04. Data type

목차

1. Data type 개요

▣ PL/SQL Data type

- SQL Data type + PL/SQL 고유의 Data type

▣ Data type 분류

DATA TYPE 유형	주요특징
Scalar type	단일값을 저장하는 DATA TYPE
Composite type	복수값을 저장하는 DATA TYPE
Reference type	다른 DATA TYPE을 참조하는 DATA TYPE 타언어의 포인터와 유사한 개념으로 커서 참조 유형과 객체 참조 유형으로 구분
LOB(Large Object) type	Large Object를 저장하는 DATA TYPE
Object type	객체지향 언어의 객체를 저장하는 DATA TYPE

2. Scalar data type

▣ Scalar data type

- Scalar type으로 선언된 변수는 1개의 변수에 1개 값 저장 (Hold a single value)

데이터 형	설 명
VARCHAR2(n)	- 가변 길이 (variable-Length) 문자 데이터 타입 - TABLE 의 COLUMN : 1 ~ 4000 Bytes PL/SQL의 변수 : 1 ~ 32767 Bytes
NUMBER(p,s)	- 가변 길이(variable-length) 숫자 데이터 타입 - TABLE 의 COLUMN 과 PL/SQL 변수가 동일한 특성을 가진다. - 38 자리 이하의 유효 숫자 자리
DATE	- 고정 길이(Fixed-length) 날짜 데이터 타입 - TABLE 의 COLUMN 과 PL/SQL 변수가 동일한 특성을 가진다.
CHAR(n)	- 고정 길이(Fixed-length) 문자 데이터 타입 - TABLE 의 COLUMN : 1 ~ 2000 Bytes PL/SQL의 변수 : 1 ~ 32767 Bytes
LONG	- 가변 길이 (variable-Length) 문자 데이터 타입 - TABLE 의 COLUMN : 1 ~ 2G Bytes PL/SQL의 변수 : 1 ~ 32760 Bytes
LONG RAW	- 가변 길이 (variable-Length) 바이너리 데이터 타입 - TABLE 의 COLUMN : 1 ~ 2G Bytes PL/SQL의 변수 : 1 ~ 32760 Bytes
BOOLEAN	- 논리연산에 사용되는 BOOLEAN 데이터 타입 - TABLE 의 COLUMN : 지원하지 않음 PL/SQL의 변수 : TRUE,FALSE,NULL 의 3가지 값 허용
LOB	- LOB(Large Object, 4G) 유형의 데이터를 저장 문자(CLOB) 또는 바이너리(BLOB) 유형
BINARY_INTEGER	- TABLE 의 COLUMN : 지원하지 않음 PL/SQL의 변수 : -2147483647~2147483647 사이의 정수에 대한 기본 형
PLS_INTEGER	- TABLE 의 COLUMN : 지원하지 않음 PL/SQL의 변수 : -2147483647~2147483647사이의 signed정수에 대한 기본형으로 BINARY_INTEGER Type 보다 빠른 연산 성능개선을 위한 유형이며 BINARY_INTEGER를 대체하는 Data Type 유형

* 12c alter system set max_string_size=extended & utl32k.sql , varchar2: 4000 >> 32767

2. Scalar data type

<참고>

Scalar data type은 개발시점에 빈번히 사용 하게 되는 유형으로 아래의 2가지 사항 유의

① Type의 고유 특성 ② Type의 최대크기(해당 데이터 Type의 최대 할당크기)

SQL Data Type 과 PL/SQL Data Type은

① Type의 고유 특성은 동일 ② 제한길이는 일부 차이

<참고>

PLS_INTEGER 와 BINARY_INTEGER는 PL/SQL내에만 있는 Data Type 으로

PLS_INTEGER는 BINARY_INTEGER를 개선한 Data Type이며 빠른 숫자 연산을 위한 용도로 사용.

NUMBER는 Packed Decimal 유형으로 DBMS내에 저장

십진수 1자리를 4 Bit 단위로 저장. NUMBER Type을 가지고 숫자 연산을 하게 되면

Packed Decimal → Binary로 변환 → 연산 → Packed Decimal로 변환.

NUMBER는 빠른 연산보다는 저장 및 비교를 위해 최적화 되어 있다.

PL/SQL 내에서 DBMS내에 저장할 필요가 없이 빠른 수치적 연산이 필요한 경우를 위해서 Binary형태로 저장되는 BINARY_INTEGER Type을 제공. BINARY_INTEGER 에서 저장 공간을 줄이고 및 연산속도를 향상시킨 것이 PLS_INTEGER Type. PL/SQL로 프로그램을 작성하게 되면 대부분은 테이블에 저장되어 있는 데이터에 대한 연산 이기 때문에 일부 특별한 연산이 아니면 PLS_INTEGER, BINARY_INTEGER 유형을 사용하는 경우가 적다.

3. Composite data type

▣ Composite data type

- Composite Datatype은 내부 구성 요소를 가지고 있어서 1개의 변수가 여러 개의 값을 저장.
  타언어의 배열 변수(Array variable) 나 구조체 변수(Structure/Record variable)와 유사 (Hold Multiple Value)

데이터 형	설 명
RECORD	C언어의 구조체(Structure)와 동일한 개념 서로 다른 데이터형을 논리적인 하나의 그룹으로 정의 PL/SQL에서만 지원
INDEX-BY TABLE	C언어의 배열(Array)와 동일한 개념 동일 데이터형을 하나의 그룹으로 정의 PL/SQL 에서만 지원
NESTED TABLE	INDEX-BY TABLE 의 확장 유형 (최대 2G) TABLE 의 COLUMN : 지원(관계형 테이블의 컬럼으로 저장가능) PL/SQL의 변수 : 지원
VARRAY	C언어의 배열(ARRAY)와 동일한 개념(최대 2G) 동일 데이터형을 하나의 그룹으로 정의 TABLE 의 COLUMN : 지원(관계형 테이블의 컬럼으로 저장가능한) PL/SQL의 변수 : 지원

* 12c alter system set max_string_size=extended & utl32k.sql , varchar2: 4000 >> 32767

3. Composite data type

▣ Record data type

- Record는 관계형 데이터베이스의 Row와 동일한 개념
  PL/SQL에서 Record type의 주요 용도는 테이블의 Row를 변수에 저장후 데이터 처리에 사용.
  테이블의 1 ROW ➔ 1개의 Record변수에 저장 하는 방식을 사용 하면 데이터 처리시 편리
  테이블이 여러 컬럼으로 구성되듯 PL/SQL의 Record는 여러 Field로 구성.

```
TYPE  type_name  IS  RECORD
    (field_name1 {scalar_type |record_type |table_type} [NOT NULL] [{:= | DEFAULT} expr],
    (field_name2 {scalar_type |record_type |table_type} [NOT NULL] [{:= | DEFAULT} expr],
    . . . . . .);

variable_name  type_name;
```

* 12c alter system set max_string_size=extended & utl32k.sql , varchar2: 4000 >> 32767

3. Composite data type

▣ Record data type

```
SET  SERVEROUTPUT ON
DECLARE
        TYPE   T_ADDRESS  IS  RECORD(                  -- Record type 정의
                    ADDR1       VARCHAR2(60),         -- 주소1
                    ADDR2       VARCHAR2(60),         -- 주소2
                    ZIP         VARCHAR2(7),          -- 우편번호
                    PHONE       VARCHAR2(14)          -- 전화번호
        );
        TYPE  T_EMP_RECORD  IS  RECORD (               -- Record type 정의
                     EMPNO     NUMBER(4)  ,            -- 사번
                     ENAME     VARCHAR2(10),           -- 이름
                     JOB       VARCHAR2(9),            -- 직업
                     ADDRESS   T_ADDRESS,              -- 주소     ??? Field에 Record type 정의가 가능한지
                     HIREDATE  DATE);                  -- 입사일

        REC_EMP  T_EMP_RECORD;                         -- Record 변수 선언

BEGIN
        -- Record의 field에 값을 대입(저장)
        REC_EMP.EMPNO  := 1234;
        REC_EMP.ENAME  := 'XMAN';
        REC_EMP.JOB    := 'DBA';
        REC_EMP.ADDRESS.ADDR1 :=  '강남구 역삼동';
        REC_EMP.ADDRESS.ZIP   :=  '150-036';
```

3. Composite data type

▣ Record data type

```
        REC_EMP.HIREDATE        :=    SYSDATE - 365;

        -- -- Record의 field값 조회
        DBMS_OUTPUT.PUT_LINE('*****************************************************************');
        DBMS_OUTPUT.PUT_LINE('사번 :'||REC_EMP.EMPNO);
        DBMS_OUTPUT.PUT_LINE('이름 :'||REC_EMP.ENAME);
        DBMS_OUTPUT.PUT_LINE('직업 :'||REC_EMP.JOB);
        DBMS_OUTPUT.PUT_LINE('주소 :'||REC_EMP.ADDRESS.ADDR1);
        DBMS_OUTPUT.PUT_LINE('주소 :'||REC_EMP.ADDRESS.ZIP);
        DBMS_OUTPUT.PUT_LINE('입사일 :'||TO_CHAR(REC_EMP.HIREDATE,'YYYY/MM/DD'));
        DBMS_OUTPUT.PUT_LINE('*****************************************************************');
END;
/
```

① T_ADDRESS는 사용자가 정의한 Record type으로 Record의 각 field 정의는 관계형 테이블의 컬럼 정의와 동일한 방식

② Syntax 구조를 보면 Field의 Data type에 scalar ,table, record 유형 정의 가능
ADDRESS Field의 data type에 T_ADDRESS 라는 Record type 정의.

③ **REC_EMP T_EMP_RECORD; 는** T_EMP_RECORD TYPE의 변수 선언

④ Record내의 Field에 데이터를 저장하거나 Field 데이터에 접근은
RECORD변수명.FIELD명(EX REC_EMP.EMPNO)

⑤ NESTED RECORD(중첩 레코드)의 FIELD에 접근하는 방법은
RECORD변수명.RECORD변수명.FIELD명 (EX REC_EMP.ADDRESS.ZIP).

3. Composite data type

▣ Index by table data type

- 관계형 데이터베이스의 테이블과 단어상 혼동을 피하기 위해서 PL/SQL 테이블 또는 INDEX BY TABLE로 호칭
- 배열(Array)처럼 액세스하기 위해 배열의 첨자와 유사한 키(KEY)를 사용
- 배열의 데이터에 접근 하기 위해서는 배열의 첨자를 사용하듯 PL/SQL 테이블의 데이터에 접근하기 위해서는 키(KEY) 사용
- 데이터 유형으로는 Scalar , Composite 사용
- 일반 배열과는 달리 크기가 미리 정의되지 않고 동적으로 자유롭게 증가할 수 있어 동적 배열과 유사

①SIMILAR TO A ONE-DIMENSIONAL ARRAY OF SCALAR, COMPOSITE
②BINARY_INTEGER를 ARRAY의 첨자(KEY)로 사용
③크기가 동적으로 증가

```
TYPE table_type_name IS TABLE OF
                {scalar_type | record_type |} [NOT  NULL]
      [INDEX  BY  BINARY_INTEGER];
variable_name          table_type_name;
```

3. Composite data type

▣ Index by table data type

```
SET  SERVEROUTPUT ON

DECLARE
        TYPE  T_EMP_LIST  IS  TABLE  OF  VARCHAR2(20)   -- TABLE TYPE 정의 ,  Dynamic array와 유사
                INDEX  BY  BINARY_INTEGER;

        TBL_EMP_LIST        T_EMP_LIST;          -- TABLE 변수 선언
        V_TMP               VARCHAR2(20);
        V_INDEX             NUMBER(10);
BEGIN

        TBL_EMP_LIST(1)      :=  'SCOTT';      -- TABLE에 DATA 입력한다
        TBL_EMP_LIST(1000)   :=  'MILLER';     -- Index 첨자의 범위 -2147483647 ~ 2147483647
        TBL_EMP_LIST(-2134)  :=  'ALLEN';      -- 불연속적 첨자 사용

        TBL_EMP_LIST(0)      :=  'XMAN';

        V_TMP :=  TBL_EMP_LIST(1000);          -- TABLE에서 DATA조회

        --  TABLE에 있는 DATA를 조회해서 RETURN한다.
        DBMS_OUTPUT.PUT_LINE('DATA OF KEY  1000   IS '||TBL_EMP_LIST(1000));
        DBMS_OUTPUT.PUT_LINE('DATA OF KEY -2134   IS '||TBL_EMP_LIST(-2134));
        DBMS_OUTPUT.PUT_LINE('DATA OF KEY     1   IS '||TBL_EMP_LIST(1));
```

3. Composite data type

▣ Index by table data type

```
        -- Method 사용 ex) FIRST,LAST,NEXT,PRIOR, EXISTS, COUNT, DELETE ..
         IF NOT TBL_EMP_LIST.EXISTS(888) THEN
               DBMS_OUTPUT.PUT_LINE('DATA OF KEY 888 IS NOT EXIST ');
        END IF;

        -- LOOP를 사용하여 데이타 조회
        V_INDEX := TBL_EMP_LIST.FIRST;          --  EX) PRIOR,FIRST,LAST
        LOOP
                DBMS_OUTPUT.PUT_LINE('LOOP:'||TO_CHAR(V_INDEX)||' ==>'||TBL_EMP_LIST(V_INDEX));
                V_INDEX := TBL_EMP_LIST.NEXT(V_INDEX);                          -- 다음번 유효한 INDEX RETURN
                EXIT WHEN V_INDEX IS NULL;
        END LOOP;

        DBMS_OUTPUT.PUT_LINE('DATA OF KEY   999   IS '||TBL_EMP_LIST(999));       -- 존재하지 않는값을 조회
        DBMS_OUTPUT.PUT_LINE('DATA OF KEY     0   IS '||TBL_EMP_LIST(0));
EXCEPTION
        WHEN  NO_DATA_FOUND  THEN
                DBMS_OUTPUT.PUT_LINE('ERROR CODE=>'||TO_CHAR(SQLCODE));
                DBMS_OUTPUT.PUT_LINE('ERROR MSG =>'||SQLERRM);
END;
/
```

3. Composite data type

▣ Index by table data type

① T_EMP_LIST는 사용자가 정의한 TABLE TYPE으로 VARCHAR2(20)은 ARRAY의 1 CELL에 해당.
INDEX BY BINARY_INTEGER는 PL/SQL TABLE의 KEY에 사용되는 값을 BINARY_INTEGER DATA TYPE으로 사용하겠다는 의미로 SYNTAX 문법상 사용자가 임의적으로 다른 DATA TYPE을 사용할수 있는 것처럼 보이지만 Data Type을 변경할수 없다. INDEX BY BINARY_INTEGER는 PL/SQL TABLE을 사용할때는 항상 사용하는 구문.

<참고>PL/SQL TABLE 은 Oracle Server Process(서버 프로세스)가 사용하는 메모리 영역(PGA)영역에 할당.
즉 DBMS 가 설치된 H/W시스템의 메모리 영역 사용. 개발하는 PL/SQL프로그램이 동시 사용성이 높고 PL/SQL TABLE을 과다하게 사용하는 경우 일시적인 메모리 자원 경합 발생 가능

② TBL_EMP_LIST 라는 사용자 정의 PL/SQL TABLE 변수를 선언

③ KEY를 사용하여 TABLE 변수의 각 CELL 접근 **(Key-Value pair)**

```
TBL_EMP_LIST(1) :=  'SCOTT';
```

예제는 비연속적인 또는 상호 관련 없는 값이 KEY 로 사용될수 있음을 보여주기 위한 것이지만 실제 프로그래밍시 위와 같은 의미 없는 불연속적이며 무의미한 KEY 값을 사용하시 데이터 접근 불편

④ **PL/SQL BLOCK내에 특정 데이터에 접근(SELECT, PL/SQL TABLE)시 해당 데이터가 없는 경우 NO_DATA_FOUND Exception 발생**

⑤ **SQLCODE 와 SQLERRM** 주요 용도 2가지

(a) PL/SQL 개발시 디버깅 용도
DBMS_OUTPUT 과 함께 사용되며 실행시 발생하는 에러의 정보를 개발툴 화면에 출력하는 용도 (예제)

(b) **PL/SQL로 개발된 모듈 (ex Procedure , Function등)의 실행 에러 기록**
예제 처럼 화면에 결과를 출력 하는 것이 아니라 로그 테이블등에 INSERT를 사용하여 실행시 발생한 에러를 기록하는 방법

4. 참조 연산자

▣용도

데이터베이스 프로그램의 추상적 구조는 관계형 데이터베이스 테이블내 저장된 데이터를
(1) 조회 (2) 연산(가공처리) (3) 반영(저장)

PL/SQL에서 사용하는 대부분의 변수는 관계형 테이블의 컬럼 데이터를 저장하기 위한 용도 이기에 테이블의 컬럼 정의와 동일하게 선언한다. 테이블의 컬럼 정의가 변경되는 경우 관련된 P/SQL 변수들도 변경해주어야 하나 참조 연산자를 사용하는 경우는 실행시 동적으로 참조 하기 때문에 변경 필요 없다.

참조연산자 2가지 장점

① 편리성
② 변경 적응성

▣ 유형

① %TYPE
② %ROWTYPE

4. 참조 연산자

▣ %TYPE

- 관계형 데이터베이스 **Table내 Column의 Data type , data length만을 참조하여 변수 선언**
- 참조하는 Column의 NOT NULL 제약사항은 참조하지 않는다.

```
EX)  DECLARE
          V_ENAME          VARCHAR2(40);
          V_EMPNO          EMP.EMPNO%TYPE;
     BEGIN
```

- EMPNO 컬럼의 Data type 과 Data length는 NUMNER(4)로 정의
- 실행 시점에
  V_EMPNOEMP.EMPNO%TYPE; ➔ V_EMPNO NUMBER(4)로 선언

*** 개발시 편리성 제공**

4. 참조 연산자

▣ %ROWTYPE

- **TABLE,VIEW,CURSOR내의 여러 Column을 참조하여 Record type 변수 생성**
- 테이블의 구조적 정보를 일일이 확인 하지 않고 컬럼,데이터 타입,데이터 길이등을 일괄 생성
- **Row 단위의 데이터 fetch시 편리**

*데이터베이스의 테이블 또는 VIEW의 일련의 열을 Record로 선언하기 위하여 %ROWTYPE 사용.

```
EX) DECLARE
        R_EMP       EMP%ROWTYPE;
    BEGIN
```

- EMP라는 테이블 전체를 참조(Reference)해서 R_EMP Record 생성 하며
- %TYPE 과 Record type를 하나의 기능으로 구현
- **EMP 테이블내의 모든 컬럼을 %TYPE으로 참조 한후 내부적으로 Record로 정의**

4. 참조 연산자

▣ %TYPE & %ROWTYPE

```
SET  SERVEROUTPUT ON
DECLARE
        REC_EMP          EMP%ROWTYPE;        -- Row     참조 연산자
        V_EMPNO          EMP.EMPNO%TYPE;     -- Column 참조 연산자
BEGIN
        -- 1개의 Row를 SELECT 해서 Record 변수에 저장
        SELECT  *  INTO  REC_EMP  FROM EMP WHERE   EMPNO = 7369;

        DBMS_OUTPUT.PUT_LINE('EMPNO    =>'||REC_EMP.EMPNO);
        DBMS_OUTPUT.PUT_LINE('ENAME    =>'||REC_EMP.ENAME);
        DBMS_OUTPUT.PUT_LINE('JOB      =>'||REC_EMP.JOB);
        DBMS_OUTPUT.PUT_LINE('MGR      =>'||REC_EMP.MGR);
        DBMS_OUTPUT.PUT_LINE('HIREDATE =>'||REC_EMP.HIREDATE);
        DBMS_OUTPUT.PUT_LINE('SAL      =>'||REC_EMP.SAL);

        -- Record의 개개 Field를  독립적으로 사용
        SELECT  EMPNO,ENAME  INTO  V_EMPNO, REC_EMP.ENAME
        FROM    EMP
        WHERE   EMPNO = 7369;

        DBMS_OUTPUT.PUT_LINE('--------------------------------');
        DBMS_OUTPUT.PUT_LINE('EMPNO    =>'||V_EMPNO);
        DBMS_OUTPUT.PUT_LINE('ENAME    =>'||REC_EMP.ENAME);
END;
/
```

● 4. 참조 연산자

▣ %TYPE & %ROWTYPE

① %ROWTYPE 을 사용하여 **Row 형태의 데이터(ex Table,Cursor,View)를 참조하여 변수 선언**
%TYPE을 사용하여 Column 형태의 데이터를 참조하여 변수 선언

② **PL/SQL 실행부(BEGIN ~ END)에서 SELECT를 사용하는 경우 항상 INTO 구문을 사용해야 한다.**
INTO 이하의 변수에 SELECT결과 저장

[질문] EMPXXX 테이블이 80개의 컬럼으로 구성되어 있는데 %ROWTYPE의 참조 연산자가 없다면 ?
80개의 변수를 정의해야 하고 INTO 이하에 80개의 변수를 나열

③ %ROWTYPE을 사용하여 선언한 REC_EMP 변수의 정체는 ? RECORD 변수

④ SELECT 에서 2개의 컬럼을 조회하기 때문에 INTO 이하에도 2개의 변수 사용해야 한다.

5. Block 실행부에서 Select

▣ Exception

Exception Name	Error code	Error Message
NO_DATA_FOUND	**ORA-01403**	**-데이터를 찾을 수 없습니다. -조회하는 데이터가 0건인 경우 발생하는 EXCEPTION**
NOT_LOGGED_ON	ORA-01012	데이터베이스에 연결되지 않은 상태에서 SQL문을 실행하려는 경우
TOO_MANY_ROWS	**ORA-01422**	**-실제 인출은 요구된 것보다 많은 수의 행을 추출합니다. -조회하는 데이터가 >1건인 경우 발생하는 EXCEPTION**
ZERO_DEVIDE	ORA-01476	제수가 0 입니다
	ORA-00904	부적합한 식별자
	ORA-00001	무결성 제약 조건(SCOTT.EMP_EMPNO_PK)에 위배됩니다

Oracle DBMS내에 모든 Error는 Error Code 와 Error Message가 사전 정의되어 있다.

① PL/SQL 내의 예외처리부(EXCEPTION SECTION)에서 **빈번히 제어 해야하는 Error들에게는 Error의 이름을 부여 해서 명료 하게 하고 사용을 쉽게 한다.** 이름이 숫자 코드보다 인식성이 좋아 식별성이 좋다.

② PL/SQL 내의 예외처리부(EXCEPTION SECTION)에서 빈번히 제어할 필요가 없는Error들은 이름을 부여 하지 않는다.
PL/SQL내에 이름이 없는 ERROR들도 사용자(개발자가) 임의의 이름 부여 가능

5. Block 실행부에서 Select

▣ NO_DATA_FOUND

```
SET SERVEROUTPUT ON
DECLARE
        V_EMPNO     EMP.EMPNO%TYPE;
        V_ENAME     EMP.ENAME%TYPE;
        V_HIREDATE  EMP.HIREDATE%TYPE;
BEGIN
        -- SELECT 되는 대상 데이타가 없는 조회
        SELECT EMPNO,ENAME,HIREDATE  INTO   V_EMPNO,V_ENAME,V_HIREDATE
        FROM   EMP
        WHERE  EMPNO = 10000;

        DBMS_OUTPUT.PUT_LINE( ' selected exactly one row  by  '||V_EMPNO );
EXCEPTION
        WHEN NO_DATA_FOUND  THEN
                DBMS_OUTPUT.PUT_LINE('NO DATA FOUND !!!!!');
        WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE('TOO MANY ROWS FOUND  !!!!');
END;
/
```

[질문] NO DATA FOUND를 왜 Error로 간주 하는가?

5. Block 실행부에서 Select

▣ TOO_MANY_ROWS

```
SET SERVEROUTPUT ON
DECLARE
        V_EMPNO     EMP.EMPNO%TYPE;
        V_ENAME     EMP.ENAME%TYPE;
        V_HIREDATE  EMP.HIREDATE%TYPE;
BEGIN
        --  SELECT 되는 대상 데이타가 1개 이상인 조회
        SELECT EMPNO,ENAME,HIREDATE  INTO   V_EMPNO,V_ENAME,V_HIREDATE
        FROM   EMP
        WHERE  EMPNO >= 1;

        DBMS_OUTPUT.PUT_LINE( ' selected exactly one row  by  '||V_EMPNO );
EXCEPTION
        WHEN NO_DATA_FOUND  THEN
                DBMS_OUTPUT.PUT_LINE('NO DATA FOUND !!!!!');
        WHEN TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE('TOO MANY ROWS FOUND  !!!!');
END;
/
```

[질문] TOO MANY ROWS는 왜 Error 인가?