

레코드와 컬렉션

오라클 데이터베이스에서는 한 번에 여러 데이터를 관리하거나 저장하기 위해 레코드, 컬렉션 같은 자료형을 제공합니다. 이 장에서는 레코드와 컬렉션의 정의 및 사용법에 대해 살펴보겠습니다.

17-1 자료형이 다른 여러 데이터를 저장하는 레코드

17-1 자료형이 같은 여러 데이터를 저장하는 컬렉션

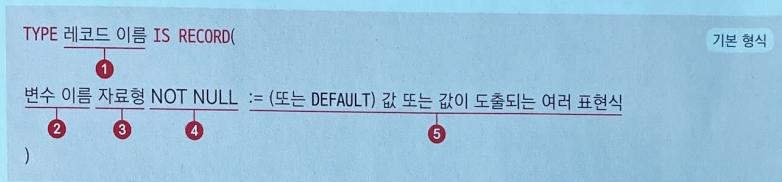
이 장에서 꼭 익혀야 할 것

- 레코드의 의미와 사용 방법
- 연관 배열의 의미와 사용 방법

17-1 자료형이 다른 여러 데이터를 저장하는 레코드

레코드란?

레코드(record)는 자료형이 각기 다른 데이터를 하나의 변수에 저장하는 데 사용합니다. 기본 형식은 다음과 같습니다.



번호	설명
①	저장할 레코드 이름을 지정합니다.
②	레코드 안에 포함할 변수를 지정합니다. 변수는 여러 개 지정할 수 있으며 각 변수는 쉼표(,)로 구분합니다.
③	지정한 변수의 자료형을 지정합니다. 이 자료형 역시 %TYPE, %ROWTYPE 지정이 가능합니다.
④	지정한 변수에 NOT NULL 제약 조건을 지정합니다(생략 가능).
⑤	기본값을 지정합니다(생략 가능).

☞ C, C++, Java 같은 프로그래밍 언어의 구조체(Structure), 클래스(Class) 개념과 비슷합니다.

정의한 레코드는 지금까지 다른 변수와 마찬가지로 기존 자료형처럼 사용할 수 있습니다. 레코드에 포함된 변수는 레코드 이름과 마침표(.)로 사용할 수 있습니다.

☞ 이 장의 모든 예제는 SQL*PLUS에서 실행합니다.

실습 17-1 레코드 정의해서 사용하기

• 완성 파일 17-1.sql

```
01  DECLARE
02      TYPE REC_DEPT IS RECORD(
03          deptno NUMBER(2) NOT NULL := 99,
04          dname DEPT.DNAME%TYPE,
05          loc DEPT.LOC%TYPE
06      );
07      dept_rec REC_DEPT;
08  BEGIN
09      dept_rec.deptno := 99;
10      dept_rec.dname := 'DATABASE';
11      dept_rec.loc := 'SEOUL';
12      DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || dept_rec.deptno);
13      DBMS_OUTPUT.PUT_LINE('DNAME : ' || dept_rec.dname);
14      DBMS_OUTPUT.PUT_LINE('LOC : ' || dept_rec.loc);
15  END;
16 /
```



07행의 dept_rec가 레코드
변수 이름입니다.

:: 결과 화면

```
DEPTNO : 99
DNAME : DATABASE
LOC : SEOUL
```

02~06행 레코드를 정의합니다.

07행 선언한 레코드형으로 변수를 선언합니다. 같은 레코드형으로 변수 여러 개를 선언할 수 있습니다.

09~11행 레코드형으로 선언한 변수 안에 포함된 변수에 값을 대입합니다.

12~14행 저장된 값을 사용할 때도 레코드형 변수 이름과 마침표(.)를 사용합니다.

레코드를 사용한 INSERT

PL/SQL문에서는 테이블에 데이터를 삽입하거나 수정하는 INSERT, UPDATE문에도 레코드를 사용할 수 있습니다. 레코드를 사용할 테이블 DEPT_RECORD를 먼저 만듭니다.

실습 17-2 DEPT_RECORD 테이블 생성하기

• 완성 파일 17-2.sql

```
01  CREATE TABLE DEPT_RECORD
02  AS SELECT * FROM DEPT;
```

:: 결과 화면

테이블이 만들어졌습니다.

실습 17-2 DEPT_RECORD 테이블 생성하기(생성된 테이블 조회)

• 완성 파일 17-2.sql

```
01 SELECT * FROM DEPT_RECORD;
```

:: 결과 화면

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

기존 INSERT문에서는 삽입할 데이터를 VALUES절에 하나씩 명시하였습니다. 하지만 INSERT문에 레코드를 사용하면 VALUES절에 레코드 이름만 명시해도 됩니다. 그리고 선언한 레코드와 INSERT 대상이 되는 테이블의 데이터 개수, 자료형, 순서를 맞추어야 한다는 것도 함께 기억하세요.

실습 17-3 레코드를 사용하여 INSERT하기

• 완성 파일 17-3.sql

```
01 DECLARE
02   TYPE REC_DEPT IS RECORD(
03     deptno NUMBER(2) NOT NULL := 99,
04     dname DEPT.DNAME%TYPE,
05     loc DEPT.LOC%TYPE
06   );
07   dept_rec REC_DEPT;
08 BEGIN
09   dept_rec.deptno := 99;
10   dept_rec.dname := 'DATABASE';
11   dept_rec.loc := 'SEOUL';
12
13   INSERT INTO DEPT_RECORD
14   VALUES dept_rec;
15 END;
16 /
```

실습 17-3 레코드를 사용하여 INSERT하기(테이블 조회)

· 완성 파일 17-3.sql

```
01 SELECT * FROM DEPT_RECORD;
```

:: 결과 화면

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
99	DATABASE	SEOUL

레코드를 사용한 UPDATE

레코드는 UPDATE문에도 사용할 수 있습니다. 이 경우에 SET절은 ROW 키워드와 함께 레코드 이름을 명시합니다. 기존 UPDATE문에서는 SET절을 통해 변경할 열을 하나하나 지정한 것과 달리 레코드에 저장된 데이터를 사용하여 행 전체의 데이터를 바꿔 줍니다.

실습 17-4 레코드를 사용하여 UPDATE하기

· 완성 파일 17-4.sql

```
01 DECLARE
02   TYPE REC_DEPT IS RECORD(
03     deptno NUMBER(2) NOT NULL := 99,
04     dname DEPT.DNAME%TYPE,
05     loc DEPT.LOC%TYPE
06   );
07   dept_rec REC_DEPT;
08 BEGIN
09   dept_rec.deptno := 50;
10   dept_rec.dname := 'DB';
11   dept_rec.loc := 'SEOUL';
12
13   UPDATE DEPT_RECORD
14   SET ROW = dept_rec
15   WHERE DEPTNO = 99;
16 END;
17 /
```

실습 17-4 레코드를 사용하여 UPDATE하기(테이블 조회)

• 완성 파일 17-4.sql

```
01 SELECT * FROM DEPT_RECORD;
```

:: 결과 화면

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DB	SEOUL

레코드를 포함하는 레코드

레코드에 포함된 변수의 자료형을 지정할 때 다른 레코드를 지정할 수도 있습니다. 실습 17-5는 REC_DEPT, REC_EMP 두 개 레코드를 사용하고 있습니다. REC_EMP 레코드는 dinfo 변수에 REC_DEPT 레코드를 자료형으로 지정합니다. 레코드 역시 자료형이기 때문이죠. 변수에 레코드형을 적용했으므로 두 개의 마침표(.)로 값을 사용하고 있는 것을 눈여겨보세요. 이렇게 레코드 안에 또 다른 레코드를 포함한 형태를 중첩 레코드(nested record)라고 합니다.

실습 17-5 레코드에 다른 레코드 포함하기

• 완성 파일 17-5.sql

```
01 DECLARE
02   TYPE REC_DEPT IS RECORD(
03     deptno DEPT.DEPTNO%TYPE,
04     dname DEPT.DNAME%TYPE,
05     loc DEPT.LOC%TYPE
06   );
07   TYPE REC_EMP IS RECORD(
08     empno EMP.EMPNO%TYPE,
09     ename EMP.ENAME%TYPE,
10     dinfo REC_DEPT
11   );
12   emp_rec REC_EMP;
13 BEGIN
14   SELECT E.EMPNO, E.ENAME, D.DEPTNO, D.DNAME, D.LOC
15
16   INTO emp_rec.empno, emp_rec.ename,
17       emp_rec.dinfo.deptno,
```

```
18      emp_rec.dinfo.dname,
19      emp_rec.dinfo.loc
20  FROM EMP E, DEPT D
21 WHERE E.DEPTNO = D.DEPTNO
22 AND E.EMPNO = 7788;
23 DBMS_OUTPUT.PUT_LINE('EMPNO : ' || emp_rec.empno);
24 DBMS_OUTPUT.PUT_LINE('ENAME : ' || emp_rec.ename);
25
26 DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || emp_rec.dinfo.deptno);
27 DBMS_OUTPUT.PUT_LINE('DNAME : ' || emp_rec.dinfo.dname);
28 DBMS_OUTPUT.PUT_LINE('LOC : ' || emp_rec.dinfo.loc);
29 END;
30 /
```

:: 결과 확인

```
EMPNO : 7788
ENAME : SCOTT
DEPTNO : 20
DNAME : RESEARCH
LOC : DALLAS
```

17-2 자료형이 같은 여러 데이터를 저장하는 컬렉션

컬렉션은 특정 자료형의 데이터를 여러 개 저장하는 복합 자료형입니다. 여러 종류의 데이터를 하나로 묶어 사용하는 레코드를 테이블의 한 행처럼 사용한다면, 컬렉션은 열 또는 테이블과 같은 형태로 사용할 수 있습니다. PL/SQL에서 사용할 수 있는 컬렉션은 다음과 같이 세 가지 종류가 있습니다.

- 연관 배열(associative array (or index by table))
- 중첩 테이블(nested table)
- VARRAY(variable-size array)

이 책에서는 위 세 가지 컬렉션 중 사용 빈도가 가장 높은 연관 배열을 알아보겠습니다.

☞ 그 밖에 컬렉션의 자세한 내용은 오라클 공식 문서(docs.oracle.com/cd/B28359_01/appdev.111/b28370/collections.htm#LNPLS005)를 참고하세요.

연관 배열

연관 배열은 인덱스라고도 불리는 키(key), 값(value)으로 구성되는 컬렉션입니다. 중복되지 않은 유일한 키를 통해 값을 저장하고 불러오는 방식을 사용합니다. 연관 배열을 정의할 때 자료형이 TABLE인 변수를 다음과 같이 작성합니다.

TYPE 연관 배열 이름 IS TABLE OF 자료형 [NOT NULL]
INDEX BY 인덱스형;
① ② ③

기본 형식

번호	설명
①	작성할 연관 배열 이름을 지정합니다.
②	연관 배열 열에 사용할 자료형을 지정합니다. 이 자료형에는 VARCHAR2, DATE, NUMBER와 같은 단일 자료형을 지정할 수 있고 %TYPE, %ROWTYPE 같은 참조 자료형도 사용할 수 있습니다. NOT NULL 옵션을 사용할 수 있으며 생략 가능합니다.
③	키로 사용할 인덱스의 자료형을 지정합니다. BINARY_INTEGER, PLS_INTEGER 같은 정수 또는 VARCHAR2 같은 문자 자료형도 사용할 수 있습니다.

이렇게 정의한 연관 배열은 레코드와 마찬가지로 특정 변수의 자료형으로서 사용할 수 있습니다. 먼저 간단한 형태의 연관 배열을 작성해 볼까요? 다음 연관 배열은 인덱스 값을 정수로 사용하여 값을 저장합니다.

☞ 다른 프로그래밍 언어가 익숙하다면 배열과 유사한 방식으로 사용 가능하다는 것을 알 수 있습니다.

실습 17-6 연관 배열 사용하기

· 완성 파일 17-6.sql

```
01  DECLARE
02      TYPE ITAB_EX IS TABLE OF VARCHAR2(20)
03      INDEX BY PLS_INTEGER;
04
05      text_arr ITAB_EX;
06
07  BEGIN
08      text_arr(1) := '1st data';
09      text_arr(2) := '2nd data';
10      text_arr(3) := '3rd data';
11      text_arr(4) := '4th data';
12
13      DBMS_OUTPUT.PUT_LINE('text_arr(1) : ' || text_arr(1));
14      DBMS_OUTPUT.PUT_LINE('text_arr(2) : ' || text_arr(2));
15      DBMS_OUTPUT.PUT_LINE('text_arr(3) : ' || text_arr(3));
16      DBMS_OUTPUT.PUT_LINE('text_arr(4) : ' || text_arr(4));
17  END;
18 /
```

:: 결과 화면

```
text_arr(1) : 1st data
text_arr(2) : 2nd data
text_arr(3) : 3rd data
text_arr(4) : 4th data
```

레코드를 활용한 연관 배열

연관 배열의 자료형에는 레코드를 사용할 수 있습니다. 이 경우에 다양한 자료형을 포함한 레코드를 여러 개 사용할 수 있으므로 마치 테이블과 같은 데이터 사용과 저장이 가능합니다.

실습 17-7 연관 배열 자료형에 레코드 사용하기

· 완성 파일 17-7.sql

```
01  DECLARE
02      TYPE REC_DEPT IS RECORD(
03          deptno DEPT.DEPTNO%TYPE,
```

```

04      dname DEPT.DNAME%TYPE          :: 결과 화면
05  );
06
07  TYPE ITAB_DEPT IS TABLE OF REC_DEPT
08      INDEX BY PLS_INTEGER;
09
10  dept_arr ITAB_DEPT;
11  idx PLS_INTEGER := 0;
12
13 BEGIN
14   FOR i IN (SELECT DEPTNO, DNAME FROM DEPT) LOOP
15     idx := idx + 1;
16     dept_arr(idx).deptno := i.DEPTNO;
17     dept_arr(idx).dname := i.DNAME;
18
19     DBMS_OUTPUT.PUT_LINE(
20       dept_arr(idx).deptno || ' : ' || dept_arr(idx).dname);
21   END LOOP;
22 END;
23 /

```

만약 특정 테이블의 전체 열과 같은 구성을 가진 연관 배열을 제작한다면 다음과 같이 %ROWTYPE을 사용하는 것이 레코드를 정의하는 것보다 간편합니다.

실습 17-8 %ROWTYPE으로 연관 배열 자료형 지정하기

• 완성 파일 17-8.sql

```

01  DECLARE
02    TYPE ITAB_DEPT IS TABLE OF DEPT%ROWTYPE
03      INDEX BY PLS_INTEGER;
04
05    dept_arr ITAB_DEPT;
06    idx PLS_INTEGER := 0;
07
08 BEGIN
09   FOR i IN(SELECT * FROM DEPT) LOOP
10     idx := idx + 1;
11     dept_arr(idx).deptno := i.DEPTNO;
12     dept_arr(idx).dname := i.DNAME;
13     dept_arr(idx).loc := i.LOC;
14
15     DBMS_OUTPUT.PUT_LINE(

```

```

16      dept_arr(idx).deptno || ' : ' ||
17      dept_arr(idx).dname || ' : ' ||
18      dept_arr(idx).loc);
19  END LOOP;
20 END;
21 /

```

:: 결과 화면
10 : ACCOUNTING : NEW YORK
20 : RESEARCH : DALLAS
30 : SALES : CHICAGO
40 : OPERATIONS : BOSTON

☞ 연관 배열을 이런 식으로 사용하는 것은 다음 장에서 배울 커서를 사용할 때도 동일하게 적용할 수 있습니다.

컬렉션 메서드

오라클에서는 컬렉션 사용상의 편의를 위해 몇 가지 서브프로그램을 제공하고 있습니다. 이를 컬렉션 메서드라고 합니다. 컬렉션 메서드는 컬렉션과 관련된 다양한 정보 조회 기능을 제공합니다. 이와 더불어 컬렉션 내의 데이터 삭제나 컬렉션 크기 조절을 위한 특정 조작도 가능합니다.

메서드	설명
EXISTS(n)	컬렉션에서 n인덱스의 데이터 존재 여부를 true/false로 반환합니다.
COUNT	컬렉션에 포함되어 있는 요소 개수를 반환합니다.
LIMIT	현재 컬렉션의 최대 크기를 반환합니다. 최대 크기가 없으면 NULL을 반환합니다.
FIRST	컬렉션의 첫 번째 인덱스 번호를 반환합니다.
LAST	컬렉션의 마지막 인덱스 번호를 반환합니다.
PRIOR(n)	컬렉션에서 n인덱스 바로 앞 인덱스 값을 반환합니다. 대상 인덱스 값이 존재하지 않는다면 NULL을 반환합니다.
NEXT(n)	컬렉션에서 n인덱스 바로 다음 인덱스 값을 반환합니다. 대상 인덱스 값이 존재하지 않는다면 NULL을 반환합니다.
DELETE	컬렉션에 저장된 요소를 지우는 데 사용합니다. 다음 세 가지 방식으로 사용합니다. <ul style="list-style-type: none"> • DELETE : 컬렉션에 저장되어 있는 모든 요소를 삭제합니다. • DELETE(n) : n인덱스의 컬렉션 요소를 삭제합니다. • DELETE(n, m) : n인덱스부터 m인덱스까지 요소를 삭제합니다.
EXTEND	컬렉션 크기를 증가시킵니다. 연관 배열을 제외한 중첩 테이블과 VARRAY에서 사용합니다.
TRIM	컬렉션 크기를 감소시킵니다. 연관 배열을 제외한 중첩 테이블과 VARRAY에서 사용합니다.

컬렉션 메서드는 컬렉션형으로 선언한 변수에 마침표(.)와 함께 작성하여 사용할 수 있습니다.

```

01  DECLARE
02      TYPE ITAB_EX IS TABLE OF VARCHAR2(20)
03      INDEX BY PLS_INTEGER;
04
05      text_arr ITAB_EX;
06
07  BEGIN
08      text_arr(1) := '1st data';
09      text_arr(2) := '2nd data';
10      text_arr(3) := '3rd data';
11      text_arr(50) := '50th data';
12
13      DBMS_OUTPUT.PUT_LINE('text_arr.COUNT : ' || text_arr.COUNT);
14      DBMS_OUTPUT.PUT_LINE('text_arr.FIRST : ' || text_arr.FIRST);
15      DBMS_OUTPUT.PUT_LINE('text_arr.LAST : ' || text_arr.LAST);
16      DBMS_OUTPUT.PUT_LINE('text_arr.PRIOR(50) : ' || text_arr.PRIOR(50));
17      DBMS_OUTPUT.PUT_LINE('text_arr.NEXT(50) : ' || text_arr.NEXT(50));
18
19  END;
20 /

```

:: 결과 화면

```

text_arr.COUNT : 4
text_arr.FIRST : 1
text_arr.LAST : 50
text_arr.PRIOR(50) : 3
text_arr.NEXT(50) :

```

☞ text_arr.NEXT(50)은 50번 인덱스의 다음 인덱스가 존재하지 않으므로 NULL이 반환되어 값이 비어있는 상태로 출력됩니다.

I. 본 복습

다음 빈칸을 채우며 복습해 보세요.

오라클에서는 여러 가지 데이터를 하나의 자료형으로 지정하고 사용하기 위해 직접 정의하는 ¹ 복 ² 레 ³ 컬 형은 여러 종류의 자료형을 하나의 변수에 저장할 때 사용합니다. 형은 특정 자료형의 데이터 여러 개를 하나의 변수에 저장할 때 사용합니다.

답변 1. 키워드 2. 컬렉션 3. 타입

Q1 다음과 같은 결과가 나오도록 PL/SQL문을 작성해 보세요.

- ① EMP 테이블과 같은 열 구조를 가지는 빈 테이블 EMP_RECORD를 생성하는 SQL문을 작성해 보세요.
- ② EMP_RECORD 테이블에 코드를 사용하여 새로운 사원 정보를 다음과 같이 삽입하는 PL/SQL 프로그램을 작성해 보세요.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
      ... [변수 선언 및 값 대입]
      BEGIN
          ... [PL/SQL 작성]
      END;
      /
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL> SELECT * FROM EMP_RECORD;
EMPNO ENAME JOB MGR HIREDATE SAL COMM
-----
DEPTNO
-----
1111 TEST_USER TEST_JOB 18/03/01 3000
40
SQL>
```

Q2 EMP 테이블을 구성하는 모든 열을 저장할 수 있는 레코드를 활용하여 연관 배열을 작성해 보세요. 그리고 저장된 연관 배열의 내용을 다음과 같이 출력해 보세요.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
... [변수 선언 및 값 대입]
BEGIN
... [PL/SQL 작성]
END;
/
7369 : SMITH : CLERK : 7902 : 80/12/17 : 800 :: 20
7499 : ALLEN : SALESMAN : 7698 : 81/02/20 : 1600 : 300 : 30
7521 : WARD : SALESMAN : 7698 : 81/02/22 : 1250 : 500 : 30
7566 : JONES : MANAGER : 7839 : 81/04/02 : 2975 :: 20
7654 : MARTIN : SALESMAN : 7698 : 81/09/28 : 1250 : 1400 : 30
7698 : BLAKE : MANAGER : 7839 : 81/05/01 : 2850 :: 30
7782 : CLARK : MANAGER : 7839 : 81/06/09 : 2450 :: 10
7788 : SCOTT : ANALYST : 7566 : 87/04/19 : 3000 :: 20
7839 : KING : PRESIDENT : : 81/11/17 : 5000 :: 10
7844 : TURNER : SALESMAN : 7698 : 81/09/08 : 1500 : 0 : 30
7876 : ADAMS : CLERK : 7788 : 87/05/23 : 1100 :: 20
7900 : JAMES : CLERK : 7698 : 81/12/03 : 950 :: 30
7902 : FORD : ANALYST : 7566 : 81/12/03 : 3000 :: 20
7934 : MILLER : CLERK : 7782 : 82/01/23 : 1300 :: 10
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

정답 이지스퍼블리싱 홈페이지에서 확인하세요.