데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

4차시: Data Type

[학습목표]

PL/SQL Data Type 중 주요하게 사용되는 SCALAR,COMPOSITE Datatype 에 대해 학습하며 개발시 코딩 생산성과 유지보수성을 향상에 도움을 주는 참조연산에 대해 학습 하고 PL/SQL BLOCK내의 SELECT 연산과 변수에 대해 학습 한다.

[학습목차]

4-1 DATA TYPE 개요

PL/SQL DATA TYPE	SQL의 모든 DATA TYPE
	PL/SQL 고유의 DATA TYPE

PL/SQL Data Type은 SQL DATA TYPE + PL/SQL 고유의 DATA TYPE 으로 구성. SQL의 확장된 언어가 PL/SQL

DATA TYPE 특성에 따른 분류를 해보면 5가지 유형으로 분류가 됩니다.

DATA TYPE 유형	주요특징
SCALAR DATA TYPE	단일값을 저장하는 DATA TYPE
COMPOSITE DATA TYPE	복수값을 저장하는 DATA TYPE
REFERENCE DATA TYPE	다른 DATA TYPE을 참조하는 DATA TYPE 타언어의 포인터와 유사한 개념으로 커서 참조 유형과 객체 참조 유형으로 구분
LOB(Large Object) DATA TYPE	Large Object를 저장하는 DATA TYPE
OBJECT DATA TYPE	객체지향 언어 개념의 객체를 저장하는 DATA TYPE

위의 DATA TYPE 이외에도 참조연산자에 대해서 학습을 합니다.

용어상으로 REFERENCE DATA TYPE 과 혼동이 생기기도 하지만 DATA TYPE이 아니라 DATA TYPE을 참조하는 연산자 입니다. 개발시 유연성 및 효율성을 줄수 있는 연산자.

4-2 SCALAR DATA TYPE

SCALAR DATA TYPE 으로 선언된 변수는 1개의 변수내에 단일 값(Value)을 저장 (Hold a single value)

SCALAR DATA TYPE

데이터 형	설 명
VARCHAR2(n)	- 가변 길이 (variable-Length) 문자 데이터 타입 - TABLE 의 COLUMN : 1 ~ 4000 Bytes PL/SQL 의 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/SQL 의 변수 : 1 ~ 32760 Bytes
LONG RAW	- 가변 길이 (variable-Length) 바이너리 데이터 타입 - TABLE 의 COLUMN : 1 ~ 2G Bytes PL/SQL 의 변수 : 1 ~ 32760 Bytes
BOOLEAN	- 논리연산에 사용되는 BOOLEAN 데이터 타입 - TABLE 의 COLUMN : 지원하지 않음 PL/SQL 의 변수 : TRUE,FALSE,NULL 의 3 가지 값 허용
LOB	- LOB(Large Object) 유형의 데이터를 저장 문자(CLOB) 또는 바이너리(BLOB) 유형
BINARY_INTEGER	- TABLE 의 COLUMN : 지원하지 않음

데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

	PL/SQL 의 변수 : -2147483647~2147483647 사이의 정수에 대한 기본 형
PLS_INTEGER	- TABLE 의 COLUMN : 지원하지 않음 PL/SQL 의 변수 : -2147483647~2147483647 사이의 signed 정수에 대한 기본형으로 BINARY_INTEGER Type 보다 빠른 연산 성능개선을 위한 유형이며 BINARY_INTEGER 를 대체하는 Data Type 유형

<참고>
SCALAR DATA TYPE 은 개발시점에 빈번히 사용 하게 되는 유형.
PL/SQL 로 개발시에 Data Type 에 대한 학습 사항은
① Type 의 고유 특성
② Type 의 최대크기(해당 데이터 Type 의 최대 할당크기)

정리하자면 SQL Data Type 과 PL/SQL Data Type 은
① Type 의 고유 특성은 동일
② 제한길이는 일부 차이

NUMBER 는 SQL 과 PL/SQ 이 동일
PLS_INTEGER 와 BINARY_INTEGER 는 PL/SQL 내에만 있는 Data Type 으로
PLS_INTEGER 는 BINARY_INTEGER 를 개선한 Data Type 이며 빠른 숫자 연산을
위한 용도로 사용.

NUMBER 는 Packed Decimal 유형으로 DBMS 내에 저장
십진수 1 자리를 4 Bit 단위로 저장. NUMBER Type 을 가지고 숫자 연산을 하게
되면 Packed Decimal ➔ Binary 로 변환 ➔ 연산 ➔ Packed Decimal 로 변환.
NUMBER 는 빠른 연산보다는 저장 및 비교를 위해 최적화 되어 있다.

숫자 연산을 하기 위해서 Binary 로 변환을 하고 저장 형태로 바꾸기 위해서
Packed Decimal 로 변환. PL/SQL 내에서 DBMS 내에 저장할 필요가 없이 빠른
수치적 연산이 필요한 경우를 위해서 Binary 형태로 저장되는 BINARY_INTEGER
Type 을 제공. BINARY_INTEGER 에서 저장 공간을 줄이고 및 연산속도를
향상시킨 것이 PLS_INTEGER Type. PL/SQL 로 프로그램을 작성하게 되면 대부분은
테이블에 저장되어 있는 데이터에 대한 연산 이기 때문에 일부 특별한 연산이
아니면 PLS_INTEGER, BINARY_INTEGER 유형을 사용하는 경우가 적다.

데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

BOOLEAN Type 은 PL/SQL 에만 있는 유형으로 다른 언어와 다른점이 있습니다.
이미 3 차시 학습하신 내용이지만 PL/SQL 은 논리연산으로 3 항 연산사용.

BOOLEAN Type 에 저장될수 있는 값이 TRUE,FALSE,NULL 3 가지 종류 입니다.
따라서 AND,OR,NOT 연산시 NULL 에 따른 연산 결과를 기억해야 합니다.

4-3 COMPOSITE DATA TYPE

Composite Datatype은 내부 구성 요소를 가지고 있어서 1개의 변수가 여러 개의 값을 저장.타언어의 배열변수 나 구조체 변수와 유사 (Hold Multiple Value)

COMPOSITE DATA TYPE

데이터 형	유사설 명
RECORD	C 언어의 구조체(STRUCTURE)와 동일한 개념 서로 다른 데이터형들을 논리적인 하나의 그룹으로 정의 PL/SQL 에서만 지원
INDEX-BY TABLE	C 언어의 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/SQL 의 변수 : 지원
VARRAY	C 언어의 배열(ARRAY)와 동일한 개념(최대 2G) 동일 데이터형을 하나의 그룹으로 정의 TABLE 의 COLUMN : 지원(관계형 테이블의 컬럼으로 저장가능한) PL/SQL 의 변수 : 지원

4-3-1 RECORD TYPE

RECORD 는 관계형 데이터베이스의 ROW 와 동일한 개념으로 사용됩니다.
RECORD = STRUCTURE, PL/SQL 에서 RECOR 의 주요 용도는 테이블의 ROW 를 변수에 가져 와서 데이터 처리를 할 때 사용.
테이블의 1 ROW ➔ 1 개의 RECORD 변수 저장 하는 방식을 사용 하면 데이터 처리의 편리하다. 테이블이 여러 컬럼으로 구성되듯이 PL/SQL 의 RECORD 는 여러 FIELD 로 구성 된다.

데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

RECORD 의 SYNTAX

```
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;
```

① RECORD는 여러 개의 FIELD로 구성되며
각 FIELD의 DATA TYPE에 SCALAR , RECORD,TABLE(INDEX-BY TABLE)을
사용할수 있으며 개발시 활용도가 높다.
예를들면 RECORD의 구성요소인 FIELD를 정의할 때 RECORD를 사용할수
있다는 것은 중첩 RECORD (Nested Record)를 구현할수 있다.

일반 PL/SQL 변수 정의와 동일하게 FIELD 정의시 초기값을 지정할수 있고
지정하지 않는 경우에는 NULL 값이DEFAULT로 지정.

4_RECORD_1.SQL

```
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  ??? RECORD를 FIELD로 처리가 가능한지?
                HIREDATE DATE);                    -- 입사일

REC_EMP  T_EMP_RECORD;

BEGIN
        -- RECORD의 FIELD에 값을 대입
        REC_EMP.EMPNO  := 1234;
        REC_EMP.ENAME  := 'XMAN';
        REC_EMP.JOB    := 'DBA';
        REC_EMP.ADDRESS.ADDR1   := '강남구 역삼동';
        REC_EMP.ADDRESS.ZIP     := '150-036';
        REC_EMP.HIREDATE        := SYSDATE - 365;

        -- RECORD의 FIELD값을 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 TYPE 으로 RECORD의 각 FIELD를 정의는 관계형 테이블의 컬럼 정의와 동일한 방식.

관계형 테이블	컬럼명 데이터 TYPE(길이)
RECORD FIELD	FIELD 명 데이터 TYPE(길이) ADDR1 VARCHAR2(60)

② SYNTAX 구조를 보면 FILED 의 DATA TYPE 에 SCALAR,TABLE,RECORD 유형이 올수 있다. ADDRESS FIELD 의 DATA TYPE 에 T_ADDRESS 라는 RECORD TYPE 정의.

③ 정의된 T_EMP_RECORD RECORD TYPE 의 변수를 선언하는 부분 입니다.

④ RECORD 내의 FIELD 에 데이터를 저장하거나 FIELD 의 데이터에 접근은
RECORD 변수명.FIELD 명(EX REC_EMP.EMPNO)

⑤ NESTED RECORD(중첩 레코드)의 FIELD 에 접근하는 방법은
RECORD 변수명.RECORD 변수명.FIELD 명 (EX REC_EMP.ADDRESS.ZIP).

4-3-2 TABLE TYPE

TABLE 이라는 단어로 인해 종종 혼동이 된다. 관계형 데이터베이스의 테이블과 단어상 혼동을 피하기 위해서 PL/SQL 테이블 또는 INDEX BY TABLE 이라고 호칭한다. PL/SQL 테이블은 행에 대해 배열처럼 액세스하기 위해 배열의 첨자와 유사한 개념으로 키(KEY)를 사용 한다. 배열의 데이터에 접근하기 위해서는 배열의 첨자를 사용하듯이 PL/SQL 테이블의 데이터에 접근하기 위해서는 키(KEY)를 사용한다.
사용할수 있는 데이터 유형으로는 SCALAR , COMPOSITE 유형을 사용할수 있으며 RECORD 실습에서처럼 사용할수 있는 유형에 COMPOSITE 유형이 있다는 것은 개발시 표현의 유연성을 줄수 있다.
PL/SQL 테이블은 일반 배열과는 달리 크기가 미리 정의되지 않고 동적으로 자유롭게 증가할 수 있어 타언어의 동적 배열과 유사 하다.

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

데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

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

① SCALAR를 사용하여 동적 배열을 구현 할수 있고
RECORD를 사용하면 RECORD의 동적 배열을 구현 할수 있다.

4_TABLE_1.SQL

```
DECLARE
        TYPE  T_EMP_LIST IS  TABLE  OF  VARCHAR2(20)          -- TABLE TYPE 정의
                INDEX  BY  BINARY_INTEGER;                    -- DYNAMIC ARRAY와 유사

        TBL_EMP_LIST     T_EMP_LIST;                          -- TABLE 변수 선언

        V_TMP            VARCHAR2(20);
        V_INDEX          NUMBER(10);
BEGIN

        TBL_EMP_LIST(1)      := 'SCOTT';      -- TABLE에 DATA를 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));

        -- METHOD사용 EX) DELETE,COUNT,FIRST,NEXT
        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;
/
```

데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

① T_EMP_LIST 는 사용자가 정의한 TABLE TYPE 으로 VARCHAR2(20)은 ARRAY 에 비유 한다면 1 CELL 의 크기에 해당.
INDEX BY BINARY_INTEGER 는 PL/SQL TABLE 의 KEY 에 사용되는 값을 BINARY_INTEGER DATA TYPE 으로 사용하겠다는 의미로 SYNTAX 문법상 사용자가 임의적으로 다른 DATA TYPE 을 사용할수 있는 것처럼 보이지만 Data Type 을 변경할수 없다. INDEX BY BINARY_INTEGER 는 PL/SQL TABLE 을 사용할때는 항상 사용하는 구문 이다.

<참고>PL/SQL TABLE 은 Oracle Server Process(서버 프로세스)가 사용하는 메모리 영역(PGA)영역에 할당됩니다. 즉 DBMS 가 설치된 H/W 시스템의 메모리 영역을 사용한다. 개발하는 PL/SQL 프로그램이 동시 사용성이 높고 PL/SQL TABLE 을 과다하게 사용해서 일시적인 메모리 자원 경합이 발생하는 경우도 있다.

② TBL_EMP_LIST 라는 사용자 정의 PL/SQL TABLE 변수를 선언

③ KEY 를 사용하여 TABLE 변수의 각 CELL 접근 (Key-Value pair)

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

KEY 값(1)을 통해 데이터를 저장 하고 데이터에 접근 한다.마치 관계형 데이터 베이스 TABLE 에 고유한 PRIMARY KEY 값을 통해 접근 하는것과 유사하여 TABLE ,KEY 라는 명칭을 사용한다.
일반 ARRAY 와 다른점은 KEY 에 사용되는 값이 1, 1000, -2134,0 등 연속적이지 않은 값을 사용할수 있다는 점이다.
예제는 비연속적인 또는 상호 관련 없는 값이 KEY 로 사용될수 있음을 보여주기 위한 것이지만 실제 프로그래밍시 위와 같은 의미 없는 불연속적이며 무의미한 KEY 값을 사용하게 되면 데이터에 대한 접근이 어렵게 된다.

④ TABLE METHOD
EXISTS(888)는 해당 KEY 에 데이터가 존재하는지 여부를 BOOLEAN 값으로 리턴 하는 METHOD(기능) 이다.PRIOR,FIRST,LAST,NEXT 는 PL/SQL TABLE 변수내의 데이터에 접근할 때 KEY 를 통해 순차적으로 접근할수 있는 방법을 제공 한다.
위의 예제는 METHOD 와 LOOP 를 사용하여 테이블 변수내의 모든 데이터에 순차적으로 접근 한다. PL/SQL 로 복잡한 데이터 처리시 중간 집계 데이터를 IN-LINE VIEW 나 TEMP TABLE 로 처리를 하지 못하는 경우에 유용하게 사용하는 기법이다.

⑤ PL/SQL BLOCK 내에 특정 데이터에 접근(SELECT, PL/SQL TABLE) 할때 해당 데이터가 없는 경우 NO_DATA_FOUND EXCEPTION 발생한다.

SQLCODE 와 SQLERRM 은 주로 2 가지 용도로 사용 한다.

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

(b) PL/SQL 로 개발된 모듈 (EX PROCEDURE,FUNCTION 등)의 실행 에러 기록
예제 처럼 화면에 결과를 출력 하는 것이 아니라 로그 테이블등에 INSERT 를 사용하여 실행시 발생한 에러를 기록하는 방법

<참고>
ORACLE DBMS 내의 모든 에러(ERROR)는 ERROR CODE 와 ERROR MESSAGE 가 사전 정의 되어 있다.

```
ORA-06531: 초기화되지 않은 모음을 참조합니다
ORA-06512: 줄 11에서
```

"6531"은 ERROR CODE 이며
"초기화되지 않은 모음을 참조합니다" 은 ERROR MESSAGE 입니다.
SQLCODE 는 ERROR CODE 를 SQLERRM 은 ERROR MESSAGE 를 리턴 한다.

4-4 DATA TYPE 과 참조연산자

%TYPE , %ROWTYPE 2 가지 유형의 참조 연산자.

4-4-1 %TYPE

- TABLE내 COLUMN의DATA TYPE,LENGTH만을 참조하여 변수 선언
- 참조하는 COLUMN의 NOT NULL제약사항을 참조하지 않는다.

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

EMPNO 컬럼은 NUMNER(4)로 정의 되어 있으며 실행 시점에
V_EMPNO EMP.EMPNO%TYPE;

➔ V_EMPNO NUMBER(4) 로 선언되어 사용 됩니다.

개발시 편리성 제공

PL/SQL로 개발하는 대부분의 MODULE은 관계형 데이타 베이스 테이블내 저장된 데이터를 (1) 조회 (2) 연산 (3) 반영 하는 구조 입니다.
PL/SQL에서 변수를 정의할때도 해당 변수는 관계형 테이블의 컬럼 정의와 동일하게 정의 하는 경우가 대부분 이다.
테이블의 컬럼 정의가 변경되는 경우 관련된 P/SQL 변수들도 변경을 해주어야 하나 참조 연산자를 사용하는 경우는 실행시 동적으로 참조 하기 때문에 변경을 해줄 필요가 없다.

참조연산자는 2가지 장점

① 편리성

② 유연성

4-4-2 %ROWTYPE

- TABLE,VIEW,CURSOR의 여러COLUMN을 참조하여 RECORD TYPE 생성
- 테이블내의 컬럼개수,데이터 타입,데이터 길이등을 확인하지 생성
- ROW 단위의 데이터 FETCH 시 편리하다.

데이터베이스의 테이블 또는 VIEW 의 일련의 열을 RECORD 로 선언하기 위하여 %ROWTYPE 를 사용한다. 데이터베이스 테이블 이름을 %ROWTYPE 앞에 접두어를 붙여 RECORD 를 선언하고 FIELD 는 테이블이나 VIEW 의 COLUMN 명과 데이터 타입과 LENGTH 을 그대로 가져올 수 있다.

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

EMP라는 테이블 전체를 참조(Reference)해서 REC_EMP Record 생성 하며 %TYPE 과 RECORD의 조합을 하나의 기능으로 구현.
EMP 테이블내의 모든 컬럼을 %TYPE으로 참조 한후 내부적으로 RECORD로 정의 된다.

데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

4_REF_1.SQL

```
DECLARE
        REC_EMP         EMP%ROWTYPE;            -- ROW    참조 연산자
        V_EMPNO         EMP.EMPNO%TYPE;         -- COLUMN 참조 연산자
BEGIN
        -- 1개의 ROW를 SELECT 해서 RECORD에 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 가능
        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;
/
```

① %ROWTYPE 을 사용하여 Row 형태의 데이터(ex Table,Cursor,View)를 참조하고 %TYPE 을 사용하여 Column 형태의 데이터를 참조

②

PL/SQL 실행부(BEGIN ~ END)에서 SELECT 를 사용하는 경우 항상 INTO 구문을 사용해야 한다. INTO 이하의 변수에 SELECT 결과를 저장 한다.

EMP 테이블은 8 개의 컬럼으로 구성되어 있는데 %ROWTYPE 의 참조 연산자가 없다면 ?
8 개의 변수를 정의해야 하고 INTO 이하에 8 개의 변수를 나열해야 한다.

③ %ROWTYPE 을 사용하여 선언한 REC_EMP 변수의 정체는 ?

RECORD 변수
%ROWTYPE 으로 참조 변수를 정의하면 PL/SQL 내부에서 암시적으로 RECORD 변수를 정의 한다.

④ SELECT 에서 2 개의 컬럼을 조회하기 때문에 INTO 이하에도 2 개의 변수를 사용해야 한다.
첫번째 변수는 %TYPE 을 이용하여 생성한 변수
두번째 변수는 %ROWTYPE 을 통해 선언한 변수를 구성하는 1 개의 FIELD 를 일반변수 처럼 사용하는 예.

4-5 BLOCK 내의 SELECT

4-5-1 SELECT 와 관련된 EXCEPTION

다음은 Oracle Error 중 일부분입니다.

EXCEPTION NAME	ERROR CODE	ERROR MESSAGE
NO_DATA_FOUND	ORA-01403	데이터를 찾을 수 없습니다.
NOT_LOGGED_ON	ORA-01012	데이터베이스에 연결되지 않은 상태에서 SQL 문을 실행하려는 경우
TOO_MANY_ROWS	ORA-01422	실제 인출은 요구된 것보다 많은 수의 행을 추출합니다
ZERO_DEVIDE	ORA-01476	제수가 0 입니다
	ORA-00904	부적합한 식별자
	ORA-00001	무결성 제약 조건(SCOTT.EMP_EMPNO_PK)에 위배됩니다

Oracle DBMS 내에 정의된 모든 Error 는 Error Code 와 Error Message 가 정의되어 있다.

① PL/SQL 내의 예외처리부(EXCEPTION SECTION)에서 빈번히 제어 해야하는 ERROR 들에게는 ERROR 의 이름을 부여 해서 명료 하게 하고 사용을 쉽게 한다.

② PL/SQL 내의 예외처리부(EXCEPTION SECTION)에서 빈번히 제어할 필요가 없는 ERROR 들은 ERROR 의 이름을 부여 하지 않는다.

이름이 숫자 코드보다 인식성이 좋아 식별성이 좋다.
PL/SQL 내에 이름이 없는 ERROR 들도 여러분이 이름을 부여 해줄수가 있다.

지난 3차시에 학습한 EXCEPTION을 다시 상기해보면
ORACLE DEFINED-EXCEPTION
* PREDEFINED-ORACLE- EXCEPTION
ex) NO_DATA_FOUND
TOO_MANY_ROWS,TIMEOUT_ON_RESOURCE..
* NON-PREDEFINED-ORACLE-EXCEPTION
ORACLE 에서 정의한 EXCEPTION 이 2 가지 유형이었습니다.
PREDEFINED-ORACLE- EXCEPTION이 ①번 유형이고
NON-PREDEFINED-ORACLE-EXCEPTION 이 ②번 유형입니다.

SELECT 와 관련된 주요한 EXCEPTION

NO_DATA_FOUND
조회하는 데이터가 0 건인 경우 발생하는 EXCEPTION
TOO_MANY_ROWS
조회하는 데이터가 >1 건인 경우 발생하는 EXCEPTION

데이터베이스 프로그래밍 – PL/SQL Data Type(4차시)

4_SELECT_1.SQL

```
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 = 1;

        DBMS_OUTPUT.PUT_LINE('SELECTED EXACTLY ONE ROW  '||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;
/
```

① EMP 테이블의 데이터중에 EMPNO = 1 인 데이터는 없습니다.
SELECT 문장 실행시 0 건의 데이터가 조회 되면 NO_DATA_FOUND EXCEPTION이 발생 합니다.

② 예외처리부(EXCEPTION SECTION)에서 해당 예외처리를 수행 합니다.

4_SELECT_2.SQL

```
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  '||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;
/
```

① EMP 테이블의 데이터중에 EMPNO >= 1 인 데이터는 1 개 이상입니다.
SELECT 문장 실행시 1 개이상의 데이터가 조회 되면 TOO_MANY_ROWS EXCEPTION 이 발생 합니다.
② 예외처리부(EXCEPTION SECTION)에서 해당 예외처리를 수행 합니다.

< 참고> TOO_MANY_ROWS는 스칼라변수에 복수개의 값이 저장될수 없기 때문에 RUN TIME ERROR(EXCEPTION)시 발생하는 것을 이해 할수 있습니다.
NO_DATA_FOUND는 0건의 데이터를 조회 했는데 왜 RUN TIME ERROR 일까요 ?
사실은 ERROR는 아니죠!! PL/SQL BLOCK내에서 SELECT를 수행하는 목적이 무었일까요? 대부분은 SELECT 한 데이터를 기반으로 다른 연산을 수행하기 때문입니다. SELECT 한것이 없기 때문에 후속 연산을 수행할 수가 없어 ERROR로 간주하여 예외처리를 할수 있도록 한것 입니다.