[AI초급] SQLD 자격증 코스 - 챕터 12

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- SQL의 활용 과목인 집합 연산자와 서브쿼리에 대해 학습합니다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 집합 연산자

동일 테이블에서 서로 다른 질의의 결과를 합치고자 할 때 사용하는 집합연산자에 대해 학습합니다.

▼ 1) 집합 연산자 개념과 종류

☑️ 집합 연산자 개념

집합 연산자는(SET OPERATOR)는 SELECT를 통해 얻은 결과 간의 집합 연산을 수행하는 연산자입니다. 2개 이상의 테이블에서 JOIN을 사용하지 않고 연관된 데이터를 조회하여 합쳐서 볼 수 있습니다. 즉, 2개 이상의 쿼리 수행 결과를 가지고 하나의 결과로 만들어줍니다. 동일 테이블에서 서로 다른 질의를 수행하여 결과를 합치고자 할 때 주로 사용할 수 있습니다.

집합 연산자를 사용하기 위해서는 SELECT 절의 칼럼 수가 동일해야 합니다. SELECT 절의 동일 위치에 존재하는 칼럼의 데이터 타입이 반드시 같을 필요는 없지만 암시적 데이터 형변환으로 상호 호환이 가능해야 합니다.

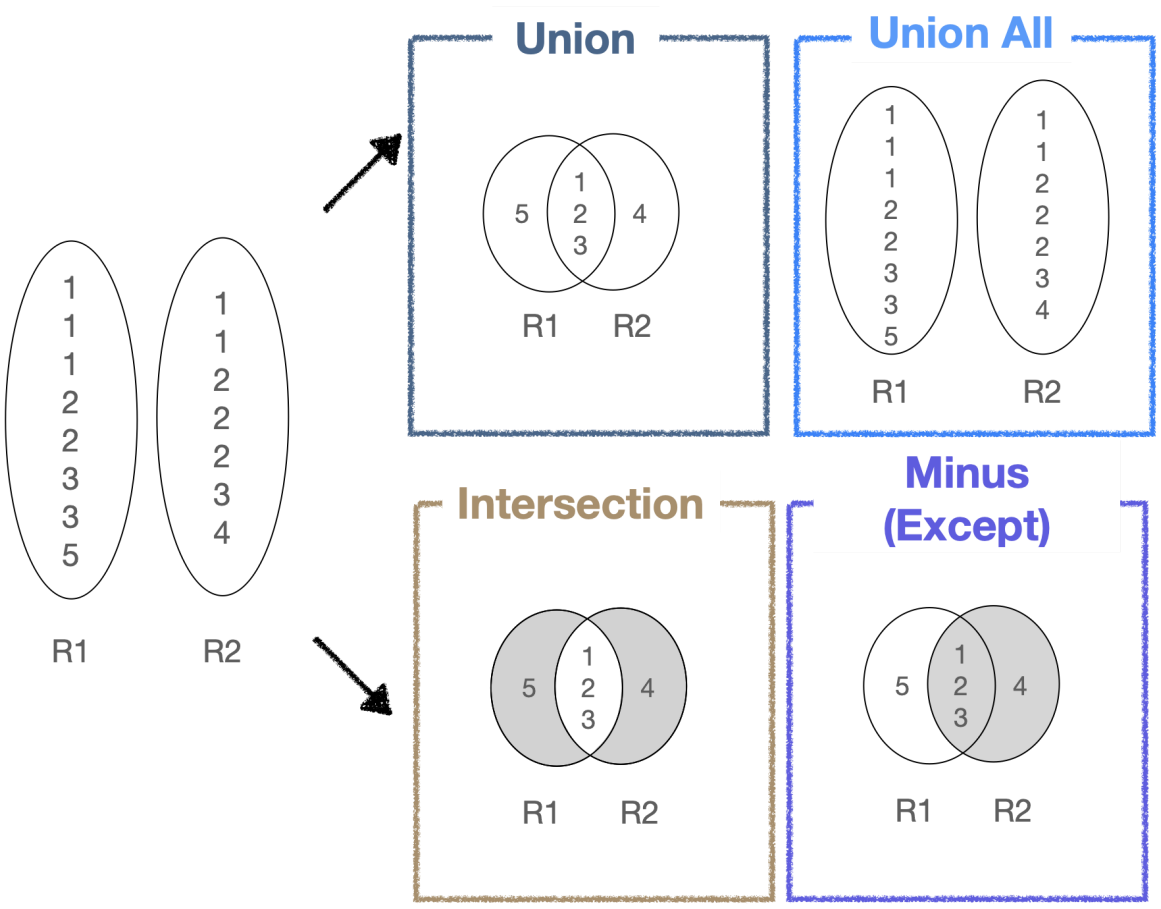
ORDER BY는 집합 연산을 적용한 최종 결과에 대한 정렬 처리이므로 가장 마지막 줄에 한 번만 기술하면 됩니다.

☑️ 집합 연산자의 종류

집합 연산자	설명
UNION (합집합 중복 허용 x)	- SQL문 결과에 대한 합집합 - 모든 중복된 행을 하나의 행으로 만들어서 합집합의 결과를 반환 - 중복 배제하기 위한 정렬 연산이 있어 시스템의 부하가 있음
UNION ALL (합집합 - 중복 허용 o)	- SQL문 결과에 대한 합집합 - 중복된 행도 그대로 포함하여 합집합 결과를 반환하는 연산자 - 주로 여러 쿼리문의 결과가 상호 배타적일 때 많이 사용

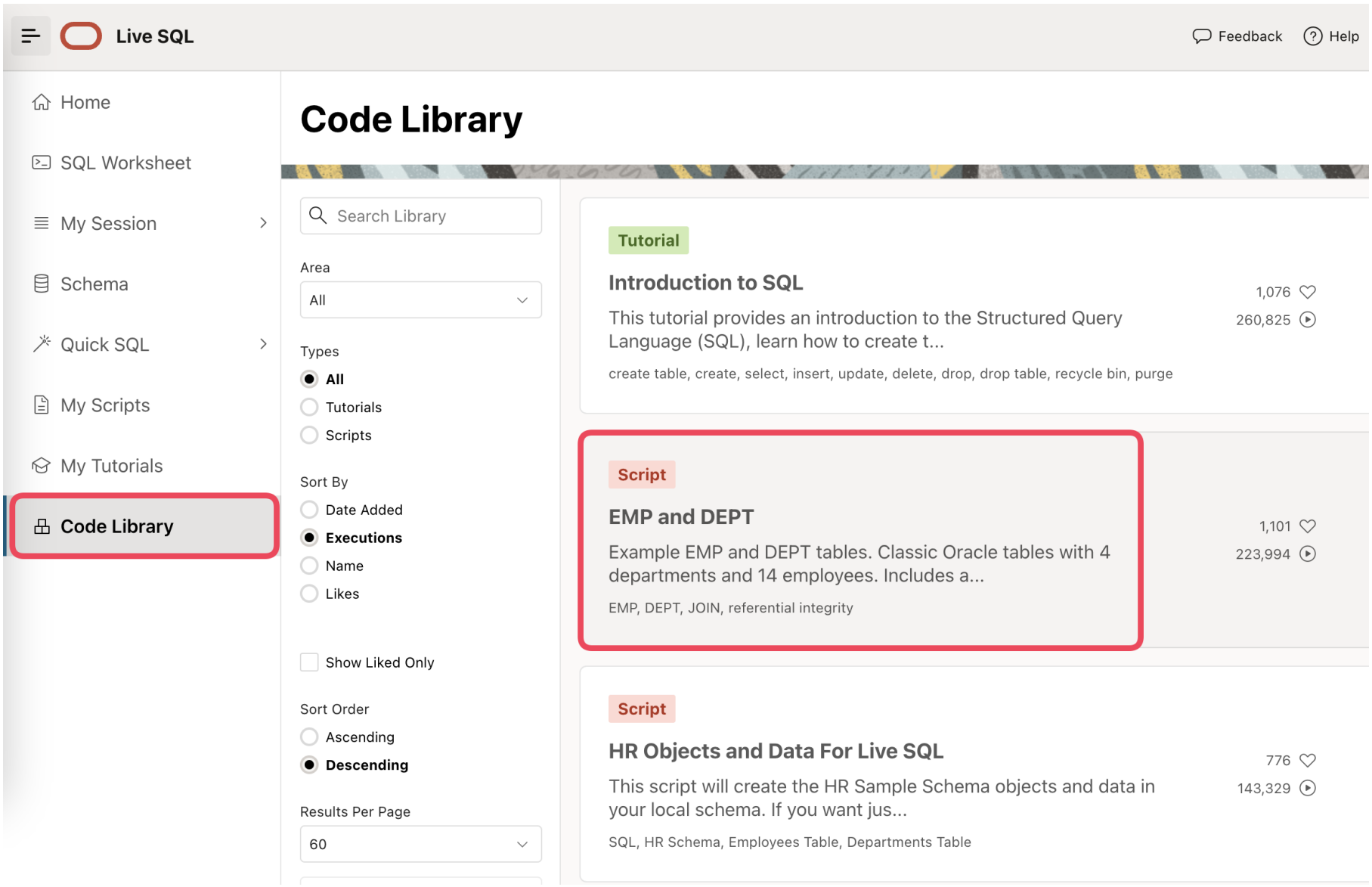
집합 연산자	설명
INTERSECT (INTERSECTION)	- SQL문 결과에 대한 교집합 - 중복을 허용하지 않고 하나의 행을 만들어 줌
MINUS(EXCEPT)	- SQL문 결과에 대한 차집합 - 특정한 SQL문 결과에서 다른 SQL문 결과를 뺀 값 - 중복을 허용하지 않고 하나의 행으로 만들어 줌

개별 결과 집합이 있다면 UNION, UNION ALL, INTERSECT, MINUS (EXCEPT)를 이용하여 집합 연산을 할 수 있습니다. UNION ALL을 제외한 나머지 집합 연산자들은 중복 데이터는 허용하지 않습니다.

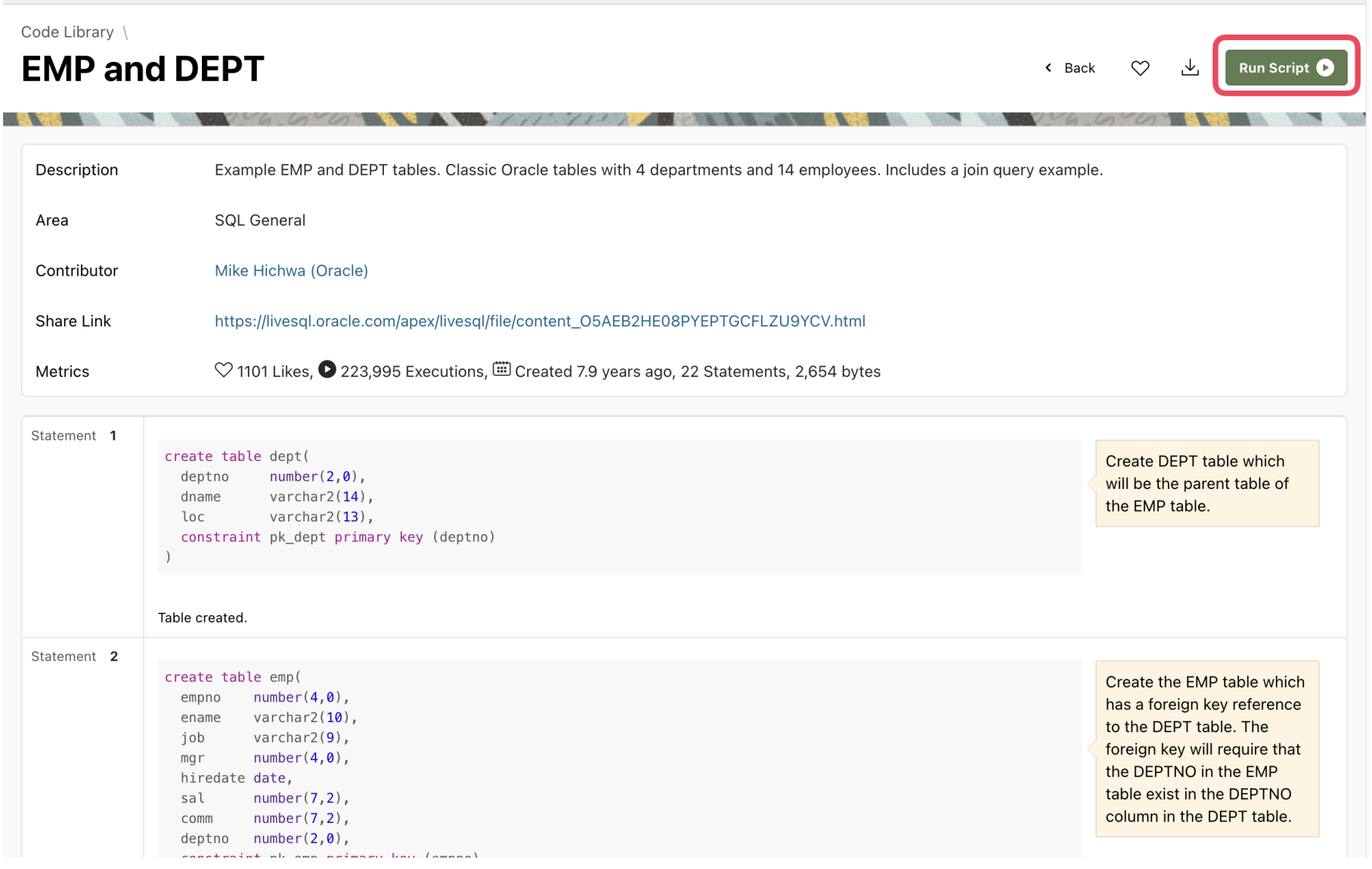


▼ 2) 집합 연산자의 예시

> 💡 ! 참고 - Oracle의 실습용 테이블인 EMP / DEPT를 생성하여 실습에 참여해 주세요.



Code Library → EMP and DEPT



Run Script 선택

(1) 시작하기 전 기본 흐름 가져가기

기본 emp 테이블 조회

```
SELECT empno, ename, job, deptno FROM emp;
```

EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7566	JONES	MANAGER	20
7788	SCOTT	ANALYST	20
7902	FORD	ANALYST	20
7369	SMITH	CLERK	20
7499	ALLEN	SALESMAN	30
7521	WARD	SALESMAN	30
7654	MARTIN	SALESMAN	30
7844	TURNER	SALESMAN	30
7876	ADAMS	CLERK	20
7900	JAMES	CLERK	30
7934	MILLER	CLERK	10

기본 dept 테이블 조회

```
SELECT deptno, dname FROM dept;
```

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

부서 번호가 10번인 job 조회

```
SELECT job FROM EMP WHERE deptno = 10;
```

부서 번호가 20번인 job 조회

```
SELECT job FROM emp WHERE deptno = 20;
```

(2) UNION

- 합집합 연산입니다.
- 결과 집합을 정렬된 순서로 중복을 제거하고 반환합니다.
- NULL 값을 포함하여 연산합니다.

- 예시

```
SELECT job FROM emp WHERE deptno = 10
UNION
SELECT job FROM emp WHERE deptno = 20;
```

JOB
PRESIDENT
MANAGER
CLERK

JOB
MANAGER
ANALYST
ANALYST
CLERK
CLERK

JOB
ANALYST
CLERK
MANAGER
PRESIDENT

- 실습

▼ 실습 데이터

```
CREATE TABLE PARTICIPANT (
partic_id NUMBER,
nation_id NUMBER,
main_sport_id NUMBER,
first_name VARCHAR2(30),
last_name VARCHAR2(30),
gender VARCHAR2(1),
height NUMBER,
weight NUMBER
);

INSERT INTO PARTICIPANT VALUES (1, 1, 101, 'John', 'Doe', 'M', 180, 75);
INSERT INTO PARTICIPANT VALUES (2, 2, 102, 'Jane', 'Smith', 'F', 165, 55);
INSERT INTO PARTICIPANT VALUES (3, 3, 103, 'Michael', 'Johnson', 'M', 175, 70);
INSERT INTO PARTICIPANT VALUES (4, 4, 104, 'Emily', 'Davis', 'F', 160, 50);
INSERT INTO PARTICIPANT VALUES (5, 5, 105, 'Chris', 'Brown', 'M', 190, 80);
INSERT INTO PARTICIPANT VALUES (6, 1, 101, 'Sarah', 'Wilson', 'F', 170, 60);
INSERT INTO PARTICIPANT VALUES (7, 2, 102, 'David', 'Lee', 'M', 175, 70);
```

```
CREATE TABLE NATION (
    nation_id NUMBER,
    country_name VARCHAR2(30),
    population NUMBER
);

INSERT INTO NATION VALUES (1, 'United States', 331002651);
INSERT INTO NATION VALUES (2, 'India', 1380004385);
INSERT INTO NATION VALUES (3, 'China', 1444216107);
INSERT INTO NATION VALUES (4, 'Brazil', 212559417);
INSERT INTO NATION VALUES (5, 'United Kingdom', 67886011);
```

```
CREATE TABLE SPORT (
    sport_id NUMBER PRIMARY KEY,
    sport_name VARCHAR2(20),
    max_weight NUMBER,
    min_weight NUMBER
);

INSERT INTO SPORT VALUES (101, 'Football', 100, 60);
INSERT INTO SPORT VALUES  (102, 'Basketball', 120, 50);
INSERT INTO SPORT VALUES  (103, 'Tennis', 90, 50);
INSERT INTO SPORT VALUES  (104, 'Swimming', 85, 45);
INSERT INTO SPORT VALUES  (105, 'Gymnastics', 80, 40);
```

> 국가ID가 1 인 선수와 주 종목 ID가 102인 선수의 이름과 키를 UNION을 이용하여 표현해보세요.

```
SELECT first_name, height FROM participant WHERE nation_id = 1
UNION
SELECT first_name, height FROM participant WHERE main_sport_id = 102;
```

(3) UNION ALL

- 합집합 연산입니다.
- 모든 쿼리문의 결과를 중복을 포함하여 반환합니다.

- 예시

```
SELECT job FROM emp WHERE deptno = 10
UNION ALL
SELECT job FROM emp WHERE deptno = 20;
```

JOB
PRESIDENT
MANAGER
CLERK
MANAGER
ANALYST
ANALYST
CLERK
CLERK

- 실습

> 성별이 ‘M’인 선수와 ‘F’인 선수의 이름과 키 데이터를 전부 표현해보세요.

```
SELECT first_name, height FROM participant WHERE gender = 'M'
UNION ALL
SELECT first_name, height FROM participant WHERE gender = 'F';
```

(4) INTERSECT

- 교집합 연산입니다.
- 모든 공통된 값을 중복없이 반환합니다.

- 예시

```
SELECT job FROM emp WHERE deptno = 10
INTERSECT
SELECT job FROM emp WHERE deptno = 20;
```

JOB
CLERK
MANAGER

- 실습

▼ 실습 데이터

```
INSERT INTO participant VALUES (8, 1, 102, 'Laura', 'Johnson', 'F', 160, 55);
```

> 국가ID가 1 인 선수와 주 종목 ID가 101인 선수의 이름과 키를 INTERSECT를 이용하여 표현해보세요.

```
SELECT first_name, height FROM participant WHERE nation_id = 1
INTERSECT
SELECT first_name, height FROM participant WHERE main_sport_id = 101;
```

(5) MINUS (EXCEPT)

- 차집합 연산입니다.
- 중복없이 반환합니다.

- 예시

```
SELECT job FROM emp WHERE deptno = 10
MINUS
SELECT job FROM emp WHERE deptno = 20;
```

JOB
PRESIDENT

- 실습

> 국가 ID가 1인 선수 중에서 주 종목ID가 101인 종목만 제외해서 선수이름과 키를 출력해보세요.

```
SELECT first_name, height FROM participant WHERE nation_id = 1
MINUS
SELECT first_name, height FROM participant WHERE main_sport_id = 101;
```

▼ 연습문제

☑️ 문제 1

Q. 문제	**집합연산자 UNION과 UNION ALL에 대한 설명으로 옳지 않은 것은?**
A. (1)	UNION은 중복된 행은 하나의 행으로 만든다
A. (2)	UNION은 중복 배제하기 위한 정렬 연산이 있어 시스템의 부하가 있다
A. (3)	UNION ALL은 중복된 행을 그대로 보여주며 중복된 행을 제외하면 UNION과 결과 집합 및 그 순서가 동일하다

A. (4)	UNION ALL은 일반적으로 여러 개의 질의 결과가 중복이 없을 때(상호 배타적일 때) 사용한다

▼ 정답

(3) UNION과 UNION ALL의 결과 집합의 순서가 다를 수 있다

☑️ 문제 2

Q. 문제	**집합연산자에 대한 설명으로 가장 부적절한 것은?**
A. (1)	UNION은 여러 개의 집합 연산에 대한 합집합을 출력한다
A. (2)	UNION ALL은 여러 개의 집합 연산에 대한 합집합을 출력한다
A. (3)	EXCEPT/MINUS는 앞의 SQL문의 결과에서 뒤의 SQL문의 결과에 차집합이다
A. (4)	UNION과UNION ALL은 모두 중복을 허용한다

▼ 정답

(4) UNION은 중복을 허용하지 않는다

☑️ 문제 3

Q. 문제	**UNION과 UNION ALL에 대한 설명으로 옳지 않은 것은?**
A. (1)	UNION은 2개의 테이블에 대해 합집합을 만들 수 있다
A. (2)	UNION은 중복을 제거한다
A. (3)	UNION ALL은 정렬을 유발하지만 UNION은 정렬을 유발하지 않는다
A. (4)	UNION과 UNION ALL을 사용할 때 2개 SELECT문에서 칼럼의 수가 일치해야 한다

▼ 정답

(3) UNION은 중복을 제거하기 때문에 정렬을 유발한다. 반면 UNION ALL은 중복을 제거하지 않기 때문에 정렬을 유발하지 않는다

02. 서브쿼리

✔️ **다양한 업무처리가 가능하여 실무에서 많이 사용되는 서브쿼리에 대해 학습합니다.**

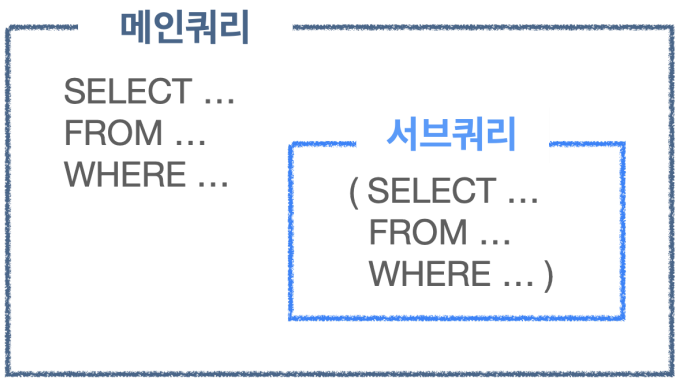
▼ 1) 서브쿼리 기본

☑️ 개념

서브쿼리(Subquery)란 하나의 SQL 문안에 포함되어 있는 또 다른 SQL 문을 의미합니다. 서브쿼리를 사용하면 SQL문을 통해 할 수 있는 일이 다양해지기 때문에 다양한 업무 처리가 가능하여 실무에서도 많이 사용합니다. 같이 상세하게 알아보도록 하겠습니다.

조인은 조인에 참여하는 모든 테이블에 대등한 관계에 있기 때문에 조인에 참여하는 모든 테이블의 칼럼을 어느 위치에서라도 자유롭게 참조가 가능합니다. 하지만 서브쿼리는 조인과 다르게 자유로운 형태의 참조가 아닌 특정 조건에 맞게 참조를 해야 합니다.

서브쿼리는 알려지지 않은 기준을 이용한 검색을 위해 사용되기 때문에 아래 그림과 같이 메인쿼리가 서브쿼리를 포함하는 형태를 띠게 됩니다. 즉, 서브쿼리는 레벨과는 상관없이 항상 메인쿼리 레벨로 결과 집합이 생성됩니다.



이러한 형태로 인해서 서브쿼리는 메인쿼리의 칼럼을 모두 사용할 수 있습니다. 그러나 반대의 경우 메인쿼리는 서브쿼리의 칼럼을 사용할 수가 없습니다. 메인쿼리는 서브쿼리에게 자신의 칼럼을 주는 것만 가능할 뿐 서브쿼리 내에 칼럼을 이용할 수가 없습니다.

포인트 정리

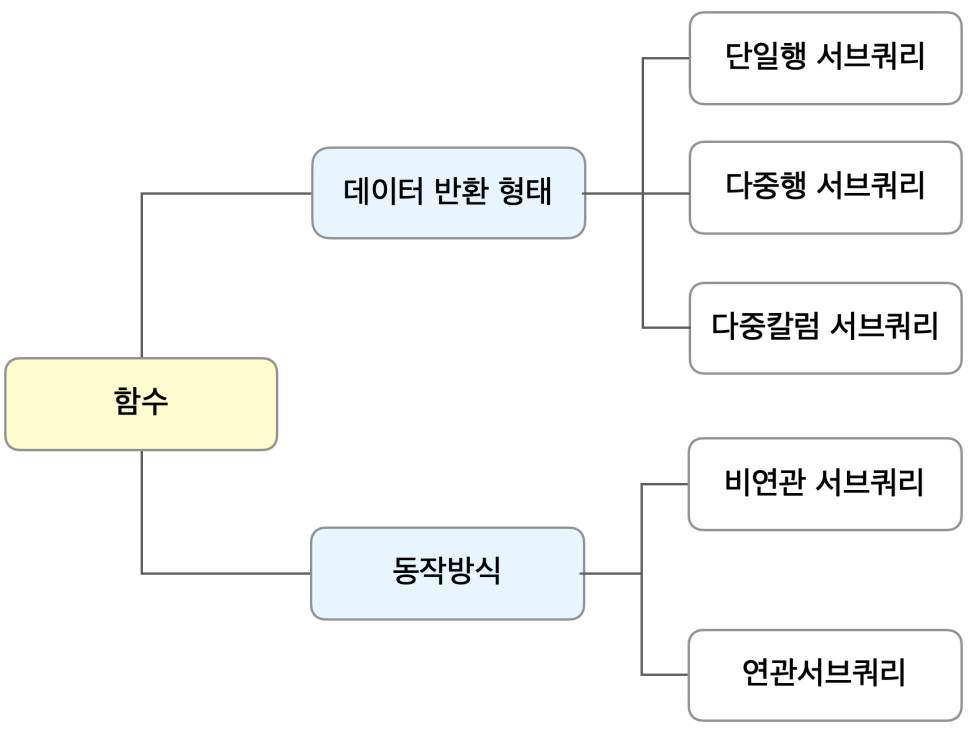
- 하나의 쿼리 안에 다른 작은 쿼리를 포함시키는 것
- 서브 쿼리는 주요 쿼리의 일부로 사용되어 조건을 만족시키거나 값을 가져올 때 유용
- 서브쿼리는 주인공 쿼리를 도와주는 작은 도우미 쿼리
- 데이터베이스에서 더 복잡한 정보를 가져오거나 원하는 조건을 충족시키기 위해 사용되는 도구

❕ 참조 - 서브쿼리 사용 시 주의사항

1. 서브쿼리는 소괄호`()`로 감싸서 사용합니다.
2. 서브쿼리는 단일행 또는 복수행 비교 연산자와 함께 사용이 가능합니다.
    - 단일 행 비교 연산자는 서브쿼리의 결과가 반드시 1건 이하여야 합니다.
    - 복수 행 비교 연산자는 서브쿼리의 결과 건수와 관련이 없습니다. 1건도 가능하고 여러 건도 가능합니다.
3. 서브쿼리에서는 ORDER BY를 사용할 수 없습니다.
    - ORDER BY 절은 SELECT 절에서 오직 한 개만 올 수 있기 때문에 메인쿼리의 마지막 문장에 위치해야 합니다.

☑️ 서브쿼리의 분류

서브쿼리는 동작방식과 데이터 반환 형태를 가지고 분류할 수 있고, 이러한 분류 기준에 따라 비연관/연관서브쿼리, 단일행/다중행/다중칼럼 서브쿼리로 나뉘게 됩니다. 상세 내용을 살펴보겠습니다.



▼ 2) 동작 방식에 따른 서브쿼리

서브쿼리는 메인쿼리 안에 포함된 종속적인 관계이기 때문에 논리적인 실행순서는 항상 메인쿼리에서 읽힌 데이터에 대해 서브쿼리에서 해당 조건이 만족하는지 확인하는 방식으로 수행되어야 합니다.

연관 서브쿼리는 서브쿼리가 메인쿼리의 값을 사용하는 경우이며 비연관 서브쿼리는 서브쿼리가 메인쿼리의 값을 사용하지 않는 경우를 의미합니다.

- **서브쿼리의 동작 방식**

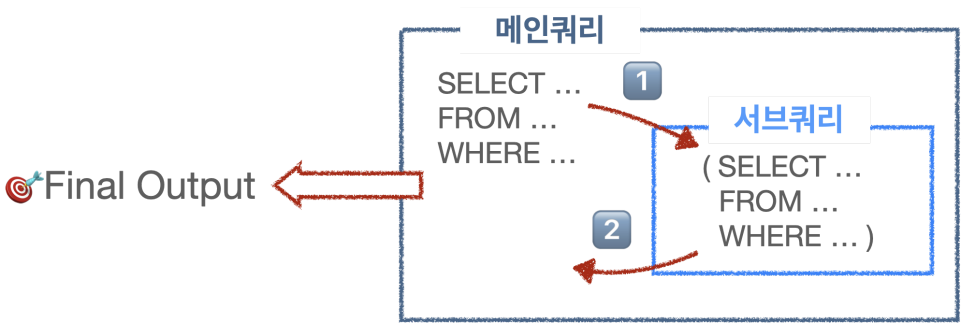
동작방식	설명
비연관 서브쿼리	- 서브쿼리가 메인쿼리 칼럼을 가지고 있지 않은 형태의 서브쿼리입니다. - 메인쿼리에 값(서브쿼리가 실행된 결과)를 제공하기 위한 목적으로 사용합니다.
연관 서브쿼리	- 서브쿼리가 메인쿼리의 값을 가지고 있는 형태의 서브쿼리입니다. - 메인쿼리가 먼저 수행되어, 읽힌 데이터를 서브쿼리에 조건이 맞는지 확인하고자 할 때 사용합니다.

☑️ 연관 서브쿼리

연관 서브쿼리는 서브쿼리 내에 메인쿼리 칼럼이 사용된 서브쿼리입니다. 즉, 서브쿼리의 값이 결정되는데 메인쿼리에 의존하는 것입니다.

- 예시
  - 각 부서별 평균 급여보다 급여 액수가 같거나 큰 직원들의 정보를 출력

```
SELECT A.EMPNO
     , A.ENAME
    , A.DEPTNO
    , A.SAL
FROM EMP A
WHERE A.SAL >= (
    SELECT AVG(X.SAL)
      FROM EMP X
     WHERE X.DEPTNO = A.DEPTNO
GROUP BY X.DEPTNO );
```



또한, EXISTS 서브쿼리는 항상 연관 서브쿼리로 사용되는데, EXISTS문은 서브쿼리의 결과가 참이라면 결과 집합에 포함시킵니다.
EXISTS 서브쿼리는 아무리 조건을 만족하는 건이 여러 건이더라도 조건을 만족하는 1건만 찾으면 추가적인 검색을 진행하지 않습니다.

** EXISTS 서브쿼리 : 서브쿼리 결과를 만족하는 값이 존재하는지 여부를 확인하는 조건을 의미합니다.*

- 실습

▼ 실습 데이터

```
CREATE TABLE PARTICIPANT (
partic_id NUMBER,
nation_id NUMBER,
main_sport_id NUMBER,
first_name VARCHAR2(30),
last_name VARCHAR2(30),
gender VARCHAR2(1),
height NUMBER,
weight NUMBER
);

INSERT INTO PARTICIPANT VALUES (1, 1, 101, 'John', 'Doe', 'M', 180, 75);
INSERT INTO PARTICIPANT VALUES (2, 2, 102, 'Jane', 'Smith', 'F', 165, 55);
INSERT INTO PARTICIPANT VALUES (3, 3, 103, 'Michael', 'Johnson', 'M', 175, 70);
INSERT INTO PARTICIPANT VALUES (4, 4, 104, 'Emily', 'Davis', 'F', 160, 50);
INSERT INTO PARTICIPANT VALUES (5, 5, 105, 'Chris', 'Brown', 'M', 190, 80);
INSERT INTO PARTICIPANT VALUES (6, 1, 101, 'Sarah', 'Wilson', 'F', 170, 60);
INSERT INTO PARTICIPANT VALUES (7, 2, 102, 'David', 'Lee', 'M', 175, 70);
INSERT INTO participant VALUES (8, 1, 102, 'Laura', 'Johnson', 'F', 160, 55);
```

```
CREATE TABLE NATION (
    nation_id NUMBER,
    country_name VARCHAR2(30),
    population NUMBER
);

INSERT INTO NATION VALUES (1, 'United States', 331002651);
INSERT INTO NATION VALUES (2, 'India', 1380004385);
INSERT INTO NATION VALUES (3, 'China', 1444216107);
INSERT INTO NATION VALUES (4, 'Brazil', 212559417);
INSERT INTO NATION VALUES (5, 'United Kingdom', 67886011);
INSERT INTO NATION VALUES (6, 'Canada', 37742154);
INSERT INTO NATION VALUES (7, 'Australia', 25788221);
```

```
CREATE TABLE SPORT (
    sport_id NUMBER PRIMARY KEY,
    sport_name VARCHAR2(20),
    max_weight NUMBER,
    min_weight NUMBER
);

INSERT INTO SPORT VALUES (101, 'Football', 100, 60);
INSERT INTO SPORT VALUES  (102, 'Basketball', 120, 50);
INSERT INTO SPORT VALUES  (103, 'Tennis', 90, 50);
INSERT INTO SPORT VALUES  (104, 'Swimming', 85, 45);
INSERT INTO SPORT VALUES  (105, 'Gymnastics', 80, 40);
```

> 대회에 참여한 국가의 이름을 출력해보세요.

```
SELECT DISTINCT country_name
FROM nation n
WHERE EXISTS (
    SELECT 1 -- * 동일
    FROM participant p
    WHERE p.nation_id = n.nation_id
);
```

COUNTRY_NAME
United States
Brazil
China
India
United Kingdom

☑️비연관 쿼리

만약 서브쿼리가 메인 쿼리의 어떤 것도 참조하지 않고 단독으로 사용되면 비연관쿼리입니다. 서브 쿼리가 우선 실행되고, 그 결과가 메인 쿼리의 WHERE 조건으로 이용되는 것입니다. 그러나 내부 쿼리는 외부 쿼리의 값과는 아무 상관이 없고, 단독으로도 쿼리가 실행될 수 있습니다.

- 예시

```
SELECT first_name, height
FROM PARTICIPANT
WHERE height > (SELECT AVG(height) FROM PARTICIPANT);
```

- 실습

> 참가 선수 중 두 번째로 키가 큰 선수를 출력해보세요.

```
SELECT MAX(height)
FROM participant
WHERE height NOT IN (SELECT MAX(height) FROM participant);
```

▼ 3) 반환 형태에 따른 서브쿼리

서브쿼리는 특정한 값을 메인쿼리에 반환하는데, 이때 몇 개의 행 혹은 칼럼을 반환하는지에 따라서 단일행, 다중행, 그리고 다중칼럼 서브쿼리로 구분할 수 있습니다.

- **반환 형태에 따른 서브쿼리**

동작방식	설명
단일행 서브쿼리 (Single Row)	- 실행 결과가 항상 1건 이하인 서브쿼리입니다. - 단일 행 비교 연산자인 =, <, ≤, >, ≥, <>과 함께 사용합니다.
다중행 서브쿼리 (Multi Row)	- 실행 결과가 여러 건인 서브쿼리입니다. - 다중 행 비교 연산자인 IN, ALL, ANY, SOME, EXISTS와 함께 사용합니다.
다중칼럼 서브쿼리 (Multi Column)	- 실행 결과로 여러 칼럼을 반환하는 서브쿼리입니다. - 메인쿼리의 조건절에 여러 칼럼을 동시 비교할 수 있습니다. - 서브쿼리와 메인쿼리에서 비교하고자 하는 칼럼 개수와 위치가 동일해야 합니다.

☑️ 단일행 서브쿼리

단일행 서브쿼리는 서브쿼리의 결과가 0건 혹은 1건인 SQL 문이 서브쿼리로 존재하는 SQL문을 말합니다. 서브쿼리가 단일행 비교 연산자(=, <, <=, >, >=, <>)와 함께 사용할 때는 서브쿼리의 결과 건수가 반드시 1건 이하여야 합니다. 만약 2건 이상을 반환한다면 런타임 오류가 발생하게 됩니다.

- 예시
  - 직원들이 받는 급여 액수의 평균보다 그 이상의 급여를 받고 있는 직원들의 정보를 출력

```
SELECT A.EMPNO, A.ENAME, A.SAL
FROM EMP A
WHERE A.SAL >= (SELECT AVG(K.SAL) AS MAX_SAL FROM EMP K)
ORDER BY A.SAL ;
```

- 실습

> 풋볼(101) 선수의 최대 몸무게 보다 무거운 사람의 이름과 주 종목 ID 그리고 몸무게를 출력해 보세요.

```
SELECT first_name, main_sport_id, weight
FROM participant
WHERE weight > (SELECT MAX(weight)
                FROM participant
                WHERE main_sport_id = 101);
```

☑️ 다중행 서브쿼리

다중행 서브쿼리는 서브 쿼리의 결과가 2건 이상인 SELECT문이 서브쿼리로 존재하는 것을 의미합니다.

서브쿼리의 결과가 2건 이상 반환될 수 있다면 반드시 다중행 비교 연산자**(IN, ALL, ANY, EXISTS 등)**와 함께 사용해야 합니다. 그렇지 않으면 SQL문은 오류를 반환하게 됩니다.

📌 **다중행 비교 연산자**

- **IN (서브쿼리)**
  - 서브쿼리 결과에 존재하는 임의의 값과 동일한 조건을 의미합니다.
- **ALL (서브쿼리)**
  - 서브쿼리 결과에 존재하는 모든 값을 만족하는 조건을 의미합니다.
- **ANY (서브쿼리) / SOME (서브쿼리)**
  - 서브쿼리 결과에 존재하는 어느 하나의 값이라도 만족하는 조건을 의미합니다.
- **EXISTS (서브쿼리)**
  - 서브쿼리 결과를 만족하는 값이 존재하는지 여부를 확인하는 조건을 의미합니다.

- 예시
  - DNAME이 'ACCOUNTING' 혹은 'SALES'인 DEPTNO에 소속되어 있는 직원들의 정보를 출력

```
--정상 실행
SELECT A.EMPNO
     , A.ENAME
     , A.DEPTNO
  FROM EMP A
 WHERE A.DEPTNO IN
                 (
                  SELECT K.DEPTNO
                    FROM DEPT K
                    WHERE K.DNAME IN ('ACCOUNTING', 'SALES')
                 )
ORDER BY A.DEPTNO ;

--error 발생
SELECT A.EMPNO
     , A.ENAME
     , A.DEPTNO
  FROM EMP A
WHERE A.DEPTNO = (
                  SELECT K.DEPTNO
                    FROM DEPT K
                   WHERE K.DNAME IN ('ACCOUNTING', 'SALES')
                 )
ORDER BY A.DEPTNO ;
```

이때 주의할 점은 다중행 서브쿼리를 사용하면서 단일행 서브쿼리용 연산자 `=` 를 사용하면 안 된다는 것입니다.

- 실습 1

> SPORT 테이블에서 각 스포츠 종목별로 키(height)가 평균 키보다 큰 선수들의 정보를 출력해 보세요

```
SELECT sport_id, sport_name, max_weight, min_weight
FROM SPORT
WHERE sport_id IN (
    SELECT DISTINCT main_sport_id
    FROM PARTICIPANT
    WHERE height > (
        SELECT AVG(height)
        FROM PARTICIPANT
    )
);
```

☑️ 다중칼럼 서브쿼리

다중칼럼 서브쿼리는 서브쿼리의 결과로 여러 개의 칼럼이 반환되어, 메인쿼리의 조건과 동시에 비교되는 것을 의미합니다. 예시를 통해서 알아보도록 하겠습니다.

- 예시
  - 부서별로 가장 높은 급여를 받는 사원을 찾는 SQL 문

```
SELECT deptno, dname, empno, ename, sal
FROM (
    SELECT e.deptno,
           d.dname,
           e.empno,
           e.ename,
           e.sal,
           RANK() OVER (PARTITION BY e.deptno ORDER BY e.sal DESC) AS sal_rank
    FROM emp e
    JOIN dept d ON e.deptno = d.deptno
)
WHERE sal_rank = 1;
```

** 다중칼럼 서브쿼리의 경우 SQL Server에서 지원하지 않습니다.*

- 실습

> 국가 ID가 1 (United States) 이고 성별이 'F' 인 선수의 정보를 출력해 보세요.

```
SELECT first_name, last_name, height, weight
FROM PARTICIPANT
WHERE (nation_id, gender) IN (
    SELECT nation_id, 'F'
    FROM NATION
    WHERE country_name = 'United States'
);
```

▼ 연습문제

☑️ 문제 1

Q. 문제	**<보기>의 설명은 어떤 서브 쿼리에 대한 설명인가?** <보기> a. 실행 결과로 여러 칼럼을 반환하는 서브쿼리이다. b.서브 쿼리와 메인쿼리에서 비교하고자하는 칼럼 개수와 칼럼의 위치가 동일해야 한다.
A. (1)	단일행 서브쿼리
A. (2)	다중행 서브쿼리
A. (3)	다중 칼럼 서브쿼리
A. (4)	연관 서브쿼리

▼ 정답

(3)

☑️ 문제 2

Q. 문제	**다중행 비교 연산자의 종류로 적절하지 않은 것은?**
A. (1)	ALL
A. (2)	ANY
A. (3)	EXISTS
A. (4)	INTEREST

▼ 정답

(4) 다중행 비교 연산자의 종류는 IN, ALL, ANY, SOME, EXISTS가 있다

☑️ 문제 3

Q. 문제	**서브쿼리에 대한 설명으로 옳은 것은?**
A. (1)	서브쿼리는 메인 쿼리의 결과와 독립적으로 실행된다
A. (2)	EXISTS는 서브쿼리의 결괏값이 존재하는지를 확인하는 조건이다
A. (3)	서브쿼리는 항상 SELECT 문에서만 사용된다
A. (4)	서브쿼리는 항상 WHERE 절에서만 사용된다

▼ 정답

(2)

03. 기타 서브쿼리와 뷰

✔️ **위치별 서브쿼리와 뷰에 대해 학습합니다.**

▼ 1) 위치별 서브쿼리

서브쿼리는 위치를 기준으로도 구분할 수 있습니다.

- SELECT
  - SELECT 절에 위치한 서브쿼리를 '**스칼라 서브쿼리**'라고 합니다.
- FROM
  - FROM 절에 위치한 서브쿼리를 '**인라인뷰 서브쿼리**'라고 합니다.
- WHERE
  - WHERE절에 위치한 서브쿼리를 '서브쿼리'라고 합니다.
- HAVING
  - HAVING절에 위치한 서브쿼리를 '서브쿼리'라고 합니다.
- INSERT문의 VALUES
  - INSERT문의 VALUES절에 위치한 서브쿼리를 '서브쿼리'라고 합니다.
- UPDATE문의 SET
  - UPDATE문의 SET절에 위치한 서브쿼리를 '서브쿼리'라고 합니다.

(1) SELECT

select절에서 사용하는 서브쿼리를 '**스칼라 서브쿼리**'라고 합니다. 스칼라 서브쿼리의 가장 큰 특징은 하나의 행과 하나의 칼럼(1 Row - 1 Column)만을 반환하는 것입니다.

- 예시

```
SELECT A.EMPNO , A.ENAME
      , A.DEPTNO
      , (SELECT L.DNAME FROM DEPT L WHERE L.DEPTNO = A.DEPTNO ) AS DNAME
FROM EMP A
WHERE A.DEPTNO IN (SELECT K.DEPTNO
                     FROM DEPT K
                    WHERE K.DNAME IN ('ACCOUNTING', 'SALES'))
ORDER BY A.DEPTNO;
```

쿼리 결과는 하나의 행으로만 나와야합니다. 만약 두 개 이상의 복수 행이 나오게 되면 위 쿼리는 실행되지 않습니다.

- 실습

> 다음 쿼리문의 결과를 비교해보세요.

```
SELECT ROWNUM, ENAME
FROM EMP
WHERE ROWNUM <=5;

SELECT ROWNUM, ENAME
FROM EMP
WHERE ROWNUM <= 5
ORDER BY ENAME;

SELECT ROWNUM, ENAME
FROM (SELECT ENAME FROM EMP ORDER BY ENAME)
WHERE ROWNUM <= 5;
```

(2) FROM

FROM 절에서 사용하는 서브쿼리를 '**인라인 뷰**'라고 합니다. FROM절에는 테이블명이 반드시 오게 되어 있는데, FROM 절에 위치한 서브쿼리의 결과는 마치 실행 시에 동적으로 생성된 테이블인 것처럼 사용할 수 있습니다.

> 📌 ❕ **참조 - 뷰(View)**
>
> - 하나 이상의 테이블이나 다른 뷰의 데이터를 볼 수 있게 하는 데이터베이스 객체로 실제 데이터를 가지고 있지 않은 상태입니다.
> - 반면에 테이블은 실제로 데이터를 가지고 있는 데이터베이스를 의미합니다.
> - 상세한 내용은 이번 챕터 마지막에 다루게 됩니다.

인라인 뷰는 SQL 문이 실행될 때만 임시적으로 생성되는 뷰이기 때문에 데이터베이스에 해당 정보가 저장되지 않습니다. 그렇기 때문에 일반적인 뷰를 정적 뷰(Static View)라고 하고 인라인 뷰를 동적 뷰(Dynamic View)라고도 합니다.

서브쿼리의 칼럼은 메인쿼리에서 사용할 수 없지만 인라인 뷰의 칼럼은 사용 가능한데, 이는 동적으로 생성된 테이블이기 때문입니다. 인라인 뷰를 사용하는 것은 조인 방식을 사용하는 것과 같기 때문에 칼럼을 자유롭게 참조할 수 있습니다.

- 예시
  - 인라인 뷰에 먼저 정렬을 수행하고 정렬된 결과 중에서 일부 데이터를 추출

```
-- Oracle

SELECT ENAME 이름, JOB 직업, COMM 수수료, SAL 급여
FROM (SELECT ENAME, JOB, COMM, SAL
      FROM emp
      WHERE COMM IS NOT NULL
      ORDER BY COMM DESC)
WHERE ROWNUM <= 4;
```

```
-- SQL Server

SELECT TOP(5) ENAME AS 이름,
       JOB          AS 직업,
       COMM         AS 수수료,
       SAL          AS 급여
FROM emp
WHERE COMM IS NOT NULL
ORDER BY COMM DESC;
```

- 실습

> EMP 테이블과 DEPT 테이블을 조인하여 직원의 정보와 부서 정보를 출력해 보세요

```
SELECT e.empno, e.ename, e.job, d.dname, d.loc
FROM (
    SELECT empno, ename, job, deptno
    FROM emp
) e
INNER JOIN dept d ON e.deptno = d.deptno;
```

(3) WHERE

WHERE 절 안에 들어있는 서브쿼리입니다. 가장 많이 사용하며 '서브쿼리'라고 했을 때 가장 먼저 떠올리는 서브쿼리입니다. 여러 개의 서브쿼리가 겹쳐 있기 때문에 '중첩서브쿼리 (nested subqueries)' 라고도 부릅니다.

- 예시
  - 각 부서별로 평균 급여를 계산하고, 이 평균 급여보다 높은 급여를 받는 직원의 정보를 출력해 보세요

```
SELECT empno, ename, deptno, sal
FROM emp
WHERE sal > (
    SELECT AVG(sal)
    FROM emp
);
```

- 실습

> 각 국가별로 평균 키를 계산하고, 이 평균 키보다 낮은 키를 가진 참가자의 정보를 출력해 보세요.

```
SELECT partic_id, first_name, last_name, nation_id, height
FROM PARTICIPANT
WHERE height < (
    SELECT AVG(height)
    FROM PARTICIPANT
);
```

(4) HAVING

WHERE 대신 HAVING 절에서도 서브쿼리를 사용할 수 있습니다. HAVING절에서 서브쿼리는 그룹함수와 함께 사용될 때 그룹핑된 결과에 대해 부가적인 조건을 주기 위해서 사용합니다.

- 예시

```
SELECT
      A.DEPTNO
      , COUNT(*) AS CNT
      , MAX(SAL) AS MAX_SAL
      , MIN(SAL) AS MIN_SAL
      , ROUND(AVG(SAL), 2) AS AVG_SAL
      , SUM(SAL) AS SUM_SAL
FROM EMP A
GROUP BY A.DEPTNO
HAVING AVG(SAL) > (SELECT AVG(K.SAL)
                     FROM EMP K
                    WHERE K.JOB = 'MANAGER' );
```

- 실습

> 전체 평균이 170이하의 평균보다 크다면 평균 몸무게 보다 무거운 선수들의 이름, 키, 몸무게, 종목 ID를 출력해보세요.

```
SELECT first_name, height, weight, main_sport_id
FROM participant
WHERE weight > (SELECT AVG(weight)
                FROM participant
```

```
                HAVING AVG(weight) > (SELECT AVG(weight)
                                      FROM participant
                                      WHERE height <= 170));
```

(5) UPDATE문의 SET

UPDATE시에도 서브쿼리를 사용하여 변경할 수 있습니다. 서브쿼리를 사용한 변경 작업을 할 때 주의할 점은 서브쿼리의 결과가 NULL을 반환할 때입니다. WHERE 절은 UPDATE 대상이 되는 데이터의 범위를 결정하게 되는데, WHERE 절이 누락되면 모든 데이터가 UPDATE 대상이 되므로 NULL 값으로 변경될 수 있기 때문입니다. 서브쿼리를 활용하여, SELECT한 데이터를 바로 UPDATE할 수도 있습니다.

- 예시
  - 지역이 DALLAS인 부서에 속한 직원들의 급여를 1000만큼 증가

```
UPDATE emp
SET sal = sal + 1000
WHERE deptno IN (
    SELECT deptno
    FROM dept
    WHERE loc = 'DALLAS'
);
```

- 실습

▼ 실습 데이터

```
CREATE TABLE Participant_B (
    partic_id NUMBER PRIMARY KEY,
    nation_id NUMBER,
    main_sport_id NUMBER,
    first_name VARCHAR2(30),
    last_name VARCHAR2(30),
    gender VARCHAR2(1),
    height NUMBER,
    weight NUMBER
);

-- 데이터 삽입
INSERT INTO Participant_B VALUES (1, 1, 3, 'John', 'Doe', 'M', 180, 75);
INSERT INTO Participant_B VALUES (2, 2, NULL, 'Jane', 'Smith', 'F', 165, 55);
INSERT INTO Participant_B VALUES (3, 3, NULL, 'Michael', 'Johnson', 'M', 175, 70);
INSERT INTO Participant_B VALUES (4, 4, 2, 'Emily', 'Davis', 'F', 160, 50);
INSERT INTO Participant_B VALUES (5, 5, NULL, 'Chris', 'Brown', 'M', 190, 80);
INSERT INTO Participant_B VALUES (6, 1, 3, 'Sarah', 'Wilson', 'F', 170, 60);
INSERT INTO Participant_B VALUES (7, 2, NULL, 'David', 'Lee', 'M', 175, 70);
```

```
CREATE TABLE sport_B (
    sport_id NUMBER PRIMARY KEY,
    sport_name VARCHAR2(30)
);

-- 데이터 삽입
INSERT INTO sport_B (sport_id, sport_name) VALUES (1, '100미터 달리기');
INSERT INTO sport_B (sport_id, sport_name) VALUES (2, '수영');
INSERT INTO sport_B (sport_id, sport_name) VALUES (3, '농구');
```

> 주 종목이 비어있는 선수들의 주 종목을 100미터 달리기의 id로 수정해 보세요.

```
UPDATE Participant_B
SET main_sport_id = (
    SELECT sport_id
    FROM Sport_B
    WHERE sport_name = '100미터 달리기'
)
WHERE main_sport_id IS NULL;
```

```
SELECT * FROM participant;
```

(6) INSERT문의 VALUES

서브쿼리를 통해 새로운 값을 집어넣을 수도 있습니다.

- 예시
  - INSERT_TEST 테이블을 생성 후 해당 테이블에 부서번호가 20인 직원의 가장 높은 급여를 삽입한다

```
CREATE TABLE INSERT_TEST (
  DEPT_NO NUMBER
, MAX_SAL_AMT NUMBER(15) );

INSERT INTO INSERT_TEST
VALUES (20, (SELECT MAX(SAL)
               FROM EMP
              WHERE DEPTNO = 20)
);
```

- 실습

> partic_id는 8 그리고 nation_id=2 남자 100미터 달리기 선수 'Ryan Thomas' 를 추가해주세요. sport 테이블에서 찾아서 main_sport_id를 작성해보세요.

```
INSERT INTO participant_B
(partic_id, nation_id, gender, first_name, last_name, main_sport_id)
VALUES (210, 2, 'M', 'Ryan', 'Thomas',
        (SELECT sport_id
         FROM sport_B
         WHERE sport_name='100미터 달리기' ));
```

서브쿼리는 여러 케이스를 함께 쓸 수도 있습니다. 스칼라 서브쿼리(SELECT절), 인라인뷰(FROM절), 서브쿼리(WHERE절)가 모두 쓰인 케이스를 통해 확인해 봅시다.

- 예시
  - 여러 서브쿼리가 섞인 개념

```
SELECT A.계좌번호 , B.고객번호 , B.고객명 , C.적립금,
      (SELECT 이름
      FROM 사원 A
       WHERE D.사원번호 = A.사원번호 ) 담당자_이름 -- 스칼라 서브쿼리
FROM 계좌 A
    (SELECT 고객번호, MAX(적립금) 적립금
    FROM 포인트
    WHERE 적립일 = '20211030') C -- 인라인 뷰
WHERE NOT EXISTS (SELECT 'X'
                  FROM 입금목록 B
                  WHERE A.계좌번호 = B.계좌번호
                  AND B.거래일자 LIKE '202109%') -- 서브쿼리
AND A.지점_위치 = '서울'
AND A.고객번호 = C.고객번호
```

▼ 2) 뷰(View)

- 데이터베이스에서 저장된 정보를 좀 더 편리하게 보여주기 위해 사용되는 가상의 테이블
- 이미 존재하는 데이터 테이블을 사용하여 원하는 정보를 필터링하거나 정리하여
  새로운 "가상" 테이블을 만드는 것

뷰는 하나 이상의 테이블이나 다른 뷰의 데이터를 볼 수 있게 하는 데이터베이스 객체입니다. 실제 데이터는 뷰를 구성하는 테이블에 담겨 있지만 마치 테이블처럼 사용할 수 있습니다. 다만, 테이블과는 조금 다른 특징이 있습니다. 테이블은 실제로 데이터를 가지고 있는 반면 뷰

(View)는 실제 데이터를 가지고 있지 않습니다. 뷰는 단지 뷰 정의(View Definition)만을 가지고 있습니다. 질의에서 뷰가 사용되면 뷰 정의를 참조해서 DBMS 내부적으로 질의를 재작성(Rewrite)하여 수행하는 것입니다.

✅ 뷰 사용의 장점

장점	설명
독립성	테이블 구조가 변경되어도 뷰를 사용하는 응용 프로그램은 변경하지 않아도 됩니다.
편리성	복잡한 질의를 뷰로 생성함으로써 관련 질의를 단순하게 작성할 수 있습니다.
보안성	숨기고 싶은 정보가 존재한다면, 뷰를 생성할 때 해당 칼럼을 빼고 생성함으로써 사용자에게 정보를 감출 수 있습니다.

뷰는 실제 데이터를 가지고 있지 않지만 테이블이 수행하는 역할을 수행하기 때문에 '가상 테이블(Virtual Table)'이라고도 합니다. 뷰는 테이블뿐만 아니라 다른 뷰를 참조해 새로운 뷰를 만들어 사용할 수 있습니다. 데이터를 본다는 의미가 있으므로 뷰의 정의는 데이터를 조회하는 SELECT문으로 구성됩니다.

- 예시
  - CREATE VIEW 문을 통해서 생성 가능

```
CREATE VIEW V_DEPT_EMP AS
SELECT E.EMPNO,
       E.ENAME,
       E.JOB,
       E.SAL,
       D.DNAME
FROM DEPT D,
     EMP E
WHERE D.DEPTNO = E.DEPTNO;
```

- 예시
  - 이미 존재하는 뷰를 참조해서도 생성 가능

```
CREATE VIEW V_DEPT_EMP_FILTER AS
SELECT ENAME,
       JOB
FROM V_DEPT_EMP
WHERE EMPNO IN (7698, 7788);
```

- 예시
  - 뷰를 사용하여 데이터를 조회

```
SELECT ENAME,
       JOB
FROM V_DEPT_EMP
WHERE EMPNO IN (7698, 7788);
```

- 예시
  - 뷰를 제거하기 위해서는 DROP VIEW 문을 사용

```
DROP VIEW V_DEPT_EMP;
DROP VIEW V_DEPT_EMP_FILTER;
```

- 실습

▼ 실습 데이터

```
CREATE TABLE PARTICIPANT (
partic_id NUMBER,
nation_id NUMBER,
```

```
main_sport_id NUMBER,
first_name VARCHAR2(30),
last_name VARCHAR2(30),
gender VARCHAR2(1),
height NUMBER,
weight NUMBER
);

INSERT INTO PARTICIPANT VALUES (1, 1, 101, 'John', 'Doe', 'M', 180, 75);
INSERT INTO PARTICIPANT VALUES (2, 2, 102, 'Jane', 'Smith', 'F', 165, 55);
INSERT INTO PARTICIPANT VALUES (3, 3, 103, 'Michael', 'Johnson', 'M', 175, 70);
INSERT INTO PARTICIPANT VALUES (4, 4, 104, 'Emily', 'Davis', 'F', 160, 50);
INSERT INTO PARTICIPANT VALUES (5, 5, 105, 'Chris', 'Brown', 'M', 190, 80);
INSERT INTO PARTICIPANT VALUES (6, 1, 101, 'Sarah', 'Wilson', 'F', 170, 60);
INSERT INTO PARTICIPANT VALUES (7, 2, 102, 'David', 'Lee', 'M', 175, 70);
INSERT INTO PARTICIPANT VALUES (8, 1, 102, 'Laura', 'Johnson', 'F', 160, 55);
```

```
CREATE TABLE NATION (
    nation_id NUMBER,
    country_name VARCHAR2(30),
    population NUMBER
);

INSERT INTO NATION VALUES (1, 'United States', 331002651);
INSERT INTO NATION VALUES (2, 'India', 1380004385);
INSERT INTO NATION VALUES (3, 'China', 1444216107);
INSERT INTO NATION VALUES (4, 'Brazil', 212559417);
INSERT INTO NATION VALUES (5, 'United Kingdom', 67886011);
INSERT INTO NATION VALUES (6, 'Canada', 37742154);
INSERT INTO NATION VALUES (7, 'Australia', 25788221);
```

> 참가자와 국가 테이블을 JOIN 한 v_p_nation VIEW 를 생성하세요.

```
CREATE VIEW v_p_nation AS
SELECT p.first_name,
       p.last_name,
       p.main_sport_id,
       p.height,
       p.weight,
       n.country_name,
       n.population
FROM participant p, nation n
WHERE p.nation_id = n.nation_id;
```

▼ 연습문제

☑️ 문제 1

Q. 문제	**아래 <보기> 에서 ㉠ 에 들어갈 알맞은 단어는?** <보기> 1. 뷰는 실제 데이터를 가지고 있지 않지만 테이블이 수행하는 역할을 수행한다. 2. 뷰는 ㉠ VIEW AS 문으로 정의가 가능하다
A. (1)	VIRTUAL
A. (2)	FROM
A. (3)	SELECT
A. (4)	CREATE

▼ 정답

(4) 뷰는 CREATE VIEW AS문으로 생성한다

☑️ 문제 2

Q. 문제	**뷰의 특징과 설명으로 잘못된 것은?**
A. (1)	독립성 : 테이블 구조가 변경되어도 뷰를 사용하는 응용프로그램은 변경하지 않아도 된다
A. (2)	편리성 : 복잡한 질의를 뷰로 생성함으로써 관련 질의를 단순하게 작성할 수 있다
A. (3)	보안성 : 직원들의 급여 정보와 같은 숨겨야하는 정보가 있다면 해당 칼럼을 빼고 뷰 생성이 가능하다

A. (4)	물리성 : DBMS 내부의 객체로 존재하여 테이블 내에 존재하는 데이터를 물리적으로 관리할 수 있다

▼ 정답

(4) 뷰는 실제 테이블이 아니며 DBMS내부의 객체로 존재하지 않는다

☑️ 문제 3

Q. 문제	**서브쿼리 중에서 반드시 한 행과 한 칼럼만 반환하는 것은?**
A. (1)	Inline view
A. (2)	Subquery
A. (3)	Scala Subquery
A. (4)	Update Subquery

▼ 정답

(3) 스칼라 서브쿼리