3 THEN
        ngay :='Tuesday';
    ELSIF n =4 THEN
        ngay :='Wednesday';
    ELSIF n =5 THEN
        ngay :='Thursday';
    ELSIF n =6 THEN
        ngay :='Friday';
    ELSIF n =7 THEN
        ngay :='Saturday';
    END IF;
    m:=n;
    dbms_output.put_line('Ngay truyen vao:' || ngay);
END;
```

7*.Ví dụ: Procedure sử dụng tham số IN, OUT (tt)

Chạy 04 trường hợp sau, cho biết kết quả và nhận xét

```
declare
m number;
begin
 P_Ngay(5,m);
 dbms_output.put_line('Tham so ra:' || m);
end;
```

```
declare
a number;
b number;
begin
a:=4;
P_Ngay(a,b);
dbms_output.put_line(a);
dbms_output.put_line(b);
end;
```

```
declare
m number;
begin
 m:=7;
 P_Ngay(5,m);
 dbms_output.put_line('Tham so ra:' || m);
end;
```

```
declare
m number;
begin
 P_Ngay(5,7);   /* cho nhan xet???? */
 dbms_output.put_line('Tham so ra:');
end;
```

7. Khai báo hàm, thủ tục và ràng buộc (10)

(Function, Procedure, Trigger)

2. Khai báo ràng buộc (Trigger)

- **Trigger được dùng để khai báo các ràng buộc toàn vẹn phức tạp mà không thể khai báo ở cấp talbe.**
- **Cú pháp:**

```
CREATE [REPLACE] TRIGGER  tên-trigger
BEFORE|AFTER  INSERT/DELETE/UPDATE  ON tên-Table
[REFERENCING [NEW AS <new_row_name>] [OLD AS
    <old_row_name>]]
    [FOR EACH  ROW]
    DECLARE  /*Tùy thuộc bài toán có khai báo biến hay ko*/
        [khai báo biến]
    WHEN <điều kiện>
        Block-của-PL/SQL
```

7. Khai báo hàm, thủ tục và ràng buộc (11)

(Function, Procedure, Trigger)

- Từ khóa REPLACE để tự động xóa và tạo mới trigger nếu trigger đó đã tồn tại. Ví dụ: REPLACE TRIGGER Tên-Trigger
- table_name để chỉ đến tên của table muốn tạo trigger.

- **:NEW chỉ giá trị dòng mới insert/update, :OLD chỉ giá trị dòng mới vừa xóa (delete).** **Note: In the trigger body, NEW and OLD must be preceded by a colon (":"), but in the WHEN clause, they do not have a preceding colon!**

- INSERT | DELETE | UPDATE ứng với sự kiện tác động lên table để trigger tự động thi hành khi sự kiện đó xảy ra.
- Tùy chọn FOR EACH ROW để chỉ rằng trigger sẽ thi hành khi câu lệnh SQL tác động lên từng dòng.

7. Khai báo hàm, thủ tục và ràng buộc (12)

Phân loại Trigger và các thao tác trên trigger

- ✓ **Create Triggers: (TẠO TRIGGER)**

  1) Insert Triggers: gồm 2 loại

  BEFORE INSERT Trigger (You can update the :NEW values)
  AFTER INSERT Trigger (You can not update the :NEW values)

  2) Update Triggers: gồm 2 loại

  BEFORE UPDATE Trigger (You can update the :NEW values)
  AFTER UPDATE Trigger (You can not update the :NEW values)

  3) Delete Triggers: gồm 2 loại

  BEFORE DELETE Trigger (You can update the :NEW values)
  AFTER DELETE Trigger (You can not update the :OLD values)

- ✓ **Drop Triggers: (XÓA TRIGGER)**

  Drop a Trigger

- ✓ **Disable/Enable Triggers: (BẬT hoặc TẮT TRIGGER)**

  Disable a Trigger
  Disable all Triggers on a table
  Enable a Trigger
  Enable all Triggers on a table

7. Khai báo hàm, thủ tục và ràng buộc (13)

Ví dụ: Trigger INSERT: Before Insert

A BEFORE INSERT Trigger means that Oracle will fire this trigger before the INSERT operation is executed.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2),
  create_date date,
  created_by varchar2(10)
);
```

(You can update the :NEW values)

We could then create a BEFORE INSERT trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_before_insert
BEFORE INSERT ON orders
   FOR EACH ROW
    BEGIN
    -- Update create_date field to current system date
   :new.create_date := sysdate;
    -- Update created_by field to the username of the person performing the INSERT
   :new.created_by := user;
END;
```

7. Khai báo hàm, thủ tục và ràng buộc (14)

Ví dụ: Trigger INSERT: After Insert

An AFTER INSERT Trigger means that Oracle will fire this trigger after the INSERT operation is executed.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2)
);
```

(You can not update the :NEW values)

We could then create a AFTER INSERT trigger as follows:

```
CREATE OR REPLACE TRIGGER orders_after_insert
AFTER INSERT  ON orders
   FOR EACH ROW
BEGIN
   -- Find username of person performing the INSERT into the table -- Insert record into audit table
      INSERT INTO orders_audit
         (order_id, quantity, cost_per_item, total_cost, username )
      VALUES  ( :new.order_id, :new.quantity, :new.cost_per_item, :new.total_cost, user);
END;
```

7. Khai báo hàm, thủ tục và ràng buộc (15)

Ví dụ: Trigger UPDATE: Before update

A BEFORE UPDATE Trigger means that Oracle will fire this trigger before the UPDATE operation is executed.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2),
  updated_date date,
  updated_by varchar2(10)
);
```

(You can update the :NEW values)

We could then create a BEFORE UPDATE trigger:

```
CREATE OR REPLACE TRIGGER orders_before_update
BEFORE UPDATE  ON orders
   FOR EACH ROW
BEGIN
   -- Find username of person performing UPDATE on the table -- Update updated_date field to current system date
   :new.updated_date := sysdate;
    -- Update updated_by field to the username of the person performing the UPDATE
   :new.updated_by := user;
END;
```

7. Khai báo hàm, thủ tục và ràng buộc (16)

Ví dụ: Trigger UPDATE: After update

An AFTER UPDATE Trigger means that Oracle will fire this trigger after the UPDATE operation is executed.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2)
);
```

(You can not update the :NEW values)

We could then create a AFTER UPDATE trigger:

```
CREATE OR REPLACE TRIGGER orders_after_update
AFTER UPDATE  ON orders
   FOR EACH ROW
BEGIN
   -- Find username of person performing UPDATE into table -- Insert record into audit table
   INSERT INTO orders_audit
   ( order_id, quantity_before, quantity_after, username)
   VALUES
   ( :new.order_id, :old.quantity,:new.quantity,user);
END;
```

7. Khai báo hàm, thủ tục và ràng buộc (17)

Ví dụ: Trigger DELETE: Before Delete

A BEFORE DELETE Trigger means that Oracle will fire this trigger before the DELETE operation is executed.

If you had a table created as follows:

```
CREATE TABLE orders
( order_id number(5),
  quantity number(4),
  cost_per_item number(6,2),
  total_cost number(8,2)
);
```

(You can update the :NEW values, in this case it's not necessary)

We could then create a BEFORE DELETE trigger:

```
CREATE OR REPLACE TRIGGER orders_before_delete
BEFORE DELETE   ON orders
   FOR EACH ROW
BEGIN

   -- Find username of person performing the
DELETE on the table -- Insert record into audit table

   INSERT INTO orders_audit

   ( order_id, quantity, cost_per_item, total_cost,
delete_date, deleted_by )

   VALUES

   ( :old.order_id, :old.quantity, :old.cost_per_item,
:old.total_cost, sysdate, user);

END;
```

7. Khai báo hàm, thủ tục và ràng buộc (18)

(Function, Procedure, Trigger)

Chú ý khi tạo trigger:

- Phần thân trigger có thể chứa các lệnh DML, nhưng lệnh SELECT phải là SELECT INTO ngoại trừ lệnh SELECT khi khai báo cursor.
- DDL không được dùng trong phần thân của trigger.
- Không cho phép các lệnh quản lý giao tác (COMMIT, ROLLBACK, SAVEPOINT) trong phần thân của trigger.
- Nếu trigger gọi một chương trình con thì chương trình con đó không được chứa các lệnh quản lý giao tác.

7. Khai báo hàm, thủ tục và ràng buộc (19)

(Function, Procedure, Trigger)

Thao tác trigger: DISABLE và ENABLE

- ALTER TRIGGER tên-trigger DISABLE;

  Để disable tất cả các trigger liên quan đến một table cụ thể, dùng lệnh:

  ALTER TABLE table_name DISABLE ALL TRIGGERS;

- Lệnh enable một trigger

  ALTER TRIGGER trigger_name ENABLE;

- Để enable tất cả các trigger liên quan đến một table cụ thể, dùng lệnh:

  ALTER TABLE table_name ENABLE ALL TRIGGERS;

- Cú pháp xóa trigger: DROP TRIGGER Tên-trigger;

7*. Khai báo hàm, thủ tục và ràng buộc (20)

(Function, Procedure, Trigger)

```
Create Trigger Tang_Bonus AFTER INSERT ON emp
FOR EACH ROW
declare
    v_sal  EMP.SAL%TYPE;
Begin
    if :new.SAL IS NOT NULL then
        /*trich 10% luong cua nguoi moi vao*/
        /*Note: In the trigger body, NEW and OLD must be preceded by a colon (":"), but in the WHEN clause, they do not have a preceding colon! */
        v_sal:= :new.Sal*10/100;
        /*bonus cho nguoi quan ly = 10% luong nguoi moi vao*/
        insert into BONUS (empno, sal) values (:new.MGR,v_sal) ;
    End if;
End;
```

Ghi chú: Trước khi tạo Trigger, mở bảng BONUS của user SCOTT sửa lại cột Ename thành Empno và đổi kiểu dữ liệu tương ứng. Chạy lệnh sau:

SQL> Exec Insert_EMP (`7790', `Nguyen Van B','2 Feb 2006', '7788',1000) ; ➔ thêm nhân viên mới ➔ràng buộc được thực hiện ➔ kết quả nhân viên 7788 được thêm bonus là 100 (table BONUS).

7*. Khai báo hàm, thủ tục và ràng buộc (21)

(Function, Procedure, Trigger)

Aborting Triggers with Error

The WHEN clause or body of the trigger can check for the violation of certain conditions and signal
an error accordingly using the Oracle built-in function RAISE_APPLICATION_ERROR.
The action that activated the trigger (insert, update, or delete) would be aborted.
For example, the following trigger enforces the constraint Person.age >= 0:
create table Person (age int);

```
CREATE TRIGGER PersonCheckAge
AFTER INSERT OR UPDATE OF age ON Person
FOR EACH ROW
BEGIN
    IF (:new.age < 0) THEN
        RAISE_APPLICATION_ERROR(-20000, 'no negative age allowed');
    END IF;
END;
```

7*. Khai báo hàm, thủ tục và ràng buộc (22)

(Function, Procedure, Trigger)

Aborting Triggers with Error

```
CREATE OR REPLACE TRIGGER PersonCheckAge
AFTER INSERT OR UPDATE OF Ngaysinh ON NHANVIEN
FOR EACH ROW
BEGIN
    IF (to_number(extract(year from sysdate)) -
     to_number(extract(year from :new.ngaysinh))<=18) THEN
        RAISE_APPLICATION_ERROR(-20000, 'no negative age
     allowed');
    END IF;
END;
```

7*. Trigger: Mutating Trigger (1)

- Mutating Table Errors
- Sometimes you may find that Oracle reports a "mutating table error" when your trigger executes. This happens when the trigger is querying or modifying a "mutating table", which is either the table whose modification activated the trigger, or a table that might need to be updated because of a foreign key constraint with a CASCADE policy. To avoid mutating table errors:

- * A row-level trigger must not query or modify a mutating table. (Of course, NEW and OLD still can be accessed by the trigger.)
- * A statement-level trigger must not query or modify a mutating table if the trigger is fired as the result of a CASCADE delete.

7*. Trigger: Mutating Trigger (2)

Mutating Trigger Demo

The insert into t1 firest the trigger which attempts to count the number of records in t1 ... **which is ambiguous.**

```
CREATE TABLE t1 (x int);
CREATE TABLE t2 (x int);
INSERT INTO t1 VALUES (1);

CREATE OR REPLACE TRIGGER t_trigger AFTER INSERT
ON t1
FOR EACH ROW

DECLARE
    i INTEGER;
BEGIN
    SELECT COUNT(*)  INTO i   FROM t1;
    INSERT INTO t2 VALUES (i);
END;
```

Tạo trigger, sau đó chạy lệnh INSERT INTO t1 VALUES (2); **cho nhận xét???**

7*. Trigger: Mutating Trigger (3)

Fix Mutating Trigger With Autonomous Transaction

Count on t1 is performed as though a different user logged on and asked the question of t1

```
CREATE OR REPLACE TRIGGER t_trigger AFTER INSERT
ON t1
FOR EACH ROW

DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
    i INTEGER;
BEGIN
    SELECT COUNT(*)  INTO i   FROM t1;
    INSERT INTO t2 VALUES (i);
    COMMIT;
END;
```

Sửa lại Trigger trên, chạy lệnh INSERT INTO t1 VALUES (2); **cho nhận xét???**

```
SELECT COUNT(*) FROM t1;
SELECT COUNT(*) FROM t2;
```