Working with Composite Data Types

Oracle SQL Structured Query Language
Working with Composite Data Types

1. 조합 데이터 유형 개요

- 스칼라(Scalar) 변수와 같이 조합 변수에도 데이터 유형이 있음
- 조합 데이터 유형
  - **RECORD**
  - **TABLE**
  - **NESTED TABLE** 및 **VARRAY**
- **RECORD** 데이터 유형은 서로 다른 데이터를 하나의 논리 단위로 처리할 때 사용
- **RECORD**는 필드에 저장된 관련 데이터 항목의 그룹으로서 레코드마다 고유한 이름과 데이터 유형을 포함
- **TABLE** 데이터 유형은 데이터 모음을 전체 객체로 참조하여 조작할 때 사용
- **TABLE**은 열과 기본 키를 포함하여 행을 배열 형태로 액세스할 수 있음
- 정의된 테이블 및 레코드는 재사용 가능

2. PL/SQL 레코드(RECORD)

2-1. 개요

- 필드(Field)라고 불리는 PL/SQL 테이블, 레코드, 임의의 스칼라 데이터 유형 중 하나 이상의 구성 요소 포함
- 3GL의 레코드 구조와 유사
- 데이터베이스 테이블의 행과 동일하지 않음
- 필드 모음을 하나의 논리 단위로 처리
- 테이블에서 데이터 행을 인출하여 처리하는 데 편리

Oracle SQL Structured Query Language
Working with Composite Data Types

2-2. PL/SQL 레코드 생성 (1)

```
TYPE type_name IS RECORD
      (field_declaration [,field_declaration] …) ;
identifier type_name;
```

- field_declaration 상세 기술

```
field_name {field_type | variable%TYPE
           | table.column%TYPE  | table%ROWTYPE}
           [[NOT NULL] {:= | DEFAULT} expr]
```

- 레코드(RECORD)를 생성하려면 레코드 유형을 정의한 후 해당 유형의 레코드 선언
- NOT NULL 제약조건이 있는 필드(Field)에는 널(Null)을 할당할 수 없으므로 NOT NULL 필드는 반드시 초기화

2-3. PL/SQL 레코드 생성 (2)

```
DECLARE
  TYPE emp_record_type IS RECORD
       (empno        NUMBER(4)  NOT NULL := 100,
        ename        emp.ename%TYPE,
        job          emp.job%TYPE);
  emp_record       emp_record_type;
...
```

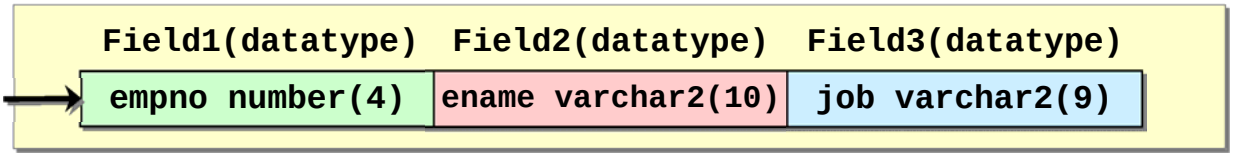
- 필드 선언은 변수 선언과 유사
- 각 필드는 고유한 이름과 특정 데이터 유형을 가짐
- PL/SQL 레코드에는 미리 정의된 데이터 유형이 없으므로 먼저 데이터 유형을 생성한 후 해당 데이터 유형을 사용하여 식별자 선언
- NOT NULL로 선언한 필드는 반드시 초기화

Oracle SQL Structured Query Language
Working with Composite Data Types

2-4. PL/SQL 레코드 생성 (3)

```
SQL> DECLARE
  2    TYPE dept_record_type IS RECORD
  3      (deptno    dept.deptno%TYPE,
  4       dname     dept.dname%TYPE,
  5       loc       dept.loc%TYPE);
  6    dept_record  dept_record_type;
  7  BEGIN
  8    SELECT deptno, dname, loc
  9    INTO   dept_record
 10    FROM   dept
 11    WHERE  deptno = 30;
 12    DBMS_OUTPUT.PUT_LINE(dept_record.deptno);
 13    DBMS_OUTPUT.PUT_LINE(dept_record.dname);
 14    DBMS_OUTPUT.PUT_LINE(dept_record.loc);
 15  END;
 16  /
```

2-4. PL/SQL 레코드 구조



- 레코드 참조 및 초기화
  - 레코드의 필드는 이름을 기준으로 액세스
  - 개별 필드를 참조하거나 초기화하려면 **record_name.field_name** 구문 사용
  - 블록 또는 하위 프로그램에서 사용자가 정의한 레코드는 블록 또는 하위 프로그램을 시작하면 인스턴스화되고 종료하면 없어짐
- 레코드에 값 할당
  - **SELECT** 또는 **FETCH** 문을 사용하여 레코드에 일반 값 할당 가능
  - 열 이름을 레코드의 필드와 동일한 순서로 표시
  - 데이터 유형이 동일하면 레코드를 다른 레코드에 할당 가능
  - 사용자가 정의한 레코드와 **%ROWTYPE** 레코드의 데이터 유형은 다름

Oracle SQL Structured Query Language
Working with Composite Data Types

2-5. %ROWTYPE

```
dept_record     dept%ROWTYPE;
emp_record      emp%ROWTYPE;
```

- **%ROWTYPE** 속성
  - 데이터베이스 테이블 또는 뷰(**VIEW**)의 열 모음에 따라 변수 선언
  - **%ROWTYPE**에 데이터베이스 테이블 이름을 접두어로 사용
  - 레코드 필드의 이름과 데이터 유형은 테이블 또는 뷰(**VIEW**)의 열에서 가져옴
- **%ROWTYPE** 사용의 장점
  - 기본 데이터베이스 열의 수 및 데이터 유형을 모르는 경우에 사용
  - 실행 중에 기본 데이터베이스 열의 수 및 데이터 유형 변경 가능
  - **SELECT** 문을 사용하여 행을 검색할 때 유용

2-5. %ROWTYPE

```
SQL> DECLARE
  2    dept_record  dept%ROWTYPE;
  3  BEGIN
  4    SELECT deptno, dname, loc
  5    INTO   dept_record
  6    FROM   dept
  7    WHERE  deptno = 30;
  8    DBMS_OUTPUT.PUT_LINE(dept_record.deptno);
  9    DBMS_OUTPUT.PUT_LINE(dept_record.dname);
 10    DBMS_OUTPUT.PUT_LINE(dept_record.loc);
 11  END;
 12  /
```


Oracle SQL Structured Query Language
Working with Composite Data Types

3. PL/SQL 테이블(TABLE)

3-1. 개요

- 데이터베이스 테이블과 동일하지는 않음
- 배열과 유사
- 다음 두 개의 구성 요소를 포함
  - **BINARY_INTERGER** 데이터 유형의 기본 키
  - 스칼라 또는 레코드 데이터 유형의 열
- 크기 제한이 없으므로 동적으로 증가

3-2. PL/SQL 테이블 생성 (1)

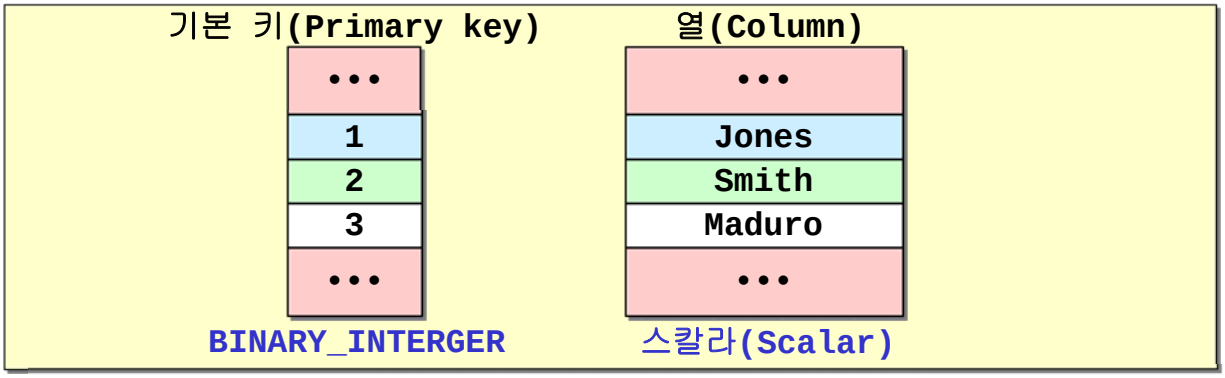
```
TYPE   type_name IS TABLE OF
       {column_type | variable%TYPE | table.column%TYPE}
       [NOT NULL] INDEX  BY  BINARY_INTEGER];
identifier   type_name;
```

- PL/SQL 테이블 생성은 다음 두 단계로 구성
  - 테이블의 데이터 유형 선언
  - 선언한 해당 데이터 유형의 변수 선언
- NOT NULL 제약조건을 사용하면 해당 유형의 PL/SQL 테이블에 널을 할당할 수 없음
- PL/SQL 테이블은 초기화하지 않음

3-3. PL/SQL 테이블 생성 (2)

```
DECLARE
  TYPE ename_table_type IS TABLE OF emp.ename%type
       INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
       INDEX BY BINARY_INTEGER;
  ename_table       ename_table_type;
  hiredate_table    hiredate_table_type;
BEGIN
  ename_table(1) := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
       IF ename_table.EXIST(1) THEN
            INSERT  INTO •••
       •••
END;
```

3-4. PL/SQL 테이블 구조



- 데이터베이스 테이블과 마찬가지로 **PL/SQL** 테이블도 크기 제한이 없음
- **PL/SQL** 테이블에는 하나의 열과 하나의 기본 키(**Primary key**)가 있는데 둘 중 어느 것에도 이름 지정 불가능
- 열은 스칼라 또는 레코드 데이터 유형일 수 있지만 기본 키(**Primary key**)는 **BINARY_INTERGER** 유형
- **PL/SQL** 테이블은 선언 부분에서 초기화 불가능

Oracle SQL Structured Query Language
Working with Composite Data Types

3-5. PL/SQL 테이블 메소드

```
table_name.method_name [ (parameter) ]
```

- 메소드를 사용해서 **PL/SQL** 테이블을 쉽게 사용할 수 있음

메소드	설 명
EXISTS(*n*)	***n*번째 요소가 존재하면 TRUE 반환**
COUNT	현재 포함하고 있는 요소 수 반환
FIRST & LAST	첫번째와 마지막 인덱스 번호 반환
PRIOR(*n*)	인덱스 ***n*** 앞의 인덱스 번호 반환
NEXT(*n*)	인덱스 ***n*** 다음의 인덱스 번호 반환
EXTEND(*n,i*)	**PL/SQL** 테이블의 크기 증가
TRIM	끝에서 요소를 하나 제거
DELETE	모든 요소를 제거

3-6. 레코드로 구성된 PL/SQL 테이블

```
DECLARE
  TYPE dept_table_type IS TABLE OF dept%ROWTYPE
    INDEX BY BINARY_INTEGER ;
  dept_table   dept_table_type ;
  -- dept_table의 각 요소들은 하나의 레코드
```

- 허용된 **PL/SQL** 데이터 유형을 사용하여 테이블을 변수를 정의
- 데이터베이스 테이블의 모든 필드에 관한 정보를 저장하는 데 하나의 테이블 정의만 필요하므로 레코드로 구성된 테이블은 **PL/SQL** 테이블의 기능을 상당히 향상

Oracle SQL Structured Query Language
Working with Composite Data Types

3-7. PL/SQL 테이블 실습

```
SQL> DECLARE
  2    TYPE dept_table_type IS TABLE OF NUMBER
  3      INDEX BY BINARY_INTEGER;
  4    dept_table  dept_table_type;
  5    v_total  NUMBER;
  6  BEGIN
  7    FOR v_counter IN 1..50 LOOP
  8      dept_table(v_counter) := v_counter;
  9    END LOOP;
 10    v_total := dept_table.COUNT;
 11    DBMS_OUTPUT.PUT_LINE(v_total);
 12  END;
 13  /
```

3-7. PL/SQL 테이블 실습

```
SQL> DECLARE
  2    TYPE test_table_type IS TABLE OF VARCHAR2(10)
  3      INDEX BY BINARY_INTEGER;
  4    test_table  test_table_type;
  5  BEGIN
  6    test_table(1) := 'One';
  7    test_table(3) := 'Three';
  8    test_table(-2) := 'Minus Two';
  9    test_table(0) := 'Zero';
 10    test_table(100) := 'Hundred';
 11    DBMS_OUTPUT.PUT_LINE(test_table(1));
 12    DBMS_OUTPUT.PUT_LINE(test_table(3));
 13    DBMS_OUTPUT.PUT_LINE(test_table(-2));
 14    DBMS_OUTPUT.PUT_LINE(test_table(0));
 15    DBMS_OUTPUT.PUT_LINE(test_table(100));
 16    test_table.DELETE(100);
 17    test_table.DELETE(1,3);
 18    test_table.DELETE;
 19  END;
 20  /
```


Oracle SQL Structured Query Language
Working with Composite Data Types

3-7. PL/SQL 테이블 실습

```
SQL> DECLARE
  2    TYPE emp_table_type IS TABLE OF VARCHAR2(13)
  3      INDEX BY BINARY_INTEGER;
  4    emp_table  emp_table_type;
  5  BEGIN
  6    emp_table(1) := 'SCOTT';
  7    emp_table(3) := 'JONE';
  8    DBMS_OUTPUT.PUT_LINE(emp_table(1));
  9    DBMS_OUTPUT.PUT_LINE(emp_table(3));
 10    IF emp_table.EXISTS(1) THEN
 11      INSERT INTO emp (empid, empname)
 12      VALUES (100, 'Row 1 있음');
 13    ELSE
 14      INSERT INTO emp (empid, empname)
 15      VALUES (100, 'Row 1 없음');
 16    END IF;
 17    IF emp_table.EXISTS(2) THEN
 18      INSERT INTO emp (empid, empname)
 19      VALUES (200, 'Row 2 있음');
 20    ELSE
 21      INSERT INTO emp (empid, empname)
 22      VALUES (200, 'Row 2 없음');
 23    END IF;
 24  END;
 25  /
```

YD 예담직업전문학교

3-7. PL/SQL 테이블 실습

```
SQL> DECLARE
  2     TYPE emp_table_type IS TABLE OF emp.empname%TYPE
  3       INDEX BY BINARY_INTEGER;
  4     emp_table  emp_table_type;
  5     v_index    BINARY_INTEGER;
  6   BEGIN
  7     emp_table(43) := 'MASON';
  8     emp_table(50) := 'JUNEBUG';
  9     emp_table(47) := 'TALLER';
 10     v_index := emp_table.FIRST;
 11     DBMS_OUTPUT.PUT_LINE(v_index);
 12     v_index := emp_table.LAST;
 13     DBMS_OUTPUT.PUT_LINE(v_index);
 14   END;
 15  /
```