7. 서브쿼리

7.1 서브 쿼리의 정의

서브 쿼리와 메인쿼리

- 메인 쿼리(외부 쿼리)
  - 결과를 반환할 때 사용할 최종 쿼리문
- 서브 쿼리(내부 쿼리)
  - SQL 문 내부에 포함되는 괄호 안의 쿼리문
  - 중첩 SELECT 문 또는 하위 SELECT 문
  - 외부 쿼리(메인쿼리)에 사용될 값만 처리
  - 서브 쿼리 결과는 화면에 반환되지 않음
  - 특별한 경우를 제외하고는 서브 쿼리는 메인 쿼리보다 먼저, 한 번만 실행

서브쿼리가 필요한 경우

- 두 개의 순차적인 쿼리를 수행하여 첫 번째 쿼리 결과를 두 번째 쿼리의 검색 값으로 사용하는 경우와 동일한 작업을 하나의 SQL에서 실행

```
SELECT salary FROM y_emp
WHERE emp_name = '오상우'
```

630

```
SELECT salary FROM y_emp
WHERE salary > 630
```

서브쿼리의 실행

- 조건 값을 알 수 없는 SQL 문을 작성하는데 매우 유용
- 주로 WHERE과 HAVING 절에서 연산자의 오른 쪽에 많이 사용
- SQL 문장에서 서브쿼리가 사용될 수 있는 경우
  - WHERE 절
  - HAVING 절
  - FROM 절
  - ORDER BY 절

서브쿼리의 구문

- 서브쿼리는 괄호로 묶어야 하며 일반적으로 비교조건의 오른쪽에 위치
- 한 개의 메인 쿼리에 여러 개의 서브 쿼리가 존재 가능
- 구문

```
SELECT select_list
FROM table
WHERE  표현식    연산자  (SELECT select_list
                          FROM table);
```

Note) 위 구문에서 연산자란 여러 가지 비교 연산자 (=, >, < 등) 들 의미

예제

- 오상우 사원보다 급여를 많이 받는 직원의 이름과 직위, 급여를 표시하시오.

```
SELECT emp_name, position, salary
FROM y_emp
WHERE salary > ( SELECT salary FROM y_emp
                 WHERE emp_name = '오상우');
```

서브 쿼리의 종류

- 반환되는 행의 수 기준
  - 단일 행 서브쿼리
  - 다중 행 서브쿼리
- 다중 열 서브쿼리
- 인라인 뷰
  - FROM 절에 사용된 서브쿼리
- 스칼라 서브쿼리
  - 하나의 단일값을 나타내기 위해 SELECT LIST, WHERE절, ORDER BY절, DML등에 사용하는 서브쿼리
- 상호관련 서브쿼리
  - 메인쿼리와 서브쿼리의 특정 열 값을 상호비교

7.2 단일 행 서브 쿼리와 다중 행 서브 쿼리

단일 행 서브쿼리와 다중 행 서브쿼리

- 서브 쿼리에서 반환되는 행 수에 의해 구분
  - 서브 쿼리로부터 단일 행을 반환 : 단일 행 서브 쿼리
  - 서브쿼리로 부터 여러 행을 반환 : 다중 행 서브 쿼리
- 서브 쿼리에서 사용될 비교연산자를 구분하여 사용
  - 단일 행 연산자(>, =, >=, <, <>, <=)
  - 여러 행 연산자(IN, ANY, ALL)

단일 행 서브 쿼리

- 내부 SELECT 문(서브 쿼리)에서 하나의 행을 반환하는 서브 쿼리
- 단일 행 비교 연산자를 사용
- 사용 가능한 단일 행 비교 연산자

=	같다
>	크다
>=	크거나 같다
<	작다
<=	작거나 같다
<>	같지않다

예제

- 다음은 1049번 사원과 직급이 동일한 사원을 표시하고 있다.

```
SELECT emp_name, position
FROM   y_emp
WHERE  position = (SELECT position
                   FROM   y_emp
                   WHERE  emp_id = 1049);
```

예제

- 다음 예제는 1044번 사원과 업무가 동일하면서 1050번 사원보다 급여를 많이 받는 사원을 반환하기 위한 구문이다. 이 예제는 1개의 메인 쿼리와 2개의 서브 쿼리로 이루어져 있다.

```
SELECT emp_name, position, salary
FROM   y_emp
WHERE  position = (SELECT position FROM   y_emp
                   WHERE  emp_id = 1044)
AND    salary > (SELECT salary FROM   y_emp
                  WHERE  emp_id = 1050);
```

예제

- 다음 예제에서 서브 쿼리의 MIN 그룹 함수를 사용하여 회사에서 가장 적은 급여를 받는 사원에 대한 정보가 출력된다.

```
SELECT emp_id,  emp_name, position, salary
FROM   y_emp
WHERE  salary = (SELECT MIN(salary)
                 FROM   y_emp);
```

- 다음은 HAVING 절에 서브쿼리를 사용하여 회사의 업무 중 평균 급여가 가장 낮은 직급과 그 평균 급여를 반환하는 예제이다.

```
SELECT   position, AVG(salary)
FROM     y_emp
GROUP BY position
HAVING   AVG(salary) = (SELECT   MIN(AVG(salary))
                        FROM     y_emp
                        GROUP BY position);
```

서브 쿼리에서 발생할 수 있는 문제

- 연산자의 잘못된 사용
  - 단일 행 비교연산자를 사용하면서 서브 쿼리에서는 여러 행을 반환하는 경우
  - 쿼리의 구조를 변경하거나 연산자의 종류 변경
- 내부 쿼리에서 행을 반환하지 않는 경우
  - 이 경우 서브 쿼리는 NULL 값을 반환하고 메인 쿼리는 서브 쿼리의 결과(NULL)를 받아 WHERE 절에서 사용
  - 결과로 아무 행이 반환되지 않음
- 잘못 작성된 서브쿼리 사용자가 잘못된 데이터를 최종 결과로 사용

예제

- 다음은 직급이 '차장'인 사원과 급여가 같은 사원을 찾는 문장이다. 서브쿼리가 여러 행을 반환하여 다일행 서브쿼리인 메인쿼리가 오류를 반환하는 문장이다.

```
SELECT emp_id, emp_name, position, salary
FROM y_emp
WHERE salary = (SELECT salary FROM y_emp
                WHERE position='차장');
WHERE salary = (SELECT salary FROM y_emp
                *
ERROR at line 3:
ORA-01427: single-row subquery returns more than one row
```

예제

- 다음 예제는 그룹 함수를 사용한 서브 쿼리에 GROUP BY 절이 포함되어 여러 행을 반환하는 단일 행 서브쿼리이다. 이 SQL 역시 연산자의 종류와 서브 쿼리에서 반환되는 행의 수가 맞지 않아 오류가 발생한다.

```
SELECT emp_id, emp_name   FROM    y_emp
WHERE  salary = (SELECT    MIN(salary)
                 FROM y_emp
                 GROUP BY dept_id);
(SELECT    MIN(salary)
          *
ERROR at line 4:
ORA-01427: single-row subquery returns more than one row
```

예제

- 다음 예제는 서브 쿼리에서 NULL을 반환하여 메인 쿼리의 결과도 없는 경우이다.

```
SELECT emp_id, emp_name, position
FROM y_emp
WHERE mgr_id = (SELECT mgr_id FROM y_emp
                WHERE emp_id = 1045);
no rows selected
```

다중 행 서브쿼리

- 여러 행을 반환하는 서브 쿼리
- 여러 값을 처리하는 다중 행 연산자를 사용
- 다중 행 연산자의 종류

IN	목록의 임의의 값과 동등 비교
ANY	값을 서브 쿼리에 의해 반환된 각 값과 비교
ALL	값을 서브 쿼리에 의해 반환된 모든 값과 비교

예제

- 다음은 200번 이하의 부서에 대하여 각 부서의 최고급여를 구한 후 그 급여와 동일한 급여를 받는 사원정보를 반환한다.

```
SELECT emp_name, salary, dept_id
FROM   y_emp
WHERE  salary IN (SELECT   MAX(salary)
                  FROM     y_emp
                  WHERE dept_id <= 200
                  GROUP BY dept_id );
```

예제

- 다음은 직책이 대리가 아니면서 급여가 임의의 대리인 사원보다 낮은 사원을 표시한다.

```
SELECT emp_id, emp_name, position, salary
FROM   y_emp
WHERE  salary < ANY
                (SELECT salary FROM   y_emp
                 WHERE  position = '대리')
AND    position <> '대리';
```

위 예제에서 연산자의 종류를 >ANY, >ALL, <ALL로 변경하여 각각의 결과를 표시하고 비교해 본다.

7.3 다중 열 서브 쿼리

다중 열 서브쿼리의 개념

- 서브 쿼리에서 반환되는 데이터가 두 개 이상의 열을 반환한다면 다중 열 서브 쿼리를 사용
- 두 개 이상의 열을 비교하는 경우
  - 논리 연산자를 사용하여 혼합 WHERE 절을 작성
  - 중복된 WHERE 조건을 하나의 WHERE 절로 결합
- 다중 열 서브 쿼리에서 열의 비교 방식
  - 쌍 비교 방식(Pairwise)
  - 비쌍 비교 방식(Unpairwise)

다중 열 서브 쿼리의 쌍 비교

- 서브쿼리에서 여러 개의 행을 여러 개의 열과 함께 반환
  - 비교할 열 들을 괄호로 묶어서 동시에 비교하는 방식
  - 비교 연산자 좌우의 열의 수가 같아야 함
  - 출력되는 결과는 두 열의 값이 동시에 일치하는 행들
- 구문

```
SELECT        column, column, ...
FROM  table
WHERE (column, column, ...) IN
                      (SELECT column, column, ...
                              FROM   table
                              WHERE  조건 );
```

다중 열 서브 쿼리의 비쌍 비교

- 둘 이상의 열의 비교에 서브 쿼리를 별도로 작성하여 각 열을 따로 비교하는 방식
- 서브쿼리의 개수가 비교되는 열 수만큼 AND를 사용
- 구문

```
SELECT column, column, ...
FROM  table
WHERE  column1 IN
                  (SELECT column1
                   FROM   table
                   WHERE  조건)
AND    column2 IN
                  (SELECT column2
                   FROM   table
                   WHERE  조건);
```

예제

- 다음 예제는 쌍 비교 방식으로 성이 차씨인 사원들과 직급 및 부서가 동일한 사원의 정보를 반환한다.

```
SELECT emp_id, emp_name, position, dept_id
FROM y_emp
WHERE (position, dept_id) IN (SELECT position,dept_id FROM y_emp
                              WHERE emp_name LIKE '차%')
                              AND emp_name  NOT LIKE '차%';
```

예제

- 다음은 비쌍 비교 방식을 통하여 성이 차씨인 사원들의 직급, 부서 번호와 직급 및 부서 번호가 동일한 사원의 정보를 반환한다.

```
SELECT emp_id, emp_name, position, dept_id
FROM y_emp
WHERE position IN (SELECT position  FROM y_emp
                   WHERE emp_name LIKE '차%')
AND dept_id IN (SELECT dept_id FROM y_emp
                WHERE emp_name LIKE '차%')
AND emp_name  NOT LIKE '차%';
```

7.4 상호 관련 서브 쿼리 (Correlated Subquery)

상호 관련 서브 쿼리

- 메인 쿼리와 서브 쿼리가 서로 연관되어 실행되는 SQL
- 서브 쿼리가 메인 쿼리에서 사용하는 테이블의 열을 참조할 때 상호관련 서브 쿼리를 수행
- 상호관련 서브 쿼리는 메인 쿼리에서 처리되는 각 행에 대해 한 번씩 평가
- 상호관련 서브 쿼리는 메인 쿼리의 모든 행에 대해 한 번씩 실행

상호 관련 서브 쿼리

- 구문

```
SELECT column1, column2, ...
FROM   table1 outer
WHERE  column1 연산자 (SELECT  colum1, column2
                        FROM    table2
                       WHERE   expr1 = outer.expr2);
```

- 서브 쿼리에 메인 쿼리와의 상관 관계를 정의
  - 서브 쿼리의 WHERE 절에서 메인 쿼리에서 사용하는 테이블의 열을 서브 쿼리의 관련된 데이터와 비교
  - 서브 쿼리가 메인 쿼리에 의해서 실행
- 메인 쿼리의 모든 후보 행의 수만큼 반복실행
  - 처리할 데이터의 양에 따라 처리 시간이 상당히 많이 소요되어 데이터베이스의 성능 저하 가능성

EXISTS 연산자

- 외부 쿼리에서 검색된 값이 내부 쿼리에서 검색된 값의 결과 집합에 존재하는지 여부를 검사하기 위한 상호관련 서브 쿼리
  - EXISTS 연산자를 사용
  - 상호 관련 서브 쿼리의 처리 성능 상의 문제 해소
- EXISTS 연산자
  - 서브 쿼리가 한 행 이상 반환하면 TRUE를 반환
  - 해당 값이 없으면 FALSE를 반환

NOT EXISTS 연산자

- 메인 쿼리에서 검색된 값이 서브 쿼리에서 검색된 값의 결과 집합에 속하지 않는지 여부를 검사
  - EXISTS 연산자와 동일한 프로세스를 실행
  - 상수 'X'를 반환하지 않는 후보 행에 대한 정보를 반환하는 연산자

예제

- 상호 관련 서브 쿼리를 이용해 소속부서의 평균급여보다 급여를 더 많이 받는 사원의 이름, 급여, 부서번호를 출력한다.

```
SELECT emp_name, salary, dept_id
FROM   y_emp outer
WHERE  salary > (SELECT AVG(salary) FROM  y_emp
                 WHERE  dept_id =outer.dept_id);
```

예제

- 다음 예제는 EXISTS 연산자를 사용하여 부하 직원이 있는 사원을 검색한다.

```
SELECT emp_id, emp_name, position, dept_id
FROM   y_emp o
WHERE  EXISTS ( SELECT 'X' FROM   y_emp
                WHERE  mgr_id = o.emp_id) ;
```

- NOT EXISTS 연산자를 사용하여 부하직원이 없는 사원, 평사원들을 검색해 보자.

```
SELECT emp_id, emp_name, position, dept_id
FROM   y_emp o
WHERE  NOT EXISTS ( SELECT 'X' FROM   y_emp
                    WHERE  mgr_id = o.emp_id);
```

7.5 FROM절에 사용하는 서브 쿼리 (Inline View)

인라인 뷰(INLINE VIEW)

- SELECT 문장의 FROM 절에 사용된 서브쿼리
- 조인 시 테이블에 별칭을 주는 것처럼 서브 쿼리에 대하여 별칭 사용
  - 하나의 SQL문장 내부에서는 서브 쿼리의 결과 집합을 하나의 테이블처럼 접근하기 때문에 인라인 뷰 라고 함
- 데이터베이스 실무에서 많이 사용

예제

- 400번 미만의 부서에 속하는 사원들에 대하여 소속 부서의 평균 급여보다 많은 급여를 받는 모든 사원의 이름, 급여, 부서 번호 및 소속 부서의 평균 급여를 표시하시오.

```
SELECT  a.emp_name, a.salary, a.dept_id, b.salavg
FROM    y_emp a
JOIN (SELECT dept_id, AVG(salary) salavg FROM y_emp
      GROUP BY dept_id) b
ON ( a.dept_id = b.dept_id )
AND a.dept_id < 400
AND a.salary > b.salavg;
```

7.6 Top-N 분석

Top-N 분석 쿼리

- 테이블에서 조건에 맞는 최상위 레코드 n개 또는 최하위 레코드 n개를 표시하는 쿼리문을 의미
- TON-N 쿼리의 예시
  - 인터넷의 포털사이트에는 검색어 순위
  - 서점이나 온라인 쇼핑몰의 베스트셀러 정보
  - 회사의 최상위 소득자 세 명
  - 회사에 가장 오래 근무한 사원 세명
- 인라인 뷰의 기본구조에 ROWNUM 등 몇 가지 필수 구성 요소를 포함하여 작성

TOP-N 분석 구문

- 구문

```
SELECT [column,…], ROWNUM
FROM   (SELECT [column,…]
        FROM table
        ORDER  BY Top-N_column)
WHERE  ROWNUM <=  N;
```

- 서브쿼리 또는 인라인 뷰에 ORDER BY 절을 반드시 포함
  - 최대값(상위순위)을 검색하려면 DESC 파라미터를 ORDER BY 열에 지정
- 메인 쿼리에 의사열 ROWNUM을 SELECT 목록에 반드시 포함
- WHERE 절에 반환될 행 수를 제한하기 위하여 의사열인ROWNUM과 < 또는 <= 연산자 사용

예제

- Y_EMP 테이블에서 급여를 가장 많이 받는 5명의 정보를 표시한다.

```
SELECT ROWNUM as RANK, emp_id, emp_name, salary
FROM  (SELECT emp_id, emp_name,salary FROM y_emp
       WHERE salary IS NOT NULL
       ORDER BY salary DESC)
       WHERE ROWNUM <= 5 ;
```

- Y_EMP 테이블에서 최장기 근무 사원 3명의 정보를 표시한다.

```
SELECT ROWNUM as RANK,E.emp_name, E.hiredate
FROM  (SELECT emp_name,hiredate FROM y_emp
       ORDER BY hiredate)E
       WHERE rownum <= 3 ;
```

7.7 기타 서브 쿼리의 사용

서브쿼리의 사용 방식

- WHERE 절 또는 HAVING 절에 많이 사용
- FROM 절에 사용하여 테이블이나 뷰처럼 활용
- CASE 구문이나 ORDER BY절에도 사용 가능
- DML 명령문에서도 서브 쿼리를 활용
  - 다른 테이블의 데이터를 이용하여 테이블에 입력, 수정, 삭제하는 경우

예제

- 다음은 CASE 구문에 서브 쿼리가 사용되는 경우이다. 서브 쿼리는 사원들에 대하여 위치 ID가 4인 부서의 부서 ID인 400을 반환한다. 메인 쿼리의 CASE 표현식에서는 서브 쿼리의 결과를 사용하여 메인 쿼리에서 검색된 행의 부서 ID가 400이면 대구를 그렇지 않으면 기타를 EMPID, EMP_NAME과 함께 표시한다.

```
SELECT emp_id, emp_name,
       (CASE WHEN dept_id =
                          (SELECT dept_id FROM y_dept
                           WHERE loc_id =4)
        THEN '대구' ELSE '기타' END) loc_name
FROM   y_emp;
```

예제

- 이번 예제는 ORDER BY 절에 사용된 서브 쿼리이다. 다음은 Y_EMP 테이블에 존재하지 않는 Y_DEPT 테이블이 DEPT_NAME열을 기준으로 Y_EMP의 결과를 정렬한다.

```
SELECT    emp_id, emp_name
FROM      y_emp e
ORDER BY (SELECT dept_name FROM y_dept d
           WHERE e.dept_id = d.dept_id) ;
```

7.8 WITH 절의 사용

WITH 절을 사용하여 쿼리 블록 정의

- 이름과 함께 정의한 WITH 절의 쿼리 블록을 SELECT 문에서 참조하는 방법
- 메인쿼리가 동일한 쿼리 블록을 여러 번 참조하거나 복잡한 조인이나 집계를 해야 하는 경우 매우 유용
- 내용이 복잡한 복합 SELECT 문에서 해당 쿼리 블록이 여러 번 발생하는 경우 유용
- 같은 쿼리를 다시 사용할 수 있어서 성능 향상

예제

- 다음은 전체 부서의 평균 총 급여보다 총 급여가 많은 부서의 부서이름(DEPT_NAME) 및 총 급여를 표시하기 위하여 WITH 절을 사용한 예제이다.

```
WITH
    dept_total_sal  AS ( SELECT  d.dept_name, SUM(e.salary) AS dept_total
                         FROM    y_emp e JOIN y_dept d
                         ON   (e.dept_id = d.dept_id)
                         GROUP BY d.dept_name),
    total_avg_sal  AS (SELECT SUM(dept_total)/COUNT(*) AS dept_avg
                       FROM   dept_total_sal)
SELECT * FROM   dept_total_sal
WHERE  dept_total > (SELECT dept_avg
                     FROM total_avg_sal);
```

실습 설명

- WITH 절의 쿼리블록
  - 모든 부서에 대해 부서 이름 별 총 급여를 계산한 DEPT_TOTAL_SAL
  - DEPT_TOTAL_SAL의 결과에 대한 평균 총 급여를 저장한 TOTAL_AVG_SAL
- SELECT 문에서 WITH 절에 정의한 쿼리 블록들을 사용
- 결과 데이터
  - 첫 번째 쿼리블록 DEPT_TOTAL_SAL에서 계산한 총 급여를 두 번째 쿼리 블록에서 계산한 평균 총 급여와 비교
  - 특정 부서의 총 급여가 전체 부서의 평균 총 급여보다 많으면 해당 부서 이름 및 총 급여를 표시한 것

WITH 절의 특장점

- WITH 절은 SELECT 문에 대해서만 사용 가능
- 쉼표로 구분하면서 여러 개의 쿼리 블록 정의 가능
- 쿼리 블록은 다른 쿼리 블록을 참조하여 작성 가능
- 내부적으로 인라인 뷰 또는 임시 테이블로 해석
- 성능을 향상시키고 보다 간단한 SQL 코드 작성 가능

THANKS!