1 다양한 서브쿼리를 이용한 데이터 검색.

◈ 학습 내용.

■ 인라인 뷰(IN-LINE VIEW) 사용법을 학습합니다. FROM절에 사용되는 서브쿼리

■ 다중 열 서브쿼리(Multi-Column Subquery)를 사용하는 방법을 학습합니다.

■ 쌍-비교(Pairwise Comparison) 및 비쌍-비교(Non-Pairwise Comparison) 방식에 대하여 학습합니다.

■ SQL문에서 스칼라 서브쿼리(Scalar Subquery)를 사용하는 방법을 학습합니다.

■ 상관관계 서브쿼리(Correlated Subquery)를 사용하여 데이터를 조회하는 방법을 학습합니다.

■ 상관관계 서브쿼리(Correlated Subquery)를 사용하여 행을 갱신하고 행을 삭제하는 방법을 학습합니다.

■ EXISTS 연산자 및 NOT EXISTS 연산자를 사용하는 방법을 학습합니다.

■ WITH절을 사용하는 방법을 학습합니다.

1-1. 인라인 뷰(IN-LINE VIEW) 사용하기. FROM절에 사용되는 서브쿼리

■ 오라클 데이터베이스 서버에서는 SELECT문, INSERT문, UPDATE문, DELETE문에서 테이블 또는 뷰 이름을 기술해야 하는 곳에 서브쿼리를 기술할 수 있습니다. 이렇게 **SELECT문, INSERT문, UPDATE문, DELETE문에서 테이블이나 뷰의 이름 대신, 기술된 서브쿼리를 인라인 뷰(In-Line View)라고 합니다.**

■ 테이블 또는 뷰 대신 기술된 서브쿼리(인라인 뷰)는 가 메인 SQL 문장에서의 데이터 소스 역할을 합니다.

■ HR.EMPLOYEES 테이블의 데이터를 이용하여, 자신이 속한 부서의 평균월급보다 많은 월급을 받는 모든 직원의 성(LAST_NAME), 월급(SALARY), 부서코드(DEPARTMENT_ID), 및 자신이 속한 부서의 평균임금을 구하시오.

```
SQL> SELECT a.last_name, a.salary, a.department_id, b.salavg
     FROM hr.employees a JOIN (SELECT e.department_id, AVG(e.salary) AS SALAVG
                               FROM hr.employees e
                               GROUP BY e.department_id) b
     ON a.department_id = b.department_id
     WHERE a.salary > b.salavg;

LAST_NAME           SALARY DEPARTMENT_ID     SALAVG
--------------- ---------- ------------- ----------
Hartstein            13000            20       9500
Higgins              12000           110      10150
King                 24000            90 19333.3333
Hunold                9000            60       5760
...
 38개의 행이 선택됨
```

☞ 위의 문장에서 **인라인 뷰의 SELECT절에 AVG(salary) 표현식에 대하여 정의된 SALAVG 컬럼별칭은, 메인-쿼리에서는 컬럼이름으로 사용됩니다.** 따라서, 메인-쿼리의 WHERE절에서 사용된 b.salavg는 컬럼별칭이 아니라 데이터소스로서의 컬럼이름입니다. 만약 위의 구문에서처럼, **인라인 뷰의 SELECT절에 사용된 표현식에 대하여 컬럼별칭을 지정하지 않으면, 아래와 같이 오류가 발생됩니다.**

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

ORA-00904: "B"."AVG": 부적합한 식별자
00904. 00000 -  "%s: invalid identifier"
*Cause:
*Action:
6행, 18열에서 오류 발생
```

■ HR.DEPARTMENTS, HR.LOCATIONS, HR.REGIONS 테이블의 데이터를 이용하여 'Europe' 지역에 있는 부서의 부서이름(DEPARTMENTS.DEPARTMENT_NAME)과 부서가 존재하는 도시명(LOCATIONS.CITY)을 표시하시오.

이 때, 다음의 정보를 이용합니다.

○ 대륙이름('Europe')은 HR.REGIONS 테이블의 REGION_NAME 컬럼을 통해 확인할 수 있습니다.

○ 국가코드는 HR.COUNTRIES 테이블의 COUNTRY_ID 컬럼을 통해 확인할 수 있습니다.

```
SQL> SELECT a.department_name, b.city
     FROM hr.departments a
     INNER JOIN (SELECT l.location_id, l.city, l.country_id
                 FROM hr.locations l JOIN hr.countries c ON (l.country_id = c.country_id)
                                     JOIN hr.regions r ON (c.region_id=r.region_id)
                 WHERE r.region_name = 'Europe') b
     ON a.location_id=b.location_id ;

DEPARTMENT_NAME              CITY
---------------------------- ------------------------------
Human Resources              London
Sales                        Oxford
Public Relations             Munich
```

☞ 위의 쿼리를 다음의 과정을 통해 수행할 수도 있습니다. 오라클 데이터베이스처럼 인라인 뷰를 사용할 수 없는 경우에는, 아래의 방법을 이용해서 같은 작업을 수행할 수 있습니다.

1> 인라인 뷰에 해당하는 결과를 표시하는 뷰(VIEW)를 생성합니다.

```
SQL> CREATE VIEW HR.EUROPE_CITY
     AS SELECT l.location_id, l.city, l.country_id
        FROM hr.locations l JOIN hr.countries c ON (l.country_id = c.country_id)
                            JOIN hr.regions r ON (c.region_id=r.region_id)
        WHERE r.region_name = 'Europe';

view HR.EUROPE_CITY이(가) 생성되었습니다.
```

2> 생성된 뷰를 DEPARTMENTS 테이블과 조인시켜 원하는 결과를 표시합니다.

```
SQL> SELECT d.department_name, e.city
     FROM departments d JOIN hr.europe_city e ON (d.location_id=e.location_id);

DEPARTMENT_NAME              CITY
---------------------------- ------------------------------
Human Resources              London
Sales                        Oxford
Public Relations             Munich
```

■ HR.LOCATIONS 테이블의 LOCATION_ID, CITY, COUNTRY_ID 컬럼에 각각 3600, 'Washington', 'US' 값을 입력하시오.
단, 국가코드인 'US' 값이, Europe 대륙에 있는 국가들 중 하나일 경우에만 행이 입력되도록 하시오.
만약, 'US' 값이 Europe 대륙에 해당하는 국가가 아닐 때는 행이 입력되지 않아야 합니다.

이 때, 다음의 정보를 이용합니다.
○ 대륙이름('Europe')은 HR.REGIONS 테이블의 REGION_NAME 컬럼을 통해 확인할 수 있습니다.
○ 국가코드는 HR.COUNTRIES 테이블의 COUNTRY_ID 컬럼을 통해 확인할 수 있습니다.

```
SQL> INSERT INTO (SELECT location_id, city, country_id
                  FROM locations
                  WHERE country_id IN (SELECT c.country_id
                                       FROM countries c JOIN regions r
                                            ON (c.region_id=r.region_id)
                                       WHERE r.region_name = 'Europe')
                        WITH CHECK OPTION
                 )
   VALUES (3600, 'Washington', 'US');
명령의 1 행에서 시작하는 중 오류 발생 -
INSERT INTO (SELECT location_id, city, country_id
             FROM locations
             WHERE country_id IN (SELECT c.country_id
                                  FROM countries c JOIN regions r
                                       ON (c.region_id=r.region_id)
                                  WHERE r.region_name = 'Europe')
             WITH CHECK OPTION
            )
VALUES (3600, 'Washington', 'US')
오류 보고 -
SQL 오류: ORA-01402: 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다
01402. 00000 -  "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

☞ INSERT문, UPDATE문 또는 DELETE문의 테이블 자리에 [WITH CHECK OPTION 키워드]가 포함된 서브쿼리를 사용하여, 해당 테이블에 대해 서브쿼리에 포함될 수 없는 행을 생성하지 못하도록 할 수 있습니다. 그렇지만, UPDATE문 이나 DELETE문은, [WITH CHECK OPTION 키워드]가 포함된 서브쿼리를 사용하지 않고, WHERE 절을 사용하여 검사 조건을 쉽게 기술할 수 있습니다.

☞ 위의 문장을 수행하면, 서브쿼리 안에 포함된 WITH CHECK OPTION에 의하여 서브쿼리의 조건 절을 사용자가 입력한 값이 만족하는지를 검사하여 만족하지 않는 경우 데이터 삽입이 수행되지 않습니다.

☞ 앞의 쿼리를 다음의 과정을 통해 수행할 수도 있습니다. 오라클 데이터베이스처럼 인라인 뷰를 사용할 수 없는 경우에는, 아래의 방법을 이용해서 같은 작업을 수행할 수 있습니다.

1> 인라인 뷰에 해당하는 결과를 표시하는 뷰(VIEW)를 생성합니다.

```
SQL> CREATE VIEW HR.EUROPE_CITIES
     AS
        SELECT location_id, city, country_id
        FROM locations
        WHERE country_id IN (SELECT c.country_id
                             FROM countries c JOIN regions r ON (c.region_id=r.region_id)
                             WHERE r.region_name = 'Europe')
        WITH CHECK OPTION ;

view HR.EUROPE_CITIES이(가) 생성되었습니다.
```

2> 생성된 뷰에 데이터 입력작업을 수행합니다.

```
SQL> INSERT INTO hr.europe_cities
     VALUES (3400,'New York','US');
명령의 1 행에서 시작하는 중 오류 발생 -
INSERT INTO europe_cities
VALUES (3400,'New York','US')
오류 보고 -
SQL 오류: ORA-01402: 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다
01402. 00000 -  "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

```
--12c 이후
SELECT EMPLOYEE_ID, last_name, salary
FROM hr.employees
ORDER BY salary DESC
OFFSET 0 ROWS FETCH FIRST 2 ROWS ONLY;
-- OFFSET 0 첫번째부터 행을 추출합니다.
-- FETCH : 행을 뽑아오세요
-- FIRST 5 ROWS ONLY : 첫번째 그룹의 5개 행만

SELECT EMPLOYEE_ID, last_name, salary
FROM hr.employees
ORDER BY salary DESC
OFFSET 0 ROWS FETCH FIRST 2 ROWS WITH TIES;


SELECT EMPLOYEE_ID, last_name, salary
FROM hr.employees
ORDER BY salary DESC
OFFSET 0 ROWS FETCH FIRST 50 PERCENT ROWS ONLY;
```

Top - N 분석 결과 질의문.

-- salary 값 상위 5개 레코드

```
SELECT rownum, employee_id, last_name, salary
FROM  (SELECT employee_id, last_name, salary
        FROM hr.employees
        ORDER BY 3 DESC )
WHERE rownum <= 5 ;
```

-- salary 값 하위 5개 레코드

```
SELECT rownum, employee_id, last_name, salary
FROM  (SELECT employee_id, last_name, salary
        FROM hr.employees
        ORDER BY 3 ASC )
WHERE rownum <= 5 ;
```

1-2. WHERE 절에서 다중 컬럼 서브쿼리(Multi-column Subquery)를 사용한 데이터 조회.

■ SQL문의 WHERE절에서, 2개 이상의 컬럼이 연산자에 의하여 비교되도록, **2개의 이상의 컬럼으로 구성된 레코드가 반환되는 서브쿼리를 다중 컬럼 서브쿼리라고 합니다.**

■ 기본문법

```
SELECT 컬럼, 컬럼, ....
FROM 스키마이름.테이블이름
WHERE (컬럼, 컬럼, ...) 연산자 (SELECT column, column, ...
                                FROM 스키마이름.테이블이름
                                WHERE 조건) ;
```

○ 위의 문법에서 WHERE 절의 연산자 오른편에 있는 서브쿼리가 다중 컬럼 서브쿼리입니다.

○ 메인 쿼리의 WHERE절에 다중 컬럼 서브쿼리를 작성할 때 다음의 사항이 지켜져야 합니다.

- 메인 쿼리의 WHERE절에서 **연산자의 왼쪽에 있는 컬럼들은 괄호로 감싸져야 합니다.**
- 메인 쿼리의 WHERE절에서 **연산자의 왼쪽에 있는 컬럼의 수 및 순서에 따른 데이터유형이, 연산자의 오른편에 있는 다중 컬럼 서브쿼리의 SELECT절에 있는 컬럼의 수와 순서에 따른 데이터유형과 일치**해야 합니다.
- **메인 쿼리의 WHERE절에 다중 컬럼 서브쿼리가 사용된 경우, 연산자는 = 연산자 또는 IN 연산자만 사용 가능합니다.**
  - ☞ = 연산자가 사용되면, 다중 컬럼 서브쿼리는 반드시 1개의 행에서 구성된 레코드만 반환해야 합니다.
  - ☞ IN 연산자가 사용되면, 다중 컬럼 서브쿼리는 1개 이상의 행에서 구성된 레코드를 반환할 수 있습니다.

○ 다중 컬럼 서브쿼리와 비교하여, 메인 쿼리의 WHERE절에 1개의 컬럼으로 구성된 레코드가 반환되도록 작성된 일반적인 서브쿼리를, 단일 컬럼 서브쿼리라고 합니다.

■ 다중 컬럼 서브쿼리가 WHERE 절에 사용된 경우, WHERE절에서 쌍-비교(Pairwise Comparison) 방식을 이용하여 비교됩니다.

1-2-1. WHERE절에서, 다중 컬럼 서브쿼리를 사용하지 않고, 단일 컬럼 서브쿼리를 사용한 경우.

■ HR.EMPLOYEES 테이블을 이용하여 성(LAST_NAME컬럼)이 'Abel'인 직원과, 같은 부서(DEPARTMENT_ID컬럼)에서 근무하고, 동일한 직책(JOB_ID컬럼)을 수행하는 다른 직원들의 사번(EMPLOYEE_ID컬럼), 성(LAST_NMAE컬럼), 월급(SALARY컬럼)을 표시하시오. 단 성이 'Abel'인 직원은 한 명만 있습니다.

```
SQL> SELECT employee_id, last_name, salary
     FROM hr.employees
     WHERE department_id = (SELECT department_id
                            FROM hr.employees
                            WHERE last_name = 'Abel')
     AND job_id = (SELECT job_id
                   FROM hr.employees
                   WHERE last_name = 'Abel')
     AND last_name <> 'Abel'
     ORDER BY 2 ;

EMPLOYEE_ID LAST_NAME                 SALARY
----------- ------------------------- ----------
        166 Ande                        6400
        167 Banda                       6200
        172 Bates                       7300
        151 Bernstein                   9500
        169 Bloom                      10000
...

 28개의 행이 선택됨
```

☞ 위의 문장에 적힌 2개의 서브쿼리를 확인하면, **테이블이름이 동일하고, 행을 추출하는 조건도 동일합니다. 즉, 서브쿼리에 의하여 반환된 값들이 동일한 행으로부터 추출되었습니다.**

☞ 위의 문장에서 메인 쿼리의 WHERE절에 2개의 조건이 AND 연산자로 기술되어 있으므로, 2개의 조건이 모두 만족해야만 데이터를 추출할 행이 선택됩니다. 또한 조건에 서브쿼리가 포함되어 있습니다.
위와 같이 WHERE절에 하나의 컬럼 값과 비교되도록 서브쿼리를 작성하는 방법은 일반적으로 많이 사용되는 방법입니다.

☞ 2개의 서브쿼리가 각각 값들을 반환하면, 최종 결과를 표시할 행을 선택하기 위하여, HR.EMPLOYEES 테이블의 행 마다, 행에 있는 DEPARTMENT_ID 컬럼과 JOB_ID 컬럼의 값이, 서브쿼리로부터 반환된 2개의 값과 각각 비교되어, 두 조건이 모두 만족하는 경우에만 행이 선택됩니다.

○ 서브쿼리가 반환하는 하나의 컬럼으로 된 값과 테이블의 행에서 가져온 하나의 컬럼 값이 비교되는 2개 이상의 조건이 AND 연산자를 이용하여 WHERE절에 기술되면, **테이블의 행에서 서브쿼리로부터 반환된 값들을 개별적으로 비교하는 방법이 사용되며, 이런 방식을 비쌍-비교(Non-pairwise comparison) 라고 합니다.**

1-2-2. WHERE절에서, 다중 컬럼 서브쿼리를 사용한 경우.

■ 앞에서 WHERE절에 AND 연산자로 연결된 2개의 조건에 서브쿼리가 각각 사용되었고, 이들 서브쿼리들에 의하여 반환된 값은 **동일한 행으로부터 추출된 경우, 오라클에서는 다중 컬럼 서브쿼리를 사용하여, 2개가 아닌 하나의 조건으로 WHERE절을 작성할 수 있습니다.**

■ 앞에서 실습한 동일한 요구사항의 SELECT문을 WHERE절에 다중 컬럼 서브쿼리를 사용하여 작성합니다.

■ HR.EMPLOYEES 테이블을 이용하여 성(LAST_NAME컬럼)이 'Abel'인 직원과, 같은 부서(DEPARTMENT_ID컬럼)에서 근무하고, 동일한 직책(JOB_ID컬럼)을 수행하는, 다른 직원들의 사번(EMPLOYEE_ID컬럼), 성(LAST_NMAE컬럼), 월급(SALARY컬럼), 부서 및 직책을 표시하시오. 단 성이 'Abel'인 직원은 제외합니다.

```
SQL> SELECT employee_id, last_name, salary, department_id, job_id
     FROM hr.employees
     WHERE (department_id, job_id) = (SELECT department_id, job_id
                                      FROM  hr.employees
                                      WHERE last_name = 'Abel')
     AND last_name <> 'Abel'
     ORDER BY 2 ;

EMPLOYEE_ID LAST_NAME                 SALARY DEPARTMENT_ID JOB_ID
----------- ------------------------- ------ ------------- ----------
        166 Ande                        6400            80 SA_REP
        167 Banda                       6200            80 SA_REP
        172 Bates                       7300            80 SA_REP
        151 Bernstein                   9500            80 SA_REP
        169 Bloom                      10000            80 SA_REP
...

 28개의 행이 선택됨
```

☞ 서브쿼리에 의하여 2개의 컬럼 값으로 구성된 1개의 레코드가 반환되면, 최종 결과를 표시할 행을 선택하기 위하여, HR.EMPLOYEES 테이블의 행 마다, 행의 DEPARTMENT_ID 컬럼과 JOB_ID 컬럼의 값들이, 1개의 레코드처럼 쌍으로 묶여서 서브쿼리가 반환한 레코드와 한 번에 비교됩니다.

○ 다음은 위의 SELET문의 이해를 돕기 위하여, WHERE절의 다중 열 서브쿼리가 반환하는 레코드를 확인한 결과입니다.

```
SQL> SELECT department_id, job_id
     FROM hr.employees
     WHERE last_name = 'Abel' ;

DEPARTMENT_ID JOB_ID
------------- ----------
           80 SA_REP
```

1-3. 쌍-비교 (Pairwise Comparison)와 비쌍-비교(Non-pairwise Comparison)

1-3-1. 쌍-비교 (Pairwise Comparison)

■ WHERE절에 다중 컬럼 서브쿼리가 사용되면, **행의 컬럼들로 구성된 값들의 조합이 하나의 레코드처럼 쌍으로 묶여서, 서브쿼리가 반환하는 레코드와 한 번에 비교됩니다. 이런 방식을 쌍-비교(Pairwise comparison) 라고 합니다.**

■ HR.EMPLOYEES 테이블을 이용하여 이름(FIRST_NAME컬럼)이 "John"인 사원(들)과 동일한 관리자(MANAGER_ID컬럼)의 감독을 받고 동일한 부서(DEPARTMENT_ID컬럼)에서 근무하는, 다른 사원들의 사번(EMPLOYEE_ID컬럼), 이름, 관리자, 부서, 월급(SALARY컬럼) 정보를 표시하시오. 단, 다중 컬럼 서브쿼리를 이용하여 WHERE절을 작성하시오.

```
SQL> SELECT employee_id, first_name, manager_id, department_id, salary
     FROM hr.employees
     WHERE (manager_id, department_id) IN (SELECT manager_id, department_id
                                           FROM hr.employees
                                           WHERE first_name = 'John')
     AND first_name <> 'John'
     ORDER BY 3,4,1;
```

108 100
123 50
100 80

EMPLOYEE_ID	FIRST_NAME	MANAGER_ID	DEPARTMENT_ID	SALARY
146	Karen	100	80	13500
147	Alberto	100	80	12000
148	Gerald	100	80	11000
149	Eleni	100	80	10500
109	Daniel	108	100	9000
111	Ismael	108	100	7700
112	Jose Manuel	108	100	7800
113	Luis	108	100	6900
137	Renske	123	50	3600
138	Stephen	123	50	3200
140	Joshua	123	50	2500
192	Sarah	123	50	4000
193	Britney	123	50	3900
194	Samuel	123	50	3200
195	Vance	123	50	2800

15개의 행이 선택됨

☞ 다중 열 서브쿼리가 실행되어, FIRST_NAME 컬럼이 'John'인 행(들)로부터 MANAGER_ID 컬럼 및 DEPARTMENT_ID 컬럼으로 구성되는 (3개의) 레코드들이 반환되면, 최종 결과를 표시할 행을 선택하기 위하여, HR.EMPLOYEES 테이블에 있는 각 행의 MANAGER_ID 컬럼과 DEPARTMENT_ID 컬럼의 값들의 조합이, 하나의 레코드처럼 쌍으로 묶여서, 서브쿼리가 반환하는 (3개의) 레코드들 중 하나와 와 같은지가, 값들의 조합 단위로 비교됩니다.

○ 다음은 WHERE절의 다중 열 서브쿼리가 반환하는 레코드를 확인한 결과입니다.

```
SQL> SELECT manager_id, department_id
     FROM hr.employees
     WHERE first_name = 'John'  ;

MANAGER_ID DEPARTMENT_ID
---------- -------------
       108           100
       123            50
       100            80
```

☞ 이름이 'John' 직원이 3명이 근무하는데 그들의 근무부서와 관리자의 조합을 확인하면, 위와 같은 같습니다.

○ 위의 다중 컬럼 서브쿼리가 반환하는 레코드들을 가지고, 메인 쿼리의 WHERE 절을 통해 행이 선택되는 의미는 아래와 같은 3가지의 경우로 정리됩니다.

```
manager_id = 108 이고, department_id = 100 또는
manager_id = 123 이고, department_id = 50  또는
manager_id = 100 이고, department_id = 80
```

☞ 즉, WHERE절에 의하여 행을 선택할 때, 테이블에 있는 행의 MANAGER_ID 컬럼과 DEPARTMENT_ID 컬럼의 값들이, 하나의 레코드처럼 쌍으로 묶여서, 서브쿼리가 반환한 위의 3 개의 레코드 중 하나와 같은지가 값들의 조합 단위로 비교됩니다.

1-3-2. 비쌍-비교 (Non-pairwise Comparison)

■ 서브쿼리가 반환하는 하나의 컬럼으로 된 값과 테이블의 행에서 가져온 하나의 컬럼 값이 비교되는 2개 이상의 조건이 AND 연산자를 이용하여 WHERE절에 기술되면, 테이블의 행에서 서브쿼리들이 반환하는 값들을 개별적으로 비교하는 방법이 사용되며, 이런 방식을 비쌍-비교(Non-pairwise comparison) 라고 합니다.

■ 앞의 쌍-비교 설명에서 실습한 SELECT문에서, 다중 컬럼 서브쿼리가 포함된 WHERE절을, 단일 컬럼 서브쿼리가 포함된 2개의 조건으로 변경한 다음, 반환 결과를 비교하여, 차이가 발생되는 이유를 확인해 봅니다.

○ 다음은 이름(FIRST_NAME)이 'John' 인 직원의 관리자 및 근무부서를 확인한 결과입니다.

```
SQL> SELECT manager_id, department_id
     FROM hr.employees                 -- HR.EMPLOYEES 테이블로부터 FIRST_NAME 의 값이 'John'인 직원은
     WHERE first_name = 'John'  ;      -- 3명 밖에 없으며, 이들 각각의 관리자(MANAGER_ID)와
                                       -- 근무부서(DEPARTMENT_ID)는 위와 같이 3가지 조합만 있습니다.
MANAGER_ID DEPARTMENT_ID
---------- -------------
       108           100
       123            50
       100            80
```

■ HR.EMPLOYEES 테이블을 이용하여 이름(FIRST_NAME컬럼)이 "John"인 사원(들)과 동일한 관리자(MANAGER_ID컬럼)의 감독을 받고 동일한 부서(DEPARTMENT_ID컬럼)에서 근무하는, 다른 사원들의 사번(EMPLOYEE_ID컬럼), 이름, 관리자, 부서, 월급(SALARY컬럼) 정보를 표시하시오. 단, 다중 컬럼 서브쿼리를 이용하지 말고, WHERE절을 작성하시오.

```
SQL> SELECT employee_id, first_name, manager_id, department_id, salary
     FROM hr.employees
     WHERE manager_id IN (SELECT manager_id              -- 이 서브쿼리가 반환하는 값은,
                          FROM hr.employees              -- 108, 123, 100 입니다.
                          WHERE first_name = 'John')
     AND department_id IN (SELECT department_id          -- 이 서브쿼리로 반환되는 값은,
                           FROM hr.employees             -- 100, 50, 80 입니다.
                           WHERE first_name = 'John')
     AND first_name <> 'John'
     ORDER BY 3,4,1 ;

EMPLOYEE_ID FIRST_NAME           MANAGER_ID DEPARTMENT_ID     SALARY
----------- -------------------- ---------- ------------- ----------
        120 Matthew                     100            50       8000
        121 Adam                        100            50       8200
        122 Payam                       100            50       7900
        123 Shanta                      100            50       6500
        124 Kevin                       100            50       5800
        146 Karen                       100            80      13500
        147 Alberto                     100            80      12000
        148 Gerald                      100            80      11000
        149 Eleni                       100            80      10500
        109 Daniel                      108           100       9000
        111 Ismael                      108           100       7700
        112 Jose Manuel                 108           100       7800
        113 Luis                        108           100       6900
        137 Renske                      123            50       3600
        138 Stephen                     123            50       3200
        140 Joshua                      123            50       2500
        192 Sarah                       123            50       4000
        193 Britney                     123            50       3900
        194 Samuel                      123            50       3200
        195 Vance                       123            50       2800

 20개의 행이 선택됨
```

☞ WHERE절에 다중 컬럼 서브쿼리를 사용한 경우와 다르게 5개의 행(붉게 표시됨)이 더 표시됩니다.

■ 다중 컬럼 서브쿼리가 사용된 하나의 조건을, 단일 컬럼 서브쿼리를 사용하는 2개의 조건으로 작성했을 때, 결과에 차이가 발생하는 이유는 다음과 같습니다.

○ 앞의 WHERE절에서 각각의 서브쿼리가 반환한 값으로 WHERE절을 다시 작성해 보면 다음과 같이 됩니다.

```
WHERE manager_id IN (108, 123, 100)
AND department_id IN (100, 50, 80)
```

○ 위의 WHERE절을, MANAGER_ID 컬럼 값과 DEPARTMENT_ID 컬럼 값의 조합으로 의미를 정리하면, 다음과 같습니다.

- manager_id = 108 이고, department_id = 100 또는
- manager_id = 123 이고 department_id = 50 또는
- manager_id = 100 이고 department_id = 80 또는
- manager_id = 108 이고, department_id = 50 또는
- manager_id = 108 이고, department_id = 80 또는
- manager_id = 123 이고 department_id = 100 또는
- manager_id = 123 이고 department_id = 80 또는
- manager_id = 100 이고 department_id = 100 또는
- manager_id = 100 이고 department_id = 50 -- 이 조합이 쿼리 결과에 추가적으로 표시되었습니다.

○ 위에서 붉게 표시된 조합은, 현재 이름이 'John'인 직원에 해당하는 관리자 및 근무하는 부서의 조합이 아닙니다. 문장의 IN 연산자와 AND 연산자에 의하여, 고려되는 추가된 조합입니다. 따라서, 앞의 SELECT문으로 표시된 결과에 추가적으로 표시된 MANAGER_ID가 100이고, DEPARTMENT_ID가 50인 5개의 행들은 이름이 'John'인 직원에 해당하는 실제 조합이 아닙니다.

○ 이와 같이 비쌍-비교 방식이 IN 연산자가 사용된 WHERE절에서 수행될 경우, 보다 많은 행들이 더 선택되어, 결과가 표시될 수 있습니다.

○ 메인쿼리의 WHERE절 또는 HAVING 절에 IN 연산자가 사용된 경우, 메인쿼리의 하나의 조건에 다중 컬럼 서브쿼리가 사용된 것과 메인쿼리의 WHERE절 또는 HAVING절에 2개 이상의 조건으로 단일 컬럼 서브쿼리가 사용된 것은 같지 않습니다.

○ 다중 컬럼 서브쿼리 앞에 = 연산자가 사용된 WHERE 절을, 단일 컬럼 서브쿼리를 이용하여 2 개의 조건으로 구성된 WHERE절로 변경한 경우에는, 위와 같은 표시 결과의 차이가 없습니다.

[참고] 다중 열 서브쿼리를 이용한 UPDATE 수행.

```
SQL> UPDATE hr.employees    -- 이 문장은 서브쿼리가 한번 실행되었습니다.
     SET (job_id, salary) = (SELECT job_id, salary
                             FROM hr.employees
                             WHERE employee_id = 205)
     WHERE employee_id = 114;

1 행 이(가) 업데이트되었습니다.


SQL> ROLLBACK ; -- 이 후 실습을 위하여 트랜잭션을 롤백 합니다.

롤백 완료.
```

☞ 오라클에서는 다중 열 서브쿼리를 UPDATE문의 SET 절에도 사용할 수 있습니다.

☞ 위의 UPDATE문은 다음의 UPDATE문과, 데이터에 대하여 동일한 데이터의 갱신을 수행합니다.

```
UPDATE hr.employees    -- 이 문장은 동일한 테이블의 행에 대하여 서브쿼리가 2번 실행되었습니다.
SET job_id = (SELECT job_id FROM hr.employees WHERE employee_id = 205),
     salary = (SELECT salary FROM hr.employees WHERE employee_id = 205)
WHERE employee_id = 114;
```

1-4. 스칼라 서브쿼리(Scalar Subquery) 표현식의 사용.

■ 스칼라 서브쿼리 표현식은 **하나의 행으로부터 오직 하나의 컬럼 값을 반환하는 서브쿼리**입니다.

■ 스칼라 서브쿼리 표현식은 다음의 부분에서 사용할 수 있습니다.

○ DECODE 및 CASE의 조건 및 표현식 부분

○ GROUP BY를 제외한 SELECT문의 모든 절

○ UPDATE문의 SET절 및 WHERE절

■ HR.EMPLOYEES 테이블로부터, 전체 직원에 대하여, 직원ID(EMPLOYEE_ID 열) 및 성(LAST_NAME 열)을 표시하되, 직원이 근무하는 부서(DEPARTMENT_ID 열)의 위치ID에 따라서 다음의 리터럴 문자열이 같이 표시되도록 하시오. 단, 부서의 위치ID는 HR.DEPARTMENTS 테이블의 LOCATION_ID 열을 통해 확인할 수 있습니다.

○ 부서의 위치ID가 1800일 때는 'Canada', 1800이 아닐 때는 'USA' 리터럴 문자열이 표시되도록 하시오.

```
SQL> SELECT employee_id, last_name, (CASE department_id
                                     WHEN (SELECT department_id      -- 스칼라서브쿼리에 의하여
                                           FROM hr.departments       -- 20 이 반환됩니다.
                                           WHERE location_id = 1800)
                                     THEN 'Canada'  ELSE 'USA' END ) AS "LOCATION"
    FROM hr.employees;

EMPLOYEE_ID LAST_NAME                 LOCATI
----------- ------------------------- ------
        198 OConnell                  USA
        199 Grant                     USA
        200 Whalen                    USA
        201 Hartstein                 Canada
        202 Fay                       Canada
...

 107개의 행이 선택됨
```

☞ 위의 문장에서 먼저 CASE 표현식에 사용된 스칼라 서브쿼리에 의하여, 위치ID가 1800인 부서의 아이디 값인 20이 반환됩니다. 이 값을 이용하여, 메인 쿼리는 각 직원에 대하여, 직원ID 및 성과 함께, 직원의 부서ID가 20일 경우는 Canada 값이, 직원의 부서ID가 20이 아닐 때는 USA 값이 표시됩니다.

☞ 위의 SELECT문에서 CASE 표현식 대신 DECODE() 함수를 사용하여 다시 작성하면, 다음과 같습니다.

```
SQL> SELECT employee_id, last_name,
           DECODE(department_id,
                  (SELECT department_id FROM hr.departments WHERE location_id = 1800),
                  'Canada',
                  'USA') AS  "LOCATION"
    FROM hr.employees ;
```

1-5. 상관관계 서브쿼리 (Correlated Subquery)를 사용한 데이터 조회.

모든 데이터베이스 제품에서 지원합니다.

■ 메인 쿼리의 WHERE절에 포함된 서브쿼리가, 메인 쿼리에 있는 테이블의 컬럼을 참조하는 경우, 이 서브쿼리를 상관관계 서브쿼리 (Correlated Subquery)라고 합니다.

■ 메인 쿼리에서 고려되는 각 후보 행에 대해, 서브쿼리가 다른 결과 또는 결과 집합을 반환하여 해당 메인 쿼리 행의 선택유무를 결정해야 할 때, 상관관계 서브쿼리 형식이 사용됩니다.

■ 메인 쿼리에서 상관관계 서브쿼리가 사용되면, 메인 쿼리의 후보 행 수만큼, 서브쿼리가 반복적으로 처리됩니다.

■ 기본문법1: 이 방법의 상관관계 서브쿼리는 대부분의 데이터베이스 제품에서 사용할 수 있습니다.

```
-- 상관관계 서브쿼리로, 단일 컬럼 서브쿼리를 사용.
SELECT A.컬럼1, A.컬럼2, ...
FROM 스키마이름.테이블1 A
WHERE A.컬럼1 연산자 (SELECT B.column1              -- 서브쿼리의 WHERE 절에 메인쿼리의 테이블의 컬럼이
                      FROM 스키마이름2.테이블2 B    -- 참조(A.표현식2)되고 있습니다.
                      WHERE B.표현식1 = A.표현식2);
```

■ 기본문법2: 이 방법의 상관관계 서브쿼리는 다른 회사의 데이터베이스 제품에서는 수행되지 않을 수도 있습니다.

```
-- 상관관계 서브쿼리로, 다중 컬럼 서브쿼리를 사용.
SELECT A.컬럼1, A.컬럼2, ...
FROM 스키마이름.테이블1 A
WHERE (A.컬럼1, A.컬럼2, ...) 연산자 (SELECT B.column1, B.column2,... -- 서브쿼리의 WHERE 절에
                                    FROM 스키마이름2.테이블2 B       -- 메인쿼리의 테이블의 컬럼이
                                    WHERE B.표현식1 = A.표현식2);    -- 참조(A.표현식2)되고 있습니다.
```

○ [기본문법1] 방법에서는 ANY 및 ALL 연산자도 메인 쿼리의 WHERE절에서 사용될 수 있습니다.

○ [기본문법2] 방법에서 사용된 다중 컬럼 서브쿼리는 다음의 사항을 지켜야 합니다.

- 메인 쿼리의 WHERE절에서 연산자의 왼쪽에 있는 컬럼들은 괄호로 감싸져야 하며, 이들 컬럼의 수 및 순서에 따른 데이터유형이 연산자의 오른편에 있는 다중 열 서브쿼리의 SELECT절에 있는 컬럼의 수와 순서에 따른 데이터유형이 일치해야 합니다.
- 메인 쿼리의 WHERE절에 다중 열 서브쿼리가 사용된 경우, 연산자는 = 연산자 또는 IN 연산자만 사용 가능합니다.

○ WHERE절에 상관관계 서브쿼리가 사용된 메인 쿼리가 처리되는 과정은 다음과 같습니다.

1) 메인 쿼리의 테이블에 있는 행(후보 행, Candidate Row) 하나로부터 서브쿼리로 값이 전달됩니다.
2) 전달받은 값을 이용하여, 서브쿼리가 실행되고, 메인 쿼리로 전달할 값이 구해집니다.
3) 서브쿼리로부터 전달받은 값을 이용하여, 메인 쿼리의 후보 행에 대한 WHERE절의 조건이 평가된 후, 후보 행이 선택되거나 선택되지 않습니다.
4) 1단계부터 3단계까지를 메인 쿼리의 각 행에 대하여 계속 수행합니다.

■ HR.EMPLOYEES 테이블의 데이터를 이용하여, 자신의 부서의 평균급여보다 많은 급여를 받는 사원을 모두 찾아, 직원ID(EMPLOYEE_ID), 성(LAST_NAME), 급여(SALARY), 부서ID(DEPARTMENT_ID), 직책ID(JOB_ID)를 표시하시오.

```
SQL> SELECT a.employee_id, a.last_name, a.salary, a.department_id, a.job_id
     FROM hr.employees a
     WHERE a.salary > (SELECT AVG(b.salary)              -- 메인쿼리의 행마다 서브쿼리 실행 후,
                       FROM hr.employees b               -- 메인쿼리의 행이 평가됩니다.
                       WHERE b.department_id = a.department_id)
     ORDER BY 4;

EMPLOYEE_ID LAST_NAME           SALARY DEPARTMENT_ID JOB_ID
----------- --------------- ---------- ------------- ----------
        201 Hartstein            13000            20 MK_MAN
        114 Raphaely             11000            30 PU_MAN
        120 Weiss                 8000            50 ST_MAN
        121 Fripp                 8200            50 ST_MAN
        122 Kaufling              7900            50 ST_MAN
...

 38개의 행이 선택됨
```

☞ 위의 SELECT문에 있는 WHERE절은 다음의 과정을 통해 처리됩니다(아래에 표시된 데이터들을 이용합니다).

메인 쿼리의 후보 행을 나타내기 위한 HR.EMPLOYEES 테이블의 일부 데이터

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
200	Whalen	4400	10
201	Hartstein	13000	20
202	Fay	6000	20
114	Raphaely	11000	30
115	Khoo	3100	30
116	Baida	2900	30
117	Tobias	2800	30
118	Himuro	2600	30
119	Colmenares	2500	30

...

107개의 행이 선택됨

서브쿼리가 실행되는 것을 나타내기 위한 HR.EMPLOYEES 테이블의 일부 데이터

EMPLOYEE_ID	SALARY	DEPARTMENT_ID
200	4400	10
201	13000	20
202	6000	20
114	11000	30
115	3100	30
116	2900	30
117	2800	30
118	2600	30
119	2500	30

...

107개의 행이 선택됨

1) 메인 쿼리의 테이블에서, 직원ID가 201인 후보 행의 부서ID 값인 20이 서브쿼리로 전달됩니다.

2) 전달받은 20 값을 이용하여, 서브쿼리가 실행되어, 부서ID가 20인 행을 찾아 평균입금(9500)을 구합니다

3) 서브쿼리로부터 전달받은 평균임금 9500 값을 이용하여, 메인 쿼리의 직원ID가 201인 후보 행에 대하여, 해당 행의 월급인 13000이 평균임금 9500보다 큰지 비교하여, 조건을 만족하므로 직원ID가 201인 행을 선택합니다.

4) 1단계부터 3단계까지를 메인 쿼리의 107 개 행에 대하여 수행합니다.

☞ 앞의 실습에서 작성된 SELECT문을 상관관계 서브쿼리를 사용하지 않고, 인라인 뷰를 이용하여 다음처럼 작성할 수도 있습니다.

```
SQL> SELECT a.employee_id, a.last_name, a.salary, a.department_id, a.job_id
     FROM hr.employees a INNER JOIN (SELECT department_id, avg(salary) AS "DEPT_AVG"
                                     FROM hr.employees
                                     GROUP BY department_id) b
     ON a.department_id = b.department_id
     WHERE a.salary > b.dept_avg
     ORDER BY 4;
```

○ 위의 문장은 인라인 뷰를 HR.EMPLOYEES 테이블과 조인하여, 직원의 급여와 인라인 뷰에서 구한 부서의 평균급여를 비교하여 행을 선택하는 방법을 사용했습니다

■ 아래에 표시된 HR.EMPLOYEES 테이블과 HR.JOB_HISTORY 테이블과 관련된 데이터를 이용하여, 입사 후, 직무가 2번 이상 바뀐 직원에 대한 현재의 직원ID(EMPLOYEE_ID), 성(LAST_NAME), 급여(SALARY), 직무ID(JOB_ID)를 표시하시오.

```
SQL> SELECT e.employee_id, e.last_name, e.salary, e.job_id
     FROM hr.employees e
     WHERE  2 <= (SELECT COUNT(*)
                  FROM hr.job_history j
                  WHERE j.employee_id = e.employee_id);

EMPLOYEE_ID LAST_NAME            SALARY JOB_ID
----------- --------------- ---------- ----------
        200 Whalen                4400 AD_ASST
        101 Kochhar              17000 AD_VP
        176 Taylor                8600 SA_REP
```

☞ 위의 SELECT문을 작성하기 위하여 다음의 테이블을 사용합니다.

○ HR.EMPLOYEES 테이블에는 현재 근무하는 직원에 대한 현재의 직원 정보가 저장되어 있습니다.

○ HR.JOB_HISTORY 테이블에는 현재 근무하는 직원에 대한 과거의 직무와 관련된 정보가 저장된 테이블입니다.

○ 요구사항과 관련된 각 테이블의 컬럼이름 및 확인할 데이터는 다음과 같습니다.

메인 쿼리의 후보 행을 나타내기 위한 HR.EMPLOYEES 테이블의 일부 데이터

```
EMPLOYEE_ID LAST_NAME       JOB_ID     DEPARTMENT_ID
----------- --------------- ---------- -------------
        101 Kochhar         AD_VP                 90
        102 De Haan         AD_VP                 90
        103 Hunold          IT_PROG               60
        104 Ernst           IT_PROG               60
        114 Raphaely        PU_MAN                30
        122 Kaufling        ST_MAN                50
        176 Taylor          SA_REP                80
        200 Whalen          AD_ASST               10
        201 Hartstein       MK_MAN                20
...

 107개의 행이 선택됨
```

서브쿼리가 실행되는 것을 나타내기 위한 HR.JOB_HISTORY 테이블의 일부 데이터

```
EMPLOYEE_ID START_DATE JOB_ID     DEPARTMENT_ID
----------- ---------- ---------- -------------
        101 97/09/21   AC_ACCOUNT           110
        101 01/10/28   AC_MGR               110
        102 01/01/13   IT_PROG                60
        114 06/03/24   ST_CLERK               50
        122 07/01/01   ST_CLERK               50
        176 06/03/24   SA_REP                 80
        176 07/01/01   SA_MAN                 80
        200 95/09/17   AD_ASST                90
        200 02/07/01   AC_ACCOUNT             90
        201 04/02/17   MK_REP                 20

 10개의 행이 선택됨
```

☞ 위의 SELECT문에 있는 WHERE절은 다음의 과정을 통해 처리됩니다.

1) HR.EMPLOYEES 테이블에서, 직원ID가 101인 후보 행의 직원ID인 101 값이 서브쿼리로 전달됩니다.

2) 전달받은 101 값을 이용하여, 서브쿼리가 실행되어, HR.JOB_HISTORY 테이블에서 행을 찾아 개수(2)를 구합니다

3) 서브쿼리로부터 전달받은 행의 개수 2 값을 이용하여, 전달받은 2 값이 상수 2 보다 크거나 같은지 확인합니다. 조건을 만족하므로, 처리 중인 후보 행(직원ID가 101인 행)을 선택합니다.

4) 1단계부터 3단계까지를 메인 쿼리의 107 개 행에 대하여 수행합니다.

☞ 앞의 실습에서 작성된 SELECT문을 상관관계 서브쿼리를 사용하지 않고, 다음처럼 작성할 수도 있습니다.

```
SQL> SELECT e.employee_id, e.last_name, e.salary, e.job_id
     FROM hr.employees e
     WHERE e.employee_id IN (SELECT j.employee_id
                             FROM hr.job_history j
                             GROUP BY j.employee_id
                             HAVING count(*) >= 2) ;
```

• 위의 문장은 메인쿼리의 WHERE절에서 일반적인 서브쿼리를 사용하여, 두 번 이상 직무가 바뀐 직원의 사번을 구하여, 이를 HR.EMPLOYEES 테이블의 사원ID 값과 비교하여 행을 선택하는 방법을 사용했습니다.

○ 원하는 결과를 나타내는 SQL문은 경우에 따라서, 여러 가지 문장이 가능합니다. 따라서, 여러 가지 방법의 SQL문 중에, 가장 효과적인 문장을 최종적으로 선택하여 사용할 필요가 있습니다. 경력이 쌓이고, SQL문을 많이 연습 및 작성해보면서, SQL-튜닝에 대하여 학습이 되면, 적합한 문장을 선택할 능력이 생깁니다. 지금은 SQL문의 기본 문법 및 다양한 문장에 대하여 학습합니다.

1-6. 상관관계 서브쿼리 (Correlated Subquery)를 사용한 데이터 변경 및 삭제

■ 테이블의 각 행에 대하여 서브쿼리가 처리한 값을 기반으로 확인한 뒤, 행을 수정하거나 삭제하길 원할 때, DELETE문의 WHERE 절과 UPDATE문에서 상관관계 서브쿼리를 이용할 수 있습니다.

■ 기본문법: 아래에서 A 와 B 는 각각 테이블 별칭입니다.

```
-- 상관관계 서브쿼리를 이용한 UPDATE문
UPDATE 스키마이름.테이블이름1 A        -- 각 행마다 서브쿼리의 고유한 결과를 통해 검사한 후, 값이 수정됩니다.
SET A.컬럼1 = (SELECT 표현식
               FROM 스키마이름2.테이블이름2 B
               WHERE B.컬럼 = A.컬럼);

-- 상관관계 서브쿼리를 이용한 DELETE문
DELETE FROM 스키마.테이블1 A          -- 각 행마다 서브쿼리의 고유한 결과를 통해 검사한 후, 삭제가 수행됩니다.
WHERE A.컬럼 연산자 (SELECT 표현식
                     FROM 스키마.테이블이름2 B
                     WHERE B.컬럼 = A.컬럼);
```

1-6-1. 상관관계 서브쿼리를 이용한 테이블의 데이터 수정.

■ 예를 들어 개발자가 프로그램을 테스트하기 위하여 사용하는 '프로그램 테스트용 테이블'에, 컬럼을 추가하고, 추가된 컬럼을 다른 테이블의 데이터를 이용하여 UPDATE 하고자 할 때, 상관관계 서브쿼리를 이용한 UPDATE 문장을 사용할 수 있습니다.

■ 다음에 나오는 일련의 실습을 통해 상관관계 서브쿼리를 이용한 UPDATE 문을 사용하는 방법을 확인합니다.

1> SQL*Developer를 이용하여, HR 계정으로 데이터베이스에 접속합니다.

2> HR.EMP_APP_TEST 테이블을 HR.EMPLOYEES 테이블에 대한 서브쿼리를 이용하여 생성합니다. 단, 생성되는 테이블에는 EMPLOYEE_ID, LAST_NAME, SALARY, DEPARTMENT_ID 컬럼만 복사해서 구성되도록 합니다.

```
SQL> CREATE TABLE HR.EMP_APP_TEST
     AS SELECT employee_id, last_name, salary, department_id
        FROM hr.employees
        ORDER BY 1 ;

Table HR.EMP_APP_TEST이(가) 생성되었습니다.
```

3> 생성한 HR.EMP_APP_TEST 테이블에, DEPARTMENT_NAME 컬럼을 추가합니다.

```
SQL> ALTER TABLE HR.EMP_APP_TEST
     ADD (DEPARTMENT_NAME VARCHAR2(30)) ;

Table HR.EMP_APP_TEST이(가) 변경되었습니다.
```

☞ 컬럼 추가는 이 후 단원에서 자세히 설명합니다.

4> HR.EMP_APP_TEST 테이블의 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.emp_app_test;

EMPLOYEE_ID LAST_NAME           SALARY DEPARTMENT_ID DEPARTMENT_NAME
----------- --------------- ---------- ------------- ------------------------------
        100 King                24000            90                                -- 추가된 DEPARTMENT_NAME
        101 Kochhar             17000            90                                -- 컬럼은 NULL 입니다.
        102 De Haan             17000            90                                -- 적당한 값으로 UPDATE
        103 Hunold               9000            60                                -- 해야 합니다.
        104 Ernst                6000            60
...

 107개의 행이 선택됨
```

5> 부서이름의 원본 데이터가 저장된 HR.DEPARTMENTS 테이블의 데이터를 확인합니다.

```
SQL> SELECT department_id, department_name
     FROM hr.departments ;

DEPARTMENT_ID DEPARTMENT_NAME
------------- ------------------------------
           10 Administration
           20 Marketing
           30 Purchasing
           40 Human Resources
           50 Shipping
...

 27개의 행이 선택됨
```

6> HR.EMP_APP_TEST 테이블의 NULL 상태인 DEPARTMENT_NAME 컬럼을 HR.DEPARTMENTS 테이블의 DEPARTMENT_NAME 컬럼의 데이터로 적절히 수정합니다. 이 때, HR.EMP_APP_TEST 테이블의 각 행마다 고유한 부서ID에 해당하는 부서이름으로 수정되어야 합니다.

```
SQL> UPDATE hr.emp_app_test a
     SET a.department_name = (SELECT b.department_name
                              FROM hr.departments b
                              WHERE b.department_id = a.department_id) ;

107개 행 이(가) 업데이트되었습니다.

SQL> COMMIT ;

커밋 완료.
```

☞ 각 행마다 고유한 부서ID에 해당하는 부서이름을 가져오기 위하여 상관관계 서브쿼리가 이용되었습니다.

☞ 앞의 상관관계 서브쿼리를 이용한 UPDATE문은 다음의 과정을 통해 처리됩니다.

1) HR.EMP_APP_TEST 테이블에서, 하나의 후보 행에 있는 직원의 부서ID 값이 값이 서브쿼리로 전달됩니다.

2) 전달된 부서ID 값을 이용하여 서브쿼리가 실행되고, HR.DEPARTMENTS 테이블에서 행을 찾아 부서ID에 해당하는 부서이름을 구합니다

3) 서브쿼리로부터 전달받은 부서이름으로 후보 행의 부서이름 컬럼의 값을 업데이트 합니다.

4) 1단계부터 3단계까지를 HR.EMP_APP_TEST 테이블의 각 행에 대하여 반복 수행하여, HR.EMP_APP_TEST 테이블의 각 행의 부서이름을 부서ID에 해당하는 부서이름으로 업데이트 합니다.

☞ 이 과정을 이해하면, 서브쿼리로부터 부서이름은 반드시 하나가 반환되어야 하며, 따라서, 서브쿼리의 WHERE절에서 HR.DEPARTMENTS 테이블의 DEPARTMENT_ID 값에 대하여 반드시 한 행만 존재해야 합니다.
서브쿼리에 있는 WHERE절의 조건에서 명시된 DEPARTMENT_ID 컬럼에는 HR.DEPARTMENTS 테이블에 대한 PRIMARY KEY 제약조건이 정의되어 있으므로, 각 부서ID에 대하여 하나의 행으로부터 하나의 부서이름이 서브쿼리에 의하여 반환될 수 있습니다.

☞ 추가된 컬럼에 대한 동일한 수정작업을 MERGE 문장을 이용하여 수행할 수도 있습니다.

```
SQL> MERGE INTO hr.emp_app_test a
     USING hr.departments d
     ON (a.department_id=d.department_id)
     WHEN MATCHED THEN
     UPDATE SET
       a.department_name=d.department_name ;

106개 행 이(가) 병합되었습니다.
```

7> HR.EMP_APP_TEST 테이블의 DEPARTMENT_NAME 컬럼의 수정 값 확인.

```
SQL> SELECT *
     FROM hr.emp_app_test ;

EMPLOYEE_ID LAST_NAME           SALARY DEPARTMENT_ID DEPARTMENT_NAME
----------- --------------- ---------- ------------- ------------------------------
        100 King                24000            90 Executive
        101 Kochhar             17000            90 Executive
        102 De Haan             17000            90 Executive
        103 Hunold               9000            60 IT
        104 Ernst                6000            60 IT
...

 107개의 행이 선택됨
```

1-6-2. 상관관계 서브쿼리를 이용한 테이블의 데이터 삭제.

■ 현재 사용중인 테이블에 있는 과거데이터의 행을, 과거 데이터를 가지는 테이블로 입력한 후에, 과거 데이터를 가지는 테이블로 입력된 데이터를 확인해서, 현재 사용중인 테이블에 있는 과거데이터를 삭제하려고 할 때, 상관관계 서브쿼리를 이용한 DELETE문을 사용할 수 있습니다.

■ 다음과 같은 경우에, 상관관계 서브쿼리를 이용한 DELETE문으로 행을 삭제하는 방법이 적용될 수 있습니다.

○ 병원에서 입원환자 테이블에 있는 퇴원한 환자정보의 행을, 퇴원환자 테이블로 입력한 후, 퇴원환자 테이블에 입력된 것을 확인하면서, 입원환자 테이블의 행을 삭제해야 하는 경우.

○ 회사에서 퇴사자 테이블에 있는 퇴사한 직원에 대한의 행을, 퇴사자 테이블로 입력한 후, 퇴사자 테이블에 입력된 것을 확인하면서, 직원 테이블의 행을 삭제해야 하는 경우.

○ 거래내역 테이블에 있는 과거의 거래내역과 관련된 행을, 과거거래내역 테이블로 입력한 후, 과거거래내역 테이블에 입력된 것을 확인하면서, 거래내역 테이블의 행을 삭제해야 하는 경우.

■ 다음에 나오는 일련의 실습을 통해 상관관계 서브쿼리를 이용한 DELETE문을 사용하는 방법을 확인합니다.

1> SQL*Developer를 이용하여, HR 계정으로 데이터베이스에 접속합니다.

2> 앞의 상관관계 서브쿼리를 이용한 UPDATE문 실습에서 사용한 HR.EMP_APP_TEST 테이블과 동일한 컬럼들로 구성되는, HR.EMP_HISTORY 테이블을 서브쿼리를 이용하여 생성하되, 부서ID가 30 및 50인 직원의 행이 같이 복사되도록 하시오. 단, 직원ID가 199, 198, 197 인 행은 복사되지 않도록 하시오

```
SQL> CREATE TABLE HR.EMP_HISTORY
     AS
         SELECT *
         FROM hr.emp_app_test
         WHERE department_id IN ( 30 , 50 )
         AND employee_id not in (199,198,197) ;

Table HR.EMP_HISTORY이(가) 생성되었습니다.
```

☞ 생성된 EMP_HISTORY 테이블은 퇴사한 직원의 데이터들이 저장된 테이블로 사용합니다.

3> HR.EMP_HISTORY 테이블의 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.emp_history ;

EMPLOYEE_ID LAST_NAME           SALARY DEPARTMENT_ID DEPARTMENT_NAME
----------- --------------- ---------- ------------- ------------------------------
        114 Raphaely             11000            30 Purchasing
        115 Khoo                  3100            30 Purchasing
        116 Baida                 2900            30 Purchasing
        117 Tobias                2800            30 Purchasing
        118 Himuro                2600            30 Purchasing
...

 48개의 행이 선택됨
```

☞ 현재 48명의 퇴사자 직원정보가 저장되어 있습니다.

☞ 오늘 직원ID가 197,198,199 인 직원 3명이 퇴사를 했고, 이 직원들과 관련된 행을 아직 HR.EMP_HISTORY 테이블에 입력하지 않았습니다.

4> HR.EMP_APP_TEST 테이블에 있는 직원정보 중 퇴사한 직원과 관련된 정보를 삭제하시오. 단, 퇴사했지만, 아직 HR.EMP_HISTORY 테이블에 입력되지 않은, 퇴사직원과 관련된 행은 삭제하지 마시오.

```
SQL> DELETE FROM hr.emp_app_test a
     WHERE a.employee_id = (SELECT b.employee_id
                            FROM emp_history b
                            WHERE b.employee_id = a.employee_id ) ;

48개 행 이(가) 삭제되었습니다.
```

☞ 앞의 상관관계 서브쿼리를 이용한 DELETE문은 다음의 과정을 통해 처리됩니다.

1) HR.EMP_APP_TEST 테이블에서, 하나의 후보 행에 있는 직원의 직원ID 값이 값이 서브쿼리로 전달됩니다.

2) 전달된 직원ID 값을 이용하여 서브쿼리가 실행되고, HR.EMP_HISTORY 테이블에서 행을 찾아, 전달된 직원ID에 해당하는 행이 있으면, 해당 직원의 동일한 직원ID가 구해집니다. 즉, HR.EMP_APP_TEST 테이블의 행의 직원ID과 동일한 직원ID를 가지는 행이 HR.EMPHISTORY 테이블에 있는 것이 확인됩니다

3) 서브쿼리로부터 해당 직원ID의 행이 있음이 확인되면, HR.EMP_APP_TEST 테이블의 행을 삭제합니다.

4) 1단계부터 3단계까지를 HR.EMP_APP_TEST 테이블의 각 행에 대하여 반복 수행하여, HR.EMP_HISTORY 테이블에 저장된 행이 있는 행만 HR.EMP_APP_TEST 테이블로부터 삭제합니다.

☞ 이 과정을 이해하면, 서브쿼리로부터 직원ID가 반드시 하나가 반환되어야 하며, 따라서, 서브쿼리의 WHERE절에서 HR.EMP_HISTORY 테이블의 EMPLOYEE_ID 값에 대하여 반드시 한 행만 존재해야 합니다.

5> HR.EMP_APP_TEST 테이블에서 부서ID가 30 및 50인 행들의 데이터를 확인합니다.

```
SQL> SELECT *
     FROM hr.emp_app_test
     WHERE department_id in (30, 50) ;

EMPLOYEE_ID LAST_NAME           SALARY DEPARTMENT_ID DEPARTMENT_NAME
----------- --------------- ---------- ------------- ------------------------------
        197 Feeney                3000            50 Shipping
        198 OConnell              2600            50 Shipping
        199 Grant                 2600            50 Shipping
```

☞ 아직 HR.EMP_HISTORY 테이블에 입력되지 않은 행은 삭제되지 않고, HR.EMP_HISTORY 테이블에 입력된 48개 행만 삭제되었습니다.

1-7. EXISTS 연산자 및 NOT EXISTS 연산자 사용.

■ EXISTS 연산자는, 서브쿼리의 결과 집합에 행이 존재하는지를 테스트합니다.

■ SQL문의 WHERE절에서 EXISTS 연산자를 사용하여, SQL문의 후보 행으로부터 전달된 값으로, 서브쿼리가 실행될 때, 서브쿼리로 검색된 값의 결과 집합이 존재하는지를 확인하기 위하여 상관관계 서브쿼리가 사용된 SQL문에서 자주 사용됩니다. 서브쿼리 결과 집합이 1개 있어도 존재하는 것이고, 10개 있어도 존재하는 것이면, 하나 찾았을 때 더 찾을 필요 없습니다.

■ SQL문의 WHERE절에 EXISTS 연산자와 서브쿼리가 사용되었을 때, EXISTS 연산자의 기능은 다음과 같습니다.

○ 서브쿼리에서, 최초의 한 행이 선택되면, 더 이상 행을 찾지 않고,
EXISTS 연산자가 포함된 조건은 TRUE로 결정됩니다.

○ 서브쿼리에서, 최초의 한 행이 선택되기 전까지 검색이 계속 진행되고, 선택되는 행이 하나도 없으면,
EXISTS 연산자가 포함된 조건은 FALSE로 결정됩니다.

■ SQL문의 WHERE절에 NOT EXISTS 연산자와 서브쿼리가 사용되었을 때, NOT EXISTS 연산자의 기능은 다음과 같습니다.

○ 서브쿼리에서, 최초의 한 행이 선택되면, 더 이상 행을 찾지 않고,
NOT EXISTS 연산자가 포함된 조건은 FALSE로 결정됩니다.

○ 서브쿼리에서, 최초의 한 행이 선택되기 전까지 검색이 계속 진행되고, 선택되는 행이 하나도 없으면,
NOT EXISTS 연산자가 포함된 조건은 TRUE로 결정됩니다.

■ HR.EMPLOYEES 테이블을 이용하여, 부하직원이 있는 직원들의 직원ID(EMPLOYEE_ID컬럼), 성(LAST_NAME컬럼), 직무ID(JOB_ID컬럼), 근무부서ID(DEPARTMENT_ID컬럼)를 표시하시오.

○ HR.EMPLOYEES 테이블의 MANAGER_ID 컬럼에는, 각 직원의 직속관리자의 직원ID가 저장되어있습니다.

```
SQL> SELECT e.employee_id, e.last_name, e.job_id, e.department_id
     FROM hr.employees e
     WHERE EXISTS (SELECT 1
                   FROM hr.employees b
                   WHERE b.manager_id = e.employee_id)
     ORDER BY 1;

EMPLOYEE_ID LAST_NAME       JOB_ID     DEPARTMENT_ID
----------- --------------- ---------- -------------
        100 King            AD_PRES               90
        101 Kochhar         AD_VP                 90
        102 De Haan         AD_VP                 90
        103 Hunold          IT_PROG               60
        108 Greenberg       FI_MGR               100
        114 Raphaely        PU_MAN                30
        120 Weiss           ST_MAN                50

 18개의 행이 선택됨
```

☞ 앞의 SELECT문에 있는 WHERE절은 다음의 과정을 통해 처리됩니다.

1) HR.EMPLOYEES 테이블에서, 직원ID가 100인 후보 행의 직원ID인 100 값이 서브쿼리로 전달됩니다.

2) 전달받은 100 값을 이용하여, 서브쿼리가 실행 중에 직원의 관리자ID가 100 인 행을, HR.EMPLOYEES 테이블에서 검색합니다. 관리자ID가 100인 최초의 행을 찾을 때까지 검색이 계속 진행됩니다.

3) 관리자ID가 100인 최초의 한 행이 확인되면, 검색은 중단되고, EXISTS 연산자에 의하여, 메인쿼리의 WHERE절이 TRUE로 결정됩니다. 그리고, 메인쿼리의 후보 행은 선택됩니다.

4) 1단계부터 3단계까지를 메인 쿼리의 107 개 행에 대하여 수행합니다.

■ HR.EMPLOYEES 테이블을 이용하여, 부하직원이 없는 직원들의 직원ID(EMPLOYEE_ID컬럼), 성(LAST_NAME컬럼), 직무ID(JOB_ID컬럼), 근무부서ID(DEPARTMENT_ID컬럼)를 표시하시오.

○ HR.EMPLOYEES 테이블의 MANAGER_ID 컬럼에는 각 직원의 직속관리자의 직원ID가 저장되어있습니다.

```
SQL> SELECT e.employee_id, e.last_name, e.job_id, e.department_id
     FROM hr.employees e
     WHERE NOT EXISTS (SELECT 1
                       FROM hr.employees b
                       WHERE b.manager_id = e.employee_id)
     ORDER BY 1;

EMPLOYEE_ID LAST_NAME       JOB_ID     DEPARTMENT_ID
----------- --------------- ---------- -------------
        104 Ernst           IT_PROG               60
        105 Austin          IT_PROG               60
        106 Pataballa       IT_PROG               60
        107 Lorentz         IT_PROG               60
        109 Faviet          FI_ACCOUNT           100
...

 89개의 행이 선택됨
```

☞ 앞의 SELECT문에 있는 WHERE절은 다음의 과정을 통해 처리됩니다.

1) HR.EMPLOYEES 테이블에서, 직원ID가 100인 후보 행의 직원ID인 100 값이 서브쿼리로 전달됩니다.

2) 전달받은 100 값을 이용하여, 서브쿼리가 실행 중에 직원의 관리자ID가 100 인 행을, HR.EMPLOYEES 테이블에서 검색합니다. 관리자ID가 100인 최초의 행을 찾을 때까지 검색이 계속 진행됩니다.

3) 관리자ID가 100인 최초의 한 행이 확인되면, 검색은 중단되고, NOT EXISTS 연산자에 의하여, 메인쿼리의 WHERE절이 FALSE로 결정됩니다. 그리고, 메인쿼리의 후보 행은 선택되지 않습니다.

   만약, HR.EMPLOYEES 테이블을 모두 검색하여 직원의 관리자ID에 해당하는 행을 하나도 찾지 못하면,
   NOT EXISTS 연산자에 의하여, 메인쿼리의 WHERE절이 TRUE로 결정되고, 메인쿼리의 후보 행은 선택됩니다.

4) 1단계부터 3단계까지를 메인 쿼리의 107 개 행에 대하여 수행합니다.

■ HR.DEPARTMENTS 테이블과 HR.EMPLOYEES 테이블을 이용하여, 직원이 없는 부서의 부서ID(DEPARTMENT_ID컬럼), 부서이름(DEPARTMENT_NAME), 부서의 관리자ID(MANAGER_ID컬럼), 부서의 위치ID(LOCATION_ID컬럼)을 표시하시오.

```
SQL> SELECT d.department_id, d.department_name, d.manager_id, d.location_id
     FROM hr.departments d
     WHERE NOT EXISTS (SELECT 1
                       FROM hr.employees e
                       WHERE e.department_id = d.department_id)
     ORDER BY 1;

DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          120 Treasury                                         1700
          130 Corporate Tax                                    1700
          140 Control And Credit                               1700
          150 Shareholder Services                             1700
          160 Benefits                                         1700
          170 Manufacturing                                    1700
          180 Construction                                     1700
          190 Contracting                                      1700
          200 Operations                                       1700
          210 IT Support                                       1700
          220 NOC                                              1700
          230 IT Helpdesk                                      1700
          240 Government Sales                                 1700
          250 Retail Sales                                     1700
          260 Recruiting                                       1700
          270 Payroll                                          1700

 16개의 행이 선택됨
```

☞ 앞의 SELECT문에 있는 WHERE절은 다음의 과정을 통해 처리됩니다.

1) HR.DEPARTMENTS 테이블에서, 부서ID가 10인 후보 행의 부서ID인 10 값이 서브쿼리로 전달됩니다.

2) 전달받은 10 값을 이용하여, 서브쿼리가 실행 중에 직원의 근무부서ID가 10 인 행을, HR.EMPLOYEES 테이블에서 검색합니다. 직원의 근무부서ID가 10인 최초의 행을 찾을 때까지 검색이 계속 진행됩니다.

3) 직원의 근무부서ID가 10인 최초의 한 행이 확인되면, 검색은 중단되고, NOT EXISTS 연산자에 의하여, 메인쿼리의 WHERE절이 FALSE로 결정됩니다. 그리고, 메인쿼리의 후보 행은 선택되지 않습니다.

   만약, HR.EMPLOYEES 테이블을 모두 검색하여 직원의 근무부서ID에 해당하는 행을 하나도 찾지 못하면, NOT EXISTS 연산자에 의하여, 메인쿼리의 WHERE절이 TRUE로 결정되고, 메인쿼리의 후보 행은 선택됩니다.

4) 1단계부터 3단계까지를 메인 쿼리의 107 개 행에 대하여 수행합니다.

☞ 앞의 실습에서 작성된 SELECT문을 상관관계 서브쿼리를 사용하지 않고, 다음처럼 작성할 수도 있습니다.

```
SQL> SELECT d.department_id, d.department_name, d.manager_id, d.location_id
     FROM hr.departments d
     WHERE d.department_id NOT IN (SELECT department_id
                                   FROM hr.employees e
                                   WHERE e.department_id  IS NOT NULL)
     ORDER BY 1;
```

○ 위의 문장을 사용 시에, 서브쿼리가 반환하는 DEPARTMENT_ID 컬럼에 대하여 NOT NULL 제약조건이 없는 경우, DEPARTMENT_ID 으로 반환되는 값 중 NULL이 배제될 수 있도록 서브쿼리의 WHERE절에 e.department_id IS NOT NULL 조건을 반드시 포함시켜야 합니다.

1-7. WITH 절의 사용

■ 하나의 SELECT문에, 다수의 서브쿼리가 사용되거나 또는 서브쿼리들이 중첩되어 사용되거나, 또는 동일한 서브쿼리가 반복적으로 두 번 이상 사용되는 경우, 문장의 작성이나 의미해석이 어려워지고, 처리성능이 좋지 못하게 됩니다.

■ **SELECT문에 사용되는 서브쿼리들을 WITH절을 이용하여 쿼리블록으로 SELECT문 앞에 정의한 다음**, SELECT문에서 호출하여 사용할 수 있습니다.

■ SELECT문에서 WITH절을 이용하면, 다음과 같은 장점이 있습니다.

○ SELECT문을 읽기 쉽게 만들고, 이해가 용이해집니다.

○ SELECT문에서 동일한 서브쿼리블록이 여러 번 호출되더라도 한 번만 평가합니다.

○ 대부분의 경우 대용량 쿼리에 대한 성능을 향상시킬 수 있습니다.

■ 내부적으로 WITH절은 인라인 뷰나 임시 테이블로 분석됩니다. 옵티마이저는 WITH절의 결과를 임시로 저장하는 비용이나 이점에 따라 적절한 분석 방법을 선택합니다.

■ WITH절은, 사용자 계정에게 지정된 임시 테이블스페이스에, 선언된 쿼리 블록 결과를 검색하여 저장합니다.

■ 기본문법

```
WITH
  이름1 AS (쿼리1) ,      -- WITH절에 선언된 SELECT문들은 서브쿼리로 고려합니다.
  이름2 AS (쿼리2) ,
  ...
  이름n AS (쿼리n)
메인SELECT문 ;            -- WITH절 다음에 작성된 SELECT문을 메인쿼리로 고려합니다.
```

○ WITH절에 쿼리문의 이름을 선언하며, 선언되는 쿼리가 2개 이상인 경우에는, 콤마(,)로 구분합니다.

○ 앞에서 선언한 쿼리의 이름을, 뒤에서 선언되는 이름의 쿼리에서 호출하여 사용할 수 있습니다.

○ WITH절은 항상 SELECCT문과 함께 사용되며, WITH절에 선언된 쿼리의 이름(들)을, SELECT문에서 호출하여 사용합니다.

○ WITH절에 선언된 쿼리의 이름들은 문장이 실행되는 동안에만 유효합니다. 즉, WITH절에 선언된 이름들은 데이터베이스에 저장되지 않습니다.

○ WITH절에 선언하는 쿼리에 대한 이름을, 테이블이름과 다르게 선언하십시오.

■ HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블의 데이터를 이용하여, 부서이름 별 급여합계가, 부서별 급여합계의 평균값보다 큰 부서에 대해서만, 부서이름과 부서별 급여합계를 표시하시오. 단, WITH절을 사용하지 않고 SELECT문을 작성하시오.

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

DEPARTMENT_NAME                DEPT_TOTAL
------------------------------ ----------
Sales                              304500
Shipping                           156400
```

☞ 위의 SELECT문에서 다음의 위치에 서브쿼리가 사용되었습니다.

○ 메인쿼리의 FROM절

○ WHERE절의 조건

○ WHERE절의 조건에 사용된 서브쿼리의 FROM절

○ WHERE절의 조건에 사용된 서브쿼리의 FROM절에 사용된 서브쿼리 FROM절

☞ 동일한 서브쿼리가 2번 사용되었습니다.

☞ WITH절을 사용하지 않은 경우, 전체적으로 문장이 길어졌고, SELECT문으로 표시되는 데이터에 대한 파악이 쉽지 않습니다.

■ HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블의 데이터를 이용하여, 부서이름 별 급여합계가, 부서별 급여합계의 평균값보다 큰 부서에 대해서만, 부서이름과 부서별 급여합계를 표시하시오. 단, WITH절을 사용하여 SELECT문을 작성하시오.

```
SQL> WITH
       DEPT_COSTS AS (SELECT d.department_name, SUM(e.salary) AS dept_total
                      FROM hr.employees e INNER JOIN hr.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 ;

DEPARTMENT_NAME           DEPT_TOTAL
------------------------- ----------
Sales                         304500
Shipping                      156400
```

☞ 위의 WITH절에 선언된 서브쿼리들은 다음의 결과들을 내용을

○ DEPT_COSTS: HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블의 행들을 조인시킨 후, 부서이름 별 직원들의 급여에 대한 합계가 결과집합으로 구해집니다.

○ AVG_COST: DEPT_COSTS의 결과집합을 이용하여, 부서이름 별 급여합계에 대한 평균값이 결과로 구해집니다.

☞ WITH절에 선언된 이름들을 이용하여 하단에 작성된 SELECT문은 다음과 같은 의미의 결과를 표시합니다.

○ 부서이름 별 급여합계가, 부서별 급여합계의 평균값보다 큰 부서에 대해서만, 부서이름과 부서별 급여합계가 표시됩니다.

☞ WITH절을 사용하지 않은 경우보다, 구문이 짧고 간결해지며, 의미파악이 보다 용이해 집니다.