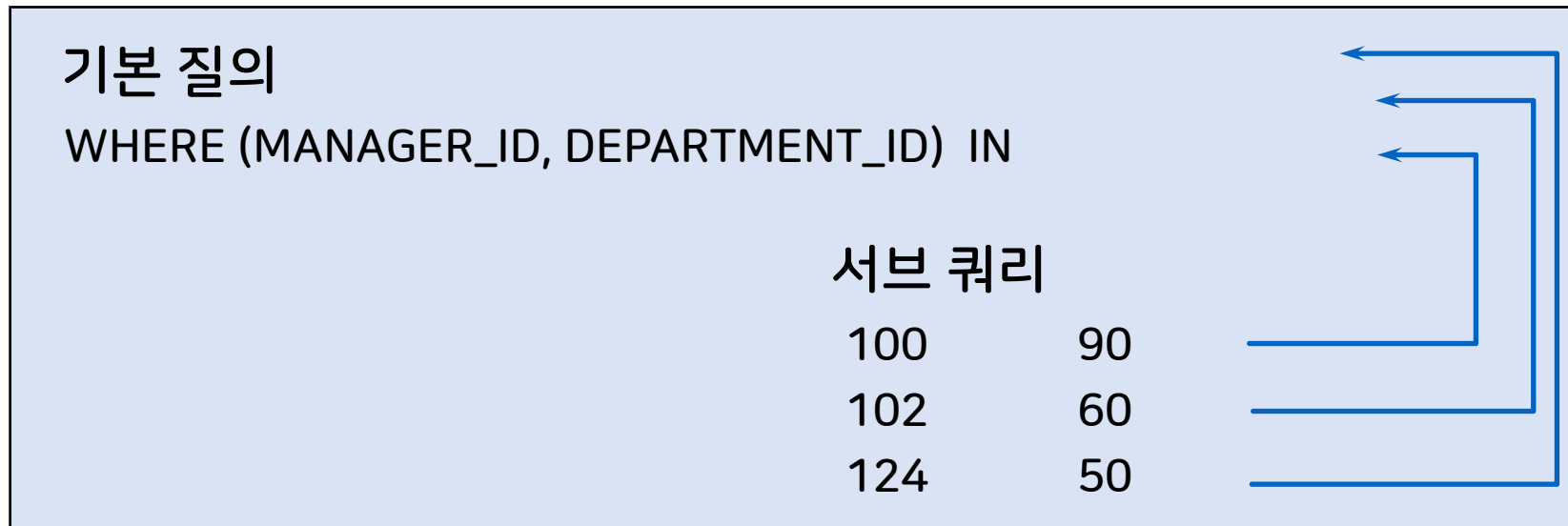


7. 고급 서브쿼리



다중 열 서브 쿼리

- 두 개 이상의 열을 비교하려면 논리 연산자를 사용하여 혼합 WHERE 절을 작성해야 합니다.
- 메인쿼리의 각 행은 Multiple-row 및 Multiple-column 서브쿼리의 값과 비교됩니다.



열 비교


- 여러 열 서브 쿼리의 열 비교 방식은 다음과 같을 수 있습니다.
 - 쌍(pairwise) 비교
 - 비쌍(nonpairwise) 비교
- 구문

```
SELECT column, column, ...  
FROM table  
WHERE(column, column, ...) IN  
      (SELECT column, column, ...  
        FROM table  
        WHERE condition)
```

쌍(pairwise) 비교 서브 쿼리

- 이름이 "John"인 사원과 동일한 관리자의 감독을 받고 동일한 부서에서 근무하는 사원에 대한 세부 정보를 표시합니다.

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE (manager_id, department_id) IN
      (SELECT manager_id, department_id
       FROM employees
       WHERE first_name = 'John')
AND first_name <> 'John';
```



	MANAGER_ID	DEPARTMENT_ID
1	108	100
2	123	50
3	100	80

비쌍 비교 서브 쿼리

- 이름이 "John"인 사원과 동일한 관리자의 감독을 받고 동일한 부서에서 근무하는 사원의 세부 정보를 표시합니다

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE manager_id IN
    (SELECT manager_id
     FROM employees
     WHERE first_name = 'John')
AND department_id IN
    (SELECT department_id
     FROM employees
     WHERE first_name = 'John')
AND first_name <> 'John';
```


스칼라 서브 쿼리식

- SELECT 문에서 열 또는 표현식처럼 사용되는 하나의 행에서 하나의 열 값만 반환하는 서브 쿼리를 스칼라 서브쿼리(scalar subquery) 표현식 이라고 합니다.
 - 스칼라 서브쿼리는 열 또는 표현식처럼 사용하므로 결과는 반드시 하나의 열에 하나의 값이어야 합니다.
 - 스칼라 서브 쿼리식의 값은 서브 쿼리의 SELECT 목록에 있는 항목의 값입니다.
 - 서브쿼리의 결과는 NULL일 수는 있지만 두 개 이상의 값을 반환하면 오류가 발생합니다.
- 스칼라 서브쿼리는 다음에서 사용할 수 있습니다.
 - SELECT 절의 DECODE 및 CASE의 조건 및 표현식 부분
 - GROUP BY를 제외한 SELECT의 모든 절

스칼라 서브 쿼리: 예제(1)

- 다음 예제는 CASE 표현식에 사용된 스칼라 서브 쿼리입니다.
 - 다음 서브 쿼리는 위치 ID가 1800인 부서의 부서 ID인 20을 반환합니다.
 - 외부 질의의 CASE 표현식에서는 외부 질의에서 검색된 레코드의 부서 ID가 20이면 Canada를 부서 ID가 20이 아니면 USA를 표시합니다.

```
SELECT employee_id, last_name,  
       CASE WHEN department_id = (SELECT department_id  
                                FROM departments  
                                WHERE location_id = 1800)  
       THEN 'Canada' ELSE 'USA' END AS location  
FROM   employees;
```



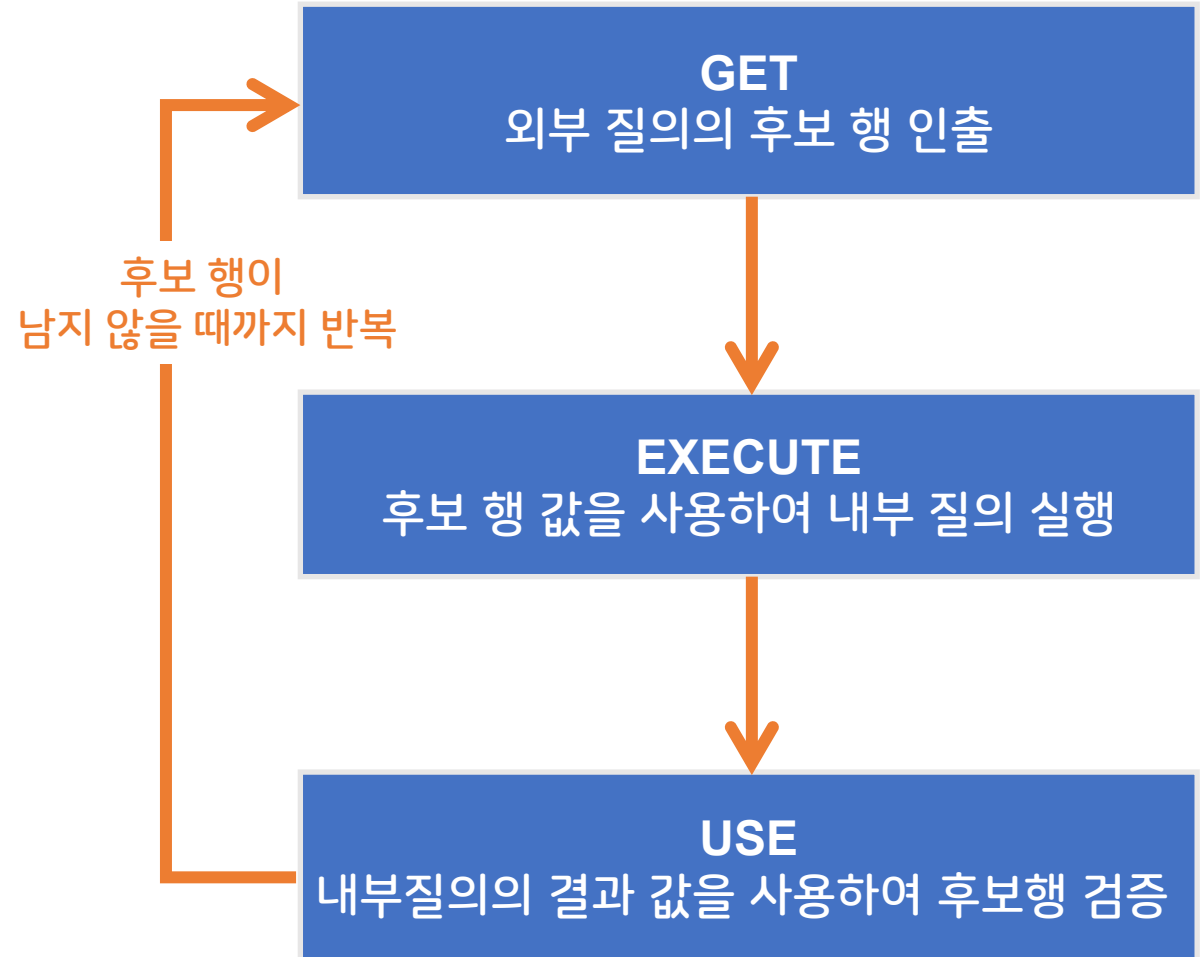
스칼라 서브 쿼리: 예제(2)

- 스칼라 서브쿼리를 사용하여 JOIN 문을 사용하지 않고 JOIN 결과를 반환할 수 있습니다.
- 만약 서브쿼리에 결과 데이터가 없는 경우 NULL을 반환합니다.
- 예제는 부서번호와 부서이름과 함께 각 부서의 최고급여를 반환합니다.

```
SELECT d.department id, d.department name,  
       (SELECT MAX(salary) FROM employees  
        WHERE department_id = d.department_id) AS 최고급여  
FROM departments d;
```


상호관련 서브 쿼리 (correlated subquery)

- 상호관련 서브쿼리는 서브 쿼리가 상위문에서 참조되는 테이블의 열을 참조합니다.
- 각 서브 쿼리는 상위 문에서 처리되는 각 행에 대해 한 번씩 평가되어, 질의의 모든 행에 대해 한 번씩 실행됩니다.



상호관련 서브 쿼리 (correlated subquery)

- 상호관련 서브쿼리에서, 서브쿼리는 상위 쿼리 테이블의 열을 참조합니다.
- 구문

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

- 상호관련 서브 쿼리는 테이블에서 모든 행을 읽어서 각 행의 값을 관련된 데이터와 비교하는 방법 중 하나입니다.

상호관련 서브 쿼리 사용

- 다음 예제는 자신의 소속 부서의 평균 급여보다 많은 급여를 받는 사원을 모두 찾습니다.

```
SELECT last_name, salary, department_id
FROM   employees outer
WHERE  salary > (SELECT AVG(salary)
                FROM   employees
                WHERE  department_id = outer.department_id) ;
```

메인쿼리의 행이 처리될 때마다 서브쿼리가 평가됩니다.

EXISTS 연산자 사용

- EXISTS 연산자는 subquery의 결과 집합에 행이 있는지 테스트합니다.
- 서브쿼리 행 값이 있을 경우:
 - 검색이 inner query에서 계속 수행되지 않습니다.
 - 조건은 TRUE로 플래그가 지정됩니다.
- 서브쿼리 행 값이 없을 경우:
 - 조건은 FALSE로 플래그가 지정됩니다.
 - 검색이 inner query에서 계속 수행됩니다.

EXISTS 연산자 사용

- 다음 예제는 EXISTS 연산자를 사용하여 부하직원이 있는 사원을 찾습니다.

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT 'X'
                FROM   employees
                WHERE  manager_id = outer.employee_id);
```

조건을 만족하는 행이 하나라도 발견될 경우
내부 질의를 더 이상 검색하지 않습니다

- IN 구문을 EXISTS 연산자 대신 사용할 수 있으며, 이에 대한 예제는 다음과 같습니다.

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  employee_id IN (SELECT manager_id
                      FROM   employees
                      WHERE  manager_id IS NOT NULL);
```

- 성능면에서 EXISTS 연산자의 사용을 더 권장합니다.

NOT EXISTS 연산자 사용

- 다음 예제는 사원이 없는 빈 부서를 검색하기 위해 NOT EXISTS 연산자를 사용합니다.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM employees
                  WHERE department_id = d.department_id);
```

- NOT IN 구문을 NOT EXISTS 연산자 대신 사용할 수 있습니다.

인라인 뷰(In-line View)

- SELECT 문의 FROM 절에 있는 서브 쿼리는 인라인 뷰라고도 합니다.
 - FROM 절에 서브 쿼리를 만들어 이 서브 쿼리에 별칭을 부여할 경우 인라인 뷰가 생성됩니다.
- SELECT 문의 FROM 절에 있는 서브 쿼리는 해당 SELECT 문에 대해서만 데이터 소스를 정의합니다.
- 이는 테이블 또는 뷰를 사용하는 방식과 매우 비슷합니다.
- 인라인 뷰는 스키마 객체가 아닙니다.

인라인 뷰의 사용 예제

- SELECT 문의 FROM 절에 서브 쿼리를 사용할 수 있습니다.
- 다음 예제는 사원의 이름, 급여, 부서 번호 및 소속 부서의 평균 급여보다 많은 급여를 받는 모든 사원의 평균 급여를 표시합니다.

```
SELECT  a.last_name, a.salary, a.department_id, b.salavg
FROM    employees a JOIN (SELECT  department_id,
                                TRUNC(AVG(salary)) salavg
                                FROM    employees
                                GROUP BY department_id) b
ON      (a.department_id = b.department_id)
WHERE   a.salary > b.salavg;
```

FROM 절에 있는 서브 쿼리를 b라고 명명하고
외부 질의가 이 별칭을 사용하여
SALAVG 열을 참조합니다.

“Top-N” 분석

- Top-N 질의는 열에서 가장 큰 n개의 값 또는 가장 작은 n개의 값을 요청합니다.
- 테이블에서 조건에 맞는 최상위 레코드 n개 또는 최하위 레코드 n개를 표시하는 시나리오에 유용합니다.
- Top-N 분석을 사용하여 다음 유형의 질의를 수행할 수 있습니다.
 - 회사의 최상위 소득자 세 명
 - 회사에 가장 최근에 입사한 신입 사원 네 명
 - 제품을 가장 많이 판매한 영업 사원 두 명
 - 최근 6개월 동안 가장 많이 팔린 제품 세 가지

“Top-N” 분석 수행

- Top-N 분석 질의의 구조는 다음과 같습니다.
 - 정렬된 데이터 목록을 생성하는 ORDER BY 절이 포함된 서브 쿼리 또는 인라인 뷰.
 - 최종 결과 집합의 행 수를 제한하는 메인쿼리 구성 요소
 - ROWNUM 의사 열: 서브 쿼리에서 반환되는 각 행에 1부터 시작하는 순차 값을 할당
 - < 또는 <= 연산자를 사용하는 WHERE 절: 반환될 n개 행을 지정
- 구문

```
SELECT [column_list], ROWNUM
FROM   (SELECT [column_list]
        FROM table
        ORDER BY Top-N_column [ASC|DESC])
WHERE  ROWNUM <= n;
```

“Top-N” 분석 수행의 예제

- 다음은 EMPLOYEES 테이블에서 최상위 소득자 세 명의 이름 및 급여를 순위와 함께 표시합니다.

```
SELECT ROWNUM as RANK, last_name, salary
FROM (SELECT last_name,salary FROM employees
      ORDER BY salary DESC)
WHERE ROWNUM <= 3;
```

- 다음 예제는 인라인 뷰를 사용하여 회사의 최장기 근무 직원 네 명을 표시합니다.

```
SELECT ROWNUM as SENIOR,E.last_name, E.hire_date
FROM (SELECT last_name,hire_date FROM employees
      ORDER BY hire_date)E
WHERE rownum <= 4;
```

WITH 절

- WITH 절을 사용하면 복합 쿼리 내에서 여러 번 발생하는 동일한 쿼리 블록을 SELECT 문에서 사용할 수 있습니다.
- WITH 절은 질의 블록의 결과를 검색한 다음 이를 사용자 임시 테이블스페이스에 저장합니다.
- WITH 절은 같은 쿼리 블록을 여러 번 참조하거나 조인 및 집계를 해야 하는 경우 매우 유용합니다.
- WITH 절의 이점
 - 질의를 읽기 쉽게 만듭니다.
 - 해당 절이 질의에서 여러 번 사용될지라도 한 번만 평가하므로 성능이 향상됩니다.

WITH 절: 예제

- WITH 절을 사용하여 전체 부서의 평균 총 급여보다 총 급여가 많은 부서의 부서 이름 및 총 급여를 표시하는 질의를 작성합니다.
- 이 문제 해결을 위해 다음 중간 계산이 필요합니다.
 - WITH 절을 사용하여 모든 부서의 총 급여를 계산한 다음 결과를 저장합니다.
 - WITH 절을 사용하여 전체 부서의 평균 총 급여를 계산한 다음 결과를 저장합니다.
 - 1단계에서 계산한 총 급여를 2단계에서 계산한 평균 총 급여와 비교합니다. 특정 부서의 총 급여가 전체 부서의 평균 총 급여보다 많으면 해당 부서 이름 및 총 급여를 표시합니다.

WITH 절: 예제

```
WITH
dept_costs AS (
    SELECT d.department_name, SUM(e.salary) AS dept_total
    FROM   employees e JOIN departments d
    ON     (e.department_id = d.department_id)
    GROUP BY d.department_name),
avg_cost AS (
    SELECT SUM(dept_total)/COUNT(*) AS dept_avg
    FROM   dept_costs)
SELECT *
FROM   dept_costs
WHERE  dept_total >
      (SELECT dept_avg
       FROM avg_cost)
ORDER BY department_name;
```

Thank you 😊