PL/SQL

# Giới thiệu chung

PL/SQL ( Procedural Language / SQL ) là một ngôn ngữ lập trình dùng để truy nhập vào CSDL Oracle từ nhiều môi trường khác nhau. PL/SQL được tích hợp trên Server CSDL cho nên nó có thể được thực hiện nhanh và hiệu quả.

Các đặc điểm của PL/SQL :

* Cấu trúc khối: đơn vị cơ bản của PL/SQL là một khối. Tất cả những chương trình PL/SQL được xây dựng từ những khối. Mỗi khối là một đơn vị công việc logic trong một chương trình. Cấu trúc của một khối như sau :

DECLARE

/\* Phần khai báo - các biến, kiểu, cursor và chương trình con \*/

BEGIN

/\* Phần thực hiện - các thủ tục và câu lệnh SQL.Đây là phần chính và bắc buộc phải có. \*/

EXCEPTION

/\* Phần kiểm soát lỗi. \*/

END;

* Biến và kiểu: PL/SQL hỗ trợ các biến và kiểu, nó cũng cho phép các kiểu người dùng tự định nghĩa: tables và record.
* Cấu trúc vòng lặp và rẽ nhánh: if.. then..else, for, loop………
* Cursor: được dùng để thao tác với nhiều hàng dữ liệu lấy từ CSDL (dùng câu lệnh Select). Bằng cách sử dụng cursor, chương trình có thể duyệt một cách dễ dàng toàn bộ hàng dữ liệu.

# Cấu trúc

PL/SQL là ngôn ngữ cấu trúc khối (block-structure). Đơn vị cơ bản trong mỗi chương trình PL/SQL là khối. Tất cả các chương trình PL/SQL đều được hợp thành từ những khối. Các khối có thể nằm lần lượt hoặc lồng nhau. Mỗi khối liên quan đến một vấn đề hoặc một vấn đề con cần được giải quyết. Do đó, PL/SQL hỗ trợ cách tiếp cận giải quyết vấn đề theo kiểu "chia để trị".

Có một vài kiểu khối bao gồm :

* Khối không tên (Anonymous block): thường được xây dựng động và được thực hiện duy nhất 1 lần
* Khối có tên (Named block) : là những khối không tên với nhãn được gán cho tên của khối.
* Chương trình con (Subprogram) : là những thủ tục (procedures), gói (package) và hàm (function) được lưu trong CSDL. Nói chung những khối này thường không thay đổi một khi nó đã được xây dựng và chúng được thực hiện nhiều lần. Thực hiện chương trình con bằng lời gọi tới thủ tục, gói hoặc hàm cần thực hiện.
* Trigger: tương tự như chương trình con, chúng cũng được lưu trong CSDL và được thực hiện nhiều lần, đồng thời ít thay đổi sau khi tạo ra. Trigger được thực hiện mỗi khi có một sự kiện nhất định xảy ra

Trong mỗi khối của một chương trình PL/SQL đều có những phần tách biệt nhau : phần khai báo, phần thực hiện và phần kiểm soát lỗi. Chỉ có phần thực hiện là bắt buộc phải có, còn 2 phần kia có thể có hoặc không:

* Phần khai báo: nơi mà tất cả các biến, cursors và các kiểu dùng trong khối được khai báo. Những hàm và thủ tục địa phương cũng có thể được khai báo tại đây.
* Phần thực hiện là phần chính của khối, là nơi thực hiện công việc của khối. Phần này bao gồm các câu lệnh SQL và những câu lệnh gọi thủ tục.
* Lỗi được kiểm soát trong phần kiểm soát lỗi. Mã chương trình trong phần này chỉ được thực hiện khi có lỗi xảy ra.

Những từ khóa DECLARE, BEGIN, EXCEPTION và END ngăn cách những khối với nhau.

* Khối không tên (Anonymous block) :

DECLARE

/\* Bắt đầu phần khai báo \*/

v\_StudentID NUMBER(5) := 10000;

v\_FirstName VARCHAR2(20);

BEGIN

/\* Bắt đầu phần thực hiện \*/

-- Tìm tên của sinh viên có ID = 10,000

SELECT first\_name

INTO v\_FirstName

FROM students

WHERE id= v\_StudentID;

EXCEPTION

/\* Bắt đầu phần kiểm soát lỗi \*/

WHEN NO\_DATA\_FOUND THEN

INSERT INTO log\_table (info) VALUES ('Sinh viên không tồn tại')

END;

* Khối có tên (Named block)

Để gán tên cho khối này, ta đặt một nhãn trước từ khoá DECLARE như sau :

<<FindNameStudent>>

DECLARE

..............

BEGIN

..............

END FindNameStudent;

* Để tạo thành một Store Procedure từ khối này, ta thay từ khóa DECLARE bằng từ khoá CREATE OR REPLACE PROCEDURE.

CREATE OR REPLACE PROCEDURE FindNameStudent AS

/\* Bắt đầu phần khai báo \*/

.........................................

BEGIN

/\* Bắt đầu phần thực hiện \*/

.........................................

END FindNameStudent;

* Cuối cùng, cấu trúc một Trigger có dạng như sau :

CREATE OR REPLACE TRIGGER OnlyPositive

BEFORE INSERT OR UPDATE OF num\_col

ON Temp\_Table

FOR EACH ROW

BEGIN

IF :new.num\_col < 0 THEN

RAISE\_APPLICATION\_ERROR(-20100,'Nhập vào giá trị dương')

END IF;

END OnlyPositive;

Trigger này thực hiện trên bảng Temp\_table, được gọi mỗi khi một hàng mới được chèn vào trong bảng Temp\_table hoặc một hàng trong Temp\_table được cập nhập để đảm bảo mọi giá trị trong trong trường num\_col đều là số dương.

# **Các thành phần**

## **PL/SQL Variables**

**Cú pháp:**

variable\_name datatype [NOT NULL := value ];

* + *variable\_name* là tên của biến.
  + *datatype* là kiểu của biến ( valid PL/SQL datatype )
  + NOT NULL : optional : ko cho phép biến này rỗng
  + *value*: optional : giá trị khởi tạo của biến.

Mỗi một biến được khai báo trên một dòng. Ví dụ:

DECLARE

salary number (6);

\* “salary” là một biến kiểu number và có độ dài là 6 ký tự

Khi thêm tham số NOT NULL thì phải gán giá trị ngay cho nó. Ví dụ:

DECLARE

salary number(4);

dept varchar2(10) NOT NULL := “HR Dept”;

**Gán giá trị**

Có 2 cách

1. Gán giá trị thực tiếp

variable\_name:= value;

1. Gán giá trị gián tiếp bằng câu lệnh SELECT.. INTO. Cú pháp :

SELECT column\_name

INTO variable\_name

FROM table\_name

[WHERE condition];

Ví dụ: Gán giá trị vào biến *var\_salary*

DECLARE

var\_salary number(6);

var\_emp\_id number(6) = 1116;

BEGIN

SELECT salary

INTO var\_salary

FROM employee

WHERE emp\_id = var\_emp\_id;

dbms\_output.put\_line(var\_salary);

dbms\_output.put\_line('The employee '

|| var\_emp\_id || ' has salary ' || var\_salary);

END;

/

VARCHAR2(20);

Phạm vi của biến bắt đầu từ phần khai báo biến đến cuối khối. Khi một biến ra khỏi phạm vi của nó, PL/SQL sẽ giải phóng bộ nhớ chứa biến này và coi như biến này không còn tồn tại.

DECLARE

v\_Number NUMBER(3,2)

BEGIN

DECLARE

v\_Character VARCHAR2(10) BEGIN

................

END;

END;

Phạm vi biến v\_Number

Phạm vi biến v\_Character

## PL/SQL Constants

**Cú pháp**:

|  |
| --- |
| constant\_name CONSTANT datatype := VALUE; |

*constant\_name là tên của hằng số*

Giá trị: bắt buộc phải gán giá trị trực tiếp

Ví dụ:

DECLARE

salary\_increase CONSTANT number (3) := 10;

## PL/SQL Records

**Record** là 1 loại kiểu dữ liệu được người sử dụng định nghĩa. Nó có thể được kết hợp bởi nhiều kiểu khác nhau.

**Cú pháp:**

TYPE record\_type\_name IS RECORD

( first\_col\_name table\_name.column\_name%type,

second\_col\_name column\_datatype, ...);

*record\_type\_name*: tên của record muốn định nghĩa .

*first\_col\_name, second\_col\_name*: kiểu của column

Định nghĩa kiểu:

* Theo từng kiểu của column. Cú pháp :

TYPE employee\_rec IS RECORD

( first\_col\_name table\_name.column\_name%type,…)

Ví dụ:

DECLARE

TYPE employee\_type IS RECORD

(employee\_id number(5),

employee\_first\_name varchar2(25),

employee\_last\_name employee.last\_name%type,

employee\_dept employee.dept%type);

employee\_salary employee.salary%type;

employee\_rec employee\_type;

* Hoặc theo tất cả các kiểu colum của bản. Cú pháp:

DECLARE

employee\_rec employee%ROWTYPE;

Làm thế này thì có 2 ưu điểm

* + Code ngắn gọn
  + Ko cần update code nếu như alter table

Nhược điểm :

* + Tốn bộ nhớ

**Gán giá trị cho Record**

Cú pháp:

record\_name.col\_name := value;

record\_name.column\_name := value;

Cũng có thể truyền giá trị cho record bằng câu lệnh Select. Ví dụ:

SELECT col1, col2

INTO record\_name.col\_name1, record\_name.col\_name2

FROM table\_name

[WHERE clause];

Nếu dùng %ROWTYPE thì có thể sử dụng select \* như sau:

SELECT \* INTO record\_name

FROM table\_name

[WHERE clause];

**Lấy giá trị từ record**

Cú pháp:

var\_name := record\_name.col\_name;

## PL/SQL Biểu thức điều kiện - vòng lặp

Ngôn ngữ PL/SQL có hỗ trợ tính năng biểu thức điều kiện và vòng lặp:

**Biểu thức điều kiện**

IF THEN ELSE STATEMENT

1)

IF condition

THEN

statement 1;

ELSE

statement 2;

END IF;

2)

IF condition 1

THEN

statement 1;

statement 2;

ELSIF condtion2 THEN

statement 3;

ELSE

statement 4;

END IF

3)

IF condition 1

THEN

statement 1;

statement 2;

ELSIF condtion2 THEN

statement 3;

ELSE

statement 4;

END IF;

4)

IF condition1 THEN

ELSE

IF condition2 THEN

statement1;

END IF;

ELSIF condition3 THEN

statement2;

END IF;

**Vòng lặp**

Có 3 kiểu vòng lặp trong PL/SQL

* Simple Loop
* While Loop
* For Loop

**Simple loop**: sử dụng khi mệnh đề lặp cần sử dụng ít nhất 1 lần

* Cú pháp:

LOOP

statements;

EXIT;

{or EXIT WHEN condition;}

END LOOP;

**While loop**: sử dụng khi mệnh đề lặp khi điều kiện lặp là đúng:

* Cú pháp:

WHILE <condition>

LOOP statements;

END LOOP;

**For loop:** sử dụng khi mệnh đề lặp thực hiện theo mỗi đơn vị của biến đếm. Biến đếm mặc định sẽ tăng 1: val1 ->val2

* Cú pháp

FOR counter IN val1..val2

LOOP statements;

END LOOP;

## PL/SQL Cursor

Để xử lý một câu lệnh SQL, Oracle tạo ra một vùng nhớ gọi là vùng ngữ cảnh (context area). Vùng ngữ cảnh chứa những thông tin cần thiết để để hoàn thành một quá trình, bao gồm số hàng được xử lý bởi câu lệnh, con trỏ tới câu lệnh và trong trường hợp một query là một tập hợp các hàng được trả về bởi query đó (active set).

Cursor là một thẻ (handle) hoặc một con trỏ (pointer) trỏ tới vùng ngữ cảnh. Thông qua cursor, một chương trình PL/SQL có thể điều khiển vùng ngữ cảnh.

4 bước cần thiết để xử lý cursor :

* Khai báo cursor
* Mở cursor để thực hiện query.
* Đưa kết quả vào biến PL/SQL
* Đóng cursor

**Câu lệnh khai báo cursor:**

CURSOR tên\_cursor IS Câu\_lệnh\_SELECT

Ví dụ :

DECLARE

v\_Departement Classes.department%TYPE

v-Course Classes.course%TYPE

CURSOR c\_Classes IS

SELECT \* From classes

WHERE department = v\_Departement

AND course = v\_Course;

**Câu lệnh mở Cursor**

OPEN tên\_cursor;

Sau lệnh này, tập các hàng dữ liệu được xác định và con trỏ sẽ trỏ tới phần tử đầu tiên của tập này.

**Câu lệnh đưa kết quả vào biến PL/SQL**

#Có 2 dạng như sau :

FETCH tên\_cursor INTO danh\_sách\_biến;

FETCH tên\_cursor INTO PL/SQL\_record;

Sau mỗi lệnh FETCH, con trỏ được trỏ tới hàng tiếp theo của tập dữ liệu.

Câu lệnh đóng cursor

CLOSE tên\_cursor;

Có 4 thuộc tính được áp dụng cho cursor :

* %FOUND : kiểu BOOL. Trả lại giá trị TRUE nếu câu lệnh FETCH trước đó trả về một hàng dữ liệu, ngược lại nhận giá trị FALSE
* %NOTFOUND : Ngược với %FOUND.
* %ISOPEN : Trả lại TRUE nếu cursor đang mở, ngược lại nhận giá trị FALSE.
* %ROWCOUNT : Trả lại số hàng trong tập dữ liệu.
* Cũng có thể sử dụng %TYPE và %ROWTYPE để xác định kiểu dữ liệu.

Thao tác cơ bản nhất đối với cursor là thực hiện lấy dữ liệu từ tập dữ liệu (active set). Việc này thường được thực hiện nhờ vòng lặp fetch loop. Vòng lặp này sẽ xử lý đối với từng hàng trong tập dữ liệu (active set).

Các vòng lặp được sử dụng :

* Simple Loop
* While Loop
* Cursor For Loop

PL/SQL cung cấp cú pháp rất thuận tiện để thực hiện việc thay đổi dữ liệu trong một hàng khi thu nhận được bởi cursor. Đó là câu lệnh FOR UPDATE và WHERE CURRENT OF.

Cách sử dụng FOR UPDATE được minh họa qua ví dụ sau :

CURSOR c\_AllStudents IS

SELECT \*

FROM students

FOR UPDATE OF first\_name, last\_name;

Qua khai báo như này, ta có thể sửa đổi dữ liệu trường first\_name, last\_name trong bảng students thông qua cursor c\_AllStudents

Nếu cursor được khai báo cùng với mệnh đề FOR UPDATE thì mệnh đề WHERE CURRENT OF có thể được dùng trong câu lệnh UPDATE hoặc DELETE. Cú pháp:

WHERE CURRENT OF cursor

Mệnh đề này chỉ ra hàng dữ liệu vừa được nhận từ cursor. Xét ví dụ sau :

DECLARE

v\_Numcredits classes.num\_credits%TYPE;

/\* Cursor chọn ra những sinh viên hiện tại đã đăng ký cho HIS 101 \*/

CURSOR c\_RegisteredStudents IS

SELECT \*

FROM Students

WHERE id IN (SELECT student\_id FROM registered\_students

WHERE department = HIS AND course = 101)

FOR UPDATE OF current\_credits;

BEGIN

-- Tạo vòng lặp

FOR v\_StudentInfo IN c\_RegisteredStudents LOOP

/\* Xác định số lượng credits của HIS 101 \*/

SELECT num\_credits

INTO v\_NumCredits

FROM classes

WHERE department = HIS AND course = 101;

/\* UPDATE những hàng dữ liệu vừa nhận được bởi cursor \*/

/\* Câu lệnh này chỉ được thực hiện khi có khai báo FOR UPDATE OF trong câu lệnh khai báo cursor \*/

UPDATE students

SET current\_credits = current\_credits + v\_NumCredits

WHERE CURRENT OF c\_RegisteredStudents;

END LOOP

COMMIT;

END;

**Biến cursor (Cursor Variable)**

Biến cursor là kiểu tham chiếu tương tự như con trỏ (pointer) trong ngôn ngữ lập trình Pascal hay C. Khai báo như sau :

TYPE tên\_kiểu IS REF CURSOR RETURN kiểu\_trả\_về;

Ví dụ :

-- Khai báo sử dụng %ROWTYPE

TYPE t\_StudentsRef IS REF CURSOR RETURN students%ROWTYPE;

-- Khai báo 1 kiểu record mới

TYPE t\_NameRecord IS RECORD (

first\_name students.first\_name%TYPE,

last\_name students.last\_name%TYPE);

--Khai báo biến cursor sử dụng kiểu record này

TYPE t\_NameRef ID REF CURSOR RETURN t\_NameRecord;

-- Khai báo những biến sử dụng các kiểu trên

v\_StudentCV t\_StudentsRef;

v\_NameCV t\_NamesRef;

Để mở một biến Cursor, thực hiện theo câu lệnh :

OPEN biến\_cursor FOR câu\_lệnh\_SELECT;

Các thao tác khác trên biến cursor thực hiện tương tự như đối với cursor tĩnh.

## PL/SQL Procedure

Chương trình con được tạo ra nhờ câu lệnh CREATE OR REPLACE và có thể được gọi từ những khối PL/SQL khác. Một chương trình con cũng có thể được định nghĩa trong vùng khai báo của một khối, trong trường hợp này ta gọi nó là local subprogram (chương trình con địa phương).

Có 3 cách để truyền tham số vào procedures:

* IN - parameters : tham số vào chỉ được đọc
* OUT - parameters : tham số vào chỉ được ghi vào
* IN OUT – parameters: tham số vao vừa được đọc và ghi

|  |  |
| --- | --- |
| Kiểu | Mô tả |
| IN | Giá trị hiện tại của tham số được truyền vào trong thủ tục khi thủ tục đó được gọi. Bên trong thủ tục, dạng của tham số được coi là read-only, nghĩa là không thể được thay đổi. Khi kết thúc thủ tục và quyền điều khiển được trả về cho môi trường , giá trị của tham số không đổi. |
| OUT | Mọi giá trị của tham số khi bắt đầu lời gọi thủ tục đều bị bỏ qua. Trong thủ tục, dạng của tham số được coi là write-only. Khi kết thúc thủ tục, nội dung của tham số trong thủ tục được chuyển lại cho môi trường hiện tại. |
| IN OUT | Kiểu này kết hợp cả hai kiểu IN và OUT. Giá trị của tham số hiện tại được chuyển vào trong thủ tục khi thủ tục được gọi. Bên trong thủ tục, tham số này có thể được đọc và ghi. Khi kết thúc thủ tục, nội dung của tham số trong thủ tục được chuyển lại cho môi trường hiện tại. |

Cú pháp:

CREATE [OR REPLACE] PROCEDURE proc\_name [list of parameters]

IS

Declaration section

BEGIN

Execution section

END;

Ví dụ:

1> CREATE OR REPLACE PROCEDURE employer\_details

2> IS

3> CURSOR emp\_cur IS

4> SELECT first\_name, last\_name, salary FROM emp\_tbl;

5> emp\_rec emp\_cur%rowtype;

6> BEGIN

7> FOR emp\_rec in sales\_cur

8> LOOP

9> dbms\_output.put\_line(emp\_cur.first\_name || ' ' ||emp\_cur.last\_name

10> || ' ' ||emp\_cur.salary);

11> END LOOP;

12>END;

13> /

Thực thi procedure:

SQL> EXECUTE [or EXEC] procedure\_name;

## PL/SQL **Function**

Tương tự procedure, nhưng function khi thực hiện xong sẽ bắt buộc trả lại kết quả

Cú pháp:

CREATE [OR REPLACE] FUNCTION function\_name [parameters]

RETURN return\_datatype;

IS

Declaration\_section

BEGIN

Execution\_section

Return return\_variable;

END;

Ví dụ :

1> CREATE OR REPLACE FUNCTION employer\_details\_func

2> RETURN VARCHAR(20);

3> IS

5> emp\_name VARCHAR(20);

6> BEGIN

7> SELECT first\_name INTO emp\_name

8> FROM emp\_tbl WHERE empID = '100';

9> RETURN emp\_name;

10> END;

11> /

Thực thi fuction:

* Gán kết quả của fuction cho biến

employee\_name := employer\_details\_func;

* Bằng câu lệnh select

SQL> SELECT employer\_details\_func FROM dual;

* Hiện thị ra màn hình

SQL> set serveroutput on

SQL> dbms\_output.put\_line(employer\_details\_func);

## **PL/SQL Exception**

Trong PL/SQL có 2 loại lỗi chính là :

* Compile-time : lỗi phát sinh khi dịch
* Run-time : lỗi phát sinh khi chạy.

PL/SQL kiểm soát lỗi thông qua exceptions và exception handlers. Exception được thiết kế để xử lý những lỗi dạng Run-time, còn đối với những lỗi phát sinh khi chạy sẽ được phát hiện bởi PL/SQL và trả về thông báo cho người sử dụng.

Khi có một lỗi phát sinh, một exception được bật lên, quyền điều khiển được chuyển sang phần kiểm soát lỗi (exception handler). Nhờ việc thiết kế phần kiểm soát lỗi một cách độc lập nên tính logic của chương trình dễ hiểu hơn, đồng thời nó cũng bảo đảm tất cả mọi lỗi sẽ được kiểm soát.

**Khai báo Exception**

Exception được khai báo trong phần khai báo của khối, được kích hoạt trong phần thực hiện và được xử lý trong phần kiểm soát lỗi. Có 2 kiểu exception là : user-defined (người dùng định nghĩa) và predefined (định nghĩa trước).

* User-defined exception: Lỗi được định nghĩa bởi chương trình. Nó được khai báo trong phần khai báo của khối tương tự như một biến có kiểu Exception
* Predefined exception: Oracle đã định nghĩa sẵn những lỗi chung hay gặp, chúng được gọi là Predefined exception. Ví dụ :

INVALID\_CURSOR: Thao tác trên cursor sai

NOT\_LOGGED\_ON : Không thể kết nối với Oracle

TOO\_MANY\_ROWS : Câu lệnh SELECT ... INTO trả về hơn một dòng dữ liệu.

Để thực hiện bắt lỗi thông qua User-defined excepion, cần thực hiện như sau :

DECLARE

A EXCEPTION; // Khai báo exception A tại đây.

BEGIN

......

-- Khi có lỗi A, thực hiện câu lệnh này để kích hoạt lỗi

-- Quyền điều khiển được chuyển về cho phần EXCEPTION

RAISE A;

-- Mã chương trình ở đây sẽ không được thực hiện khi có lỗi A

......

EXCEPTION

WHEN A THEN

-- Mã chương trình ở đây sẽ được thực hiện khi có lỗi A

.......

WHEN OTHERS THEN

....... // Những lỗi còn lại sẽ được thực hiện trong phần này.

END;

Khi sử dụng OTHERS Exception sẽ không trả lại lỗi cụ thể, để xác định xem lỗi nào đã xảy ra, sử dụng các hàm định nghĩa sẵn trong PL/SQL bao gồm: SQLCODE và SQLERRM. SQLCODE trả về mã lỗi hiện tại và SQLERRM trả về thông báo lỗi hiện tại.

Sử dụng EXCEPTION\_INIT Pragma cho phép bẫy những lỗi đặc biệt :

PRAGMA EXCEPTION\_INT (Tên\_exception, mã\_lỗi\_oracle)

Ví dụ :

DECLARE

e\_MissingNull EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_MissingNull, -1400);

BEGIN

INSERT INTO students(id) VALUES (NULL);

EXCEPTION

WHEN e\_MissingNull then

INSERT INTO log\_table(info) VALUES (lỗi ORA-1400 );

END;

Người sử dụng cũng có thể tạo ra một thông báo lỗi riêng của mình thông qua lệnh :

**RAISE\_APPLICATION\_ERROR(error\_number,error\_message,[keep\_error]);**

* error\_number : tham số nằm trong khoảng –20,000 và -20,99
* error\_message : thông báo lỗi
* keep\_error : giá trị Boolean. Nhận giá trị TRUE khi một lỗi mới được thêm vào danh sách lỗi đã tồn tại, bằng FALSE (mặc định) khi lỗi mới thay thế lỗi đã tồn tại trong danh sách lỗi.

Như vậy chúng ta đã nghiên cứu phương pháp bắt lỗi trong phần thực hiện của khối. Tuy nhiên, trong nhiều trường hợp lỗi có thể xảy bất kỳ đâu trong khối : trong phần khai báo, trong phần thực hiện hay ngay trong phần kiểm soát lỗi.

**Exception xuất hiện trong phần thực hiện của khối**

Các bước để xác định lỗi như sau :

* Nếu trong khối hiện tại có một handler cho exception này, thực hiện nó và khối kết thúc một cách thành công
* Nếu trong khối không có handler cho exception hiện tại, chuyển quyền điều khiển lỗi cho khối cha của nó (khối bao lấy khối này) (enclosing block).

**Exception xuất hiện trong phần khai báo của khối**

Nếu có lỗi trong phần khai báo tạo nên một exception, exception này lập tức được chuyển đến cho khối cha của nó, thậm chi ngay cả khi có một điều khiển lỗi trong khối hiện tại nó cũng không được thực hiện.

**Exception xuất hiện trong phần kiểm soát lỗi**

Exception có thể được kích hoạt ngay trong phần kiểm soát lỗi thông qua câu lệnh RAISE hoặc bởi một lỗi khi chạy. Trong cả hai trường hợp đó, exception lập tức được chuyển đến cho khối cha của nó giống như khi exception xuất hiện trong phần khai báo.Lý do vì PL/SQL không cho phép nhiều hơn một exception kích hoạt đồng thời.

Một thói quen tốt khi lập trình là tránh những exception không được kiểm soát, điều này

## **PL/SQL Trigger**

Trigger là 1 khối pl/sql được tự kích hoạt khi có một sự kiện nhất định xảy ra.

Ví dụ như insert, delete, update vào 1 bảng của database…

Cú pháp:

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

BEGIN

--- sql statements

END;

Ví dụ:

#Tạo 2 bảng mẫu để test

SQL>

CREATE TABLE product\_price\_history

(product\_id number(5),

product\_name varchar2(32),

supplier\_name varchar2(32),

unit\_price number(7,2) );

SQL> CREATE TABLE product

(product\_id number(5),

product\_name varchar2(32),

supplier\_name varchar2(32),

unit\_price number(7,2) );

#Tạo Trigger để thực thi

SQL> CREATE or REPLACE TRIGGER price\_history\_trigger

BEFORE UPDATE OF unit\_price

ON product

FOR EACH ROW

BEGIN

INSERT INTO product\_price\_history

VALUES

(:old.product\_id,

:old.product\_name,

:old.supplier\_name,

:old.unit\_price);

END;

/

#Thực thi update dữ liệu

SQL> UPDATE PRODUCT SET unit\_price = 800 WHERE product\_id = 100;

#Dữ liệu sẽ được insert vào product\_price\_history thay vì PRODUCT

* Xem trigger tại: dba\_triggers;