Writing Explicit Cursors

1. 커서(Cursor)

커서: 커서는 메모리 영역이라고 생각하면 됨

1-1. 개요

- Oracle Server는 전용 SQL 영역이라고 하는 작업 영역에서 SQL 문을 실행하여 처리 정보를 저장
- PL/SQL 커서를 사용하여 전용 SQL 영역의 이름을 지정하고 이 영역에 저장된 정보 액세스
- 커서는 모든 처리 단계를 지시

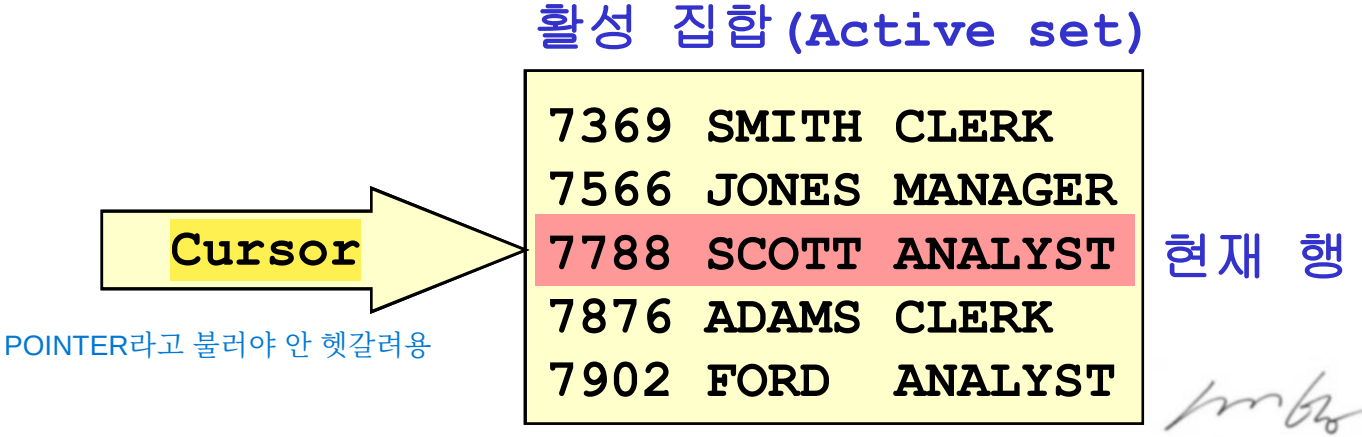
1-2. 커서 유형

커서 유형	설 명
암시적 커서	- 하나의 행만 반환하는 질의를 포함 - 모든 DML 및 PL/SQL SELECT 문에 대해 PL/SQL에서 암시적으로 선언
명시적 커서	- 둘 이상의 행을 반환하는 질의에 대해 선언 - 프로그래머가 선언하여 이름을 지정 - 블록의 실행 가능한 작업에 있는 특정 명령문을 통하여 조작

2. 명시적 커서

2-1. 명시적 커서 개요

- 명시적 커서를 사용하여 다중 행 **SELECT** 문에 의해 반환된 각 행을 개별적으로 처리 가능
- 다중 행 질의에 의해 반환되는 행 집합을 활성 집합(**Active set**)이라 함
- 커서는 활성 집합에서 현재 위치를 표시
- 명시적 커서를 사용하면 인출 작업을 여러 번 수행하고 작업 영역에서 구문이 분석된 질의를 재실행 할 수 있음 (암시적 커서에 대한 인출은 두번째 행이 있으면 **TOO_MANY_ROWS** 예외사항 발생)

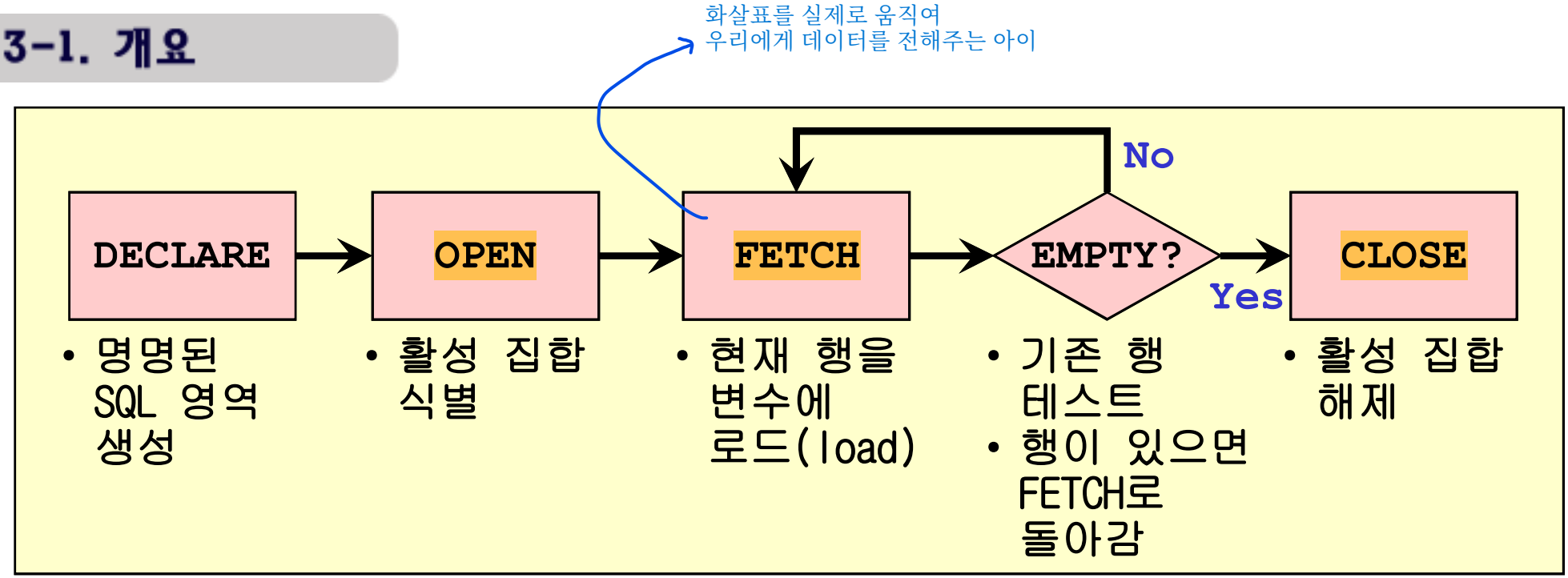


2-2. 명시적 커서 기능

- 질의에 의해 반환된 행 중 첫 번째 행은 건너뛰고 나머지는 행 단위로 차례로 처리할 수 있음
- 현재 처리되고 있는 행 추적
- 프로그래머가 **PL/SQL** 블록에서 수동으로 제어할 수 있음

3. 명시적 커서 제어

3-1. 개요



- 네 개의 명령을 사용하여 명시적 커서 제어
  - 커서 선언
  - 커서 열기
  - 커서에서 데이터 인출
  - 커서 닫기

3-2. 커서 선언

```
CURSOR cursor_name IS
     select_statement;
```

```
DECLARE
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM   emp;
    CURSOR dept_cursor IS
        SELECT  *
        FROM   dept
        WHERE  depnot = 10;
BEGIN
    ...
```

- 커서 선언 부분에서는 INTO 절을 포함하지 않음
- 특정 시퀀스로 행을 처리해야 하는 경우 질의에 ORDER BY 절 사용
- 질의에 있는 변수를 참조할 수 있지만 CURSOR 문 앞에 변수를 선언

3-3. 커서 열기 (실행)

```
OPEN cursor_name;
```

- 커서를 열어 질의를 실행하고 활성 집합(Active set) 식별
- 질의에 의해 반환되는 행이 없으면 예외사항이 발생하지 않음
- 커서 속성을 사용하여 인출 후의 결과를 테스트
- OPEN은 다음 작업을 수행하는 실행문
  - 중요한 처리 정보를 포함하는 문맥(Context) 영역에 대한 메모리를 동적으로 할당
  - SELECT 문의 구문 분석
  - 입력 변수를 바인드 (해당 메모리 주소를 확보하여 입력 변수의 값을 설정)
  - 활성 집합 식별 (OPEN 문 실행시 활성 집합의 행을 변수로 가져오는 것이 아니라 FETCH 문 실행 시 해당 행을 변수로 가져 옴)
  - 포인트(Pointer)의 위치를 활성 집합의 첫 번째 행 앞으로 지정

3-4. 커서에서 데이터 인출 (1)

```
FETCH cursor_name INTO [variable1, variable2, •••]
                       | record_name];
```

```
FETCH emp_cursor  INTO v_empno, v_ename;
```

- 현재 행 값을 변수로 가져 옴
- 동일한 개수의 변수를 포함
- 각 변수를 해당 열에 위치적으로 대응
- 커서의 행 포함 여부를 테스트
- **FETCH** 문은 다음 작업을 수행
  - 활성 집합(**Active set**)에서 포인터를 다음 행으로 이동
  - 현재 행을 데이터를 **PL/SQL**의 출력 변수로 읽어 들임

포인터가 가리키는 행을 INTO절에 넘김

3-5. 커서에서 데이터 인출 (2)

```
DECLARE
    v_empno    emp.empno%TYPE;
    v_ename    emp.ename%TYPE;
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM   emp;
BEGIN
    OPEN emp_cursor;
    FOR i IN 1..10 LOOP
        FETCH emp_cursor INTO v_empno, v_ename;
        ...
    END LOOP;
END;
```

3-6. 커서 닫기

활성집합 깨트림 / 메모리 삭제 (깔끔하게 데이터 날림)
커서별 주어지는 메모리의 영역은 '단하나'

```
CLOSE cursor_name;
```

```
...
   FOR i IN 1..10 LOOP
      FETCH emp_cursor INTO v_empno, v_ename;
      ...
   END LOOP;
   CLOSE emp_cursor;
END;
```

- 행 처리를 완료한 후 커서를 닫음
- 필요한 경우 커서를 다시 열 수 있음
- 닫혀진 커서에서 데이터 인출을 시도할 경우 **INVALID_CURSOR** 예외사항 발생

4. 명시적 커서 속성

4-1. 개요

- 암시적 커서와 마찬가지로 커서의 상태 정보를 제공
- 명시적 커서 속성을 커서 변수 이름에 추가하면 데이터 조작문 실행에 관한 유용한 정보를 반환
- SQL 문에서 커서 속성을 직접 참조할 수 없음

속성	유형	유형
%ISOPEN	부울	커서가 열려 있으면 TRUE
%NOTFOUND	부울	가장 최근의 인출에서 반환하는 행이 없으면 TRUE
%FOUND	부울	가장 최근의 인출에서 반환하는 행이 있으면 TRUE %NOTFOUND의 보수
%ROWCOUNT	숫자	지금까지 반환된 전체 행 수

재오픈하면 ERROR나면서 STOP걸리기 때문

새로운 행이 없으면

FETCH로 가지고 온 총 행의 수 / 사용이유 2가지

4-2. 다중 인출(FETCH) 제어

- 루프를 사용하여 명시적 커서에서 다중 행을 처리
- 반복적으로 행을 인출(**FETCH**)
- **%NOTFOUND** 속성을 사용하여 실행되지 못한 인출에 대한 테스트 작성
- 명시적 커서 속성을 사용하여 각 인출 작업이 성공적으로 수행되었는지 테스트
- 종료 조건을 생략하면 무한 루프 발생

4-3. %ISOPEN 속성

```
IF NOT emp_cursor%ISOPEN THEN
    OPEN emp_cursor;
END IF;
LOOP
    FETCH emp_cursor •••
```

- 커서가 열려 있는 경우에만 행을 인출할 수 있으므로 필요한 경우%ISOPEN 커서 속성을 사용하여 커서가 열려 있는지 확인
- 일반적으로 %ISOPEN 검사를 수행하지 않아도 됨

4-4. %NOTFOUND 및 %ROWCOUNT 속성

```
DECLARE
    v_empno     emp.empno%TYPE;
    v_ename     emp.ename%TYPE;
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM   emp;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_empno, v_ename;
        EXIT WHEN
            emp_cursor%ROWCOUNT > 10 OR emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_empno)||' '|| v_ename);
    END LOOP;
    CLOSE emp_cursor;
END;
```

- %ROWCOUNT 커서 속성을 사용하여 정확한 행 수를 검색
- %NOTFOUND 커서 속성을 사용하여 루프 종료 시기 결정