#Section7
Cursors & Collections

Cursors & Collections 프롤로그

단일필드 변수

- v_name varchar2(20)
- v_name cst_info.name%type

전체필드 (ROWTYPE record)

- r_cst_info cst_info%rowtype

CST_ID	NAME	BIRTH	MOBILE	POINT	REG_DAY
C001	홍길동1	2008	010-0000-1111	5000	23/08/16
C002	홍길동2	2000	010-0000-1112	6600	23/08/16
C003	홍길동3	2010	010-0000-1113	11000	23/08/16
C004	홍길동4	1970	010-0000-1114	0	23/08/16
C005	홍길동5	1960	010-0000-1115	0	23/08/16

단일 row 의 값이 아닌
Multi Row 의 값

Type Record (사용자 정의)

- Type recname IS Record(
  cst_id varchar2(20,
  name cst_info.name%type;
)

Cursors & Collections 프롤로그

6.2 Cursors Overview

A **cursor** is a pointer to a private SQL area that stores information about processing a specific `SELECT` or DML statement.

CST_ID	NAME	BIRTH	MOBILE	POINT	REG_DAY
C001	홍길동1	2008	010-0000-1111	5000	23/08/16
C002	홍길동2	2000	010-0000-1112	6600	23/08/16
C003	홍길동3	2010	010-0000-1113	11000	23/08/16
C004	홍길동4	1970	010-0000-1114	0	23/08/16
C005	홍길동5	1960	010-0000-1115	0	23/08/16

5.1 Collection Types

PL/SQL has three collection types—associative array, `VARRAY` (variable-size array), and nested table.

Implicit Cursor (묵시적 커서)

6.2.1 Implicit Cursors

An **implicit cursor** is a session cursor that is constructed and managed by PL/SQL.

PL/SQL opens an implicit cursor every time you run a SELECT or DML statement.

You cannot control an implicit cursor, but you can get information from its attributes.

묵시적 커서는 PL/SQL에 의해 생성되고 관리되는 세션 커서입니다.

PL/SQL은 SELECT 또는 DML 문을 실행할 때마다 묵시적 커서를 엽니다.

묵시적 커서를 제어할 수는 없지만 해당 속성에서 정보를 얻을 수 있습니다.

SQL%ISOPEN Attribute: Is the Cursor Open?

SQL%FOUND Attribute: Were Any Rows Affected?

SQL%NOTFOUND Attribute: Were No Rows Affected

SQL%ROWCOUNT Attribute: How Many Rows Were Affected?

Explicit Cursors (명시적 커서)

6.2.2 Explicit Cursors

An **explicit cursor** is a session cursor that you construct and manage. You must declare and define an explicit cursor,

giving it a name and associating it with a query (typically, the query returns multiple rows).

Then you can process the query result set in either of these ways:

Open the explicit cursor (with the OPEN statement), **fetch** rows from the result set (with the FETCH statement),

and **close** the explicit cursor (with the CLOSE statement).

Use the explicit cursor in a cursor FOR LOOP statement (see "Processing Query Result Sets With Cursor FOR LOOP Statements".

[Sample]

```
declare
     Cursor cur_cst_info
          is select cst_id, name from cst_info;

    v_cst_id cst_info.cst_id%type;
    v_name cst_info.name%type;
  begin
   Open  cur_cst_info;
      Loop
         FETCH cur_cst_info Into v_cst_id,v_name ;
         Exit When cur_cst_info%NOTFOUND;
         dbms_output.put_line(v_cst_id ||'-'||v_name);
      End Loop;
   Close cur_cst_info;
   dbms_output.put_line(chr(10)||chr(13));
End;
```

[Example 6-24 Cursor Variable Declarations]

```
DECLARE
  TYPE empcurtyp IS REF CURSOR RETURN employees%ROWTYPE;  -- strong type
  TYPE genericcurtyp IS REF CURSOR;                       -- weak type

  cursor1  empcurtyp;       -- strong cursor variable
  cursor2  genericcurtyp;   -- weak cursor variable
  my_cursor SYS_REFCURSOR;  -- weak cursor variable

  TYPE deptcurtyp IS REF CURSOR RETURN departments%ROWTYPE;  -- strong type
  dept_cv deptcurtyp;  -- strong cursor variable
BEGIN
  NULL;
END;
/
```

Collections 개념

Collection = Array ?

Table 5-1 PL/SQL Collection Types

Collection Type	Number of Elements	Index Type	Dense or Sparse	Uninitialized Status	Where Defined	Can Be ADT Attribute Data Type
Associative array (or index-by table)	Unspecified	String or `PLS_INTEGER`	Either	Empty	In PL/SQL block or package	No
`VARRAY` (variable-size array)	Specified	Integer	Always dense	Null	In PL/SQL block or package or at schema level	Only if defined at schema level
Nested table	Unspecified	Integer	Starts dense, can become sparse	Null	In PL/SQL block or package or at schema level	Only if defined at schema level

Key – Value 로 구성

길이가 정해지는 배열

길이가 고정되지 않은 가변배열

Non-PL/SQL Composite Type	Equivalent PL/SQL Composite Type
Hash table	Associative array
Unordered table	Associative array
Set	Nested table
Bag	Nested table
Array	`VARRAY`

☞ **실제 실습해보면서 이해하기 ~**

Collections 개념

/* Associative array Sample */

```
DECLARE
  -- Associative array indexed by string:

  TYPE Capital IS TABLE OF VARCHAR2(50)  -- Associative array type
    INDEX BY VARCHAR2(64);               --  indexed by string

  city_capital  Capital;         -- Associative array variable
  v_index  VARCHAR2(64);                    -- Scalar variable

BEGIN
  -- Add elements (key-value pairs) to associative array:

  city_capital('한국')  := '서울';
  city_capital('프랑스')  := '파리';
  city_capital('영국')  := '런던';

  DBMS_Output.PUT_LINE(city_capital('한국'));


  -- Print associative array:

  v_index := city_capital.FIRST;  -- Get first element of array

  WHILE v_index IS NOT NULL LOOP
    DBMS_Output.PUT_LINE('Population of ' || v_index || ' is ' || city_capital(v_index));
    v_index := city_capital.NEXT(v_index);  -- Get next element of array
  END LOOP;


END;
/
```

/* Varrays (Variable-Size Arrays) Sample */

```
DECLARE

  TYPE Capital IS VARRAY(10) OF VARCHAR2(50);  --Varrays (Variable-Size Arrays) type

  city_capital Capital := Capital('서울','파리','런던')  ;         -- Varrays variable

BEGIN

  FOR i IN 1..city_capital.count LOOP
      DBMS_OUTPUT.PUT_LINE(city_capital(i));
  END LOOP;


END;
/
```

/* Nested Tables Sample */

```
DECLARE
  TYPE Capital IS TABLE OF VARCHAR2(20);  -- nested table type

  -- nested table variable initialized with constructor:
  city_capital Capital := Capital('서울','파리','런던');

Begin

    FOR i IN city_capital.FIRST .. city_capital.LAST LOOP  -- For first to last element
      DBMS_OUTPUT.PUT_LINE(city_capital(i));
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('---');

END;
/
```

Collections – 객체(Table , Record, Cursor ..) 사용

단순 문자열등 사용

Associative array

```
TYPE Capital IS TABLE OF VARCHAR2(50)
    INDEX BY VARCHAR2(64);
city_capital   Capital;

city_capital('한국')  := '서울';
city_capital('프랑스') := '파리';
city_capital('영국')  := '런던';
```

Varrays

```
TYPE Capital IS VARRAY(10) OF VARCHAR2(50);
 city_capital Capital := Capital('서울','파리','런던')  ;
```

Nested Tables

```
TYPE Capital IS TABLE OF VARCHAR2(20);

  -- nested table variable initialized with constructor:
  city_capital Capital := Capital('서울','파리','런던');
```

배열에
객체 넣기

Table or Record 활용

Collections - BULK COLLECT INTO / FORALL

Collections 확장 – DB Table 입력

```
TYPE Capital IS TABLE OF VARCHAR2(20);          -- varchar2(20) 문자열
TYPE Capital IS TABLE OF CST_INFO%ROWTYPE;      -- CST_INFO Table
```

[Sample 1]

```
DECLARE
    TYPE nl_cst_info IS TABLE OF cst_info%rowtype;
    l_cst_info nl_cst_info := nl_cst_info() ; -- 초기화
BEGIN
    For fc In (SELECT *  FrOM cst_info)
    Loop
      l_cst_info.EXTEND;
      l_cst_info(l_cst_info.LAST).cst_id := fc.cst_id;
      l_cst_info(l_cst_info.LAST).name := fc.name;
      l_cst_info(l_cst_info.LAST).birth := fc.birth;

      dbms_output.put_line(fc.cst_id);
    End Loop;

    DBMS_OUTPUT.put_line (chr(10)||chr(13));

    FOR indx IN l_cst_info.first .. l_cst_info.COUNT
    LOOP
      DBMS_OUTPUT.put_line (l_cst_info (indx).cst_id);
      DBMS_OUTPUT.put_line (l_cst_info (indx).name);
      DBMS_OUTPUT.put_line (chr(10)||chr(13));

    END LOOP;

END;
```

대량의 데이터를

처리할 경우

BULK COLLECT INTO

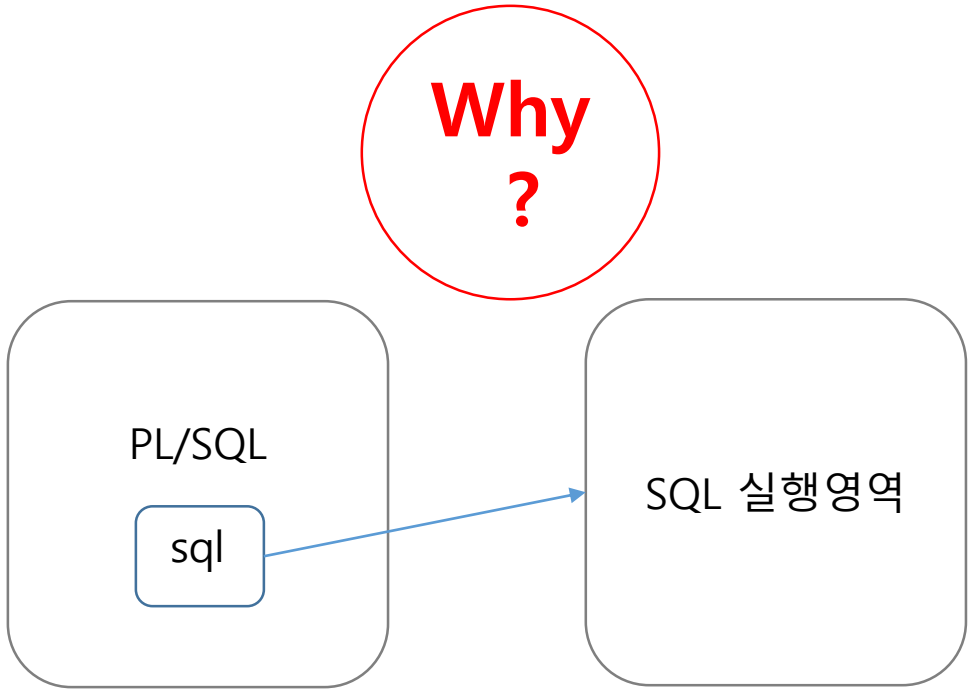
FORALL 사용

[Sample 2]

```
DECLARE
    TYPE nl_cst_info IS TABLE OF cst_info%rowtype;
    l_cst_info nl_cst_info := nl_cst_info() ; -- 초기화
BEGIN

    SELECT *
     BULK COLLECT INTO l_cst_info
    FrOM cst_info;

    DBMS_OUTPUT.put_line (chr(10)||chr(13));

    FOR indx IN l_cst_info.first .. l_cst_info.COUNT
    LOOP
     insert into cst_info2(cst_id, name, birth, mobile, point,reg_day)
     values (l_cst_info(indx).cst_id, l_cst_info(indx).name, l_cst_info(indx).birth,
             l_cst_info(indx).mobile, l_cst_info(indx).point, l_cst_info(indx).reg_day);
     END LOOP;

END;
```

```
FORALL indx IN l_cst_info.first .. l_cst_info.COUNT
    insert into cst_info2(cst_id, name, birth, mobile, point,reg_day)
    values (l_cst_info(indx).cst_id, l_cst_info(indx).name, l_cst_info(indx).birth,
            l_cst_info(indx).mobile, l_cst_info(indx).point, l_cst_info(indx).reg_day);
```

Collections – 객체(Table , Record, Cursor ..) 사용의 배경




```
DECLARE

BEGIN

    For fc In (SELECT * FROM cst_info)
    Loop

     insert into cst_info2(cst_id, name, birth, mobile, point,reg_day)
     values (fc.cst_id, fc.name, fc.birth, fc.mobile, fc.point, fc.reg_day);

    End Loop;

END;
```

Oracle Database 개발자가 작성하는 거의 모든 프로그램에는 PL/SQL 및 SQL 문이 모두 포함되어 있습니다. PL/SQL 문은 PL/SQL 문 실행기에 의해 실행됩니다.
SQL문은 SQL문 실행 프로그램에 의해 실행됩니다.
PL/SQL 런타임 엔진은 SQL 문을 발견하면 중지하고 SQL 문을 SQL 엔진으로 전달합니다.
SQL 엔진은 SQL 문을 실행하고 정보를 다시 PL/SQL 엔진으로 반환합니다.
이러한 제어 전송을 컨텍스트 스위치라고 하며 이러한 스위치 각각은
프로그램의 전체 성능을 저하시키는 오버헤드를 발생시킵니다.

☞ **cst_info 의 개수만큼 context switch 발생**

만일 cst_info 의 개수가 10만개라면 10만번의 Switch 발생

https://livesql.oracle.com/apex/livesql/file/tutorial_IEHP37S6LTWIIDQIR436SJ59L.html

Collections - BULK COLLECT INTO - Limit

Array 객체의 사이즈가 커질수록
메모리의 부담이 증가

Solution -> LIMIT

[LIMIT Sample]

```
DECLARE
    TYPE nl_cst_info IS TABLE OF cst_info%rowtype;
    l_cst_info nl_cst_info := nl_cst_info() ; -- 초기화

    my_cursor SYS_REFCURSOR;  -- weak cursor variable

BEGIN
    Open my_cursor For select * from cst_info;

    Loop
    FETCH my_cursor BULK COLLECT INTO l_cst_info LIMIT 2;

        FORALL indx IN l_cst_info.first .. l_cst_info.COUNT
         insert into cst_info2(cst_id, name, birth, mobile, point,reg_day)
         values (l_cst_info(indx).cst_id, l_cst_info(indx).name, l_cst_info(indx).birth,
                 l_cst_info(indx).mobile, l_cst_info(indx).point, l_cst_info(indx).reg_day);

    EXIT WHEN my_cursor%NOTFOUND;
    End Loop;

    Close my_cursor;
END;
```