ch18. 커서와 예외 처리

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 공간

- 커서를 사용하면 실행된 SQL문의 결과 값을 사용할 수 있다.
  - SELECT문의 결과 값이 여러 행으로 나왔을 때 각 행별로 특정 작업을 수행하도록 기능을 구현하는 것이 가능하다.
- 오라클에 저장된 데이터 활용을 극대화할 수 있는 강력한 기능.
- 커서의 사용 방법에 따른 분류
  - 명시적 *explicit* 커서
  - 묵시적 *inplicit* 커서

SELECT INTO 방식

SELECT INTO문은 조회되는 데이터가 단 하나의 행일 때 사용 가능한 방식이다.

커서는 결과 행이 하나이든 여러 개이든 상관 없이 사용할 수 있다.

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

- SELECT INTO 를 사용한 단일행 데이터 저장하기

```
DECLARE
   V_DEPT_ROW DEPT%ROWTYPE;
BEGIN
   SELECT DEPTNO, DNAME, LOC INTO V_DEPT_ROW
     FROM DEPT
    WHERE DEPTNO = 40;
   DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO);
   DBMS_OUTPUT.PUT_LINE('DNAME : ' || V_DEPT_ROW.DNAME);
```

```
   DBMS_OUTPUT.PUT_LINE('LOC : ' || V_DEPT_ROW.LOC);
END;
/
```

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

명시적 커서

명시적 커서는 사용자가 직접 커서를 선언하고 사용하는 커서를 뜻한다.

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

- 명시적 커서 사용방법

```
DECLARE
  CURSOR 커서 이름 IS SQL문; -- 커서 선언
BEGIN
  OPEN 커서 이름;             -- 커서 열기
  FETCH 커서 이름 INTO 변수  -- 커서로부터 읽어온 데이터 사용
  CLOSE 커서 이름;            -- 커서 닫기
```

하나의 행만 조회되는 경우

커서의 효용성은 조회되는 행이 여러 개일 때 극대화된다.

- 단일행 데이터를 저장하는 커서 사용하기

```
DECLARE
   -- 커서 데이터를 입력할 변수 선언
   V_DEPT_ROW DEPT%ROWTYPE;

   -- 명시적 커서 선언(Declaration)
   CURSOR c1 IS
      SELECT DEPTNO, DNAME, LOC
        FROM DEPT
       WHERE DEPTNO = 40;

BEGIN
   -- 커서 열기(Open)
   OPEN c1;

   -- 커서로부터 읽어온 데이터 사용(Fetch)
   FETCH c1 INTO V_DEPT_ROW;

   DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO);
   DBMS_OUTPUT.PUT_LINE('DNAME : ' || V_DEPT_ROW.DNAME);
   DBMS_OUTPUT.PUT_LINE('LOC : ' || V_DEPT_ROW.LOC);

   -- 커서 닫기(Close)
   CLOSE c1;

END;
/
```

여러 행이 조회되는 경우 사용하는 LOOP문

```
DECLARE
   -- 커서 데이터를 입력할 변수 선언
   V_DEPT_ROW DEPT%ROWTYPE;

   -- 명시적 커서 선언(Declaration)
   CURSOR c1 IS
      SELECT DEPTNO, DNAME, LOC
        FROM DEPT;

BEGIN
   -- 커서 열기(Open)
   OPEN c1;

   LOOP
      -- 커서로부터 읽어온 데이터 사용(Fetch)
      FETCH c1 INTO V_DEPT_ROW;

      -- 커서의 모든 행을 읽어오기 위해 %NOTFOUND 속성 지정
      EXIT WHEN c1%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO
                           || ', DNAME : ' || V_DEPT_ROW.DNAME
                           || ', LOC : ' || V_DEPT_ROW.LOC);
```

```
   END LOOP;

   -- 커서 닫기(Close)
   CLOSE c1;

END;
/
```

여러 개의 행이 조회되는 경우 (FOR LOOP문)

- LOOP문을 사용하여 커서를 처리하는 방식은 커서 속성을 사용하여 반복 수행을 제어해야 한다.

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

- FOR LOOP문을 활용하여 커서 사용하기

```
DECLARE
   -- 명시적 커서 선언(Declaration)
   CURSOR c1 IS
   SELECT DEPTNO, DNAME, LOC
     FROM DEPT;

BEGIN
   -- 커서 FOR LOOP 시작 (자동 Open, Fetch, Close)
   FOR c1_rec IN c1 LOOP
      DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || c1_rec.DEPTNO
                        || ', DNAME : ' || c1_rec.DNAME
                        || ', LOC : ' || c1_rec.LOC);
   END LOOP;

END;
/

SELECT * FROM DEPT_RECORD;
```

커서에 파라미터 사용하기

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

- 파라미터를 사용하는 커서 알아보기

```
DECLARE
   -- 커서 데이터를 입력할 변수 선언
   V_DEPT_ROW DEPT%ROWTYPE;
   -- 명시적 커서 선언(Declaration)
   CURSOR c1 (p_deptno DEPT.DEPTNO%TYPE) IS
      SELECT DEPTNO, DNAME, LOC
        FROM DEPT
       WHERE DEPTNO = p_deptno;
BEGIN
   -- 10번 부서 처리를 위해 커서 사용
   OPEN c1 (10);
      LOOP
         FETCH c1 INTO V_DEPT_ROW;
         EXIT WHEN c1%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('10번 부서 - DEPTNO : ' || V_DEPT_ROW.DEPTNO
                                    || ', DNAME : ' || V_DEPT_ROW.DNAME
                                    || ', LOC : ' || V_DEPT_ROW.LOC);
      END LOOP;
   CLOSE c1;
   -- 20번 부서 처리를 위해 커서 사용
   OPEN c1 (20);
      LOOP
         FETCH c1 INTO V_DEPT_ROW;
         EXIT WHEN c1%NOTFOUND;
         DBMS_OUTPUT.PUT_LINE('20번 부서 - DEPTNO : ' || V_DEPT_ROW.DEPTNO
                                    || ', DNAME : ' || V_DEPT_ROW.DNAME
                                    || ', LOC : ' || V_DEPT_ROW.LOC);
      END LOOP;
   CLOSE c1;
END;
/
```

- 커서 실행에 필요한 파라미터 값을 사용자에게 직접 입력받기

```
DECLARE
   -- 사용자가 입력한 부서 번호를 저장하는 변수선언
   v_deptno DEPT.DEPTNO%TYPE;
   -- 명시적 커서 선언(Declaration)
   CURSOR c1 (p_deptno DEPT.DEPTNO%TYPE) IS
      SELECT DEPTNO, DNAME, LOC
        FROM DEPT
       WHERE DEPTNO = p_deptno;
```

```
BEGIN
   -- INPUT_DEPTNO에 부서 번호 입력받고 v_deptno에 대입
   v_deptno := &INPUT_DEPTNO;
   -- 커서 FOR LOOP 시작. c1 커서에 v_deptno를 대입
   FOR c1_rec IN c1(v_deptno) LOOP
      DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || c1_rec.DEPTNO
                      || ', DNAME : ' || c1_rec.DNAME
                      || ', LOC : ' || c1_rec.LOC);
   END LOOP;
END;
/
```

묵시적 커서

- 묵시적 커서는 별다른 선언 없이 SQL문을 사용했을 때 오라클에서 자동으로 선언되는 커서
  - 따라서 사용자가 OPEN, FETCH, CLOSE를 지정하지 않는다.
  - PL/SQL문 내부에서 DML명령어나 SELECT INTO문 등이 실행될 때 자동으로 생성 및 처리된다.

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

- 묵시적 커서의 속성 사용하기

```
BEGIN
   UPDATE DEPT SET DNAME='DATABASE'
    WHERE DEPTNO = 50;

   DBMS_OUTPUT.PUT_LINE('갱신된 행의 수 : ' || SQL%ROWCOUNT);

   IF (SQL%FOUND) THEN
      DBMS_OUTPUT.PUT_LINE('갱신 대상 행 존재 여부 : true');
```

```
   ELSE
      DBMS_OUTPUT.PUT_LINE('갱신 대상 행 존재 여부 : false');
   END IF;

   IF (SQL%ISOPEN) THEN
      DBMS_OUTPUT.PUT_LINE('커서의 OPEN 여부 : true');
   ELSE
      DBMS_OUTPUT.PUT_LINE('커서의 OPEN 여부 : false');
   END IF;

END;
/
```

18-2. 오류가 발생해도 프로그램이 비정상 종료되지 않도록 하는 예외처리

> 오류란?

- 오라클에서 SQL또는 PL/SQL이 정상 수행되지 못하는 상황을 오류 *error* 라고 한다.
- 오류의 두가지 분류
  - 컴파일 오류 *compile error* / 문법 오류 *syntax error* : 문법이 잘못됬거나 오타로 인한 오류
  - 런타임 오류 *runtime error* / 실행 오류 *execute error* : 명령문의 실행 중 발생한 오류
- 오라클에서는 런타임 오류 또는 실행 오류를 예외라고 부른다.

- 예외가 발생하는 PL/SQL

```
DECLARE
   v_wrong NUMBER;
BEGIN
   SELECT DNAME INTO v_wrong
     FROM DEPT
    WHERE DEPTNO = 10;
END;
/
```

⇒ 예외 발생으로 인해 비정상 종료 되는 것을 막기 위해 특정 명령어를 작성하는 것이 '예외 처리'

- 예외를 처리하는 PL/SQL (예외 처리 추가)

```
DECLARE
   v_wrong NUMBER;
BEGIN
   SELECT DNAME INTO v_wrong
     FROM DEPT
    WHERE DEPTNO = 10;
EXCEPTION
   WHEN VALUE_ERROR THEN
      DBMS_OUTPUT.PUT_LINE('예외 처리 : 수치 또는 값 오류 발생');
END;
/
```

⇒ EXCEPTION 키워드 뒤에 예외 처리를 위해 작성한 코드 부분을 예외 처리부 또는 예외 처리절 이라 한다.

- 예외 발생 후에 코드 실행 여부 확인하기

```
DECLARE
   v_wrong NUMBER;
BEGIN
   SELECT DNAME INTO v_wrong
     FROM DEPT
    WHERE DEPTNO = 10;

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

EXCEPTION
   WHEN VALUE_ERROR THEN
      DBMS_OUTPUT.PUT_LINE('예외 처리 : 수치 또는 값 오류 발생');
END;
/
```

예외 종류

- 예외의 두가지 분류
  - 내부 예외 internal exception : 오라클에서 미리 정의한 예외

    - 사용자 정의 예외 user-defined exception : 사용자가 필요에 따라 추가로 정의한 예외

- 내부 예외
    - 사전 정의된 예외 : 내부 예외 중 예외 번호에 해당하는 이름이 존재하는 예외
    - 이름이 없는 예외 : 내부 예외 중 이름이 존재하지 않는 예외 (사용자 필요에 따라 이름지정)
- 사용자 정의 예외 : 사용자가 필요에 따라 직접 정의한 예외

예외 처리부 작성

```
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;
```

- WHEN으로 시작하는 절을 예외 핸들러 exception handler 라고 하며, 발생한 예외 이름과 일치하는 WHEN절의 명령어를 수행한다.
- OTHERS는 먼저 작성한 어느 예외와도 일치하는 예외가 없을 경우에 처리할 내용을 작성한다.

사전 정의된 예외 사용

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

```
DECLARE
   v_wrong NUMBER;
BEGIN
   SELECT DNAME INTO v_wrong
     FROM DEPT
    WHERE DEPTNO = 10;

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

```
EXCEPTION
   WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('예외 처리 : 요구보다 많은 행 추출 오류 발생');
   WHEN VALUE_ERROR THEN
      DBMS_OUTPUT.PUT_LINE('예외 처리 : 수치 또는 값 오류 발생');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('예외 처리 : 사전 정의 외 오류 발생');
END;
/
```

이름 없는 예외 사용

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

사용자 정의 예외 사용

```
DECLARE
  사용자 예외 이름1 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;
```

오류 코드와 오류 메시지 사용

- 오류 처리부가 작 작성되어 있다면 오류가 발생해도 PL/SQL은 정상 종료된다.

- PL/SQL문의 정상 종료 여부와 상관없이 발생한 오류 내역을 알고 싶을 때 SQLCODE, SQLERRM 함수를 사용한다. (SQL문에서는 사용할 수 없음)

- 오류 코드와 오류 메시지 사용하기

```
DECLARE
   v_wrong NUMBER;
BEGIN
   SELECT DNAME INTO v_wrong
     FROM DEPT
    WHERE DEPTNO = 10;

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

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('예외 처리 : 사전 정의 외 오류 발생');
      DBMS_OUTPUT.PUT_LINE('SQLCODE : ' || TO_CHAR(SQLCODE));
      DBMS_OUTPUT.PUT_LINE('SQLERRM : ' || SQLERRM);
END;
/
```

Q

1. LOOP를 사용한 방식

```
DECLARE
   V_EMP_ROW EMP%ROWTYPE;
   CURSOR c1 IS
      SELECT *
        FROM EMP;
BEGIN
   OPEN c1;
   LOOP
       FETCH c1 INTO V_EMP_ROW;
       EXIT WHEN c1%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE('EMPNO : '    || V_EMP_ROW.EMPNO
                       || ', ENAME : '    || V_EMP_ROW.ENAME
                       || ', JOB : '      || V_EMP_ROW.JOB
                       || ', SAL : '      || V_EMP_ROW.SAL
                       || ', DEPTNO : '   || V_EMP_ROW.DEPTNO
           );
   END LOOP;
   CLOSE c1;
END;
/
```

```
DECLARE
   CURSOR c1 IS
      SELECT *
        FROM EMP;
BEGIN
   FOR c1_rec IN c1 LOOP
      DBMS_OUTPUT.PUT_LINE('EMPNO : '    || c1_rec.EMPNO
                      || ', ENAME : '    || c1_rec.ENAME
                      || ', JOB : '      || c1_rec.JOB
                      || ', SAL : '      || c1_rec.SAL
                      || ', DEPTNO : '   || c1_rec.DEPTNO);
   END LOOP;
END;
/
```

2.

```
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
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('오류가 발생하였습니다.'
                        || TO_CHAR(SYSDATE, '[YYYY"년"MM"월"DD"일" HH24"시"mm"분"S
S"초"]'));

      DBMS_OUTPUT.PUT_LINE('SQLCODE : ' || TO_CHAR(SQLCODE));
      DBMS_OUTPUT.PUT_LINE('SQLERRM : ' || SQLERRM);
END;
/
```