

## 커서와 예외 처리

PL/SQL은 16장에서 소개한 기본 프로그래밍 언어의 특징뿐만 아니라 데이터를 더욱 효과적으로 사용할 수 있도록 여러 기능을 제공합니다. 이 장에서는 필요에 따라 PL/SQL 내부에서 SQL문의 실행 및 결과를 사용할 수 있는 강력한 기능을 제공하는 커서와 PL/SQL문을 실행할 때 발생할 수 있는 오류를 처리하는 예외 처리에 대해 알아보겠습니다.

18-1 특정 열을 선택하여 처리하는 커서

18-2 오류가 발생해도 프로그램이 비정상 종료되지 않도록 하는 예외 처리

### 이 장에서 꼭 익혀야 할 것

- 커서의 의미와 사용 방법
- 예외 처리의 의미와 사용 방법

# 18-1 특정 열을 선택하여 처리하는 커서

## 커서란?

커서(cursor)는 SELECT문 또는 데이터 조작어 같은 SQL문을 실행했을 때 해당 SQL문을 처리하는 정보를 저장한 메모리 공간을 뜻합니다.

☞ 메모리 공간은 Private SQL Area라고 부르며 정확히 표현하자면 커서는 이 메모리의 포인터를 말합니다.

커서를 사용하면 실행된 SQL문의 결과 값을 사용할 수 있습니다. 예를 들어 SELECT문의 결과 값이 여러 행으로 나왔을 때 각 행별로 특정 작업을 수행하도록 기능을 구현하는 것이 가능합니다. 오라클에 저장된 데이터 활용을 극대화할 수 있는 강력한 기능이죠. 커서는 사용 방법에 따라 명시적(explicit) 커서와 묵시적(implicit) 커서로 나뉩니다. 커서를 사용하기에 앞서 PL/SQL문에서 SELECT INTO를 통해 결과 행을 사용하는 방식을 먼저 살펴보겠습니다.

☞ 묵시적 커서는 암시적 커서라고도 부릅니다.

## SELECT INTO 방식

16장의 참조형을 설명하는 예제에서 SELECT INTO문을 이미 사용해 보았습니다. SELECT INTO문은 조회되는 데이터가 단 하나의 행일 때 사용 가능한 방식입니다. 커서는 결과 행이 하나이든 여러 개이든 상관없이 사용할 수 있습니다. SELECT INTO문은 SELECT절에 명시한 각 열의 결과 값을 다음과 같이 변수에 대입해 줍니다. 당연한 이야기이지만 SELECT절에 명시한 열과 INTO절에 명시한 변수는 그 개수와 자료형이 일치해야 합니다. 또한 INTO절을 제외한 나머지 부분은 SELECT문과 사용법이 같다는 것도 기억하세요. 그리고 WHERE, GROUP BY절도 사용할 수 있습니다.

```
SELECT 열1, 열2, ..., 열n INTO 변수1, 변수2, ..., 변수n  
FROM ...
```

기본 형식

다음 실습은 SELECT INTO문을 사용하여 DEPT 테이블의 40번 부서 데이터를 조회하고 각 열의 결과 값을 변수에 저장한 후 출력합니다.

### 실습 18-1 SELECT INTO를 사용한 단일행 데이터 저장하기

• 완성 파일 18-1.sql

```
01  DECLARE
02      V_DEPT_ROW DEPT%ROWTYPE;
03  BEGIN
04      SELECT DEPTNO, DNAME, LOC INTO V_DEPT_ROW
05      FROM DEPT
06      WHERE DEPTNO = 40;
07      DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO);
08      DBMS_OUTPUT.PUT_LINE('DNAME : ' || V_DEPT_ROW.DNAME);
09      DBMS_OUTPUT.PUT_LINE('LOC : ' || V_DEPT_ROW.LOC);
10  END;
11 /
```

:: 결과 화면

```
DEPTNO : 40
DNAME : OPERATIONS
LOC : BOSTON
```

☞ 출력 결과가 나오지 않는다면 SET SERVEROUTPUT ON 명령어의 실행을 확인해 주세요.

데이터 조회의 결과 값은 하나인 경우보다 여러 개인 경우가 흔하며 결과 행이 하나일지 여러 개일지 알 수 없는 경우도 존재하므로 대부분 커서를 활용합니다.

## 명시적 커서

먼저 명시적 커서의 사용법을 알아보겠습니다. 명시적 커서는 사용자가 직접 커서를 선언하고 사용하는 커서를 뜻합니다. 다음과 같은 단계를 거쳐 사용합니다.

단계	명칭	설명
1단계	커서 선언 (declaration)	사용자가 직접 이름을 지정하여 사용할 커서를 SQL문과 함께 선언합니다.
2단계	커서 열기 (open)	커서를 선언할 때 작성한 SQL문을 실행합니다. 이때 실행한 SQL문에 영향을 받는 행을 active set라 합니다.
3단계	커서에서 읽어온 데이터 사용 (fetch)	실행된 SQL문의 결과 행 정보를 하나씩 읽어 와서 변수에 저장한 후 필요한 작업을 수행합니다. 각 행별로 공통 작업을 반복해서 실행하기 위해 여러 종류의 LOOP문을 함께 사용할 수 있습니다.
4단계	커서 닫기 (close)	모든 행의 사용이 끝나고 커서를 종료합니다.

위 단계에 따라 PL/SQL문에서 명시적 커서를 작성하는 방법은 다음과 같습니다.

```
DECLARE
    CURSOR 커서 이름 IS SQL문; -- 커서 선언(Declaration)
BEGIN
    OPEN 커서 이름;           -- 커서 열기(Open)
    FETCH 커서이름 INTO 변수 -- 커서로부터 읽어온 데이터 사용(Fetch)
    CLOSE 커서이름;          -- 커서 닫기(Close)
END;
```

기본 형식

만약 커서에 여러 행이 조회되는 SELECT문을 사용할 때 필요에 따라 다양한 LOOP문을 활용하여 각 행에 필요한 작업을 반복 수행할 수 있는데요. 이제부터 명시적 커서를 사용하는 여러 방식을 알아봅시다.

#### 하나의 행만 조회되는 경우

우선 커서에 지정한 SELECT문이 하나의 행만을 조회하는 때를 살펴보겠습니다. 앞에서 살펴보았던 SELECT INTO 방식과 차이점을 확인해 주세요.

하나의 행만이 조회되는 SELECT문을 커서로 지정하여 사용할 경우 SELECT INTO문을 사용할 때보다 복잡한 여러 단계를 작성해야 하므로 다소 번거로워 보입니다. 커서의 효용성은 조회되는 행이 여러 개일 때 극대화됩니다.

#### 실습 18-2 단일행 데이터를 저장하는 커서 사용하기

• 완성 파일 18-2.sql

```
01  DECLARE
02      -- 커서 데이터를 입력할 변수 선언
03      V_DEPT_ROW DEPT%ROWTYPE;
04
05      -- 명시적 커서 선언(Declaration)
06  CURSOR c1 IS
07      SELECT DEPTNO, DNAME, LOC
08          FROM DEPT
09      WHERE DEPTNO = 40;
10
11  BEGIN
12      -- 커서 열기(Open)
13      OPEN c1;
14
15      -- 커서로부터 읽어온 데이터 사용(Fetch)
16      FETCH c1 INTO V_DEPT_ROW;
```

```

17
18 DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO);
19 DBMS_OUTPUT.PUT_LINE('DNAME : ' || V_DEPT_ROW.DNAME);
20 DBMS_OUTPUT.PUT_LINE('LOC : ' || V_DEPT_ROW.LOC);
21
22 -- 커서 닫기(Close)
23 CLOSE c1;
24
25 END;
26 /

```

:: 결과 화면

```

DEPTNO : 40
DNAME : OPERATIONS
LOC : BOSTON

```

03행 커서의 DEPT 테이블 조회 데이터를 저장할 변수를 선언합니다.

06~09행 사용할 SELECT문을 지정하여 커서의 이름(c1)을 선언합니다.

13행 c1 커서를 열어 Active Set를 식별합니다.

16행 FETCH INTO 문을 사용하여 Active Set에서 결과 행을 추출하고 INTO절에 명시한 V\_DEPT\_ROW 변수에 대입합니다.

18~20행 V\_DEPT\_ROW 변수에 저장한 데이터를 출력(사용)합니다.

23행 커서에 지정한 SELECT문의 결과 행 처리가 모두 끝나면 커서를 닫습니다. 이후에 필요하다면 커서를 다시 열어서 사용할 수 있습니다.

여러 행이 조회되는 경우 사용하는 LOOP문

커서에 지정한 SELECT문이 여러 행을 결과 값으로 가질 경우에 여러 방식의 LOOP문을 사용할 수 있습니다. 먼저 LOOP문을 사용해 보죠.

### 실습 18-3 여러 행의 데이터를 커서에 저장하여 사용하기(LOOP문 사용)

• 완성 파일 18-3.sql

```

01 DECLARE
02   -- 커서 데이터를 입력할 변수 선언
03   V_DEPT_ROW DEPT%ROWTYPE;
04
05   -- 명시적 커서 선언(Declaration)
06   CURSOR c1 IS
07     SELECT DEPTNO, DNAME, LOC
08       FROM DEPT;
09

```

```

10  BEGIN
11  -- 커서 열기(Open)
12  OPEN c1;
13
14  LOOP
15      -- 커서로부터 읽어온 데이터 사용(Fetch)
16      FETCH c1 INTO V_DEPT_ROW;
17
18      -- 커서의 모든 행을 읽어오기 위해 %NOTFOUND 속성 지정
19      EXIT WHEN c1%NOTFOUND;
20
21  DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO
22                  || ', DNAME : ' || V_DEPT_ROW.DNAME
23                  || ', LOC : ' || V_DEPT_ROW.LOC);
24  END LOOP;
25
26  -- 커서 닫기(Close)
27  CLOSE c1;
28
29  END;
30 /

```

:: 결과 화면

```

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

```

**03행** 커서의 DEPT 테이블 조회 데이터를 저장할 변수를 선언합니다.

**06~08행** 사용할 SELECT문을 지정하여 커서의 이름(c1)을 선언합니다.

**12행** c1 커서를 열어 Active Set를 식별합니다.

**14~24행** FETCH문으로 추출한 한 행씩 LOOP문으로 반복 작업(출력)을 수행합니다.

**19행** 커서의 모든 행을 사용한 후 LOOP문을 빠져나오기 위해 EXIT WHEN c1%NOT FOUND를 지정합니다.

여기에서 %NOTFOUND는 실행된 FETCH문에서 행을 추출했으면 false, 추출하지 않았으면 true를 반환합니다. 즉 FETCH문을 통해 더 이상 추출한 데이터가 없을 경우에 LOOP 반복이 끝납니다. %NOTFOUND 외에도 몇 가지 속성을 사용할 수 있으므로 다음 표를 참고하세요.

속성	설명
커서 이름%NOTFOUND	수행된 FETCH문을 통해 추출된 행이 있으면 false, 없으면 true를 반환합니다.
커서 이름%FOUND	수행된 FETCH문을 통해 추출된 행이 있으면 true, 없으면 false를 반환합니다.
커서 이름%ROWCOUNT	현재까지 추출된 행 수를 반환합니다.
커서 이름%ISOPEN	커서가 열려(open) 있으면 true, 닫혀(close) 있으면 false를 반환합니다.

### 여러 개의 행이 조회되는 경우(FOR LOOP문)

LOOP문을 사용하여 커서를 처리하는 방식은 커서 속성을 사용하여 반복 수행을 제어해야 합니다. 커서에 FOR LOOP문을 사용하면 좀 더 간편하게 여러 행을 다룰 수 있습니다. 커서에서 FOR LOOP문은 다음과 같이 사용합니다.

```
FOR 루프 인덱스 이름 IN 커서 이름 LOOP
    결과 행별로 반복 수행할 작업;
END LOOP;
```

기본 형식

루프 인덱스(loop index)는 커서에 저장된 각 행이 저장되는 변수를 뜻하며 ‘.’을 통해 행의 각 필드에 접근할 수 있습니다. 예를 들어 커서에 저장할 SELECT문에 DEPTNO 열이 존재하고 이 커서를 사용하는 루프 인덱스 이름이 c1\_rec일 경우, c1\_rec.DEPTNO는 SELECT문을 통해 조회된 데이터의 각 행에 해당하는 DEPTNO 열의 데이터를 가리키게 됩니다.

커서에 FOR LOOP문을 사용하면 OPEN, FETCH, CLOSE문을 작성하지 않습니다. FOR LOOP를 통해 각 명령어를 자동으로 수행하므로 커서 사용 방법이 간단하다는 장점이 있습니다.

#### 실습 18-4 FOR LOOP문을 활용하여 커서 사용하기

• 완성 파일 18-4.sql

```
01  DECLARE
02      -- 명시적 커서 선언(Declaration)
03      CURSOR c1 IS
04          SELECT DEPTNO, DNAME, LOC
05              FROM DEPT;
06
07      BEGIN
08          -- 커서 FOR LOOP 시작 (자동 Open, Fetch, Close)
09          FOR c1_rec IN c1 LOOP
10              DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || c1_rec.DEPTNO
```

```
11          || ', DNAME : ' || c1_rec.DNAME  
12          || ', LOC : ' || c1_rec.LOC);  
13      END LOOP;  
14  
15  END;  
16 /
```

#### :: 결과 화면

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

03~05행 사용할 SELECT문을 지정하여 커서의 이름(c1)을 선언합니다.

09~13행 커서에 FOR LOOP문을 통해 반복 작업(출력)을 수행합니다. 자동으로 OPEN, FETCH, CLOSE가 수행되므로 명시하지 않습니다.

커서 각 행을 c1\_rec 루프 인덱스에 저장하므로 결과 행을 저장하는 변수 선언도 필요하지 않습니다.

#### 커서에 파라미터 사용하기

지금까지 살펴본 커서에 지정한 SQL문은 작성한 그대로 사용합니다. 하지만 고정 값이 아닌 직접 입력한 값 또는 상황에 따라 여러 값을 번갈아 사용하려면 다음과 같이 커서에 파라미터를 지정할 수 있습니다.

```
CURSOR 커서 이름(파라미터 이름 자료형, ...) IS  
SELECT ...
```

기본 형식

만약 DEPT 테이블의 부서 번호가 10번 또는 20번일 때 다른 수행을 하고 싶다면 다음과 같이 커서의 OPEN을 각각 명시하여 실행합니다.

#### 실습 18-5 파라미터를 사용하는 커서 알아보기

• 완성 파일 18-5.sql

```
01  DECLARE  
02      -- 커서 데이터를 입력할 변수 선언  
03      V_DEPT_ROW DEPT%ROWTYPE;  
04      -- 명시적 커서 선언(Declaration)
```

```

05  CURSOR c1 (p_deptno DEPT.DEPTNO%TYPE) IS
06      SELECT DEPTNO, DNAME, LOC
07      FROM DEPT
08      WHERE DEPTNO = p_deptno;
09  BEGIN
10      -- 10번 부서 처리를 위해 커서 사용
11      OPEN c1 (10);
12      LOOP
13          FETCH c1 INTO V_DEPT_ROW;
14          EXIT WHEN c1%NOTFOUND;
15          DBMS_OUTPUT.PUT_LINE('10번 부서 - DEPTNO : ' || V_DEPT_ROW.DEPTNO
16                               || ', DNAME : ' || V_DEPT_ROW.DNAME
17                               || ', LOC : ' || V_DEPT_ROW.LOC);
18      END LOOP;
19      CLOSE c1;
20      -- 20번 부서 처리를 위해 커서 사용
21      OPEN c1 (20);
22      LOOP
23          FETCH c1 INTO V_DEPT_ROW;
24          EXIT WHEN c1%NOTFOUND;
25          DBMS_OUTPUT.PUT_LINE('20번 부서 - DEPTNO : ' || V_DEPT_ROW.DEPTNO
26                               || ', DNAME : ' || V_DEPT_ROW.DNAME
27                               || ', LOC : ' || V_DEPT_ROW.LOC);
28      END LOOP;
29      CLOSE c1;
30  END;
31  /

```

:: 결과 화면

10번 부서 - DEPTNO : 10, DNAME : ACCOUNTING, LOC : NEW YORK

20번 부서 - DEPTNO : 20, DNAME : RESEARCH, LOC : DALLAS

05행 커서에서 사용할 파라미터(p\_deptno)를 선언합니다.

08행 선언한 파라미터 p\_deptno는 커서에 지정한 SELECT문의 WHERE절에서 DEPTNO열의 비교 값으로 사용됩니다.

11행 c1 커서에 파라미터가 지정되었으므로 OPEN할 때 c1(10)과 같이 자료형을 맞추어 파라미터에 값을 지정합니다. c1(10)은 10이 p\_deptno 변수에 저장되므로 실행되는 SELECT문은 WHERE 절 조건식에서 WHERE DEPTNO = 10으로 대입됩니다.

21행 c1 커서에 20을 대입하여 실행합니다. 이 경우에 SELECT문을 실행하여 20번 부서 데이터를 조회합니다.

만약 커서 실행에 필요한 파라미터 값을 사용자에게 직접 입력받고 싶다면 & 기호와 치환 변수를 사용할 수 있습니다.

### 실습 18-6 커서에 사용할 파라미터 입력받기

· 완성 파일 18-6.sql

```
01  DECLARE
02      -- 사용자가 입력한 부서 번호를 저장하는 변수선언
03      v_deptno DEPT.DEPTNO%TYPE;
04      -- 명시적 커서 선언(Declaration)
05      CURSOR c1 (p_deptno DEPT.DEPTNO%TYPE) IS
06          SELECT DEPTNO, DNAME, LOC
07              FROM DEPT
08             WHERE DEPTNO = p_deptno;
09  BEGIN
10      -- INPUT_DEPTNO에 부서 번호 입력받고 v_deptno에 대입
11      v_deptno := &INPUT_DEPTNO;
12      -- 커서 FOR LOOP 시작. c1 커서에 v_deptno를 대입
13      FOR c1_rec IN c1(v_deptno) LOOP
14          DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || c1_rec.DEPTNO
15                                || ', DNAME : ' || c1_rec.DNAME
16                                || ', LOC : ' || c1_rec.LOC);
17      END LOOP;
18  END;
19 /
```

:: 결과 화면

```
input_deptno의 값을 입력하십시오: 10
구 11: v_deptno := &INPUT_DEPTNO;
신 11: v_deptno := 10;
DEPTNO : 10, DNAME : ACCOUNTING, LOC : NEW YORK
```

03행 사용자가 직접 입력한 부서 번호를 저장하는 v\_deptno 변수를 선언합니다.

05행 커서에 사용할 파라미터(p\_deptno)를 선언합니다.

08행 선언한 파라미터 p\_deptno는 커서에 지정된 SELECT문의 WHERE절에서 DEPTNO 열의 비교 값으로 사용됩니다.

11행 &INPUT\_DEPTNO를 사용하면 실행할 때 사용자에게 INPUT\_DEPTNO에 들어갈 값의 입력을 요구할 수 있습니다. 사용자가 입력한 값을 그대로 v\_deptno 변수에 저장합니다.

13행 c1 커서에 사용자가 입력한 부서 번호 값이 저장된 v\_deptno 변수를 대입하여 실행합니다. WHEREEE절에서 이 값을 사용합니다.

이 책에서는 따로 설명하고 있지 않지만 명시적 커서는 앞에서 살펴본 방식 외에도 커서를 사용할 때 커서 안의 행이 다른 세션에 의해 변경되지 않도록 잠금(lock) 기능을 제공하는 FOR UPDATE절, 커서를 통해 추출한 행에 DML명령어를 사용하는 WHERE CURRENT OF절 등 다양한 사용 방식이 있습니다.

☞ 명시적 커서의 좀 더 자세한 사용법은 오라클 공식 문서([docs.oracle.com/cd/E18283\\_01/appdev.112/e17126/explicit\\_cursor.htm](https://docs.oracle.com/cd/E18283_01/appdev.112/e17126/explicit_cursor.htm))를 참고하세요.

## 묵시적 커서

묵시적 커서는 별다른 선언 없이 SQL문을 사용했을 때 오라클에서 자동으로 선언되는 커서를 뜻합니다. 따라서 사용자가 OPEN, FETCH, CLOSE를 지정하지 않습니다. PL/SQL문 내부에서 DML명령어나 SELECT INTO문 등이 실행될 때 자동으로 생성 및 처리됩니다.

☞ 여러 행의 결과를 가지는 커서는 명시적 커서로만 사용 가능합니다.

자동으로 생성되어 실행되는 묵시적 커서는 별다른 PL/SQL문을 작성하지 않아도 되지만, 다음 묵시적 커서의 속성을 사용하면 현재 커서의 정보를 확인할 수 있습니다. 커서가 자동으로 생성되므로 커서 이름을 지정하지 않고 SQL 키워드로 속성을 지정하며, 명시적 커서의 속성과 유사한 기능을 갖습니다.

속성	설명
SQL%NOTFOUND	묵시적 커서 안에 추출한 행이 있으면 false, 없으면 true를 반환합니다. DML 명령어로 영향을 받는 행이 없을 경우에도 true를 반환합니다.
SQL%FOUND	묵시적 커서 안에 추출한 행이 있으면 true, 없으면 false를 반환합니다. DML 명령어로 영향을 받는 행이 있다면 true를 반환합니다.
SQL%ROWCOUNT	묵시적 커서에 현재까지 추출한 행 수 또는 DML 명령어로 영향받는 행 수를 반환합니다.
SQL%ISOPEN	묵시적 커서는 자동으로 SQL문을 실행한 후 CLOSE되므로 이 속성은 항상 false를 반환합니다.

다음 예제는 묵시적 커서의 각 속성을 사용하여 현재 커서 상태를 확인합니다.

### 실습 18-7 묵시적 커서의 속성 사용하기

• 완성 파일 18-7.sql

```
01 BEGIN
02   UPDATE DEPT SET DNAME='DATABASE'
03   WHERE DEPTNO = 50;
04
05   DBMS_OUTPUT.PUT_LINE('갱신된 행의 수 : ' || SQL%ROWCOUNT);
06
```

```

07  IF (SQL%FOUND) THEN
08      DBMS_OUTPUT.PUT_LINE('갱신 대상 행 존재 여부 : true');
09  ELSE
10      DBMS_OUTPUT.PUT_LINE('갱신 대상 행 존재 여부 : false');
11  END IF;
12
13  IF (SQL%ISOPEN) THEN
14      DBMS_OUTPUT.PUT_LINE('커서의 OPEN 여부 : true');
15  ELSE
16      DBMS_OUTPUT.PUT_LINE('커서의 OPEN 여부 : false');
17  END IF;
18
19 END;
20 /

```

:: 결과 화면

```

갱신된 행의 수 : 0
갱신 대상 행 존재 여부 : false
커서의 OPEN 여부 : false

```

**02~03행** DML을 실행하므로묵시적 커서가 자동으로 생성된 후 실행됩니다. 50번 부서는 존재하지 않으므로 실제로 갱신되는 데이터는 없습니다.

**05행** SQL%ROWCOUNT 속성을 사용하여 DML 대상이 되는 행 수를 반환합니다. 이번 예제는 50번 부서가 없으므로 대상 행은 0이 됩니다.

**07~11행** 조건문에 SQL%FOUND 속성을 대입하여 현재 갱신 대상이 존재하는지 검사합니다. 이 실습에서는 갱신 대상 행이 없으므로 false가 됩니다.

**13~17행** 조건문에 SQL%ISOPEN 속성을 대입하여 생성된 커서가 OPEN되어 있는지 여부를 검사합니다. 묵시적 커서는 자동으로 실행된 후 CLOSE되므로 결과는 false가 됩니다.

**1분  
복습**

다음 빈칸을 채우며 복습해 보세요.

오라클에서는 SELECT문과 DML 명령어의 실행 정보와 결과를 사용하기 위해 <sup>1</sup> 커 라는 메모리 공간을 사용할 수 있습니다.

<sup>1</sup> 커 에는 두 가지 종류가 존재하는데요. 사용자가 직접 선언하여 사용하는

<sup>2</sup> 명 , 그리고 별다른 선언 없이 SQL문을 사용했을 때 오라클에서 자동으로 선언되는 <sup>3</sup> 묵 가 있습니다.

## 18-2 오류가 발생해도 프로그램이 비정상 종료 되지 않도록 하는 예외 처리

### 오류란?

오라클에서 SQL 또는 PL/SQL이 정상 수행되지 못하는 상황을 오류(error)라고 합니다. 이 오류는 크게 두 가지로 구분됩니다. 하나는 문법이 잘못되었거나 오타로 인한 오류로 컴파일 오류(compile Error), 문법 오류(syntax error)라고 합니다. 또 하나는 명령문의 실행 중 발생한 오류가 있습니다. 이 오류를 런타임 오류(runtime error) 또는 실행 오류(execute error)라고 부릅니다. 오라클에서는 이 두 가지 오류 중 후자, 즉 프로그램이 실행되는 도중 발생하는 오류를 예외(exception)라고 합니다. 먼저 예외가 발생하는 다음 PL/SQL문을 실행해 보죠.

#### 실습 18-8 예외가 발생하는 PL/SQL

• 완성 파일 18-8.sql

```
01  DECLARE
02      v_wrong NUMBER;
03  BEGIN
04      SELECT DNAME INTO v_wrong
05          FROM DEPT
06          WHERE DEPTNO = 10;
07  END;
08  /  
  
:: 결과 화면  
DECLARE
*  
1행에 오류 :  
ORA-06502 : PL/SQL : 수치 또는 값 오류 : 문자를 숫자로 변환하는데 오류입니다  
ORA-06512 : 4행
```

02행 숫자 데이터를 저장하는 v\_wrong 변수를 선언합니다.

04행 SELECT INTO문을 통해 DEPT 테이블의 DNAME 열 값을 v\_wrong 변수에 대입합니다. 하지만 DNAME 열은 문자열(VARCHAR2) 데이터이므로, NUMBER 자료형인 v\_wrong 변수에 대입할 수 없고 예외가 발생합니다.

문자열 데이터를 숫자 자료형 변수에 대입하려고 했기 때문에 위 PL/SQL문은 예외가 발생하

고 비정상 종료됩니다. 이렇게 PL/SQL 실행 중 예외가 발생했을 때 프로그램이 비정상 종료되는 것을 막기 위해 특정 명령어를 PL/SQ문 안에 작성하는데 이를 ‘예외 처리’라고 합니다. 예외 처리는 PL/SQ문 안에서 EXCEPTION 영역에 필요 코드를 작성하는 것을 뜻합니다. 다음 실습은 실습 18-8에 예외 처리 코드를 추가한 예제입니다.

#### 실습 18-9 예외를 처리하는 PL/SQL(예외 처리 추가)

• 완성 파일 18-9.sql

```
01  DECLARE
02      v_wrong NUMBER;
03  BEGIN
04      SELECT DNAME INTO v_wrong
05          FROM DEPT
06      WHERE DEPTNO = 10;
07  EXCEPTION
08      WHEN VALUE_ERROR THEN
09          DBMS_OUTPUT.PUT_LINE('예외 처리 : 수치 또는 값 오류 발생');
10  END;
11  /
```

:: 결과 확인

예외 처리 : 수치 또는 값 오류 발생

07행 예외 처리부를 작성하였기 때문에 PL/SQL문을 실행할 때 오류가 발생하여도 프로그램이 비정상 종료되지 않습니다.

결과를 살펴보면 예외가 발생하였지만 PL/SQL문은 정상 처리되었음을 확인할 수 있습니다. 이렇게 EXCEPTION 키워드 뒤에 예외 처리를 위해 작성한 코드 부분을 예외 처리부 또는 예외 처리절이라고 합니다. 이 예외 처리부가 실행되면 예외가 발생한 코드 이후의 내용은 실행이 되지 않는다는 것을 기억하세요.

#### 실습 18-10 예외 발생 후의 코드 실행 여부 확인하기

• 완성 파일 18-10.sql

```
01  DECLARE
02      v_wrong NUMBER;
03  BEGIN
04      SELECT DNAME INTO v_wrong
05          FROM DEPT
06      WHERE DEPTNO = 10;
07
08      DBMS_OUTPUT.PUT_LINE('예외가 발생하면 다음 문장은 실행되지 않습니다');
09  /
```

```

10  EXCEPTION
11    WHEN VALUE_ERROR THEN
12      DBMS_OUTPUT.PUT_LINE('예외 처리 : 수치 또는 값 오류 발생');
13  END;
14 /

```

:: 결과 화면

예외 처리 : 수치 또는 값 오류 발생

**08행** 발생한 예외를 EXCEPTION 작성을 통해 처리해도 이미 예외가 발생한 코드 이후의 내용은 실행되지 않습니다.

## 예외 종류

오라클에서 예외는 크게 내부 예외(internal exceptions)와 사용자 정의 예외(user-defined exceptions)로 나뉩니다. 내부 예외는 오라클에서 미리 정의한 예외를 뜻하며 사용자 정의 예외는 사용자가 필요에 따라 추가로 정의한 예외를 의미합니다. 내부 예외는 이름이 정의되어 있는 예외인 사전 정의된 예외(predefined name exceptions)와 이름이 정해지지 않은 예외로 다시 나뉩니다.

예외 종류		설명
내부 예외 (internal exceptions)	사전 정의된 예외 (predefined name exceptions)	내부 예외 중 예외 번호에 해당하는 이름이 존재하는 예외
	이름이 없는 예외 (unnamed exceptions)	내부 예외 중 이름이 존재하지 않는 예외(사용자가 필요에 따라 이름을 지정할 수 있음)
사용자 정의 예외(user-defined exceptions)		사용자가 필요에 따라 직접 정의한 예외

사전 정의된 예외는 비교적 자주 발생하는 예외에 이름을 붙여 놓은 것입니다. 다음은 그 예입니다.

예외 이름	예외 번호(SQLCODE)	설명
ACCESS_INTO_NULL	ORA-06530 : -6530	초기화되지 않은 객체 속성 값 할당
CASE_NOT_FOUND	ORA-06592 : -6592	CASE문의 WHERE절에 조건이 없고 ELSE절도 없을 경우
COLLECTION_IS_NULL	ORA-06531 : -6531	초기화되지 않은 중첩 테이블, VARRAY에 EXIT 외 컬렉션 메서드를 사용하려 할 경우 또는 초기화되지 않은 중첩 테이블이나 VARRAY에 값을 대입하려 할 경우

CURSOR_ALREADY_OPEN	ORA-06511 : -6511	이미 OPEN된 커서를 OPEN 시도할 경우
DUP_VAL_ON_INDEX	ORA-00001 : -1	UNIQUE 인덱스가 있는 열에 중복된 값을 저장하려고 했을 경우
INVALID_CURSOR	ORA-01001 : -1001	OPEN되지 않은 커서를 CLOSE 시도하는 것과 같이 잘 못된 커서 작업을 시도하는 경우
INVALID_NUMBER	ORA-01722 : -1722	문자에서 숫자로의 변환이 실패했을 경우
LOGIN_DENIED	ORA-01017 : -1017	사용자 이름이나 패스워드가 올바르지 않은 상태에서 로그인을 시도할 경우
NO_DATA_FOUND	ORA-01403 : +100	SELECT INTO문에서 결과 행이 하나도 없을 경우
NOT_LOGGED_ON	ORA-01012 : -1012	데이터베이스에 접속되어 있지 않은 경우
PROGRAM_ERROR	ORA-06501 : -6501	PL/SQL 내부 오류가 발생했을 경우
ROWTYPE_MISMATCH	ORA-06504 : -6504	호스트 커서 변수와 PL/SQL 커서 변수의 자료형이 호환되지 않는 경우
SELF_IS_NULL	ORA-30625 : -30625	초기화되지 않은 오브젝트의 MEMBER 메서드를 호출한 경우
STORAGE_ERROR	ORA-06500 : -6500	PL/SQL 메모리가 부족하거나 문제가 발생한 경우
SUBSCRIPT_BEYOND_COUNT	ORA-06533 : -6533	컬렉션의 요소 수보다 큰 인덱스를 사용하여 중첩 테이블이나 VARARRAY의 요소 참조를 시도할 경우
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532 : -6532	정상 범위외 인덱스 번호를 사용하여 중첩 테이블이나 VARARRAY 요소 참조를 시도할 경우
SYS_INVALID_ROWID	ORA-01410 : -1410	문자열을 ROWID로 변환할 때 값이 적절하지 않은 경우
TIMEOUT_ON_RESOURCE	ORA-00051 : -51	자원 대기 시간을 초과했을 경우
TOO_MANY_ROWS	ORA-01422 : -1422	SELECT INTO문의 결과 행이 여러 개일 경우
VALUE_ERROR	ORA-06502 : -6502	산술·변환·잘림·제약 조건 오류가 발생했을 경우
ZERO_DIVIDE	ORA-01476 : -1476	숫자 데이터를 0으로 나누려고 했을 경우

이와 달리 이름 없는 예외는 ORA-XXXXX식으로 예외 번호는 있지만 이름이 정해져 있지 않은 예외를 뜻합니다. 이름이 없는 예외는 예외 처리부에서 사용하기 위해 이름을 직접 붙여서 사용합니다.

## 예외 처리부 작성

예외 처리부는 앞의 예제에서 보았듯이 EXCEPTION절에 필요한 코드를 사용하여 작성합니다. 다음과 같이 여러 예외를 명시하여 작성할 수 있습니다. WHEN으로 시작하는 절을 예외 핸들러(exception handler)라고 하며, 발생한 예외 이름과 일치하는 WHEN절의 명령어를 수행합니다(IF THEN문처럼 여러 예외 핸들러 중 일치하는 하나의 예외 핸들러 명령어만 수행합니다). 수행 할 명령어는 PL/SQL 실행부와 마찬가지로 여러 문법을 사용할 수 있습니다. OTHERS는 먼저 작성한 어느 예외와도 일치하는 예외가 없을 경우에 처리할 내용을 작성합니다(IF 조건문의 ELSE와 비슷하죠).

```
EXCEPTION
  WHEN 예외 이름1 [OR 예외 이름2 - ] THEN
    예외 처리에 사용할 명령어;
  WHEN 예외 이름3 [OR 예외 이름4 - ] THEN
    예외 처리에 사용할 명령어;
  ...
  WHEN OTHERS THEN
    예외 처리에 사용할 명령어;
```

기본 형식

### 사전 정의된 예외 사용

예외 핸들러에 사전 정의된 예외만을 사용할 때는 앞에서 살펴본 작성 방식대로 발생할 수 있는 예외를 명시합니다.

실습 18-11 사전 정의된 예외 사용하기

· 완성 파일 18-11.sql

```
01  DECLARE
02      v_wrong NUMBER;
03  BEGIN
04      SELECT DNAME INTO v_wrong
05          FROM DEPT
06         WHERE DEPTNO = 10;
07
08      DBMS_OUTPUT.PUT_LINE('예외가 발생하면 다음 문장은 실행되지 않습니다');
09
10  EXCEPTION
11      WHEN TOO_MANY_ROWS THEN
12          DBMS_OUTPUT.PUT_LINE('예외 처리 : 요구보다 많은 행 추출 오류 발생');
13      WHEN VALUE_ERROR THEN
14          DBMS_OUTPUT.PUT_LINE('예외 처리 : 수치 또는 값 오류 발생');
```

```

15 WHEN OTHERS THEN
16   DBMS_OUTPUT.PUT_LINE('예외 처리 : 사전 정의 외 오류 발생');
17 END;
18 /

```

#### :: 결과 확인

예외 처리 : 수치 또는 값 오류 발생

### 이름 없는 예외 사용

만약 이름이 없는 내부 예외를 사용해야 한다면 이름을 직접 지정해 주어야 예외 처리부에서 사용할 수 있습니다. 이름을 직접 지어 줄 때 오른쪽과 같이 선언부에서 오라클 예외 번호와 함께 이름을 붙입니다. 이름이 정해진 예외는 사전 정의된 예외를 사용할 때와 마찬가지로 예외 처리부에서 지정한 이름으로 예외 핸들러에 작성합니다.

```

DECLARE
  예외 이름1 EXCEPTION;
  PRAGMA EXCEPTION_INIT(예외 이름1, 예외 번호);
  :
EXCEPTION
  WHEN 예외 이름1 THEN
    예외 처리에 사용할 명령어;
  ...
END;

```

기본 형식

### 사용자 정의 예외 사용

사용자 정의 예외는 오라클에 정의되어 있지 않은 특정 상황을 직접 오류로 정의하는 방식입니다. 오른쪽과 같이 예외 이름을 정해 주고 실행부에서 직접 정의한 오류 상황이 생겼을 때 RAISE 키워드를 사용하여 예외를 직접 만들 수 있습니다. 이렇게 직접 만든 예외 역시 앞의 예외 처리와 마찬가지로 예외 처리부에서 예외 이름을 통해 수행할 내용을 작성해 줌으로써 처리합니다.

```

DECLARE
  사용자 예외 이름 EXCEPTION;
  :
BEGIN
  IF 사용자 예외를 발생시킬 조건 THEN
    RAISE 사용자 예외 이름
  ...
END IF;
EXCEPTION
  WHEN 사용자 예외 이름 THEN
    예외 처리에 사용할 명령어;
  ...
END;

```

기본 형식

## 오류 코드와 오류 메시지 사용

오류 처리부가 잘 작성되어 있다면 오류가 발생해도 PL/SQL은 정상 종료됩니다. PL/SQL문의 정상 종료 여부와 상관없이 발생한 오류 내역을 알고 싶을 때 SQLCODE, SQLERRM 함수를 사용합니다.

함수	설명
SQLCODE	오류 번호를 반환하는 함수
SQLERRM	오류 메시지를 반환하는 함수

SQLCODE와 SQLERRM은 PL/SQL에서만 사용 가능한 함수로 SQL문에서는 사용할 수 없다는 것도 기억하세요.

### 실습 18-12 오류 코드와 오류 메시지 사용하기

• 완성 파일 18-12.sql

```
01  DECLARE
02      v_wrong NUMBER;
03  BEGIN
04      SELECT DNAME INTO v_wrong
05          FROM DEPT
06         WHERE DEPTNO = 10;
07
08      DBMS_OUTPUT.PUT_LINE('예외가 발생하면 다음 문장은 실행되지 않습니다');
09
10  EXCEPTION
11      WHEN OTHERS THEN
12          DBMS_OUTPUT.PUT_LINE('예외 처리 : 사전 정의 외 오류 발생');
13          DBMS_OUTPUT.PUT_LINE('SQLCODE : ' || TO_CHAR(SQLCODE));
14          DBMS_OUTPUT.PUT_LINE('SQLERRM : ' || SQLERRM);
15  END;
16  /
```

#### :: 결과 확인

예외 처리 : 사전 정의 외 오류 발생

SQLCODE : -6502

SQLERRM : ORA-06502: PL/SQL: 수치 또는 값 오류 : 문자를 숫자로 변환하는데

오류입니다

**Q1** 명시적 커서를 사용하여 EMP 테이블의 전체 데이터를 조회한 후 커서 안의 데이터가 다음과 같아 출력되도록 PL/SQL문을 작성해 보세요.

- ① LOOP를 사용한 방식
- ② FOR LOOP를 사용한 방식

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
... [변수 선언 및 값 대입]
BEGIN
... [PL/SQL 작성]
END;
/
EMPNO : 7369, ENAME : SMITH, JOB : CLERK, SAL : 800, DEPTNO : 20
EMPNO : 7499, ENAME : ALLEN, JOB : SALESMAN, SAL : 1600, DEPTNO : 30
EMPNO : 7521, ENAME : WARD, JOB : SALESMAN, SAL : 1250, DEPTNO : 30
EMPNO : 7566, ENAME : JONES, JOB : MANAGER, SAL : 2975, DEPTNO : 20
EMPNO : 7654, ENAME : MARTIN, JOB : SALESMAN, SAL : 1250, DEPTNO : 30
EMPNO : 7698, ENAME : BLAKE, JOB : MANAGER, SAL : 2850, DEPTNO : 30
EMPNO : 7782, ENAME : CLARK, JOB : MANAGER, SAL : 2450, DEPTNO : 10
EMPNO : 7788, ENAME : SCOTT, JOB : ANALYST, SAL : 3000, DEPTNO : 20
EMPNO : 7839, ENAME : KING, JOB : PRESIDENT, SAL : 5000, DEPTNO : 10
EMPNO : 7844, ENAME : TURNER, JOB : SALESMAN, SAL : 1500, DEPTNO : 30
EMPNO : 7876, ENAME : ADAMS, JOB : CLERK, SAL : 1100, DEPTNO : 20
EMPNO : 7900, ENAME : JAMES, JOB : CLERK, SAL : 950, DEPTNO : 30
EMPNO : 7902, ENAME : FORD, JOB : ANALYST, SAL : 3000, DEPTNO : 20
EMPNO : 7934, ENAME : MILLER, JOB : CLERK, SAL : 1300, DEPTNO : 10
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

● 이 장에서 배운 내용을 실습하며 정리하세요.

**Q2** 다음 PL/SQL문의 실행 중 발생하는 예외를 다음 결과와 같이 처리하는 예외 처리부를 완성하세요.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
      v_wrong DATE;
    BEGIN
      SELECT ENAME INTO v_wrong
      FROM EMP
      WHERE EMPNO = 7369;

      DBMS_OUTPUT.PUT_LINE('예외가 발생하면 다음 문장은 실행되지 않습니다');

      EXCEPTION
        ... [예외 처리부 작성]
      END;
    /
오류가 발생하였습니다.[2018년03월26일 21시03분55초]
SQLCODE : -1841
SQLERRM : ORA-01841 : 년은 영이 아닌 -4713과 +4713 사이 값으로 지정해야 합니다.
```

PL/SQL 처리가 정상적으로 완료되었습니다.

SQL>

정답 이지스파블리싱 홈페이지에서 확인하세요.