

65. 다음 중 **순수 관계 연산자**에 해당하지 않는 것은?

- ① SELECT
- ② UPDATE
- ③ JOIN
- ④ DIVIDE

핵심정리)

### 순수 관계 연산자와 SQL 문장 비교

- SELECT 연산은 WHERE 절로 구현
- PROJECT 연산은 SELECT 절로 구현
- (NATURAL) JOIN 연산은 다양한 JOIN 기능으로 구현
- DIVIDE 연산은 현재 사용되지 않음

해설 :

순수 관계 연산자에는 **SELECT, PROJECT, JOIN, DIVIDE** 가 있다.

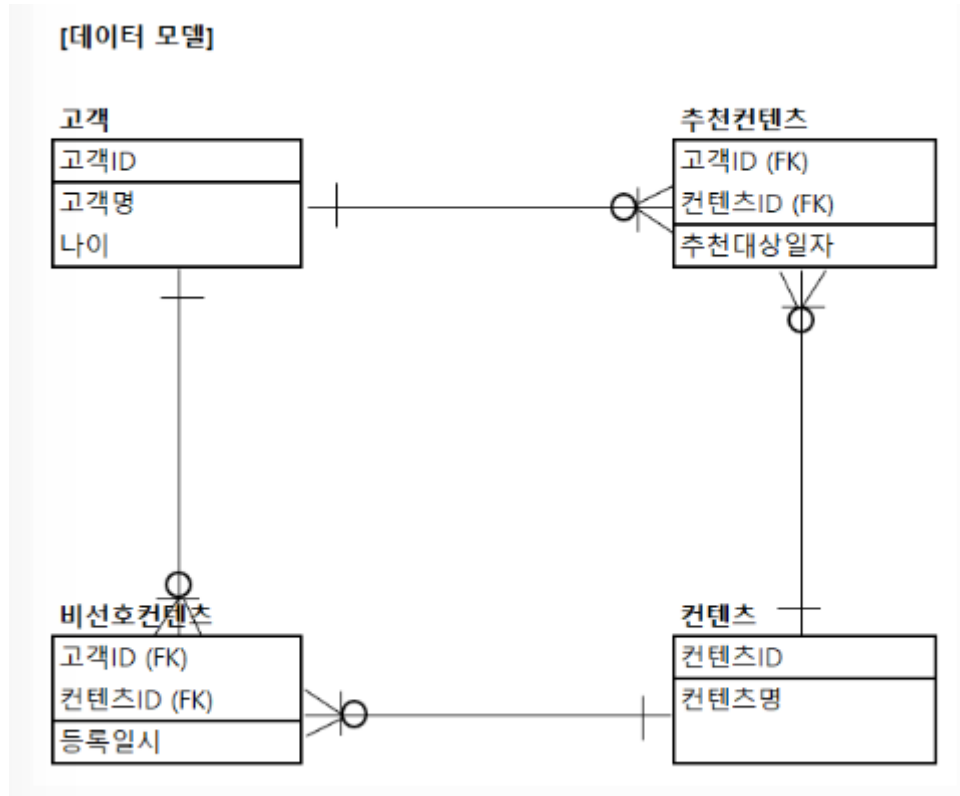
정답 : ②

---

66. 다음 중 아래 데이터 모델을 참고하여 설명에 맞게 올바르게 작성한 SQL 문장을 2개 고르시오.

- 아 래 -

[ 데이터 모델 ]



[ 설명 ]

우리는 매일 배치작업을 통하여 고객에게 추천할 컨텐츠를 생성하고 고객에게 추천서비스를 제공한다.

추천 컨텐츠가 엔터티에서 언제 추천을 해야 하는지를 정의하는 추천대상일자가 있어 해당일자에만 컨텐츠를 추천해야 한다.

또한 고객이 컨텐츠를 추천 받았을 때 선호하는 컨텐츠가 아닌 경우에는 고객이 비선호 컨텐츠로 분류하여 더 이상 추천받기를 원하지 않는다. 그러므로 우리는 비선호 컨텐츠 엔터티에 등록된 데이터에 대해서는 추천을 수행하지 않아야 한다.

※ 배치작업이란? 어떤 처리를 연속적으로 하는 것이 아니고 일정량씩 나누어 처리하는 경우 그 일정량을 배치(batch)라고 한다.

배치의 원뜻은 한 묶음이라는 의미다. [기계공학용어사전]

예) 상품을 주문하는 로직은 그당시에 발생하는 트랜잭션에 대한 처리이므로 배치작업이라 표현하지는 않는다. 하지만 상품별 주문량을 집계하는 로직의 경우 특정조건(기간등)으로 일과월리를 해야함으로 배치작업이라 표현할 수 있다.

① SELECT C.컨텐츠 ID, C.컨텐츠명  
FROM 고객 A INNER JOIN 추천콘텐츠 B  
ON (A.고객 ID = B.고객 ID) INNER JOIN 컨텐츠 C  
ON (B.컨텐츠 ID = C.컨텐츠 ID)  
WHERE A.고객 ID = #custId#  
AND B.추천대상일자 = TO\_CHAR(SYSDATE, 'YYYY.MM.DD')  
AND NOT EXISTS (SELECT X.컨텐츠 ID  
FROM 비선호컨텐츠 X  
WHERE X.고객 ID = B.고객 ID);

② SELECT C.컨텐츠 ID, C.컨텐츠명  
FROM 고객 A INNER JOIN 추천콘텐츠 B  
ON (A.고객 ID = #custId# AND A.고객 ID = B.고객 ID) INNER JOIN 컨텐츠 C  
ON (B.컨텐츠 ID = C.컨텐츠 ID) RIGHT OUTER JOIN 비선호컨텐츠 D  
ON (B.고객 ID = D.고객 ID AND B.컨텐츠 ID = D.컨텐츠 ID)  
WHERE B.추천대상일자 = TO\_CHAR(SYSDATE, 'YYYY.MM.DD')  
AND B.컨텐츠 ID IS NOT NULL;

③ SELECT C.컨텐츠 ID, C.컨텐츠명  
FROM 고객 A INNER JOIN 추천콘텐츠 B  
ON (A.고객 ID = B.고객 ID) INNER JOIN 컨텐츠 C  
ON (B.컨텐츠 ID = C.컨텐츠 ID) LEFT OUTER JOIN 비선호컨텐츠 D  
ON (B.고객 ID = D.고객 ID AND B.컨텐츠 ID = D.컨텐츠 ID)  
WHERE A.고객 ID = #custId#  
AND B.추천대상일자 = TO\_CHAR(SYSDATE, 'YYYY.MM.DD')  
AND D.컨텐츠 ID IS NULL;

④ SELECT C.컨텐츠 ID, C.컨텐츠명  
FROM 고객 A INNER JOIN 추천콘텐츠 B  
ON (A.고객 ID = #custId# AND A.고객 ID = B.고객 ID) INNER JOIN 추천콘텐츠 B  
ON (B.컨텐츠 ID = C.컨텐츠 ID)  
WHERE B.추천대상일자 = TO\_CHAR(SYSDATE, 'YYYY.MM.DD')  
AND NOT EXISTS (SELECT X.컨텐츠 ID  
FROM 비선호컨텐츠 X  
WHERE X.고객 ID = B.고객 ID  
AND X.컨텐츠 ID = B.컨텐츠 ID)

핵심정리)

## ANSI/ISO SQL 에서 표시하는 FROM 절의 JOIN 형태

- INNER JOIN
- NATURAL JOIN
- USING 조건절
- ON 조건절
- CROSS JOIN
- OUTER JOIN(LEFT, RIGHT, FULL)

해설 :

① NOT EXIST 절의 연관서브쿼리에 X.컨텐츠 ID = B.컨텐츠 ID 가 존재하지 않아  
단 하나의 컨텐츠라도 비선호로 등록한 고객에 대해서는 모든 컨텐츠가 추천에서  
배제 된다.

② 추천컨텐츠를 기준으로 비선호컨텐츠와의 LEFT OUTER JOIN 이 수행되고  
비선호컨텐츠의 컨텐츠 ID 에 대해서 IS NULL 조건(③ 번과 같이)이 있다면 정확히  
비선호 컨텐츠만 필터링할 수 있다.

(고객이 비선호로 등록하지 않은 컨텐츠는 추천컨텐츠에만 등록 되어있으므로)

정답 : ③, ④

67. 아래는 어느 회사의 생산설비를 위한 데이터 모델의 일부에 대한 설명으로  
가장 적절한 것을 2 개 고르시오.



① 제품, 생산제품, 생산라인 엔터티를 Inner Join 하기 위해서 생산제품 엔터티는  
WHERE 절에 최소 2 번이 나타나야 한다.

② 제품과 생산라인 엔터티를 Join 시 적절한 Join 조건이 없으므로 카티시안  
곱(Cartesian Product)이 발생한다.

③ 제품과 생산라인 엔터티에는 생산제품과 대응되지 않는 레코드는 없다.

④ 특정 생산라인번호에서 생산되는 제품의 제품명을 알기위해서는 제품, 생산제품, 생산라인까지 3 개의 엔터티의 Inner Join 이 필요하다.

핵심정리)

## INNER JOIN

INNER JOIN 은 OUTER(외부) JOIN 과 대비하여 내부 JOIN 이라고 하며 JOIN 조건에서 동일한 값이 있는 행만 반환한다.

해설 :

③ 데이터 모델을 보면 제품과 생산라인 엔터티에는 생산제품과 대응되지 않는 레코드가 있을 수 있다.

④ 특정 생산라인에서 생산되는 제품의 제품명을 알기위해서는 제품과 생산제품까지 2 개의 엔터티만을 Inner Join 하면 된다.

정답 : ①, ②

68. 아래의 테이블 스키마 정보를 참고하여,  
다음 중 '구매 이력이 있는 고객 중 구매 횟수가 3 회 이상인 고객의 이름과 등급을 출력하시오.'

라는 질의에 대해 아래 SQL 문장의 ㉠, ㉡ 에 들어 갈 구문으로 가장 적절한 것은?

- 아 래 -

### [ 테이블 ]

고객(고객번호(PK), 이름, 등급)

구매정보(구매번호(PK), 구매금액, 고객번호(FK))

\* 구매정보 테이블의 고객번호는 고객 테이블의 고객번호를 참조하는 외래키(Foreign Key)이다.

### [ SQL 문장 ]

SELECT A.이름, A.등급

FROM 고객 A

㉠

GROUP BY A.이름, A.등급

㉡

① ㉠ : INNER JOIN 구매정보 B ON A.고객번호 = B.고객번호

㉡ : HAVING SUM(B.구매번호) >=3

② ㉠ : INNER JOIN 구매정보 B ON A.고객번호 = B.고객번호

㉡ : HAVING COUNT(B.구매번호) >=3

③ ㉠ : LEFT OUTER JOIN 구매정보 B ON A.고객번호 = B.고객번호

㉡ : HAVING SUM(B.구매번호) >=3

④ ㉠ : INNER JOIN 구매정보 B ON A.고객번호 = B.고객번호

㉡ : WHERE B.구매번호 >=3

해설 :

구매이력이 있어야 하므로 **INNER JOIN** 이 필요하며, 구매 횟수이므로 **COUNT** 함수를 사용한다.

정답 : ②

69. 아래는 어느 회사의 정산 데이터 모델의 이불이며 고객이 서비스를 사용한 시간대에 따라 차등 단가를 적용하려고 한다.

다음 중 시간대별사용량 테이블을 기반으로 고객별 사용금액을 추출하는 SQL 로 가장 적절한 것은?



① SELECT A.고객 ID, A.고객명, SUM(B.사용량 \* C.단가) AS 사용금액  
FROM 고객 A INNER JOIN 시간대별사용량 B  
ON (A.고객 ID = B.고객 ID) INNER JOIN 시간대구간 C  
ON (B.사용시간대 <= C.시작시간대 AND B.사용시간대 >= C.종료시간대)  
GROUP BY A.고객 ID, A.고객명  
ORDER BY A.고객 ID, A.고객명;

② SELECT A.고객 ID, A.고객명, SUM(B.사용량 \* C.단가) AS 사용금액  
FROM 고객 A INNER JOIN 시간대별사용량 B INNER JOIN 시간대구간 C  
ON (A.고객 ID = B.고객 ID AND B.사용시간대  
BETWEEN C.시작시간대 AND C.종료시간대)  
GROUP BY A.고객 ID, A.고객명  
ORDER BY A.고객 ID, A.고객명;

③ SELECT A.고객 ID, A.고객명, SUM(B.사용량 \* C.단가) AS 사용금액  
FROM 고객 A INNER JOIN 시간대별사용량 B  
ON (A.고객 ID = B.고객 ID) INNER JOIN 시간대구간 C  
ON B.사용시간대 BETWEEN C.시작시간대 AND C.종료시간대  
GROUP BY A.고객 ID, A.고객명  
ORDER BY A.고객 ID, A.고객명;

④ SELECT A.고객 ID, A.고객명, SUM(B.사용량 \* C.단가) AS 사용금액  
FROM 고객 A INNER JOIN 시간대별사용량 B  
ON (A.고객 ID = B.고객 ID) BETWEEN JOIN 시간대구간 C

GROUP BY A.고객 ID, A.고객명  
ORDER BY A.고객 ID, A.고객명;

해설 :

- ① 두 번째 ON 절이 B.사용시간대 BETWEEN C.시작시간대 AND C.시작시간대 가 되어야 한다.
- ② INNER JOIN 구문 오류가 발생한다.
- ④ BETWEEN JOIN 이란 구문은 없다. 구문 오류가 발생한다.

정답 : ③

---

70. 다음 중 팀(Team) 테이블과 구장(STADIUM) 테이블의 관계를 이용해서 소속팀이 가지고 있는 전용구장의 정보를 팀의 정보와 함께 출력하는 SQL 을 작성할 때 결과가 다른 것은?

- ① SELECT T.REGION\_NAME, T.TEAM\_NAME, T.STADIUM\_ID,  
S.STADIUM\_NAME  
FROM TEAM T INNER JOIN STADIUM S  
USING (T.STADIUM\_ID = S.STADIUM\_ID);
- ② SELECT TEAM REGION\_NAME, TEAM.TEAM\_NAME,  
TEAM.STADIUM\_ID, STADIUM.STADIUM\_NAME  
FROM TEAM INNER JOIN STADIUM\_NAME  
ON (TEAM.STADIUM\_ID =  
STADIUM.STADIUM\_ID);
- ③ SELECT T. REGION\_NAME, T.TEAM\_NAME, T.STADIUM\_ID,  
S.STADIUM\_NAME  
FROM TEAM T, STADIUM S  
WHERE T.STADIUM\_ID = S.STADIUM\_ID;
- ④ SELECT TEAM.REGION\_NAME, TEAM.TEAM\_NAME,  
TEAM. STADIUM\_ID, STADIUM.STADIUM\_NAME  
FROM TEAM, STADIUM  
WHERE TEAM.STADIUM\_ID = STADIUM.STADIUM\_ID;

해설 : **TEAM, STADIUM** 두 테이블을 조인하여 사용한다.

- ① **USING** 조건절을 이용한 **EQUI JOIN** 에서도

**NATURAL JOIN** 과 마찬가지로 **JOIN** 칼럼에 대해서는 **ALIAS** 나 테이블 이름과 같은 접두사를 붙일 수 없다.

지문 1 은 SYNTAX 에러 발생함.

USING T.STADIUM\_ID = S.STADIUM\_ID

→ USING (STADIUM\_ID)

SELECT T.REGION\_NAME, T.TEAM\_NAME, T.STADIUM\_ID, S.STADIUM\_NAME

→ SELECT T.REGION\_NAME, T.TEAM\_NAME, STADIUM\_ID, S.STADIUM\_NAME

정답 : ①

---

71. 아래의 사례 1 은 **Cartesian Product** 를 만들기 위한 **SQL** 문장이며

사례 1 과 같은 결과를 얻기 위해 사례 2 **SQL** 문장의 ㉠ 안에 들어갈 내용을 작성하시오.

- 아 래 -

[ 사례 1 ]  
SELECT ENAME, DNAME  
FROM EMP, DEPT  
ORDER BY ENAME;  
[ 사례 2 ]  
SELECT ENAME, DNAME  
FROM EMP Ⓢ DEPT  
ORDER BY ENAME;

핵심정리)

## CROSS JOIN

테이블 간 JOIN 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 말한다.

결과는 양쪽 집합의  $M \times N$  건의 데이터 조합이 발생한다.

해설 :

**CROSS JOIN** 은 E.F.CODD 박사가 언급한 일반 집합 연산자의 **PRODUCT** 의 개념으로

테이블 간 **JOIN** 조건이 없는 경우 생길 수 있는 모든 데이터의 조합을 말한다.

조건절이 없거나 **CROSS JOIN** 키워드를 사용할 수 있다.

정답 : **CROSS JOIN**

72. 다음 중 아래 테이블들을 대상으로 SQL 문장을 수행한 결과로 가장 적절한 것은?

- 아 래 -

[ 테이블 : OS ]

OSID(PK)	OS 명	
100	Android	
200	iOS	



300	Bada	
[ 테이블 : 단말기 ]		
단말기 ID(PK)	단말기명	OSID(FK)
1000	A1000	100
2000	B2000	100
3000	C3000	200
4000	D3000	300
[ 테이블 : 고객 ]		
고객번호(PK)	고객명	단말기 ID(FK)
11000	홍길동	1000
12000	강감찬	NULL
13000	이순신	NULL
14000	안중근	3000

15000	고길동	4000
16000	이대로	4000

**[ SQL ]**  
**SELECT** A.고객번호, A.고객명, B.단말기 ID, B.단말기명, C.OSID, C.OS 명  
**FROM** 고객 A **LEFT OUTER JOIN** 단말기 B  
**ON** (A.고객번호 IN (11000, 12000) **AND** A.단말기 ID = B.단말기 ID) **LEFT OUTER JOIN** OS C  
**ON** (B.OSID = C.OSID)  
**ORDER BY** A.고객번호;

①

고객번호	고객명	단말기 ID	단말기명	OSID	OS 명
11000	홍길동	1000	A1000	100	Android
12000	강감찬	NULL	NULL	NULL	NULL
13000	이순신	NULL	NULL	NULL	NULL
14000	안중근	NULL	NULL	NULL	NULL
15000	고길동	NULL	NULL	NULL	NULL
16000	이대로	NULL	NULL	NULL	NULL

②

고객번호	고객명	단말기 ID	단말기명	OSID	OS 명
------	-----	--------	------	------	------

11000	홍길동	1000	A1000	100	Android
12000	강감찬	NULL	NULL	NULL	NULL

③

고객번호	고객명	단말기 ID	단말기명	OSID	OS 명
11000	홍길동	1000	A1000	100	Android

④

고객번호	고객명	단말기 ID	단말기명	OSID	OS 명
11000	홍길동	1000	A1000	100	Android
12000	강감찬	NULL	NULL	NULL	NULL
13000	이순신	NULL	NULL	NULL	NULL
14000	안중근	3000	C3000	200	iOS
15000	고길동	4000	D4000	300	Bada
16000	이대로	4000	D4000	300	Bada

해설 :

**WHERE 절**에 A.고객번호 IN(11000, 12000) 조건을 넣었다면 정답은 ② 번이 되었을 것이나,

**ON 절**에 A.고객번호 IN (11000, 12000) 조건을 넣었기 때문에 모든 고객에 대해서 출력을 하되

**JOIN** 대상 데이터를 고객번호 11000 과 12000 으로 제한되어 ① 번과 같은 결과가 출력된다.

정답 : ①

73. 다음 중 아래(1), (2), (3)의 SQL 에서 실행결과가 같은 것은?

아 래

```
(1) SELECT A.ID, B.ID
FROM TBL1 A FULL OUTER JOIN TBL2 B
ON A.ID = B.ID
(2) SELECT A.ID, B.ID
FROM TBL1 A LEFT OUTER JOIN TBL2 B
ON A.ID = B.ID
UNION
SELECT A.ID, B.ID
FROM TBL1 A RIGHT OUTER JOIN TBL2 B
ON A.ID = B.ID
(3) SELECT A.ID, B.ID
FROM TBL1 A, TBL2 B
WHERE A.ID = B.ID
UNION ALL
SELECT A.ID, NULL
FROM TBL1 A
WHERE NOT EXISTS (SELECT 1 FROM TBL2 B WHERE A.ID = B.ID)
UNION ALL
SELECT NULL, B.ID
FROM TBL2 B
WHERE NOT EXISTS (SELECT 1 FROM TBL1 A WHERE B.ID = A.ID)
```

① 1, 2

② 1, 3

③ 2, 3

④ 1, 2, 3

핵심정리)

### **FULL OUTER JOIN**

조인 수행시 좌측, 우측 테이블의 모든 데이터를 읽어 **JOIN** 하여 결과를 생성한다.

즉, **TABLE A** 와 **B** 가 있을 때 (**TABLE 'A', 'B'** 모두 기준이 됨),

**RIGHT OUTER JOIN** 과 **LEFT OUTER JOIN** 의 결과를 합집합으로 처리한 결과와 동일하다.

해설 :

보기의 3 개의 SQL 은 모두 Full Outer Join 과 동일한 결과를 반환한다.

정답 : ④

74. 아래의 **EMP** 테이블과 **DEPT** 테이블에서 밑줄 친 속성은 주키이며 **EMP.C**는 **DEPT**와 연결된 외래키이다.

**EMP** 테이블과 **DEPT** 테이블을 LEFT, FULL, RIGHT 외부조인(outer join)하면 생성되는 결과 건수로 가장 적절한 것은?

- 아 래 -		
<b>EMP</b> 테이블		
<b><u>A</u></b>	<b>B</b>	<b>C</b>
1	b	w
3	d	w
5	y	y
<b>DEPT</b> 테이블		
<b><u>C</u></b>	<b>D</b>	<b>E</b>
w	1	10
z	4	11
v	2	22

① 3 건, 5 건, 4 건

② 4 건, 5 건, 3 건

③ 3 건, 4 건, 4 건

④ 3 건, 4 건, 5 건

핵심정리)

## OUTER JOIN 문장 예시

- **LEFT OUTER JOIN**

```
SELECT X.KEY1,  
Y.KEY2  
FROM TAB1 X LEFT OUTER JOIN  
TAB2 Y  
ON (X.KEY1=Y.KEY2)
```

- **RIGHT OUTER JOIN**

```
SELECT X.KEY1,  
Y.KEY2  
FROM TAB1 X RIGHT OUTER JOIN  
TAB2 Y  
ON (X.KEY1=Y.KEY2)
```

- **FULL OUTER JOIN**

```
SELECT X.KEY1,  
Y.KEY2  
FROM TAB1 X FULL OUTER JOIN  
TAB2 Y  
ON (X.KEY1=Y.KEY2)
```

해설 : 주키와 외래키는 영향을 미치지 않는다.

### LEFT OUTER JOIN

A	B	C	D	E
1	b	w	1	10
3	d	w	1	10
5	y	y		

### FULL OUTER JOIN

A	B	C	D	E
1	b	w	1	10

3	d	w	1	10
5	y	y		
		z	4	11
		v	2	22

#### RIGHT OUTER JOIN

A	B	C	D	E
1	b	w	1	10
3	d	w	1	10
		z	4	11
		v	2	22

정답 : ①

75. 신규 부서의 경우 일시적으로 사원이 없는 경우도 있다고 가정하고  
**DEPT** 와 **EMP** 를 조인하되 사원이 없는 부서 정보도 같이 출력하도록 할 때,  
아래 SQL 문장의 ㉠ 안에 들어갈 내용을 기술하시오.

- 아 래 -

```
SELECT E.ENAME D.DEPTNO, D.DNAME
FROM DEP D ㉠ EMP E
ON D.DEPTNO = E.DEPTNO;
```

해설 :

**LEFT OUTER JOIN** 은 좌측 테이블이 기준이 되어 결과를 생성한다.

즉, TABLE A 와 B 가 있을 때 (TABLE 'A'가 기준이 됨),

A 와 B 를 비교해서 B 의 JOIN 칼럼에서 같은 값이 있을 때 B 테이블에서 해당 데이터를 가져오고,

B 의 JOIN 칼럼에서 같은 값이 없는 경우에는 B 테이블에서 가져오는 칼럼들은 **NULL** 값으로 채운다.

그리고 **LEFT JOIN** 으로 **OUTER** 키워드를 생략해서 사용할 수 있다.

정답 : **LEFT** 또는 **LEFT OUTER JOIN**

76. 다음 중 아래와 같은 데이터 상황에서 **SQL** 의 수행 결과로 가장 적절한 것은?

- 아 래 -			
TAB1		TAB2	
C1	C2	C1	C2
A	1	B	2
B	2	C	3
C	3	D	4
D	4		
E	5		
SELECT * FROM TAB1 A LEFT OUTER JOIN TAB2 B ON ( A.C1 = B.C1 AND B.C2 BETWEEN 1 AND 3)			



C1	C2	C1	C2
A	1		
B	2	B	2
C	3	C	3
D	4	D	4
E	5		

②

C1	C2	C1	C2
A	1		
B	2	B	2
C	3	C	3
D	4		
E	5		

③

C1	C2	C1	C2
A	1		
B	2	B	2
C	3	C	3

④

C1	C2	C1	C2
A	1		
B	2	B	2
C	3	C	3
D	4	D	4

해설 :

아우터 조인에서 ON 절은 조인할 대상을 결정한다. 그러나 기준 테이블은 항상 모두 표시된다.

결과 건에 대한 필터링은 WHERE 절에서 수행된다.

정답 : ②


---

77. 아래와 같은 데이터 모델에서 **ORACLE** 을 기준으로 **SQL** 을 작성하였다.

그러나 SQL Server 에서도 동일한 결과를 보장할 수 있도록 ANSI 구문으로 SQL 을 변경하려고 한다.

다음 중 아래의 SQL 을 ANSI 표준 구문으로 변경한 것으로 가장 적절한 것은?

- 아 래 -

게시판		게시글	
게시판 ID		게시글 ID	
게시판명 등록일시 사용여부		게시판 ID(FK) 제목 내용 등록일시 등록자명 삭제여부	

**[ SQL ]**  

```
SELECT A.게시판 ID, A.게시판명, COUNT(B.게시글 ID) AS CNT
FROM 게시판 A, 게시글 B
WHERE A.게시판 ID = B.게시판 ID(+)
AND B.삭제여부(+) = 'N'
AND A.사용여부 = 'Y'
GROUP BY A.게시판 ID, A.게시판명
ORDER BY A.게시판 ID;
```

- ① SELECT A.게시판 ID, A.게시판명, COUNT(B.게시글 ID) AS CNT  
FROM 게시판 A LEFT OUTER JOIN 게시글 B  
ON (A.게시판 ID = B.게시판 ID AND B.삭제여부 = 'N')  
WHERE A.사용여부 = 'Y'  
GROUP BY A.게시판 ID, A.게시판명  
ORDER BY A.게시판 ID;
- ② SELECT A.게시판 ID, A.게시판명, COUNT(B.게시글 ID) AS CNT  
FROM 게시판 A LEFT OUTER JOIN 게시글 B  
ON (A.게시판 ID = B.게시판 ID AND A.사용여부 = 'N')  
WHERE B.삭제여부 = 'Y'  
GROUP BY A.게시판 ID, A.게시판명  
ORDER BY A.게시판 ID;
- ③ SELECT A.게시판 ID, A.게시판명, COUNT(B.게시글 ID) AS CNT  
FROM 게시판 A LEFT OUTER JOIN 게시글 B  
ON (A.게시판 ID = B.게시판 ID)

WHERE A.사용여부 = 'Y'

AND B.삭제여부 = 'N'

GROUP BY A.게시판 ID, A.게시판명

ORDER BY A.게시판 ID;

④ SELECT A.게시판 ID, A.게시판명, COUNT(B.게시글 ID) AS CNT

FROM 게시판 A RIGHT OUTER JOIN 게시글 B

ON (A.게시판 ID = B.게시판 ID AND A.사용여부 = 'Y' AND B.삭제여부 = 'N')

GROUP BY A.게시판 ID, A.게시판명

ORDER BY A.게시판 ID;

해설 :

보기는 게시판별 게시글의 개수를 조회하는 SQL 이다. 이때 게시글이 존재하지 않는 게시판도 조회되어야 한다.

**ORACLE**에서는 **OUTER JOIN** 구문을 (+) 기호를 사용하여 처리할 수도 있으며, 이를 ANSI 문장으로 변경하기 위해서는 Inner 쪽 테이블(게시글)에 조건절을 ON 절에 함께 위치시켜야 정상적인 **OUTER JOIN** 을 수행할 수 있다.

②번의 경우는 Outer 대상이 되는 테이블(게시판)의 조건절이 ON 절에 위치하였으므로 원하는 결과가 출력되지 않는다.

정답 : ①

78. 다음과 같은 2 개의 릴레이션이 있다고 가정하자.

**student** 의 기본키는 **st\_num** 이고, **department** 의 기본키는 **dept\_num** 이다.

또한 **student** 의 **d\_num** 은 **department** 의 **dept\_num** 을 참조하는 외래키이다.

아래 SQL 문의 실행 결과 건수는?

- 아 래 -		
SELECT count(st_name) FROM student s WHERE not exists (SELECT * FROM department d WHERE s.d_num = d.dept_num and dept_name = '전자계산학과');		
Student		Department

st_num	st_name	d_num	dept_num	dept_name
1001	Yoo	10	10	컴퓨터공학과
1002	Kim	30	20	컴퓨터공학과
1003	Lee	20	30	컴퓨터공학과
1004	Park	10		
1005	Choi	20		
1006	Jeong	10		

해설 : 조건에 맞는 Student 데이터는 다음과 같다.

st_num	st_name	d_num
1001	Yoo	10
1003	Lee	20
1004	Park	10
1005	Choi	20
1006	Jeong	10

정답 : 5

79. (SQL Server) 다음 중 아래의 SQL 과 동일한 결과를 추출하는 SQL 은? (단, 테이블 TAB1, TAB2 의 PK 컬럼은 A,B 이다)

아 래

```
[ SQL ]
SELECT A, B
FROM TAB1
EXCEPT
SELECT A, B
FROM TAB2;
```

- ① SELECT TAB2. A, TAB2.B  
FROM TAB1, TAB2  
WHERE TAB1.A <> TAB2.A  
AND TAB1.B <> TAB2.B
- ② SELECT TAB2. A, TAB2.B  
FROM TAB1  
WHERE TAB1.A NOT IN (SELECT TAB2.A FROM TAB2)  
AND TAB1.B NOT IN (SELECT TAB2.B FROM TAB2);
- ③ SELECT TAB2. A, TAB2.B  
FROM TAB1, TAB2  
WHERE TAB1.A = TAB2.A  
AND TAB1.B = TAB2.B
- ④ SELECT TAB2. A, TAB2.B  
FROM TAB1  
WHERE NOT EXISTS (SELECT 'X'  
FROM TAB2  
WHERE TAB1.A = TAB2.A  
AND TAB1.B = TAB2.B);

### 집합 연산자의 종류

집합 연산자	연산자의 의미
<b>UNION</b>	여러 개의 SQL 문의 결과에 대한 합집합으로 결과에서 모든 중복된 행은 하나의 행으로 만든다.

<b>UNION ALL</b>	<p>여러 개의 <b>SQL</b> 문의 결과에 대한 합집합으로 중복된 행도 그대로 결과로 표시된다.</p> <p>즉, 단순히 결과만 합쳐놓은 것이다.</p> <p>일반적으로 여러 질의 결과가 상호 배타적인(<b>Exclusive</b>)일 때 많이 사용한다.</p> <p>개별 <b>SQL</b> 문의 결과가 서로 중복되지 않는 경우, <b>UNION</b> 과 결과가 동일하다.</p> <p>( 결과의 정렬 순서에는 차이가 있을수 있음)</p>
<b>INTERSECT</b>	<p>여러 개의 <b>SQL</b> 문의 결과에 대한 교집합이다. 중복된 행은 하나의 행으로 만든다.</p>
<b>EXCEPT</b>	<p>앞의 <b>SQL</b> 문의 결과에서 뒤의 <b>SQL</b> 문의 결과에 대한 차집합이다. 중복된 행은 하나의 행으로 만든다. (일부 데이터베이스는 <b>MINUS</b> 를 사용함)</p>

해설 :

**EXCEPT** 는 차집합에 대한 연산이므로 **NOT IN** 또는 **NOT EXISTS** 로 대체하여 처리가 가능하다.

②는 **NOT IN** 을 사용하였으나, **PK 컬럼 A, B** 에 대하여 각각 **NOT IN** 연산을 수행하여 다른 결과가 생성된다.

정답 : ④

80. 아래와 같은 데이터 모델에 대해 **SQL** 을 수행 하였다. 다음 중 수행된 **SQL** 과 동일한 결과를 도출하는 **SQL** 은?

- 아 래 -				
서비스		서비스이용		회원
서비스 ID		회원 ID(FK) 서비스 ID(FK)		회원 ID
서비스명 서비스 URL		이용일시		회원명
<b>[ 수행 SQL ]</b> <b>SELECT A.서비스 ID, B.서비스명, B.서비스 URL</b>				

```
FROM (SELECT 서비스 ID
FROM 서비스
INTERSECT
SELECT 서비스 ID
FROM 서비스이용) A, 서비스 B
WHERE A.서비스 ID = B.서비스 ID;
```

① SELECT B.서비스 ID, A.서비스명, A.서비스 URL  
FROM 서비스 A, 서비스이용 B  
WHERE A.서비스 ID = B.서비스 ID;

② SELECT X.서비스 ID, X.서비스명, X.서비스 URL  
FROM 서비스 X  
WHERE NOT EXISTS (SELECT 1  
FROM ( SELECT 서비스 ID  
FROM 서비스  
MINUS  
SELECT 서비스 ID  
FROM 서비스이용) Y  
WHERE X.서비스 ID = Y.서비스 ID);

③ SELECT B.서비스 ID, A.서비스명, A.서비스 URL  
FROM 서비스 A LEFT OUTER JOIN 서비스이용 B  
ON (A.서비스 ID = B.서비스 ID)  
WHERE B.서비스 ID IS NULL  
GROUP BY B.서비스 ID, A.서비스명, A.서비스 URL;

④ SELECT A.서비스 ID, A.서비스명, A.서비스 URL  
FROM 서비스 A  
WHERE 서비스 ID IN (SELECT 서비스 ID  
FROM 서비스이용  
MINUS  
SELECT 서비스 ID  
FROM 서비스);

해설 :

수행한 SQL 은 이용된 적이 있었던 서비스를 추출하는 SQL 이다.

① 이용된 적이 있었던 서비스를 추출하는 것은 동일하나 서비스와 서비스이용은 1:N  
관계이므로  
서비스이용건수 만큼 추출되므로 전체 결과가 다르다.  
GROUP BY 를 수행하면 동일한 결과를 출력할 수 있다.



② 전체 서비스에서 이용된 적이 있었던 서비스를 MINUS 하였으므로 이용된 적이 없었던 서비스가 서브쿼리에서 추출된다.

그러므로 NOT EXISTS 구문을 적용하면 이용된 적이 있었던 서비스가 출력된다. (정답)

③ 서비스를 기준으로 OUTER JOIN 을 수행하였으므로, 이용된 적이 없었던 서비스만 출력된다.

B.서비스 ID IS NOT NULL 로 변경해야 동일한 결과가 출력된다.

④ 서비스와 서비스이용 테이블의 순서를 변경하고 IN 절을 NOT IN 으로 변경하면 동일한 결과를 출력할 수 있다.

---

81. SET OPERATOR 중에서 수학의 교집합과 같은 기능을 하는 연산자로 가장 적절한 것은?

- ① UNION
- ② INTERSECT
- ③ MINUS
- ④ EXCEPT

해설 :

**SET OPERATOR : 합집합은 UNION, 교집합은 INTERSECT, 차집합은 MINUS/EXCEPT 이다.**

정답 : ②

---

82. 다음 중 아래의 EMP 테이블의 데이터를 참조하여 실행한 SQL 의 결과로 가장 적절한 것은?

## 아 래

```

SELECT ENAME AAA, JOB AAB
FROM EMP
WHERE EMPNO = 7369
UNION ALL
SELECT ENAME BBA, JOB BBB
FROM EMP
WHERE EMPNO = 7566
ORDER BY 1, 2;
    
```

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980-12-17	800		20
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30
7566	JONES	MANAGER	7839	1981-04-02	2975		20
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30
7782	CLARK	MANAGER	7839	1981-06-09	2450		10
7788	SCOTT	ANALYST	7566	1987-07-13	3000		20
7839	KING	PRESIDENT		1981-11-17	5000		10
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30
7876	ADAMS	CLERK	7788	1987-07-13	1100		20
7900	JAMES	CLERK	7698	1981-12-03	950		30
7902	FORD	ANALYST	7566	1981-12-03	3000		20
7934	MILLER	CLERK	7782	1982-01-23	1300		10

①

AAA	AAB
SMITH	CLERK
JONES	MANAGER

②

AAA	AAB
JONES	MANAGER
SMITH	CLERK

③

BBA	BBB
SMITH	CLERK
JONES	MANAGER

④

BBA	BBB
JONES	MANAGER
SMITH	CLERK

핵심정리)

SELECT 칼럼명 1,

칼럼명 2, ...

FROM 테이블명 1

[WHERE 조건식 ]

[GROUP BY 칼럼(Column)이나 표현식

[HAVING 그룹조건식 ]]

집합 연산자

SELECT 칼럼명 1,

칼럼명 2, ...

FROM 테이블명 2

[WHERE 조건식 ]

[GROUP BY 칼럼(Column)이나 표현식

[HAVING 그룹조건식 ]]

[ORDER BY 1,2

[ASC 또는 DESC];

→ ORDER BY 는 집합연산을 적용한 최종 결과에 대한 정렬처리이므로 가장 마지막 줄에 한번만 기술한다.

해설 : **UNION ALL** 을 사용하는 경우 칼럼 **ALIAS** 는 첫번째 **SQL** 모듈 기준으로 표시되며, 정렬 기준은 마지막 **SQL** 모듈에 표시하면 됨.

정답 : ②

83. 다음 중 아래 TBL1, TBL2 테이블에 대해 SQL 을 수행한 결과인 것은?

- 아 래 -			
[ 테이블 : TBL1]		[테이블 : TBL2]	
COL1	COL2	COL1	COL2
AA	A1	AA	A1
AB	A2	AB	A2
		AC	A3
		AD	A4
[SQL] SELECT COL1, COL2, COUNT(*) AS CNT FROM (SELECT COL1, COL2 FROM TBL1			

```
UNION ALL
SELECT COL1, COL2
FROM TBL2
UNION
SELECT COL1, COL2
FROM TBL1)
GROUP BY COL1, COL2;
```

①

COL1	COL2	CNT
AA	A1	1
AB	A2	1
AC	A3	1
AD	A4	1

②

COL1	COL2	CNT
AA	A1	2
AB	A2	2
AC	A3	1
AD	A4	1

③

COL1	COL2	CNT
AA	A1	3
AB	A2	3
AC	A3	1
AD	A4	1

④

COL1	COL2	CNT
AA	A1	3
AB	A2	3
AC	A3	2
AD	A4	2

해설 :

**집합 연산자는 SQL 에서 위에 정의된 연산자가 먼저 수행**된다.

그러므로 UNION 이 나중에 수행되므로 결과적으로 중복 데이터가 모두 제거되어 ①과 같은 결과가 도출된다.

**만일 UNION 과 UNION ALL 의 순서를 바꾼다면 ②과 같은 결과가 도출된다.**

정답 : ①

---

84. 다음 중 아래에서 테이블 T1, T2 에 대한 가, 나 두개의 쿼리 결과 조회되는 행의 수로 가장 적절한 것은?

- 아 래 -						
T1(A,B,C)				T2(A,B,C)		
A	B	C		A	B	C
A3	B2	C3		A1	B1	C1
A1	B1	C1		A3	B2	C3
A2	B1	C2				
가. SELECT A, B, C FROM R1 UNION ALL SELECT A, B, C FROM R2 나. SELECT A, B, C FROM R1 UNION SELECT A, B, C FROM R2						

- ① 가 : 5 개, 나 : 3 개
- ② 가 : 5 개, 나 : 5 개
- ③ 가 : 3 개, 나 : 3 개
- ④ 가 : 3 개, 나 : 5 개

해설 :

㉠ SELECT A, B, C FROM R1  
UNION ALL  
SELECT A, B, C FROM R2;  
(중복 레코드 유지, 정렬 안함)

R(A,B,C)

A	B	C
---	---	---

<b>A3</b>	<b>B2</b>	<b>C3</b>
<b>A1</b>	<b>B1</b>	<b>C1</b>
<b>A2</b>	<b>B1</b>	<b>C2</b>
<b>A1</b>	<b>B1</b>	<b>C1</b>
<b>A3</b>	<b>B2</b>	<b>C3</b>

④ SELECT A, B, C FROM R1  
UNION  
SELECT A, B, C FROM R2;  
(중복 레코드 제거함, 정렬 발생)

**R(A,B,C)**

<b>A</b>	<b>B</b>	<b>C</b>
<b>A1</b>	<b>B1</b>	<b>C1</b>
<b>A2</b>	<b>B1</b>	<b>C2</b>
<b>A3</b>	<b>B2</b>	<b>C3</b>

정답 : ①

85. 다음 중 아래와 같은 집합이 존재 할 때,  
집합 A 와 B 에 대하여 **집합연산**을 수행한 결과 집합 C 가 되는 경우 이용되는  
**데이터베이스 집합연산**은?

아 래



집합 A = [가, 나, 다, 라],  
집합 B = [다, 라, 마, 바],  
집합 C = [다, 라]

- ① Union
- ② Difference
- ③ Intersection
- ④ Product

핵심정리)

### 일반 집합 연산자를 SQL 과 비교

- UNION 연산은 UNION 기능으로,
- INTERSECTION 연산은 INTERSECTION 기능으로,
- DEFFERENCE 연산은 EXCEPT(Oracle 은 MINUS) 기능으로,
- PRODUCT 연산은 CROSS JOIN 기능으로 구현되었다.

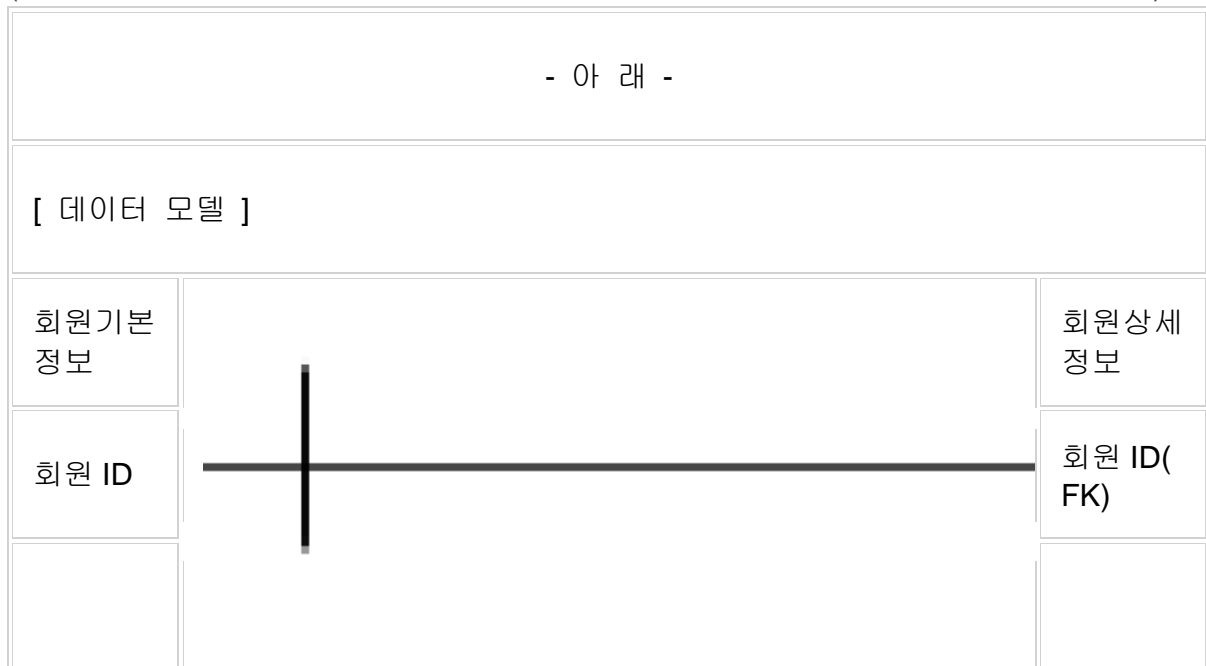
해설 :

집합 C 는 집합 A 와 집합 B 의 교집합이며, 데이터베이스에서 교집합 기능을 하는 집합 연산은 **Inersection** 이다.

정답 : ③

86. 아래와 같은 데이터 모델에 대한 설명으로 가장 적절한 것은?

(단, 시스템적으로 회원기본정보와 회원상세정보는 1:1, 양쪽 필수 관계임을 보장한다.)



- ① 회원 ID 컬럼을 대상으로 (회원기본정보 EXCEPT 회원상세정보) 연산을 수행하면 회원상세정보가 등록되지 않은 회원 ID 가 추출된다.
- ② 회원 ID 컬럼을 대상으로 (회원기본정보 UNION ALL 회원상세정보) 연산을 수행한 결과의 건수는 회원기본정보의 전체건수와 동일하다.
- ③ 회원 ID 컬럼을 대상으로 (회원기본정보 INTERSECT 회원상세정보) 연산을 수행한 결과의 건수와 두 테이블을 회원 ID 로 JOIN 연산을 수행한 결과의 건수는 동일하다.
- ④ 회원 ID 컬럼을 대상으로 (회원기본정보 INTERSECT 회원상세정보) 연산을 수행한 결과와 (회원기본정보 UNION 회원상세정보) 연산을 수행한 결과는 다르다.

해설 :

- ① 1:1, 양쪽 필수 관계를 시스템적으로 보장하므로 두 엔터티간의 EXCEPT 결과는 항상 공집합이다.
- ② 1:1, 양쪽 필수 관계를 시스템적으로 보장하므로 UNION 을 수행한 결과는 회원기본정보의 전체건수와 동일하지만, UNION ALL 을 수행하였으므로 결과건수는 회원기본정보의 전체건수에 2 배가 된다.
- ④ 1:1, 양쪽 필수 관계를 시스템적으로 보장하므로 연산 수행결과는 같다.

정답 : ③

87. 아래와 같은 데이터 상황에서 아래의 SQL 을 수행할 경우 정렬 순서상 2 번째 표시될 값을 적으시오.

- 아 래 -			
TAB1			
<b>C1</b>	<b>C2</b>	<b>C3</b>	SELECT C3 FROM TAB1 START WITH C2 IS NULL CONNECT BY PRIOR C1 = C2 ORDER SIBLINGS BY C3 DESC
1		A	
2	1	B	

3	1	C	
4	2	D	

핵심정리)

**PRIOR** : CONNECT BY 절에 사용되며, 현재 읽은 칼럼을 지정한다.

**PRIOR 자식 = 부모 형태**를 사용하면 **계층구조에서** 부모 데이터에서 자식 데이터(부모 → 자식) 방향으로 전개하는 순방향 전개를 한다.

그리고 **PRIOR 부모 = 자식 형태**를 사용하면 반대로 자식 데이터에서 부모데이터(자식 → 부모) 방향으로 전개하는 역방향 전개를 한다.

해설 :

SQL 의 실행결과는 다음과 같다.

<b>C3</b>
A
C
B
D

정답 : C

---

88. 다음 중 **Oracle 계층형 질의에 대한 설명**으로 가장 **부적절한** 것은?

- ① **START WITH** 절은 계층 구조의 시작점을 지정하는 구문이다.
- ② **ORDER SIBLINGS BY** 절은 형제 노드 사이에서 정렬을 지정하는 구문이다.
- ③ 순방향전개란 부모 노드로부터 자식 노드 방향으로 전개하는 것을 말한다.
- ④ 루트 노드의 **LEVEL** 값은 0 이다.

핵심정리)

- **START WITH 절**은 계층 구조 전개의 시작 위치를 지정하는 구문이다. 즉, 루트\_데이터를 지정한다.(엑세스)
- **ORDER SIBLINGS BY** : 형제 노드(동일 LEVEL) 사이에서 정렬을 수행한다.

해설 :

**Oracle 계층형 질의에서 루트 노드의 LEVEL 값은 1 이다.**

정답 : ④

89. 다음 중 아래와 같은 사원 테이블에 대해서 SQL 을 수행하였을 때의 결과로 가장 적절한 것은?

- 아 래 -			
[테이블 : 사원]			
사원번호(PK)	사원명	입사일자	매니저사원번호(FK)
001	홍길동	2012-01-01	NULL
002	강감찬	2012-01-01	001
003	이순신	2013-01-01	001
004	이민정	2013-01-01	001
005	이병헌	2013-01-01	NULL
006	안성기	2014-01-01	005
007	이수근	2014-01-01	005

008	김병만	2014-01-01	005
-----	-----	------------	-----

```
[SQL]
SELECT 사원번호, 사원명, 입사일자, 매니저사원번호
FROM 사원
START WITH 매니저사원번호 IS NULL
CONNCT BY PRIOR 사원번호 = 매니저사원번호
AND 입사일자 BETWEEN '2013-01-01' AND '2013-12-31'
ORDER SIBLINGS BY 사원번호;
```

①

사원번호(PK)	사원명	입사일자	매니저사원번호(FK)
001	홍길동	2012-01-01	NULL
003	이순신	2013-01-01	001
004	이민정	2013-01-01	001
005	이병헌	2013-01-01	NULL

②

사원번호(PK)	사원명	입사일자	매니저사원번호(FK)
003	이순신	2013-01-01	001
004	이민정	2013-01-01	001
005	이병헌	2013-01-01	NULL

③

사원번호(PK)	사원명	입사일자	매니저사원번호(FK)
001	홍길동	2012-01-01	NULL

④

사원번호(PK)	사원명	입사일자	매니저사원번호(FK)
001	홍길동	2012-01-01	NULL
005	이병헌	2013-01-01	NULL
006	안성기	2014-01-01	005
007	이수근	2014-01-01	005
008	김병만	2014-01-01	005

해설 :

CONNECT BY 절에 작성된 조건절은 WHERE 절에 작성된 조건절과 다르다. START WITH 절에서 필터링된 시작 데이터는 결과목록에 포함되며, 이후 CONNECT BY 절에 의해 필터링 된다. 그러므로 매니저 사원번호가 NULL 인 데이터는 결과목록에 포함되며, 이후 리커시브 조인에 의해 입사일자가 필터링 된다.

④번은 AND PRIOR 입사일자 BETWEEN '2013-01-01' AND '2013-12-31' 에 대한 결과이다.

정답 : ①

---

90. 다음 중 계층형 질의문에 대한 설명으로 가장 부적절한 것은?

① SQL Server 에서의 계층형 질의문은 CTE(Common Table Expression)를 재귀 호출함으로써 계층 구조를 전개한다.

- ② SQL Server 에서의 계층형 질의문은 앵커 멤버를 실행하여 기본 결과 집합을 만들고 이후 재귀 멤버를 지속적으로 실행한다.
- ③ 오라클의 계층형 질의문에서 WHERE 절은 모든 전개를 진행한 이후 필터 조건으로서 조건을 만족하는 데이터만을 추출하는데 활용된다.
- ④ 오라클의 계층형 질의문에서 PRIOR 키워드는 CONNECT BY 절에만 사용할 수 있으며

'PRIOR 자식 = 부모' 형태로 사용하면 순방향 전개로 수행 된다.

핵심정리)

테이블에 계층형 데이터가 존재하는 경우 데이터를 조회하기 위해서

**계층형질의(Hierarchical Query)**를 사용한다.

**계층형 데이터**란 동일 테이블에 계층적으로 상위와 하위 데이터가 포함된 데이터를 말한다.

예를 들어, 사원 테이블에서는 사원들 사이에 상위 사원(관리자)와 하위사원 관계가 존재하고

조직 테이블에서는 조직들 사이에 상위 조직과 하위 조직 관계가 존재한다.

해설 :

- ④ 오라클 계층형 질의문에서 PRIOR 키워드는 SELECT, WHERE 절에서도 사용할 수 있다.

정답 : ④

91. 아래 [부서]와 [매출] 테이블에 대해서 SQL 문장을 실행하여 아래 [결과]와 같이 데이터가 추출 되었다.

다음 중 동일한 결과를 추출하는 SQL 문장은?

- 아 래 -				
[ 테이블 : 부서 ]			[ 테이블 : 매출 ]	
부서코드 (PK)	부서명	상위 부서코드(FK)	부서코드	매출액
100	아시아부	NULL	111	1000

110	한국지사	100	112	2000
111	서울지점	110	121	1500
112	부산지점	110	122	1000
120	일본지사	100	131	1500
121	도쿄지점	120	132	2000
1222	오사카지점	120	211	2000
130	중국지사	100	212	1500
131	베이징지점	130	221	1000
132	상하이지점	130	222	2000
200	남유럽지부	NULL		
210	스페인지사	200		
211	마드리드지점	210		
212	그라나다지점	210		



220	포르투칼지사	200	
221	리스본지점	220	
222	포르투지점	220	

**[결과]**

부서코드	부서명	상위부서코드	매출액	LVL
100	아시아지부	NULL	NULL	2
120	일본지사	100	NULL	1
121	도쿄지점	120	1500	2
122	오사카지점	120	1000	2

```

① SELECT A.부서코드, A.부서명, A.상위부서코드, B.매출액, LVL
FROM (SELECT 부서코드, 부서명, 상위부서코드, LEVEL AS LVL
FROM 부서
START WITH 부서코드 = '120'
CONNECT BY PRIOR 상위부서코드 = 부서코드
UNION
SELECT 부서코드, 부서명, 상위부서코드, LEVER AS LVL
FROM 부서
START WITH 부서코드 = '120'
CONNECT BY 상위부서코드 = PRIOR 부서코드) A LEFT
OUTER JOIN 매출 B
ON (A.부서코드 = B.부서코드)

```

ORDER BY A.부서코드;

② SELECT A.부서코드, A.부서명, A.상위부서코드, B.매출액, LVL  
FROM (SELECT 부서코드, 부서명, 상위부서코드, LEVEL AS LVL  
FROM 부서  
START WITH 부서코드 = '100'  
CONNECT BY 상위부서코드 = PRIOR 부서코드) A LEFT  
OUTER JOIN 매출 B  
ON (A.부서코드 = B.부서코드)  
ORDER BY A.부서코드;

③ SELECT A.부서코드, A.부서명, A.상위부서코드, B.매출액, LVL  
FROM (SELECT 부서코드, 부서명, 상위부서코드, LEVEL AS LVL  
FROM 부서  
START WITH 부서코드 = '121'  
CONNECT BY PRIOR 상위부서코드 = 부서코드) A LEFT  
OUTER JOIN 매출 B  
ON (A.부서코드 = B.부서코드)  
ORDER BY A.부서코드;

④ SELECT A.부서코드, A.부서명, A.상위부서코드, B.매출액, LVL  
FROM (SELECT 부서코드, 부서명, 상위부서코드, LEVEL AS LVL  
FROM 부서  
START WITH 부서코드 = (SELECT 부서코드  
FROM 부서  
WHERE 상위부서코드 IS NULL  
START WITH 부서코드 = '120'  
CONNECT BY PRIOR 상위부서코드 = 부서코드)  
CONNECT BY 상위부서코드 = PRIOR 부서코드) A LEFT  
OUTER JOIN 매출 B  
ON (A.부서코드 = B.부서코드)  
ORDER BY A.부서코드;

해설 :

위의 결과는 **중간 레벨인 도쿄지점(120)을 시작으로  
상위의 전체 노드(역방향 전개)와 하위의 전체 노드(순방향 전개)를 검색하여  
매출액을 추출하는 SQL** 이다.

부서 테이블의 전체 데이터를 보면 LEVEL 은 1~3 까지 이지만  
추출된 데이터의 LEVEL 은 1 과 2 만 추출된 것으로 보면 중간 LEVEL 에서 추출된  
것을 짐작할 수 있다.

② 최상위 노드인 아시아지부(100)를 시작으로 하위의 모든 부서를 추출(순방향 전개)하므로 아래와 같은 결과가 추출된다.

부서코드	부서명	상위부서코드	매출액	LVL
100	아시아지부	NULL	NULL	2
110	한국지사	100	NULL	2
111	서울지점	110	1000	3
112	부산지점	110	2000	3
120	일본지사	100	NULL	2
121	도쿄지점	120	1500	3
122	오사카지점	120	1000	3
130	중국지사	100	NULL	2
131	베이징지점	130	1500	3
132	상하이지점	130	2000	3

③ 최하위 노드인 도쿄 지점(121)에서 상위의 모든 노드(역방향 전개)를 추출하게 되므로 아래와 같은 결과가 추출된다.

부서코드	부서명	상위부서코드	매출액	LVL
100	아시아지부	NULL	NULL	3
120	일본지사	100	NULL	2
121	도쿄지점	120	1500	1

④ WHERE 절의 서브쿼리를 보면 일본 지사(120)를 시작으로 역방향 전개하여 최상위 노드를 추출하여 다시 순방향 전개를 수행하고 있다.

이렇게 되면 ② 와 동일한 결과를 추출하게 된다.

정답 : ①

92. 다음 중 **SELF JOIN** 을 수행해야 할 경우로 가장 적절한 것은?

- ① 한 테이블 내에서 두 칼럼이 연관 관계가 있다.
- ② 두 테이블에 연관된 칼럼은 없으나 JOIN 을 해야 한다.

- ③ 두 테이블에 공통 칼럼이 존재하고 두 테이블이 연관 관계가 있다.
- ④ 한 테이블 내에서 연관된 칼럼은 없으나 JOIN 을 해야 한다.

핵심정리)

**셀프 조인(Self Join)**이란 동일테이블 사이의 조인을 말한다.

따라서 **FROM 절에 동일 테이블이 두 번 이상 나타난다.**

동일 테이블 사이의 조인을 수행하면 테이블과 칼럼 이름이 모두 동일하기 때문에 **식별을 위해 반드시 테이블 별칭(Alias)를 사용**해야 한다.

**셀프 조인(Self Join) 문장**

```
SELECT
  ALIAS 명 1.칼럼명,
  ALIAS 명 2.칼럼명, ...
FROM
  테이블 ALIAS 명 1,
  테이블 ALIAS 명 2
WHERE
  ALIAS 명 1.칼럼명 2 =
  ALIAS 명 2.칼럼명 1;
```

해설 :

**SELF JOIN** 은 하나의 테이블에서 두 개의 칼럼이 연관 관계를 가지고 있는 경우에 사용한다.

정답 : ①

93. 아래와 같이 일자별매출 테이블이 존재할 때 아래 결과처럼 **일자별 누적매출액**을 **SQL**로 구하려고 한다.

**WINDOW FUNCTION을 사용하지 않고** 일자별 누적매출액을 구하는 SQL로 옳은 것은?

- 아 래 -			
[ 테이블 : 일자별매출 ]		[ 결과 : 일자별 누적매출액 ]	
일자	매출액	일자	누적매출액
2015.11.01	1000	2015.11.01	1000

2015.11.02	1000		2015.11.02	2000
2015.11.03	1000		2015.11.03	3000
2015.11.04	1000		2015.11.04	4000
2015.11.05	1000		2015.11.05	5000
2015.11.06	1000		2015.11.06	6000
2015.11.07	1000		2015.11.07	7000
2015.11.08	1000		2015.11.08	8000
2015.11.09	1000		2015.11.09	9000
2015.11.10	1000		2015.11.10	10000

① SELECT A.일자, SUM(A.매출액) AS 누적매출액

FROM 일자별매출 A

GROUP BY A.일자

ORDER BY A.일자;

② SELECT B.일자, SUM(B.매출액) AS 누적매출액

FROM 일자별매출 A JOIN 일자별매출 B ON (A.일자) >= B.일자)

GROUP BY B.일자

ORDER BY B.일자;

③ SELECT A.일자, SUM(B.매출액) AS 누적매출액

FROM 일자별매출 A JOIN 일자별매출 B ON (A.일자) >= B.일자)

GROUP BY A.일자

ORDER BY A.일자;

```

④ SELECT A.일자
,(SELECT SUM(B.매출액)
FROM 일자별매출 B WHERE B.일자 >= A.일자) AS 누적매출액
FROM 일자별매출 A
GROUP BY A.일자
ORDER BY A.일자;

```

해설 :

①은 일자별매출액에 일자별 매출 테이블과 동일하게 출력된다.

②, ④는 작은 날짜쪽에 제일 큰 누적금액이 출력된다.

③은 일자별매출 테이블을 Self Join 하여, A Alias 쪽에 먼저 읽혔다고 가정하면 다음처럼 데이터가 생성될 것이다.

- A 가 {2015.11.01. 1000}일때 B 는 {2015.11.01. 1000}
- A 가 {2015.11.02., 1000}일때 B 는 {{2015.11.01.1000}, {2015.11.02. 1000}}
- A 가 {2015.11.03., 1000} 일때 B 는 {{2015.11.01. 1000}, {2015.11.02. 1000}, {2015.11.03. 1000}}

위의 Self Join 은 Equi Join 이 아닌 Range Join 이므로 A 의 레코드는 B 의 레코드 수 만큼 증가하게된다.

(A \* B) 그러므로 위의 3 번의 경우 A 는 B 의 레코드 개수와 동일하게 되므로 SUM(매출금액)을 하면 3,00 이 된다.

이런 식으로 A Alias 의 모든 레코드 개수를 Scan 하면 누적 값을 출력하게 된다.

정답 : ③

---

94. 다음 중 아래의 SQL 수행 결과로 가장 적절한 것은?

- 아 래 -

```

SELECT COUNT(DISTINCT A || B)
FROM EMP
WHERE D = (SELECT D FROM DEPT WHERE E = 'i');

```

**EMP 테이블**

A	B	C	D

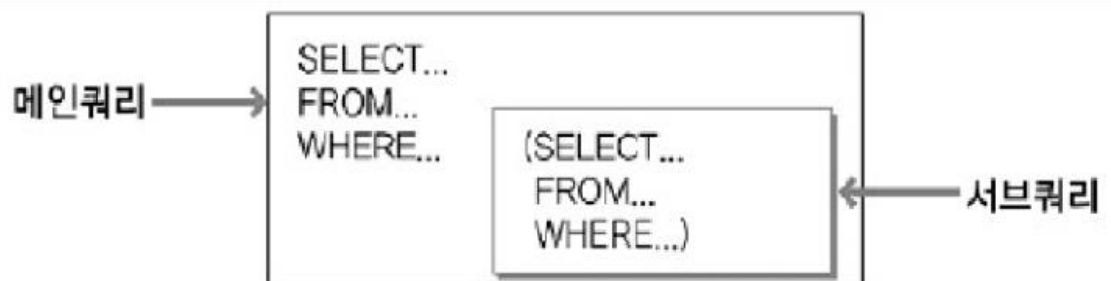
1	a	1	x
2	a	1	x
3	b	2	y

#### DEPT 테이블

D	E	F	
x	i	5	
y	m	6	

- ① 0
- ② 1
- ③ 2
- ④ 3

#### ★ 메인쿼리와 서브쿼리



반환되는 데이터의 형태에 따른 서브쿼리 분류

서브쿼리 종류	설명
Single Row 서브쿼리 (단일 행 서브쿼리)	서브쿼리의 실행 결과가 항상 1 건 이하인 서브쿼리를 의미한다. 단일 행 서브쿼리는 단일 행 비교 연산자와 함께 사용된다. 단일 행 비교 연산자에는 =, <, <=, >, >=, <>이 있다.
Multi Row 서브쿼리 (다중 행 서브쿼리)	서브쿼리의 실행 결과가 여러 건인 서브쿼리를 의미한다. 다중행 서브쿼리는 다중 행 비교 연산자와 함께 사용된다. 다중 행 비교 연산자에는 IN, ALL, ANY, SOME, EXISTS 가 있다.
Multi Column 서브쿼리 (다중 칼럼 서브쿼리)	서브쿼리의 실행 결과로 여러 칼럼을 반환한다. 메인쿼리의 조건절에 여러 칼럼을 동시에 비교할 수 있다. 서브쿼리와 메인쿼리에서 비교하고자 하는 칼럼 개수와 칼럼의 위치가 동일해야 한다.

### 해설 :

WHERE 절의 단일행 서브쿼리인 (SELECT D FROM DEPT WHERE E ='i')에 의해서  
DEPT 테이블의 D 컬럼 값이 x 인 행이 선택되고,

D = (SELECT D FROM DEPT WHERE E ='i') 조건에 의해 EMP 테이블의 (A=1, B=a),  
(A=2, B=a) 인 2 건이 출력된다.

출력된 결과가 모두 UNIQUE 하기 때문에 DISTINCT 연산자는 결과 건수에 영향을 주지  
않는다.

정답 : ③

95. 아래는 서브쿼리에 대한 설명이다. 다음 중 올바른 것끼리 묶인 것은?

- 아 래 -

- 가) 서브쿼리는 단일 행(Single Row) 또는 복수 행(Multi Row) 비교연산자와 함께 사용할 수 있다.
- 나) 서브쿼리는 SELECT 절, FROM 절, HAVING 절, ORDER BY 절 등에서 사용이 가능하다.
- 다) 서브쿼리의 결과가 복수 행(Multi Row) 결과를 반환하는 경우에는 '=', '<=', '>=' 등의 연산자와 함께 사용이 가능하다.
- 라) 연관(Correlated) 서브쿼리는 서브쿼리가 메인쿼리 컬럼을 포함하고 있는 형태의 서브쿼리이다.
- 마) 다중 컬럼 서브쿼리는 서브쿼리의 결과로 여러 개의 컬럼이 변환되어 메인쿼리의 조건과 동시에 비교되는 것을 의미하여



Oracle 및 SQL Server 등의 DBMS 에서 사용 할 수 있다.

- ① 나, 라, 마
- ② 가, 나, 라
- ③ 나, 다, 라
- ④ 가, 나, 마

해설 :

다) 서브쿼리의 결과가 복수 행 결과를 반환하는 경우에는 IN, ALL, ANY 등의 복수 행 비교 연산자와 사용하여야 한다.

마) 다중 컬럼 서브쿼리는 서브쿼리의 결과로 여러 개의 컬럼이 반환되어 메인 쿼리의 조건과 비교되는데,

SQL Server에서는 현재 지원하지 않는 기능이다.

정답 : ②

96. 아래 테이블은 어느 회사의 직원들과 이들이 부양하는 가족에 대한 것으로 밑줄친 칼럼은 기본키(Primary Key)를 표시한 것이다.

다음 중 '현재 부양하는 가족들이 없는 직원들의 이름을 구하라'는 질의에 대해 아래 SQL 문장의 ㉠, ㉡ 에 들어 갈 내용으로 가장 적절한 것은?

- 아 래 -

**[테이블]**

직원 (사번, 이름, 나이)

가족 (이름, 나이, 부양사번)

※ 가족 테이블의 부양사번은 직원 테이블의 사번을 참조하는 외래키(Foreign Key)이다.

**[SQL 문장]**

SELECT 이름

FROM 직원

WHERE ㉠ (SELECT \* FROM 가족 WHERE ㉡ )

- ① ㉠ : EXISTS ㉡ : 사번 = 부양사번
- ② ㉠ : EXISTS ㉡ : 사번 <> 부양사번
- ③ ㉠ : NOT EXISTS ㉡ : 사번 = 부양사번
- ③ ㉠ : NOT EXISTS ㉡ : 사번 <> 부양사번

해설 :

'현재 부양하는 가족들이 없는 직원들의 이름을 구하라'를 구현하는 방법은

가족 테이블에 부양사번이 없는 사원 이름을 사원 테이블에서 추출 하면되고,  
SQL 문장으로 **NOT EXISTS, NOT IN, LEFT OUTER JOIN** 을 사용하여 구현 할 수 있다.

**NOT EXISTS**

SELECT 이름

FROM 사원

WHERE NOT EXISTS (SELECT \* FROM 가족 WHERE 사번 = 부양사번)

2. NOT IN

SELECT 이름

FROM 사원

WHERE 사번 NOT IN (SELECT 부양사번 FROM 가족)

3. LEFT OUTER JOIN

SELECT 이름

FROM 사원 LEFT OUTER JOIN 가족 ON (사번 = 부양사번)

WHERE 부양사번 IS NULL

정답 : ③

97. 다음 중 아래의 ERD 를 참조하여 아래 **SQL** 과 동일한 결과를 출력하는 **SQL** 로 가장 부적절한 것은?

- 아 래 -



**[SQL]**

SELECT A.회원번호, A.회원명

FROM 회원 A, 동의항목 B

WHERE A.회원번호 = B. 회원번호

GROUP BY A.회원번호, A.회원명

HAVING COUNT(CASE WHEN B.동의여부 = 'N' THEN 0 ELSE NULL END) >= 1

ORDER BY A.회원번호;

```

① SELECT A.회원번호, A.회원명
FROM 회원 A
WHERE EXISTS (SELECT 1 FROM 동의항목 B
WHERE A.회원번호 = B.회원번호 AND
B.동의여부 = 'N')
ORDER BY A.회원번호;

② SELECT A.회원번호, A.회원명
FROM 회원 A
WHERE A.회원번호 IN (SELECT B.회원번호 FROM 동의항목 B
WHERE B.동의여부 = 'N')
ORDER BY A.회원번호;

③ SELECT A.회원번호, A.회원명
FROM 회원 A
WHERE 0 < (SELECT COUNT(*)
FROM 동의항목 B WHERE B.동의여부 = 'N')
ORDER BY A.회원번호;

④ SELECT A.회원번호, A.회원명
FROM 회원 A, 동의항목 B
WHERE A.회원번호 = B.회원번호 AND B.동의여부 = 'N')
GROUP BY A.회원번호, A.회원명
ORDER BY A.회원번호;

```

핵심정리)

## 서브쿼리를 사용시 주의사항

- ① 서브쿼리를 괄호로 감싸서 사용한다.
- ② 서브쿼리는 단일행 (Single Row) 또는 복수 행(Multiple Row) 비교 연산자와 함께 사용 가능하다.  
단일 행 비교 연산자는 서브쿼리의 결과가 반드시 1 건 이하이어야 하고  
복수행 비교 연산자는 서브쿼리의 결과 건수와 상관없다.
- ③ 서브쿼리에서는 ORDER BY 를 사용하지 못한다.  
ORDER BY 절은 SELECT 절에서 오직 한 개만 올 수 있기 때문에 ORDER BY 절은 메인쿼리의 마지막 문장에 위치해야 한다.

해설 :

위의 SQL 은 **약관항목 중 단 하나라도 동의를 하지 않은 회원**을 구하는 SQL 이다.  
HAVING 절에서 동의여부가 N 인 데이터가 한 건이라도 존재하는 데이터를 추출한다.

①은 회원 테이블과 동의항목 테이블의 회원번호 컬럼으로 연관 서브쿼리를 수행하여

동의여부 컬럼의 값이 N 인 데이터가 한 건이라도 존재하면 회원 데이터를 출력하게 된다.

②는 동의항목 테이블에서 동의여부가 N 인 한 건이라도 존재하는 회원을 추출하여 회원테이블과 IN 연산을 수행한다.

③의 회원 테이블과 동의항목 테이블간에 회원번호 컬럼으로 연관 서브쿼리로 처리되어야 정상적으로 처리할 수 있다.

④는 HAVING 절로 처리되던 조건을 WHERE 절에 위치시켜 더 간편하게 Join 으로 처리하였다.

또한 회원과 동의항목은 1:N 관계이므로 JOIN 된 결과는 N 건으로 발생됨에 따라 GROUP BY 를 추가하여 중복을 제거 하였다.

정답 : ③