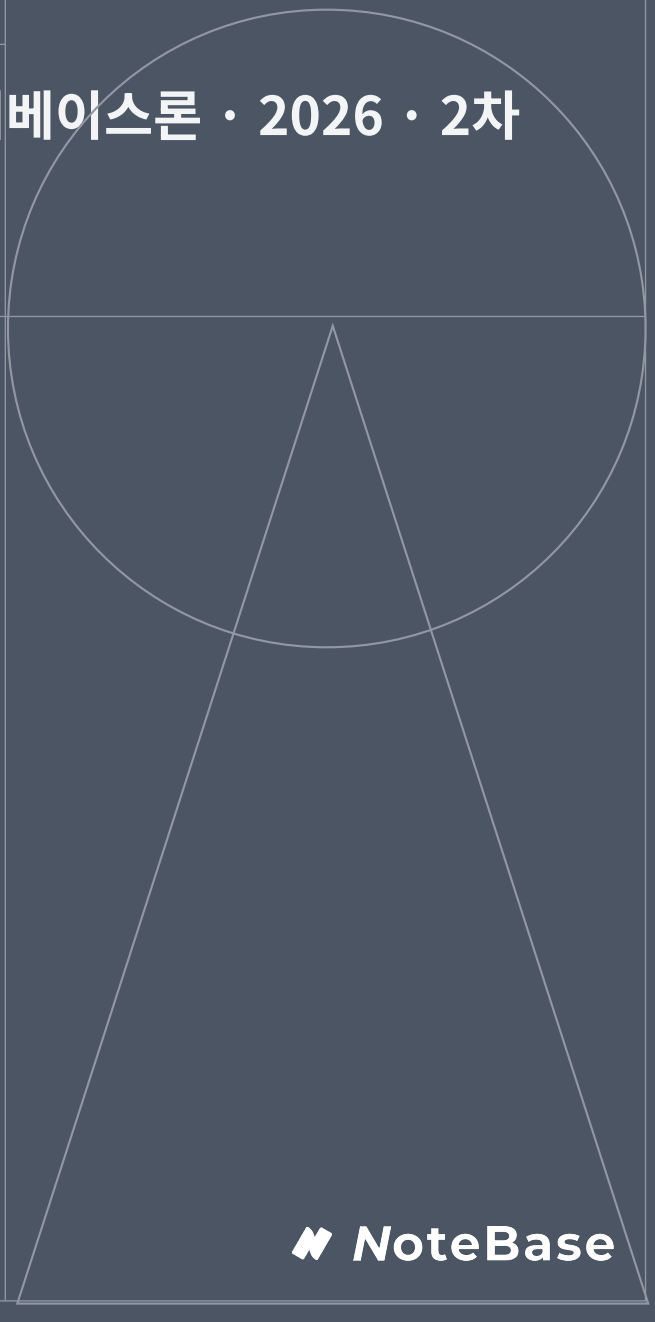
기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2026 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2026 · 2차

공무원/국가직/7급 · 데이터베이스론 · 2026 · 2차

문 1. 테이블을 생성하는 SQL 문에서 속성의 제약조건에 대한 설명으로 옳지 않은 것은?

① CHECK는 속성이 가질 수 있는 값의 범위를 제한한다.

② DEFAULT는 데이터 삽입 시 속성의 값이 명시되지 않을 때 지정된 값이 입력되도록 제한한다.

③ NOT NULL은 속성에 NULL이 입력되지 않도록 제한하며, FOREIGN KEY로 지정된 속성에 대해서도 지정할 수 있다.

④ UNIQUE는 기본키를 대체하는 속성에 부여하는 것으로, 중복되지 않고 NULL이 아닌 값이 입력되도록 제한한다.

정답 4번

간단한 해설

④의 UNIQUE는 **중복을 제한하지만 NULL을 금지하지는 않는다**. NULL까지 금지하려면 NOT NULL을 함께 지정해야 하며, PRIMARY KEY는 중복과 NULL을 모두 허용하지 않는다.

NoteBase

선지별 해설

① O: CHECK는 속성값이 만족해야 하는 조건을 지정하여 **허용되는 값의 범위**를 제한한다. 예를 들어 점수에 'CHECK (점수 BETWEEN 0 AND 100)'을 지정하면 100을 넘는 점수는 저장할 수 없다.

② O: DEFAULT는 행을 삽입할 때 **해당 속성값을 생략하면 적용되는 기본값**을 지정한다. 예를 들어 수량의 기본값이 1이면 수량을 명시하지 않고 삽입한 행에는 수량 1이 저장된다. 모든 행에 지정된 값만 입력하도록 제한한다는 뜻은 아니다.

③ O: NOT NULL은 해당 속성에 NULL을 저장하지 못하게 한다. **외래키 속성에도 NOT NULL을 함께 지정할 수 있다**. FOREIGN KEY는 참조 대상에 존재하는 값인지를 제한하고, NOT NULL은 그 속성을 NULL로 남길 수 없게 하므로 두 제약은 서로 다른 역할을 한다.

④ X: UNIQUE는 대체키의 유일성을 보장하는 데 사용할 수 있지만, **UNIQUE만으로는 NULL을 금지하지 않는다**. 따라서 '중복되지 않고 NULL이 아닌 값'이라는 설명에서 NULL 금지 부분이 잘못되었다. 중복과 NULL을 모두 금지하려면 UNIQUE와 NOT NULL을 함께 지정해야 한다. PRIMARY KEY는 이 두 조건을 모두 요구한다.

NoteBase

문 2. 데이터베이스의 무결성 제약조건에 대한 설명으로 옳지 않은 것은?

① 참조 무결성 제약조건은 외래키는 참조할 수 없는 값을 가질 수 없다는 규칙이다.

② 개체 무결성 제약조건은 기본키를 구성하는 모든 속성은 NULL값을 가지면 안 된다는 규칙이다.

③ 기본키로 정의된 속성은 중복된 값이 허용되지 않는다.

④ 참조 무결성 제약조건에서 자식 릴레이션은 부모 릴레이션의 제약을 받지 않는다.

정답 4번

간단한 해설

④의 '제약을 받지 않는다'는 설명과 달리, 자식 릴레이션의 외래키는 **부모 릴레이션에 존재하는 키 값**을 참조해야 한다. 부모에 없는 값을 임의로 저장하면 참조 무결성을 위반한다.

선지별 해설

① O: 참조 무결성은 외래키를 통해 연결된 릴레이션 사이의 일관성을 유지하는 규칙이다. 외래키에 NULL이 아닌 값을 저장하려면 **참조 대상 키에 같은 값이 존재**해야 한다. 다만 별도의 NOT NULL 제약이 없다면 외래키에 NULL을 허용할 수 있다.

② O: 기본키는 각 튜플을 식별하는 기준이므로 값이 누락되어서는 안 된다. 개체 무결성에 따라 **기본키를 구성하는 모든 속성에 NULL이 금지**되며, 여러 속성으로 이루어진 복합 기본키도 그중 어느 속성에든 NULL을 넣을 수 없다.

③ O: 기본키 값은 서로 다른 튜플을 유일하게 구별해야 하므로 중복될 수 없다. 하나의 속성으로 된 기본키라면 그 속성 값이 유일해야 한다. 복합 기본키라면 **구성 속성 값의 조합**이 유일해야 하며, 개별 속성 값은 중복될 수 있다.

④ X: 부모 릴레이션은 참조 대상 키를 가진 릴레이션이고, 자식 릴레이션은 이를 참조하는 외래키를 가진 릴레이션이다. 따라서 자식 릴레이션의 외래키 값은 **부모 릴레이션의 키 값에 제약을 받는다**. 예를 들어 부모인 부서 릴레이션에 부서번호 10만 있다면, 자식인 사원 릴레이션의 외래키인 부서번호에 20을 저장할 수 없다. '제약을 받지 않는다'는 '참조 무결성 제약을 받는다'로 고쳐야 한다.

NoteBase

문 3. 데이터와 사용자를 다양한 보안등급(1급 비밀, 2급 비밀, 3급 비밀, 일반 정보 등)으로 분류하고 해당 조직에 적합한 보안 정책을 적용하여 다단계 보안을 시행하기 위해 사용하는 접근 제어 기법은?

① 속성 기반 접근 제어(ABAC)

② 임의 접근 제어(DAC)

③ 강제 접근 제어(MAC)

④ 역할 기반 접근 제어(RBAC)

정답 3번

간단한 해설

데이터의 보안등급과 사용자의 인가등급을 대조하여 조직이 정한 정책을 강제하는 방식은 **강제 접근 제어(MAC)**이다. 보안등급에 따라 접근을 제한하여 다단계 보안을 구현한다.

핵심 풀이

강제 접근 제어(MAC)는 데이터에 부여한 보안등급과 사용자가 접근하도록 인가받은 등급을 기준으로 접근을 통제한다. 문제의 '데이터와 사용자를 보안등급으로 분류'하고 '다단계 보안'을 시행한다는 조건이 핵심 단서이다. **조직이 정한 보안 정책을 시스템이 강제**하므로, 사용자가 임의로 권한을 부여하여 등급 제한을 우회할 수 없다. 예를 들어 3급 비밀까지만 열람하도록 인가된 사용자는 1급 비밀 자료를 읽을 수 없다.

임의 접근 제어(DAC)는 자원 소유자의 권한 부여를, 역할 기반 접근 제어(RBAC)는 사용자가 맡은 역할에 부여된 권한을 중심으로 접근을 결정한다. 속성 기반 접근 제어(ABAC)는 사용자·자원·환경 등의 속성을 정책 조건과 대조한다. ABAC에서도 보안등급을 속성으로 사용할 수 있지만, 이 문제처럼 **등급별 보안 정책을 강제하는 다단계 보안**을 대표하는 기법은 MAC이다.

NoteBase

문 4. 뷰(view)에 대한 설명으로 옳지 않은 것은?

① 일반 테이블과 동일한 방법으로 검색을 수행할 수 있다.

② 집계 함수로 계산된 내용을 포함하고 있는 뷰는 변경(삽입, 삭제, 갱신)이 가능하다.

③ 뷰를 생성할 때, 뷰를 통한 투플의 삽입 또는 갱신 연산에 제한을 둘 수 있다.

④ DISTINCT 키워드를 포함하여 정의된 뷰는 변경(삽입, 삭제, 갱신)이 제한된다.

정답 2번

간단한 해설

집계 함수로 계산한 값은 여러 투플을 요약한 결과이므로, 그 값을 바꾸더라도 **기본 테이블의 어느 투플을 어떻게 변경할지** 정할 수 없다. 따라서 집계 결과를 포함한 뷰는 일반적인 뷰 갱신 규칙에서 **삽입·삭제·갱신이 제한된다**.

선지별 해설

① O: 뷰는 기본 테이블에 대한 질의로 정의되는 가상 테이블이다. 일반 테이블을 조회할 때처럼 **SELECT 문의 FROM 절에 뷰 이름**을 지정하고, 필요한 열이나 조건을 지정하여 검색할 수 있다.

② X: 집계 함수를 포함한 뷰는 일반적인 뷰 갱신 규칙에서 삽입·삭제·갱신이 제한된다. 집계 결과와 기본 테이블의 개별 투플 사이에 **변경 대상을 명확히 대응시키기 어렵기** 때문이다. 예를 들어 부서별 평균 급여를 보여주는 뷰에서 평균값을 바꾸라는 요청만으로는 어느 직원의 급여를 얼마만큼 바꿀지 정할 수 없다. 따라서 '변경이 가능하다'는 '변경이 제한된다'로 고쳐야 한다.

③ O: 갱신 가능한 뷰를 정의할 때 **WITH CHECK OPTION**을 지정하면, 뷰를 통한 삽입·갱신 결과가 뷰의 검색 조건을 만족하도록 제한할 수 있다. 예를 들어 영업부 직원만 보여 주는 뷰라면 다른 부서 직원을 삽입하거나 기존 직원의 부서를 다른 부서로 바꾸는 연산을 막을 수 있다.

④ O: DISTINCT는 조회 결과에서 중복 행을 제거한다. 따라서 뷰에 보이는 한 행이 기본 테이블의 여러 투플에 대응할 수 있어, 그 행을 변경하거나 삭제할 때 **기본 테이블의 변경 대상을 명확히 정하기 어렵다**. 이 때문에 DISTINCT를 포함하여 정의한 뷰는 일반적인 뷰 갱신 규칙에서 삽입·삭제·갱신이 제한된다.

NoteBase

문 5. (고객번호, 이름, 지역) 스키마로 정의된 '고객' 릴레이션을 지역이 '서울'인 투플은 사이트 1에, 지역이 '부산'인 투플은 사이트 2에 분산하여 저장하는 경우에 대한 설명으로 옳은 것은? (단, 각 사이트에는 세 속성이 모두 저장된다)

① 투플 단위로 나누었으므로 수평 단편화이다.

② '지역'이라는 속성 단위로 나누었으므로 수직 단편화이다.

③ '지역'에 따라 투플을 분산하여 저장하므로 혼합 단편화이다.

④ 자연 조인 연산을 통해 두 사이트의 릴레이션을 원래의 릴레이션으로 복원할 수 있다.

정답 1번

간단한 해설

각 사이트에 고객번호·이름·지역을 모두 유지하면서 지역 값에 따라 **투플(행)을 나누어 저장**하므로 수평 단편화이다. 분할 조건에 속성을 사용하더라도, 실제로 나뉘는 대상이 열이 아니라 행이라는 점이 기준이다.

선지별 해설

① O: 수평 단편화는 조건에 따라 릴레이션의 투플을 나누는 방식이다. 서울인 투플과 부산인 투플을 서로 다른 사이트에 저장하되 **각 투플의 세 속성을 모두 유지**하므로 수평 단편화에 해당한다.

② X: 수직 단편화는 릴레이션을 **속성(열) 단위로 나누는** 방식이다. 이 사례에서는 '지역' 속성을 따로 떼어 저장한 것이 아니라 그 속성의 값을 기준으로 투플을 구분했다. 따라서 수직 단편화가 아니라 수평 단편화이다.

③ X: 혼합 단편화는 **수평 단편화와 수직 단편화를 함께 적용**하는 방식이다. 이 사례에서는 지역에 따라 행만 나누었고 열은 나누지 않았으므로, 혼합 단편화가 아니라 수평 단편화이다.

④ X: 이렇게 나눈 두 단편은 **합집합 연산**으로 투플을 모아 원래 릴레이션으로 복원한다. 자연 조인은 공통 속성의 값이 같은 투플을 연결하는 연산이다. 두 사이트의 릴레이션은 세 속성이 모두 공통인데 지역 값이 각각 서울과 부산으로 달라 서로 일치하는 투플이 없으므로, 자연 조인의 결과는 빈 릴레이션이 된다.

NoteBase

문 6. '주문', '제품1', '제품2' 테이블에 대하여 '주문 ÷ 제품1', '주문 ÷ 제품2'의 연산 결과로 옳은 것은?

주문

주문고객	제품이름	제조업체
C1	라면	P1
C2	과자	P2
C3	음료	P1
C1	음료	P1
C2	음료	P1

제품1

제품이름	제조업체
음료	P1

제품2

제품이름
라면
음료

① 주문 ÷ 제품1

주문고객
C3
C1
C2

주문 ÷ 제품2

주문고객	제조업체
C1	P1

② 주문 ÷ 제품1

주문고객
C1
C2

주문 ÷ 제품2

주문고객	제조업체
C2	P2

③ 주문 ÷ 제품1

주문고객	제품이름	제조업체
C3	음료	P1
C1	음료	P1
C2	음료	P1

주문 ÷ 제품2

주문고객	제품이름	제조업체
C1	라면	P1
C1	음료	P1

④ 주문 ÷ 제품1

주문고객	제품이름	제조업체
C1	라면	P1
C2	과자	P2

주문 ÷ 제품2

주문고객	제품이름	제조업체
C2	과자	P2

정답 1번

간단한 해설

디비전(÷)은 나누는 테이블의 **모든 행과 대응하는 값**을 찾는 연산이다. 제품1의 (음료, P1)을 주문한 고객은 C1·C2·C3이고, 제품2의 라면과 음료를 **같은 제조업체에서 모두 주문한 고객**은 C1뿐이므로 두 번째 결과는 (C1, P1)이다.

NoteBase

계산 과정

디비전은 나누는 테이블의 모든 행에 대하여, 남는 속성값과 결합한 행이 원래 테이블에 존재하는지를 확인하는 연산이다. 결과에는 **나누는 테이블에 없는 속성만** 남는다. 따라서 '주문 ÷ 제품1'에는 주문고객만, '주문 ÷ 제품2'에는 주문고객과 제조업체가 남는다.

주문 ÷ 제품1: 제품1의 행은 (제품이름=음료, 제조업체=P1) 하나이다. 주문 테이블에서 이 두 값이 함께 나타나는 행의 주문고객은 C3, C1, C2이다. 세 고객 모두 제품1의 모든 행과 대응하므로 결과는 주문고객 열에 C1·C2·C3이 있는 3행의 테이블이다. 관계대수의 결과에서 행의 나열 순서는 의미가 없으므로 C3·C1·C2로 표시해도 같다.

주문 ÷ 제품2: 제품2에는 라면과 음료가 있으므로, 남는 속성인 **주문고객·제조업체의 조합을 고정**한 상태에서 두 제품을 모두 주문했는지 확인한다. (C1, P1)은 라면과 음료 주문이 모두 있어 조건을 충족한다. (C2, P2)는 과자만, (C2, P1)과 (C3, P1)은 음료만 주문했으므로 조건을 충족하지 못한다. 따라서 결과는 주문고객·제조업체 열에 (C1, P1)만 있는 1행의 테이블이다.

두 연산 결과는 각각 주문고객 C3·C1·C2와 주문고객·제조업체 (C1, P1)을 제시한 ①과 일치한다.

NoteBase

문 7. 트랜잭션 T_1, T_2에 대한 설명으로 옳지 않은 것은? (단, X의 초기값은 1,000이다)

시간 ↓

T_1	T_2	
read(X);		
	read(X);	
$X = X + 500$;		(가)
	$X = X - 200$;	
write(X);		(나)
	write(X);	(다)
COMMIT;		(라)
	COMMIT;	

① (가)에서 T_1의 작업 수행 후 메모리 버퍼에 저장된 X값은 1,000이다.

② (나)에서 T_1의 작업 수행 후 메모리 버퍼에 저장된 X값은 1,500이다.

③ (다)에서 T_2의 작업 수행 후 갱신 손실 문제가 발생한다.

④ (라)에서 “COMMIT”이 “ROLLBACK”으로 변경될 경우 오손 읽기 문제가 발생한다.

정답 4번

간단한 해설

오손 읽기는 다른 트랜잭션이 **아직 확정하지 않은 갱신값을 읽을 때** 발생한다. 여기서는 T_2가 T_1의 쓰기 전에 초기값 1,000을 읽었으므로, (라)를 ROLLBACK으로 바꾸어도 오손 읽기는 발생하지 않는다.

NoteBase

선지별 해설

① O: (가)는 T_1이 읽어 온 값에 500을 더하는 단계이다. T_1 내부의 계산 결과는 1,000 + 500 = 1,500이지만, **아직 write(X)를 실행하지 않았으므로** 메모리 버퍼의 X값은 초기값 1,000이다.

② O: (나)에서는 T_1이 write(X)를 실행하여 자신의 계산 결과 1,500을 메모리 버퍼에 기록한다. 따라서 이 작업 직후 버퍼의 X값은 1,500이다. write는 계산한 값을 반영하는 작업이며, 트랜잭션을 확정하는 COMMIT과는 구별된다.

③ O: T_2는 T_1이 값을 쓰기 전에 초기값 1,000을 읽었으므로, 내부 계산 결과는 1,000 − 200 = 800이다. (다)에서 이 800을 쓰면 T_1이 앞서 기록한 1,500을 덮어써 **500을 더한 갱신 효과가 사라진다**. 이것이 갱신 손실이다. 두 갱신이 모두 반영되었다면 X값은 1,000 + 500 − 200 = 1,300이어야 한다.

④ X: 오손 읽기는 다른 트랜잭션이 갱신했지만 아직 COMMIT하지 않은 값을 읽는 현상이다. 그림에서 T_2의 read(X)는 T_1의 write(X)보다 먼저 실행되어 초기값 1,000을 읽는다. 따라서 T_2는 T_1**의 미확정 갱신값을 읽지 않았다**. (라)의 COMMIT을 ROLLBACK으로 바꾸더라도 이러한 읽기 순서는 달라지지 않으므로, 오손 읽기가 발생한다는 설명은 성립하지 않는다.

NoteBase

문 8. (가)~(다)의 빅데이터 분석 기법과 A~C의 설명이 바르게 연결된 것은?

(가) 군집 분석 (나) 분류 분석 (다) 연관 분석
A. 새로운 데이터가 미리 정해진 기준에 따라 어떤 그룹 또는 등급에 속하는지 판단하는데 사용되는 분석 기법
B. 미리 정해진 기준이 없는 상태에서 유사한 특성이 있는 데이터들을 독립적인 그룹으로 분류하는 분석 기법
C. 데이터 간의 발생 빈도를 분석하여 데이터에서 함께 발생하는 항목 간 숨겨진 규칙을 찾아내는 분석 기법

① (가): A, (나): B, (다): C
② (가): B, (나): A, (다): C
③ (가): B, (나): C, (다): A
④ (가): C, (나): A, (다): B

정답 2번

간단한 해설

군집 분석은 **미리 정한 그룹 없이** 데이터의 유사성을 바탕으로 그룹을 찾는다. 분류 분석은 **미리 정해진 그룹이나 등급** 중 새로운 데이터가 어디에 속하는지 판단한다.

대응 관계 해설

(가) 군집 분석 → B: **미리 정해진 분류 기준 없이** 유사한 특성을 가진 데이터끼리 묶는 기법이다. 예를 들어 고객의 구매 성향을 분석하여 서로 비슷한 고객 집단을 찾아내는 경우이다. B의 '유사한 특성이 있는 데이터들을 독립적인 그룹으로 분류'한다는 표현이 이에 해당한다.

(나) 분류 분석 → A: **미리 정해진 그룹이나 등급**에 새로운 데이터를 배정하는 기법이다. 예를 들어 스팸과 정상 메일로 구분된 기존 자료를 학습한 뒤 새 메일이 어느 쪽인지 판단한다. 그룹 자체를 찾아내는 군집 분석과 달리, A처럼 정해진 범주 중 어디에 속하는지를 판단한다.

(다) 연관 분석 → C: **항목들이 함께 발생하는 관계**를 찾아 규칙으로 나타내는 기법이다. 예를 들어 구매 기록에서 우유와 빵이 함께 구매되는 경향을 찾는 경우이다. C는 데이터의 발생 빈도를 바탕으로 함께 나타나는 항목 간의 숨겨진 규칙을 찾으므로 연관 분석에 해당한다.

(가)는 B, (나)는 A, (다)는 C에 대응하므로 ②의 연결과 일치한다.

NoteBase

문 9. DBMS의 3단계 스키마 구조에서 사상(mapping)과 데이터 독립성에 대한 설명으로 옳지 않은 것은?

① 외부/개념 사상은 외부 스키마의 데이터가 개념 스키마의 어떤 부분에 해당하는지 대응 관계를 정의한 것이다.

② 개념/내부 사상은 개념 스키마의 데이터가 내부 스키마를 통해 물리적 장치에 저장되는 대응 관계를 정의한 것이다.

③ 내부 스키마가 변경되어도 개념/내부 사상을 수정하면 개념 스키마가 영향을 받지 않는다.

④ 개념 스키마가 변경되어도 외부 스키마에 영향이 없는 성질을 물리적 데이터 독립성이라 한다.

정답 4번

간단한 해설

개념 스키마가 바뀌어도 외부 스키마를 유지할 수 있는 성질은 **논리적 데이터 독립성**이다. 물리적 데이터 독립성은 **내부 스키마의 변경**이 개념 스키마에 영향을 주지 않는 성질이다.

선지별 해설

① O: 외부 스키마는 사용자나 응용 프로그램이 필요로 하는 데이터의 모습을, 개념 스키마는 데이터베이스 전체의 논리적 구조를 나타낸다. 외부/개념 사상은 **사용자 관점의 데이터와 전체 논리 구조 사이의 대응 관계**를 정의한다.

② O: 내부 스키마는 레코드의 저장 형식, 파일 구성, 접근 경로 등 물리적 저장 구조를 나타낸다. 개념/내부 사상은 **논리적으로 정의된 데이터를 물리적으로 어떻게 저장하는지**를 연결한다.

③ O: 내부 스키마를 변경하더라도 개념/내부 사상을 수정하여 기존의 개념 스키마를 유지할 수 있다. 이것이 **물리적 데이터 독립성**이다. 예를 들어 검색을 위한 인덱스를 추가해 저장·접근 구조가 달라져도, 테이블의 논리적 구조는 그대로 유지할 수 있다.

④ X: 개념 스키마의 변경에도 외부 스키마를 유지할 수 있는 성질은 **논리적 데이터 독립성**이다. 외부/개념 사상을 조정하여 전체 논리 구조의 변경이 기존 사용자 뷰에 미치는 영향을 막는다. 선지의 '물리적 데이터 독립성'은 '논리적 데이터 독립성'으로 고쳐야 한다. 물리적 데이터 독립성은 내부 스키마와 개념 스키마 사이에서 성립하는 독립성이다.

NoteBase

문 10. 데이터를 여러 디스크에 분배하여 저장하는 구조로, 데이터의 갱신이 중복해서 일어나지 않고, 가장 좋은 쓰기 성능을 갖는 RAID 레벨은?

① RAID 레벨 0

② RAID 레벨 2

③ RAID 레벨 3

④ RAID 레벨 6

정답 1번

간단한 해설

RAID 레벨 0은 데이터를 여러 디스크에 나누어 쓰는 **스트라이핑**만 사용하며, 복제본이나 오류 복구용 정보를 함께 갱신하지 않는다. 이처럼 **추가 쓰기 부담이 없어** 제시된 레벨 중 쓰기 성능이 가장 좋다.

선지별 해설

비교 기준은 데이터를 쓸 때 복제본이나 오류 복구용 부가 정보도 함께 갱신해야 하는지이다. 이러한 중복 정보는 고장에 대비할 수 있게 하지만, 계산과 추가 기록이 필요하다.

① RAID 레벨 0: 데이터를 블록 단위로 나누어 여러 디스크에 분산 저장한다. 여러 디스크에 병렬로 쓸 수 있고, **복제본이나 패리티를 갱신하는 작업이 없다**. 따라서 쓰기 성능에 유리하지만, 디스크 고장 시 데이터를 복구할 중복 정보도 제공하지 않는다.

② RAID 레벨 2: 데이터를 비트 단위로 분산 저장하고, 오류를 검출·정정하는 **해밍 코드 정보**를 별도 디스크들에 저장한다. 데이터를 쓸 때 이 오류 정정 정보도 계산하여 기록해야 하므로, RAID 레벨 0과 달리 복구용 부가 정보의 갱신 부담이 있다.

③ RAID 레벨 3: 데이터를 바이트 단위로 분산 저장하고, 오류 복구에 쓰는 패리티를 전용 디스크에 저장한다. 데이터를 갱신할 때 **전용 디스크의 패리티도 갱신**해야 하므로, 데이터만 기록하는 RAID 레벨 0보다 추가 쓰기 부담이 있다.

④ RAID 레벨 6: 데이터를 블록 단위로 분산 저장하면서 **서로 다른 두 종류의 패리티**도 여러 디스크에 분산 저장한다. 두 디스크의 동시 고장에도 복구할 수 있지만, 데이터를 쓸 때 두 패리티를 계산·갱신하는 부담이 있다.

복구용 부가 정보의 갱신 여부를 비교하면, 이를 저장하지 않고 데이터만 분산 기록하는 RAID 레벨 0이 문제의 조건에 부합하므로 ①을 선택한다.

NoteBase

문 11. 저장 프로시저(stored procedure)에 대한 설명으로 옳지 않은 것은?

① 일반 애플리케이션 함수와 달리 매개변수를 전달할 수 없다.

② 데이터베이스 서버에서 실행되어 클라이언트와 서버 사이의 데이터 전송량을 줄일 수 있다.

③ SQL 외에 다른 언어로 작성될 수 있다.

④ CALL 문장을 가진 대화형 SQL에서 호출될 수 있다.

정답 1번

간단한 해설

저장 프로시저도 일반 애플리케이션 함수처럼 **매개변수를 전달받을 수 있다**. 호출할 때 받은 값을 조건이나 처리 대상 지정에 활용한다.

선지별 해설

① X: 저장 프로시저는 **입력 매개변수를 선언하여 호출 시 값을 전달받을 수 있다**. 예를 들어 부서 번호를 매개변수로 받는 프로시저는 전달받은 번호에 해당하는 부서의 자료를 처리할 수 있다. 따라서 '매개변수를 전달할 수 없다'를 '매개변수를 전달할 수 있다'로 고쳐야 한다.

② O: 저장 프로시저는 데이터베이스 서버에 저장되어 서버에서 실행되는 처리 절차이다. 필요한 연산을 **서버 내부에서 수행한 뒤 결과만 전달**하면, 처리에 필요한 원자료를 클라이언트로 모두 가져오는 방식보다 데이터 전송량을 줄일 수 있다.

③ O: 저장 프로시저를 작성하는 언어가 SQL로만 제한되는 것은 아니다. **DBMS가 지원하는 경우** SQL 이외의 프로그래밍 언어로도 작성할 수 있으며, 지원 언어와 작성 방식은 DBMS에 따라 다르다.

④ O: CALL은 **저장 프로시저를 호출하는 SQL 문장**이다. 이를 지원하는 대화형 SQL 환경에서는 사용자가 CALL 문장에 프로시저 이름과 필요한 인수를 지정하여 서버에 저장된 프로시저를 실행할 수 있다.

NoteBase

문 12. NoSQL 데이터베이스에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. CAP에서 C는 Consistency로, 복제 데이터를 가진 분산 시스템에서 동일한 데이터 항목에 대한 여러 복사본에서 값이 일치해야 함을 의미한다.
ㄴ. CAP의 A는 Availability로, 네트워크 고장으로 분산된 노드들 중 일부에서 작동이 멈추더라도 시스템이 정상적으로 작동하는 것을 의미한다.
ㄷ. NoSQL 데이터베이스는 관계형 데이터베이스와 비교하여 ACID 특성을 엄격히 준수하지 않는다.

① ㄱ, ㄴ
② ㄱ, ㄷ
③ ㄴ, ㄷ
④ ㄱ, ㄴ, ㄷ

정답 2번

간단한 해설

ㄴ은 **가용성(A)**과 **분할 내성(P)**을 혼동한 설명이다. 가용성은 정상 작동 중인 노드에 도달한 요청에 응답하는 성질이고, 분할 내성은 네트워크 장애로 노드 사이의 통신이 끊겨도 시스템이 동작을 계속하는 성질이다.

보기별 해설

ㄱ. O: CAP의 일관성(Consistency)은 같은 데이터에 대한 읽기가 서로 모순되지 않도록 복제본의 값을 일관되게 유지하는 성질이다. 예를 들어 어떤 값의 변경이 완료되었다면, 이후 어느 복제본에서 읽더라도 **완료된 변경을 반영한 값**을 얻어야 한다.

ㄴ. X: 가용성(Availability)은 **정상 작동 중인 노드가 받은 모든 요청에 응답**하는 성질이다. 네트워크 장애로 노드 사이의 통신이 끊긴 상황에서도 시스템이 동작을 계속하는 성질은 분할 내성(Partition tolerance)이다. 따라서 보기의 설명은 A가 아니라 P에 대응한다.

ㄷ. O: ACID는 트랜잭션의 원자성·일관성·격리성·지속성을 뜻한다. NoSQL은 일반적으로 대규모 분산 처리와 수평 확장을 위해 **ACID의 일부 보장을 완화**하는 방식을 취한다. 예를 들어 복제본에 변경이 즉시 반영되지 않더라도 시간이 지나면 값이 같아지는 최종 일관성을 사용할 수 있다. 다만 이는 NoSQL의 일반적 경향을 설명한 것으로, 모든 NoSQL 데이터베이스가 ACID 트랜잭션을 지원하지 않는다는 뜻은 아니다.

옳은 보기는 ㄱ과 ㄷ이므로, 이에 해당하는 조합은 ②이다.

NoteBase

문 13. 관계형 데이터베이스에서 질의 최적화를 위한 조인 연산 처리에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 인덱스 기반 중첩 루프 조인에서 두 테이블의 조인 속성에 모두 인덱스가 존재한다면 투플 수가 적은 테이블을 외부 릴레이션으로 사용하는 것이 비용 측면에서 유리하다.
ㄴ. 정렬 합병 조인은 투플들이 기본키 속성에 대해 정렬되어 있을 때 조인 연산을 효율적으로 수행하는 방법이다.
ㄷ. 두 테이블 R과 S에 대하여 페이지(블록) 기반 중첩 루프 조인을 수행할 경우 R⋈S와 S⋈R의 비용이 다를 수 있다.

① ㄱ, ㄴ

② ㄱ, ㄷ

③ ㄴ, ㄷ

④ ㄱ, ㄴ, ㄷ

정답 2번

간단한 해설

정렬 합병 조인에서 정렬 기준은 **기본키가 아니라 조인 속성**이다. 기본키 순으로 정렬되어 있어도 실제 조인 속성이 다르면 그 정렬을 합병 단계에 그대로 이용할 수 없다.

NoteBase

보기별 해설

ㄱ. O: 인덱스 기반 중첩 루프 조인은 외부 릴레이션의 투플을 하나씩 읽고, 그 투플과 조인할 내부 릴레이션의 투플을 인덱스로 찾는다. 두 테이블의 조인 속성에 모두 인덱스가 있으면 어느 쪽을 내부 릴레이션으로 삼아도 인덱스를 이용할 수 있다. 따라서 **투플 수가 적은 테이블을 외부 릴레이션**으로 삼아 반복되는 인덱스 탐색 횟수를 줄이는 것이 유리하다.

ㄴ. X: 정렬 합병 조인은 두 릴레이션을 **조인 속성의 값 순서로 정렬**한 뒤, 같은 값을 갖는 투플들을 찾아 결합하는 방식이다. 조인 속성이 기본키일 필요는 없다. 예를 들어 학생과 학과 테이블을 학과번호로 조인한다면, 학생 테이블이 기본키인 학번 순으로 정렬되어 있다는 사실만으로는 충분하지 않다. 두 테이블이 조인 속성인 학과번호 순으로 정렬되어 있어야 그 순서를 합병 단계에 이용할 수 있다.

ㄷ. O: 페이지 기반 중첩 루프 조인은 외부 릴레이션의 페이지를 읽은 뒤, 그 페이지의 투플들과 조인할 투플을 찾기 위해 내부 릴레이션의 페이지들을 훑는다. 내부 릴레이션 전체가 메모리에 유지되지 않으면 외부 페이지마다 내부 릴레이션을 반복해서 읽을 수 있다. 따라서 R과 S 중 어느 쪽을 외부 릴레이션으로 삼는지에 따라 페이지 수와 버퍼 크기의 영향을 받아 읽기 비용이 달라질 수 있다. 조인의 교환법칙으로 **결과가 같아도 실행 비용까지 같은 것은 아니다**.

옳은 보기는 ㄱ과 ㄷ이므로, 이에 해당하는 조합은 ②이다.

NoteBase

문 14. 인덱스(index)에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 테이블은 최대 한 개의 기본 인덱스를 가질 수 있고, 여러 개의 보조 인덱스를 가질 수 있다.
ㄴ. 기본 인덱스와 보조 인덱스는 각각 기본키와 탐색키 값에 따라 레코드들이 정렬되어 있어야 정의할 수 있다.
ㄷ. 다단계 인덱스는 단일 단계 인덱스에 비하여 인덱스의 갱신이 빠르므로 관계형 DBMS에서 많이 이용된다.
ㄹ. 클러스터링 인덱스는 물리적으로 정렬된 테이블의 레코드에 대해 생성할 수 있으므로 범위 검색에 유리하다.

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄴ, ㄹ

정답 2번

간단한 해설

기본 인덱스는 레코드의 **물리적 정렬 순서**에 대응하므로 최대 하나이지만, 보조 인덱스는 그 순서와 독립적으로 여러 개를 둘 수 있다. 클러스터링 인덱스는 비슷한 키 값의 레코드가 가까이 모여 있어 **범위 검색**에 유리하다.

NoteBase

보기별 해설

ㄱ. O: 기본 인덱스는 테이블의 물리적 정렬 순서에 대응한다. 하나의 테이블에 서로 다른 물리적 정렬 순서를 동시에 부여할 수 없으므로 **기본 인덱스는 최대 하나**이다. 보조 인덱스는 레코드의 물리적 배치를 바꾸지 않고 별도의 탐색 경로를 제공하므로 여러 개를 만들 수 있다.

ㄴ. X: 기본 인덱스는 레코드가 해당 키 순서로 정렬된 파일을 전제로 하지만, **보조 인덱스는 레코드의 물리적 정렬을 요구하지 않는다**. 보조 인덱스는 탐색키 값과 레코드 위치를 연결하여 접근한다. 예를 들어 학번순으로 저장된 학생 테이블에도 학과를 탐색키로 하는 보조 인덱스를 만들 수 있다. 인덱스 항목의 정렬과 실제 테이블 레코드의 정렬을 구별해야 한다.

ㄷ. X: 다단계 인덱스는 하위 인덱스 위에 상위 인덱스를 두어 탐색 범위를 단계적으로 좁히는 구조이다. 큰 인덱스 전체를 탐색하는 대신 필요한 하위 부분으로 접근하여 **검색에 필요한 디스크 접근을 줄이는 것**이 주된 이점이다. 삽입·삭제 시에는 하위 인덱스뿐 아니라 상위 단계의 항목도 갱신해야 할 수 있으므로, 단일 단계보다 인덱스 갱신이 빠르다는 설명은 부적절하다.

ㄹ. O: 클러스터링 인덱스의 키 순서는 테이블 레코드의 물리적 정렬 순서에 대응한다. 따라서 특정 범위의 키 값을 가진 레코드들이 가까이 모여 있다. 범위의 시작 위치를 찾은 뒤 **인접한 데이터 블록을 순차적으로 읽을 수 있어** 범위 검색에 유리하다.

옳은 보기는 ㄱ, ㄹ이며, 이 조합은 ②에 해당한다.

NoteBase

문 15. 다음 R과 S 테이블에 대해, 주어진 '관계대수'와 동일한 투플 결과를 출력하는 'SQL 문'은? (단, 밑줄로 표기된 속성은 기본키를 의미한다)

R	
A	B
a1	5
a2	10
a3	15
a4	20

S	
C	A
c1	a1
c2	a1
c3	a2
c4	a3

① 관계대수: $\sigma_{(B < 10)}(R) \bowtie_{R.A=S.A} S$
SQL 문:
```
SELECT DISTINCT R.A, R.B, S.C, S.A
FROM R INNER JOIN S
WHERE R.B < 10;
```

② 관계대수: $R \bowtie_N S$
SQL 문:
```
SELECT DISTINCT R.A, R.B, S.C
FROM R, S
WHERE R.A IN (SELECT S.A FROM S);
```

③ 관계대수: $R \times S$
SQL 문:
```
SELECT DISTINCT R.A, R.B
FROM R, S
WHERE R.A = S.A;
```

④ 관계대수: $R \rtimes S$
SQL 문:
```
SELECT DISTINCT R.A, R.B, S.C
FROM R, S
WHERE R.A = S.A;
```

정답 3번

간단한 해설

왼쪽 세미 조인은 S와 A 값이 일치하는 R의 투플을 찾아 **R의 속성만** 남긴다. a1은 S의 두 투플과 연결되지만, R.A와 R.B만 출력한 뒤 **DISTINCT로 중복을 제거**하면 (a1, 5)는 한 번만 남는다.

NoteBase

선지별 해설

① X: 관계대수는 먼저 R에서 B < 10인 (a1, 5)를 고른 뒤, R.A = S.A인 S의 투플과 조인한다. 따라서 (R.A, R.B, S.C, S.A) 순서로 (a1, 5, c1, a1), (a1, 5, c2, a1)이 나온다. SQL에는 B에 대한 조건만 있고 **R.A = S.A라는 조인 조건이 빠져 있다**. INNER JOIN S 뒤에 ON R.A = S.A를 추가해야 관계대수와 같은 결과를 얻는다.

② X: 자연 조인은 공통 속성 A의 값이 같은 투플끼리 결합하고, 중복되는 A 열을 하나만 남긴다. 이때 (A, B, C)는 (a1, 5, c1), (a1, 5, c2), (a2, 10, c3), (a3, 15, c4)이다. 선지의 IN 조건은 R.A가 S 전체의 A 값 중 하나인지만 검사하므로, **바깥쪽 S 투플의 A와 같은지는 검사하지 않는다**. 이에 따라 a1 · a2 · a3 각각이 S의 네 투플 모두와 결합하여 12개 투플을 출력한다. WHERE 조건을 R.A = S.A로 고쳐야 자연 조인과 같은 결과를 얻는다.

③ O: 왼쪽 세미 조인은 S에 같은 A 값을 가진 투플이 존재하는 R의 투플만 남기고, 출력 속성도 R의 A와 B로 한정한다. a1 · a2 · a3은 S에 존재하지만 a4는 없으므로 결과는 (a1, 5), (a2, 10), (a3, 15)이다. SQL의 WHERE R.A = S.A는 일치하는 투플끼리 연결하고, SELECT는 R.A와 R.B만 남긴다. a1이 c1 · c2와 연결되면서 생기는 동일한 (a1, 5) 두 개는 **DISTINCT로 하나가 된다**. 따라서 관계대수와 SQL의 투플 결과가 같다.

④ X: 오른쪽 세미 조인은 R에 같은 A 값을 가진 투플이 존재하는 S의 투플을 남기며, **S의 속성 C와 A만 출력**한다. S의 모든 A 값이 R에 존재하므로 결과는 (c1, a1), (c2, a1), (c3, a2), (c4, a3)이다. 선지의 SQL은 R.A, R.B, S.C를 출력하여 오른쪽 세미 조인과 출력 속성이 다르다. SELECT DISTINCT S.C, S.A로 고치고 FROM R, S 및 WHERE R.A = S.A를 유지해야 같은 결과를 얻는다.

NoteBase

문 16. 다음은 항공권 예약 시스템에서 좌석 예매와 결제 처리를 수행하는 트랜잭션 코드이다. 트랜잭션 수행 중 UPDATE 문이 끝난 뒤 (가) 지점에서 장애가 발생하여 이 트랜잭션이 성공적으로 완료되지 못할 때, 이 상황과 관련된 트랜잭션의 특성과 해결 방법으로 옳은 것은? (단, 동시에 수행 중인 다른 트랜잭션은 없다)

```
BEGIN
UPDATE 좌석
SET 상태 = “예약완료”
WHERE 항공편 = “KE011_KR” AND 좌석번호 = “22A”;
(가)
INSERT INTO 결제(결제번호, 항공편, 금액, 상태)
VALUES (“Q_PAY1011”, “KE011_KR”, 1500000, “승인완료”);
END;
```

① 원자성(Atomicity)을 보장하기 위한 ROLLBACK 처리

② 지속성(Durability)을 보장하기 위한 TRIGGER 처리

③ 고립성(Isolation)을 보장하기 위한 SAVEPOINT 처리

④ 일관성(Consistency)을 보장하기 위한 START TRANSACTION 처리

정답 1번

간단한 해설

좌석만 '예약완료'로 바뀌고 결제 기록은 생성되지 않았으므로, 실패한 트랜잭션의 일부 결과를 남겨서는 안 된다. **원자성**을 보장하려면 **ROLLBACK으로 좌석 변경을 취소**해야 한다.

NoteBase

핵심 풀이

(가)의 상태: UPDATE는 끝났지만 결제 정보를 추가하는 INSERT는 실행되기 전이다. 이때 좌석 변경만 유지하면 결제 기록 없이 좌석만 '예약완료'로 남는다.

원자성과 ROLLBACK: 원자성은 한 트랜잭션의 작업이 **모두 반영되거나 모두 취소되어야 한다**는 성질이다. 좌석 예매와 결제 처리가 하나의 트랜잭션이므로, 전체가 실패하면 이미 수행한 UPDATE도 ROLLBACK으로 취소하여 좌석 상태를 UPDATE 이전 값으로 되돌린다.

다른 특성과의 구분: 지속성은 **성공적으로 확정된 결과**가 이후 장애에도 유지되는 성질이다. 아직 완료되지 못한 작업의 취소와는 구별된다. 고립성은 동시에 실행되는 트랜잭션 사이의 간섭을 막는 성질이며, 이 문제에는 다른 트랜잭션이 없다. 일관성은 트랜잭션 전후에 데이터베이스의 무결성 조건이 유지되는 성질이다. 이 상황에서 일부 작업만 남지 않도록 전체를 취소하는 근거는 원자성이다.

처리 명령의 구분: TRIGGER는 지정된 사건에 반응하여 실행되는 동작이며, 그 자체가 지속성을 보장하는 수단은 아니다. SAVEPOINT는 트랜잭션 안에서 부분적으로 되돌아갈 지점을 지정하며 고립성을 보장하지 않는다. START TRANSACTION은 트랜잭션을 시작하는 명령으로, 이미 수행된 UPDATE를 취소하는 처리가 아니다.

NoteBase

문 17. 트랜잭션 T_1, T_2, T_3에서 데이터 x, y, z에 대한 읽기와 쓰기 작업의 스케줄이 다음과 같을 때 옳은 것만을 모두 고르면?

시간 ↓

T_1	T_2	T_3
read(y)		
read(z)		
	read(x)	
	write(x)	
		read(z)
write(y)		
	read(y)	
	write(y)	
write(z)		
		read(x)

ㄱ. T_2의 write(y) 연산이 없다면 충돌 직렬가능하지 않은 스케줄이다.

ㄴ. T_3의 read(z) 연산이 없다면 충돌 직렬가능 스케줄이다.

ㄷ. T_3의 read(x)가 write(z)로 대체된다면 충돌 직렬가능하지 않은 스케줄이다.

① ㄱ, ㄴ

② ㄱ, ㄷ

③ ㄴ, ㄷ

④ ㄱ, ㄴ, ㄷ

정답 4번

간단한 해설

ㄱ에서 T_2의 write(y)를 삭제해도 T_1의 write(y)와 T_2의 read(y) 사이의 **충돌은 남는다**. 따라서 선행 그래프의 사이클이 유지되어 충돌 직렬가능하지 않다.

NoteBase

보기별 해설

서로 다른 트랜잭션의 연산이 같은 데이터를 접근하고, 적어도 하나가 쓰기이면 두 연산은 충돌한다. 읽기끼리는 충돌하지 않는다. 선행 그래프는 트랜잭션을 정점으로 두고, 충돌하는 연산의 실행 순서에 따라 먼저 실행한 쪽에서 나중에 실행한 쪽으로 간선을 그린다. 이 그래프에 **사이클이 없어야** 충돌 직렬가능하다. 표의 시간은 위에서 아래로 흐르며, 각 보기는 원래 스케줄에 해당 변경만 적용하여 판단한다.

ㄱ. O: T_2의 write(y)를 삭제해도 T_1의 write(y)가 T_2의 read(y)보다 먼저 실행되므로 $T_1 \rightarrow T_2$가 남는다. x에서는 T_2의 write(x)가 T_3의 read(x)보다 먼저이므로 $T_2 \rightarrow T_3$가 생긴다. z에서는 T_3의 read(z)가 T_1의 write(z)보다 먼저이므로 $T_3 \rightarrow T_1$이 생긴다. 따라서 **삭제 후에도** $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$의 사이클이 남아 충돌 직렬가능하지 않다.

ㄴ. O: T_3의 read(z)를 삭제하면 z에 의한 $T_3 \rightarrow T_1$ 간선이 없어진다. 남은 충돌은 y에 의한 $T_1 \rightarrow T_2$와 x에 의한 $T_2 \rightarrow T_3$뿐이므로 **사이클이 없다**. 따라서 충돌 직렬가능하며, 충돌 연산의 순서를 보존하는 직렬 실행 순서는 $T_1 \rightarrow T_2 \rightarrow T_3$이다.

ㄷ. O: 마지막에 있는 T_3의 read(x)를 write(z)로 바꾸면, 앞서 실행되는 T_1의 read(z) 및 write(z)와 충돌하여 $T_1 \rightarrow T_3$가 생긴다. 한편 T_3의 기존 read(z)는 여전히 T_1의 write(z)보다 먼저이므로 $T_3 \rightarrow T_1$도 남는다. 이처럼 **양방향 간선**이 형성되어 $T_1 \rightarrow T_3 \rightarrow T_1$의 사이클이 생기므로 충돌 직렬가능하지 않다.

ㄱ, ㄴ, ㄷ이 모두 성립하므로 이에 해당하는 조합은 ④이다.

NoteBase

문 18. 기본키가 EMP_ID인 두 테이블 EMP1(EMP_ID, NAME, SALARY), EMP2(EMP_ID, ADDRESS)에 대하여 사용자 A1이 다음 순서대로 모두 수행하고 '순서 5'의 실행 결과가 다음과 같을 때, 옳은 설명은? (단, EMP_ID, NAME, SALARY의 데이터 타입은 각각 정수형, 문자열, 정수형이다)

순서	A1이 수행한 SQL 문
1	CREATE VIEW EMP_VIEW1 AS SELECT * FROM EMP1 WHERE SALARY>1000 WITH CHECK OPTION;
2	CREATE VIEW EMP_VIEW2 AS SELECT NAME, ADDRESS FROM EMP1, EMP2 WHERE EMP1.EMP_ID = EMP2.EMP_ID;
3	GRANT SELECT, INSERT ON EMP_VIEW1 TO A2;
4	GRANT SELECT, INSERT ON EMP_VIEW2 TO A2;
5	SELECT * FROM EMP_VIEW1;

<순서 5의 실행 결과>

EMP_ID	NAME	SALARY
1001	홍길동	1200
1002	이순신	1500
1003	신윤복	1100

① EMP1에 EMP_ID가 1004인 투플이 없을 때, A2는 EMP_VIEW1을 통해 EMP1에 새로운 투플 '(1004, "강감찬", 1000)'을 삽입할 수 있다.

② A2는 EMP_VIEW1을 통해 SALARY의 합계를 출력하는 SQL 문을 실행할 수 없다.

③ A2는 EMP_VIEW2에 새로운 투플을 삽입할 수 없다.

④ A1은 EMP_VIEW1에 대한 DELETE 권한을 A2에 부여할 수 없다.

정답 3번

NoteBase

간단한 해설

EMP_VIEW2는 두 테이블을 조인하면서 기본키 EMP_ID를 제외한 뷰이다. INSERT 권한을 받았더라도 **기본 테이블의 새 투플에 필요한 기본키 값을 지정할 수 없으므로** 주어진 뷰를 통해 새 투플을 삽입할 수 없다.

선지별 해설

① X: EMP_VIEW1에는 WITH CHECK OPTION이 지정되어 있으므로, 뷰를 통해 삽입하거나 변경하는 투플도 **SALARY > 1000 조건을 만족해야 한다**. 제시된 투플의 SALARY는 1000이므로 이 조건을 만족하지 않는다. EMP_ID 1004가 기존에 없어 기본키 중복이 발생하지 않더라도 해당 투플의 삽입은 거부된다.

② X: 순서 3에서 A2는 EMP_VIEW1에 대한 **SELECT 권한**을 받았으므로 뷰의 SALARY에 SUM을 적용하는 조회를 실행할 수 있다. SELECT SUM(SALARY) FROM EMP_VIEW1;의 결과는 순서 5에 나타난 세 투플의 SALARY를 합한 1200 + 1500 + 1100 = 3800이다. 뷰를 조회할 때 집계 함수를 사용하는 것과 뷰 자체를 집계 함수로 정의하는 것은 구별해야 한다.

③ O: EMP_VIEW2는 EMP1과 EMP2를 EMP_ID로 조인하여 NAME과 ADDRESS만 보여 주는 뷰이다. 두 기본 테이블의 **기본키 EMP_ID가 뷰에 포함되어 있지 않으므로**, NAME과 ADDRESS를 삽입하는 것만으로는 기본 테이블에 새 투플을 만들 때 필요한 기본키 값을 지정할 수 없다. 따라서 주어진 뷰 정의로는 새 투플을 삽입할 수 없다. 순서 4의 INSERT 권한 부여는 연산에 대한 접근 허용이며, 삽입할 수 없는 뷰를 삽입 가능한 구조로 바꾸지는 않는다.

④ X: A1은 EMP_VIEW1을 생성한 **뷰의 소유자**이므로 A2에게 해당 뷰의 DELETE 권한을 부여할 수 있다. GRANT DELETE ON EMP_VIEW1 TO A2;로 부여하면 된다. 순서 3에서 SELECT와 INSERT만 부여했다는 것은 A2가 현재 DELETE 권한을 받지 않았다는 뜻이며, A1이 이를 추가로 부여할 수 없다는 뜻은 아니다.

NoteBase

문 19. 상품 주문 관리 데이터베이스를 구성하는 세 테이블의 스키마가 다음과 같다. 2026년도(출시연도)에 출시된 PAD(상품종류)가 두 개 이상일 때, 해당 PAD 상품을 모두 한 번 이상 주문한 고객의 고객번호를 출력하는 SQL 문은? (단, 각 테이블의 밑줄 친 속성은 기본키이고, 주문.고객번호와 주문.상품번호는 각각 고객.고객번호와 상품.상품번호를 참조하는 외래키이다)

고객(고객번호, 이름)
상품(상품번호, 상품종류, 출시연도)
주문(주문번호, 고객번호, 상품번호, 주문일자)

①
```
SELECT DISTINCT C.고객번호
FROM 고객 C, 상품 P, 주문 O
WHERE C.고객번호 = O.고객번호 AND P.상품번호 = O.상품번호
    AND P.상품종류 = “PAD” AND P.출시연도 = 2026
GROUP BY C.고객번호
HAVING COUNT(*) = (SELECT COUNT(*) FROM 상품
    WHERE 상품종류 = “PAD” AND 출시연도 = 2026);
```

②
```
SELECT DISTINCT C.고객번호
FROM 고객 C
WHERE NOT EXISTS (
    SELECT *
    FROM 상품 P
    WHERE P.상품종류 = “PAD” AND P.출시연도 = 2026
    AND NOT EXISTS (
        SELECT *
        FROM 주문 O
        WHERE C.고객번호 = O.고객번호 AND P.상품번호 =
            O.상품번호));
```

③
```
SELECT DISTINCT C.고객번호
FROM 고객 C INNER JOIN 주문 O ON C.고객번호 =
   O.고객번호 INNER JOIN 상품 P ON P.상품번호 = O.상품번호
WHERE P.상품종류 = “PAD” AND P.출시연도 = 2026
GROUP BY C.고객번호
HAVING COUNT(*) >= 1;
```

④
```
SELECT DISTINCT C.고객번호
FROM 고객 C
WHERE C.고객번호 NOT IN (
    SELECT 고객번호
    FROM 주문
    WHERE 상품번호 NOT IN (
        SELECT 상품번호
        FROM 상품
        WHERE 상품종류 = “PAD” AND 출시연도 = 2026));
```

정답 2번

간단한 해설

주문하지 않은 대상 상품이 하나도 없는 고객을 찾으면 모든 대상 상품을 한 번 이상 주문한 고객이 된다. 이중 NOT EXISTS는 안쪽에서 상품별 주문 누락을 찾고, 바깥쪽에서 그런 누락이 없는 고객만 남긴다.

NoteBase

핵심 풀이

대상 상품: 상품종류가 'PAD'이고 출시연도가 2026인 상품 전체가 대상이며, 문제에서 이 상품이 두 개 이상이라고 주어졌다. 고객은 그 상품들을 각각 한 번 이상 주문해야 한다. **2026은 출시연도 조건**이므로 주문일자가 2026년일 필요는 없고, 대상 밖 상품을 함께 주문해도 된다.

안쪽 NOT EXISTS: ②의 가장 안쪽 주문 조회는 C.고객번호 = O.고객번호와 P.상품번호 = O.상품번호로 현재 고객 C가 현재 상품 P를 주문했는지 확인한다. 주문이 한 건이라도 있으면 NOT EXISTS는 거짓이고, 한 건도 없으면 참이다. 따라서 상품 조회에는 그 고객이 **한 번도 주문하지 않은 대상 상품**만 남는다.

바깥쪽 NOT EXISTS: 위 상품 조회의 결과가 비어 있을 때만 고객을 출력한다. 이는 '주문하지 않은 대상 상품이 없다', 즉 **대상 상품을 모두 한 번 이상 주문했다**는 뜻이다. 예를 들어 대상이 A와 B라면 A만 주문한 고객은 B가 누락 상품으로 남아 제외되고, A와 B를 모두 주문한 고객은 남는다. 같은 상품을 여러 번 주문해도 주문의 존재 여부는 바뀌지 않는다.

주문 건수와 상품 수의 차이: ①의 COUNT(*)는 고객별 대상 상품 주문 건수를 센다. 주문번호가 기본키여도 같은 고객이 같은 상품을 여러 번 주문할 수 있다. 대상이 A와 B일 때 A만 두 번 주문해도 건수는 2가 되므로 전체 상품 수와 같아질 수 있다. '모두 주문'을 개수로 확인하려면 **COUNT(DISTINCT O.상품번호)**처럼 서로 다른 대상 상품번호의 수를 세어 전체 대상 상품 수와 비교해야 한다.

일부 주문과의 차이: ③의 HAVING COUNT(*) >= 1은 대상 상품 주문이 한 건 이상인 고객을 찾는다. 대상이 둘 이상이므로 **하나 이상 주문**했다는 사실만으로 모든 상품의 주문을 보장할 수 없다.

주문 범위 제한과의 차이: ④는 대상 밖 상품을 주문한 고객을 제외하는 구조이다. 대상 상품을 모두 주문했는지는 확인하지 않으므로 대상 중 일부만 주문했거나 주문 자체가 없는 고객도 포함될 수 있다. 또한 대상 상품을 모두 주문했더라도 다른 상품을 함께 주문한 고객은 제외될 수 있다. **대상 밖 상품을 주문하지 않음**과 대상 상품을 모두 주문함은 서로 다른 조건이다.

NoteBase

문 20. 인사관리 시스템의 '사원', '퇴사예정' 테이블에 대하여 다음 요구사항을 반영하여 적절하게 구현한 트리거 생성문은? (단, 모든 SQL 문은 정상적으로 수행되며, old, new는 변경 전후 투플 지정자를 의미한다)

◯ '사원' 테이블에서 어떤 사원의 '재직상태'가 "퇴사예정"으로 값이 변경되면, 해당 사원의 투플이 '퇴사예정' 테이블에 추가된다.
◯ '퇴사예정' 테이블의 속성은 '사원번호'와 '이름'이며, 이 정보는 '사원' 테이블에도 포함되어 있다.

①
```
CREATE TRIGGER Employee_Trigger
BEFORE UPDATE ON 사원 FOR EACH ROW
BEGIN
   IF old.재직상태 = "퇴사예정" THEN
      INSERT INTO 퇴사예정 VALUES(old.사원번호, old.이름);
   END IF;
END;
```

②
```
CREATE TRIGGER Employee_Trigger
AFTER UPDATE ON 사원 FOR EACH ROW
BEGIN
   IF new.재직상태 = "퇴사예정" THEN
      INSERT INTO 퇴사예정 VALUES(old.사원번호, old.이름);
   END IF;
END;
```

③
```
CREATE TRIGGER Employee_Trigger
BEFORE INSERT ON 사원 FOR EACH ROW
BEGIN
   IF old.재직상태 = "퇴사예정" THEN
      INSERT INTO 퇴사예정 VALUES(old.사원번호, old.이름);
   END IF;
END;
```

④
```
CREATE TRIGGER Employee_Trigger
AFTER INSERT ON 사원 FOR EACH ROW
BEGIN
   IF new.재직상태 = "퇴사예정" THEN
      INSERT INTO 퇴사예정 VALUES(old.사원번호, old.이름);
   END IF;
END;
```

정답 2번

간단한 해설

기존 사원의 재직상태 변경은 **UPDATE** 사건이며, '퇴사예정'이 되었는지는 변경 후 값인 **new.재직상태**로 판단한다. 이 조건을 만족하면 해당 사원의 사원번호와 이름을 '퇴사예정' 테이블에 추가한다.

NoteBase

공통 해설

트리거는 지정한 사건이 발생하면 조건을 확인하여 정해진 동작을 자동으로 수행한다. 이 요구사항은 '사원 테이블의 행 갱신 → 갱신 후 재직상태 확인 → 퇴사예정 테이블에 행 추가'라는 관계로 판단한다. 기존 행의 값을 바꾸는 사건은 UPDATE이고, INSERT는 새 행을 넣는 사건이다. 따라서 사원 테이블의 INSERT를 감시하는 트리거는 기존 사원의 재직상태 변경에 반응하지 않는다. 본문의 INSERT INTO 퇴사예정은 조건을 만족했을 때 수행할 동작이므로, 트리거가 감시할 사건과 구별해야 한다.

old는 변경 전 투플, new는 변경 후 투플을 가리킨다. 요구사항은 변경 결과가 '퇴사예정'인지를 묻기 때문에 조건은 **new.재직상태 = '퇴사예정'**이어야 한다. old.재직상태를 검사하면 변경 전부터 '퇴사예정'이던 사원을 대상으로 삼게 된다. 예를 들어 '재직'에서 '퇴사예정'으로 바뀌는 행은 old 조건을 만족하지 않아 필요한 추가 동작이 실행되지 않는다.

AFTER UPDATE ON 사원 FOR EACH ROW는 사원 테이블에서 갱신된 각 행에 대해 갱신 후 트리거를 실행한다. 여기서 new.재직상태를 검사하고, 조건을 만족하면 INSERT INTO 퇴사예정 VALUES(old.사원번호, old.이름)으로 해당 사원의 변경 전 투플에 있던 사원번호와 이름을 추가한다. '퇴사예정' 테이블에는 이 두 속성만 있으므로 재직상태까지 넣는 것이 아니라 해당 사원을 나타내는 두 값을 넣는다.

NoteBase

문 21. 다음 SQL 문으로 오류 없이 생성된 '고객'과 '주문' 테이블에 다음과 같이 투플이 삽입되어 있을 때 성공적으로 수행되는 SQL 문은?

```
CREATE TABLE 고객 (
    고객번호 VARCHAR(2),
    이름      VARCHAR(10),
    추천고객 VARCHAR(2),
    PRIMARY KEY(고객번호),
    FOREIGN  KEY(추천고객)  REFERENCES  고객(고객번호)
    ON DELETE CASCADE
);

CREATE TABLE 주문 (
    주문번호 VARCHAR(2),
    고객      VARCHAR(2),
    제품      VARCHAR(10),
    수량      INT,
    PRIMARY KEY(주문번호),
    FOREIGN KEY(고객) REFERENCES 고객(고객번호)
);
```

고객

고객번호	이름	추천고객
C1	김유신	NULL
C2	홍길동	C1
C3	이순신	C2
C4	임꺽정	C3
C5	유관순	C1

주문

주문번호	고객	제품	수량
O1	C1	HDD	2
O2	C1	SSD	1
O3	C4	USB	5

① INSERT INTO 주문 VALUES ("O4", "C6", "VGA", 2);

② DELETE FROM 고객 WHERE 고객번호 = "C3";

③ DELETE FROM 고객 WHERE 고객번호 = "C5";

④ UPDATE 고객 SET 고객번호="C7" WHERE 고객번호="C4";

정답 3번

NoteBase

간단한 해설

C5를 추천고객으로 둔 고객도, C5의 주문도 없으므로 C5를 삭제해도 **참조 무결성을 위반하지 않는다**. C5의 추천고객이 C1이라는 사실은 C5를 삭제할 때 C1까지 삭제된다는 뜻이 아니다.

선지별 해설

외래 키의 값은 NULL이 아니라면 참조 대상 테이블에 존재해야 한다. 고객 테이블의 추천고객과 주문 테이블의 고객은 모두 고객번호를 참조한다. 이때 ON DELETE CASCADE는 참조되는 고객을 삭제하면 그 고객을 참조하는 행도 함께 삭제한다는 뜻이며, 문제에서는 고객 테이블의 추천고객 외래 키에만 지정되어 있다. 각 SQL 문은 주어진 최초 테이블 상태에서 독립적으로 판단한다.

① X: 새 주문의 고객 값 C6은 고객 테이블에 없다. 따라서 주문번호 O4가 새 값이어도 **존재하지 않는 고객을 참조**하므로 삽입할 수 없다. 삽입하려면 C6 고객을 먼저 등록하거나 주문의 고객 값을 기존 고객번호로 바꾸어야 한다.

② X: C4의 추천고객이 C3이므로 C3을 삭제하면 ON DELETE CASCADE에 따라 C4도 연쇄 삭제 대상이 된다. 그런데 주문 O3이 C4를 참조하고, 주문 테이블의 외래 키에는 연쇄 삭제가 지정되어 있지 않다. C4를 삭제하면 O3의 참조 대상이 사라지므로 **C3의 삭제 문 전체가 실패**한다. C3을 삭제하려면 먼저 O3을 삭제하는 등 주문의 참조를 해소해야 한다.

③ O: 고객 테이블의 추천고객 열에 C5가 없고 주문 테이블의 고객 열에도 C5가 없다. 따라서 C5를 삭제해도 참조 대상을 잃는 행이 없어 성공적으로 수행된다. C5의 추천고객 값 C1은 **C5가 C1을 참조**한다는 뜻이므로, C5를 삭제해도 C1은 삭제되지 않는다.

④ X: 주문 O3의 고객 값이 C4이므로 고객번호를 C4에서 C7로 바꾸면 O3이 기존 값 C4를 계속 참조하게 된다. 주문의 고객 값까지 함께 바꾸는 ON UPDATE CASCADE가 지정되어 있지 않아 참조 무결성 위반으로 갱신이 실패한다. **ON DELETE CASCADE는 갱신에 적용되지 않는다**. 이 갱신을 허용하려면 주문의 해당 외래 키에 ON UPDATE CASCADE를 지정하는 등 갱신 후에도 참조가 유지되도록 해야 한다.

NoteBase

문 22. 릴레이션 R(A, B, C, D, E)에 대한 함수 종속성 집합 F가 다음과 같을 때, 이에 대한 설명으로 옳은 것은? (단, 모든 속성의 도메인은 원자값으로 구성된다)

$F = \{(A, B) \rightarrow C, C \rightarrow D, D \rightarrow E, E \rightarrow B\}$

① 유일하게 존재하는 후보키 (A, B)가 기본키이다.

② F에서 E → B를 삭제하면 (A, B)는 후보키가 아니다.

③ (A, C, D)의 폐포(closure)는 {A, B, C, D, E}이다.

④ R은 보이스/코드 정규형(BCNF)의 요건을 충족한다.

정답 3번

간단한 해설

속성 집합 (A, C, D)에서 D → E, E → B를 적용하면 빠져 있던 E와 B를 얻는다. 따라서 **폐포에는 R의 모든 속성**인 A, B, C, D, E가 포함된다.

NoteBase

선지별 해설

속성 집합의 폐포는 주어진 함수 종속성으로 그 집합에서 결정할 수 있는 모든 속성의 집합이다. 폐포가 릴레이션의 전체 속성을 포함하면 슈퍼키이고, 그중 속성을 하나라도 빼면 슈퍼키가 되지 못하는 최소의 속성 집합이 후보키이다.

① X: A는 어떤 함수 종속성의 오른쪽에도 없으므로 모든 후보키에 반드시 포함된다. A에 B, C, D, E 중 하나를 더하면 주어진 함수 종속성을 연쇄적으로 적용하여 모든 속성을 얻지만, A만으로는 다른 속성을 얻을 수 없다. 따라서 후보키는 **(A, B), (A, C), (A, D), (A, E)의 네 개**이다. 기본키는 후보키 중 하나를 선택한 것이므로, 주어진 함수 종속성만으로 (A, B)가 기본키라고 확정할 수도 없다.

② X: E → B를 삭제해도 (A, B) → C, C → D, D → E를 차례로 적용하면 C, D, E를 모두 얻는다. 따라서 여전히 $\{A, B\}^+ = \{A, B, C, D, E\}$이다. A만으로도, B만으로도 전체 속성을 결정할 수 없으므로 최소성도 유지되어 **(A, B)는 여전히 후보키**이다.

③ O: 처음 가진 속성은 A, C, D이다. D → E로 E를 추가하고, E → B로 B를 추가하면 전체 속성 A, B, C, D, E를 얻는다. 따라서 $\{A, C, D\}^+ = \{A, B, C, D, E\}$이다. 폐포는 원래 집합에서 결정되는 속성을 모두 모은 것이므로, **원래 집합이 후보키일 필요는 없다**.

④ X: BCNF에서는 성립하는 모든 비자명한 함수 종속성의 **결정자가 슈퍼키**여야 한다. 그런데 C → D에서 C의 폐포는 $\{C\}^+ = \{B, C, D, E\}$로, A를 포함하지 않으므로 C는 슈퍼키가 아니다. 이 함수 종속성이 BCNF 조건을 위반하므로 R은 BCNF를 충족하지 않는다. 모든 도메인이 원자값이라는 조건은 제1정규형에 해당하며, 그것만으로 BCNF가 보장되지는 않는다.

NoteBase

문 23. 병렬 데이터베이스 시스템과 관련된 설명으로 옳지 않은 것은?

① 메모리 공유 구조는 이완 결합(loosely-coupled) 구조로 프로세서가 주기억 장치를 공유한다.

② 디스크 공유 구조는 프로세서가 디스크를 공유하며 각 노드에 자체 주기억장치를 가지고 있다.

③ 디스크 공유 구조에서는 네트워크 병목 현상이 발생할 수 있다.

④ 비공유 구조는 각 노드가 자체 프로세서, 메모리, 디스크를 가지며, 노드 간에는 네트워크로 연결된다.

정답 1번

간단한 해설

①의 '이완 결합'은 **밀결합(tightly-coupled)**으로 고쳐야 한다. 메모리 공유 구조는 여러 프로세서가 같은 주기억장치에 접근하고, 그 안의 데이터를 통해 서로 통신하는 구조이다.

선지별 해설

① X: 메모리 공유 구조는 여러 프로세서가 공통 주기억장치를 공유하는 **밀결합(tightly-coupled) 구조**이다. 프로세서들은 공유 메모리에 데이터를 읽고 쓰면서 통신한다. 따라서 주기억장치를 공유한다는 부분은 맞지만, 이를 '이완 결합'이라고 한 부분이 잘못되었다.

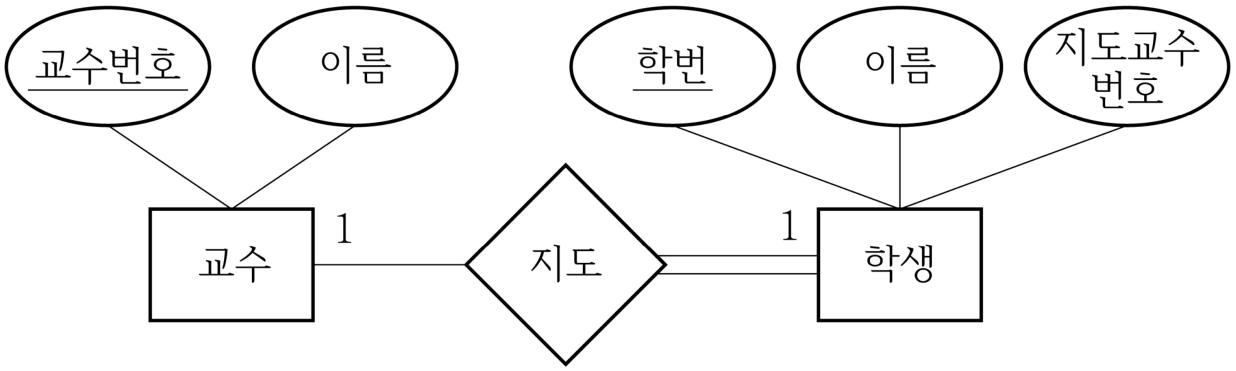
② O: 디스크 공유 구조에서는 각 노드가 **자체 주기억장치**를 가지면서 공통 디스크에 접근한다. 즉, 다른 노드와 공유하는 자원은 디스크이고, 메모리는 노드마다 독립적이다.

③ O: 디스크 공유 구조에서는 여러 프로세서의 디스크 입출력 데이터가 **공유 디스크로 연결되는 상호연결망**을 통과한다. 입출력 요구가 몰려 이 연결망의 전송 용량이 부족해지면 대기가 늘어나 네트워크 병목 현상이 발생할 수 있다.

④ O: 비공유 구조에서는 각 노드가 프로세서, 메모리, 디스크를 독립적으로 가지며, **노드 사이에 메모리와 디스크를 공유하지 않는다**. 다른 노드의 데이터가 필요하면 네트워크를 통해 요청하고 전달받으므로 노드 간 연결과 통신이 필요하다.

NoteBase

문 24. 다음은 교수와 학생의 관계를 나타낸 E-R 다이어그램이다. 이를 기준으로 '교수'와 '학생' 테이블을 생성하였을 때 옳은 것만을 모두 고르면? (단, '교수번호'와 '학번'은 기본키이고, '지도교수번호'는 학생의 지도교수 정보로, '교수번호'를 참조하는 외래키이다)



ㄱ. 기존 다른 학생과 동일한 지도교수번호를 갖는 새로운 학생 정보를 '학생' 테이블에 삽입할 수 있다.
ㄴ. '학생' 테이블의 '지도교수번호' 속성에 없는 번호를 교수번호로 갖는 새로운 교수 정보를 '교수' 테이블에 삽입할 수 있다.
ㄷ. '지도교수번호'가 NULL인 새로운 학생 정보를 '학생' 테이블에 삽입할 수 있다.
ㄹ. '교수'와 '학생' 테이블로부터 지도 학생이 없는 교수의 '교수번호'를 출력할 수 있다.

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄴ, ㄹ

정답 4번

간단한 해설

교수 쪽은 **부분 참여**이므로 지도 학생이 없는 교수도 등록하고 조회할 수 있다. 학생 쪽 이중선은 **모든 학생에게 지도교수가 필요**함을 뜻한다.

NoteBase

보기별 해설

다이어그램에서 관계 양쪽의 '1'은 교수와 학생이 1:1 관계임을 나타낸다. 학생 쪽 이중선은 모든 학생이 지도 관계에 참여하는 전체 참여를, 교수 쪽 단일선은 지도 관계에 참여하지 않는 교수도 허용하는 부분 참여를 뜻한다.

ㄱ. X: 1:1 관계이므로 한 교수가 지도할 수 있는 학생은 최대 한 명이다. 서로 다른 학생이 같은 지도교수번호를 가지면 한 교수가 여러 학생을 지도하게 되어 이 제약을 위반한다. 따라서 새 학생에게는 **다른 학생이 참조하지 않는 교수번호**를 지정해야 하며, 테이블에서도 지도교수번호의 중복을 허용하지 않아야 한다.

ㄴ. O: 교수는 지도 관계에 부분 참여하므로 **지도 학생이 없는 상태로 등록**할 수 있다. 외래키 제약은 학생의 지도교수번호가 교수 테이블에 존재하는 교수번호를 참조하도록 요구하지만, 모든 교수번호가 학생 테이블에서 참조되어야 한다고 요구하지는 않는다. 새 교수번호가 기존 교수번호와 중복되지 않는 등 기본키 제약을 만족하면 삽입할 수 있다.

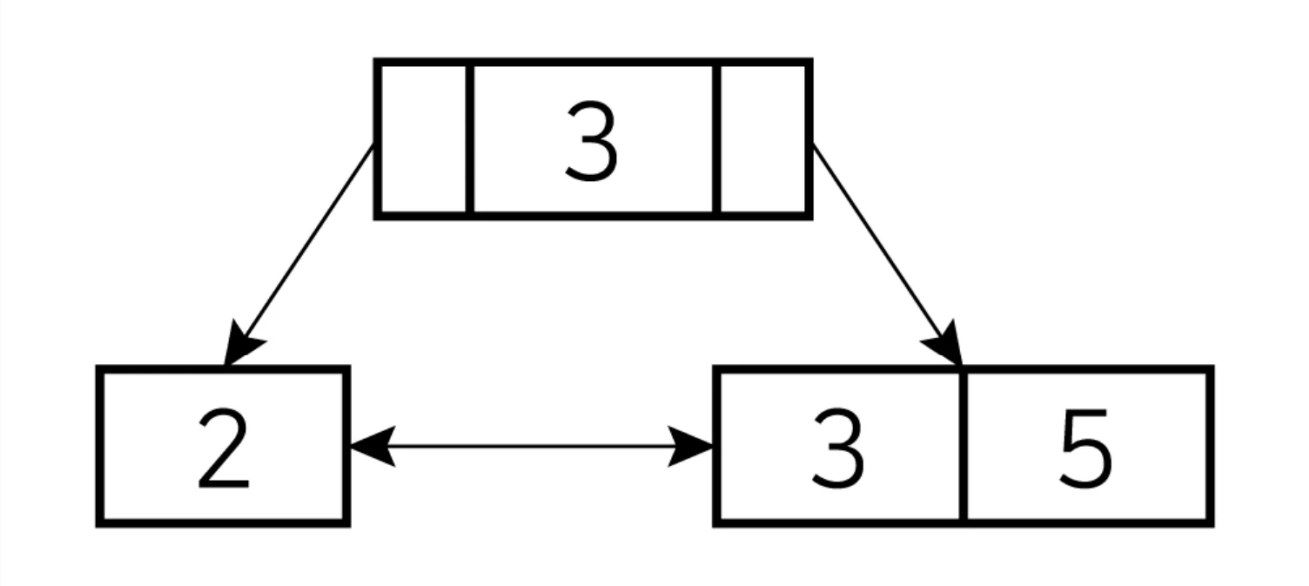
ㄷ. X: 학생 쪽 이중선은 학생의 전체 참여를 나타내므로 모든 학생에게 지도교수가 반드시 있어야 한다. 지도교수번호가 NULL이면 어떤 교수와도 지도 관계를 맺지 못하므로 삽입할 수 없다. 지도교수번호에는 **NULL이 아닌, 교수 테이블에 존재하는 교수번호**를 지정해야 한다.

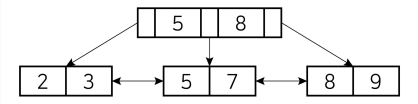
ㄹ. O: 교수 테이블의 교수번호 중 **학생 테이블의 지도교수번호에 나타나지 않는 값**을 찾으면 된다. 각 교수에 대해 그 교수번호를 참조하는 학생이 존재하지 않는지 검사하면 지도 학생이 없는 교수의 교수번호를 출력할 수 있다.

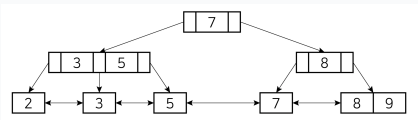
옳은 보기는 ㄴ과 ㄹ이며, 이 조합은 ④에 해당한다.

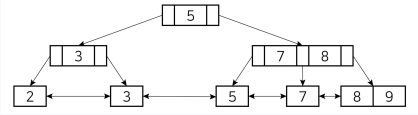
NoteBase

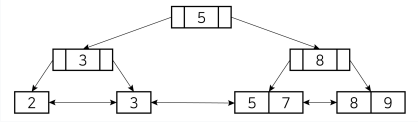
문 25. 키값 2, 3, 5를 차례대로 삽입하여 다음과 같은 B^+-트리가 생성되었다. 이후 9, 8, 7을 차례대로 삽입한 후 B^+-트리로 옳은 것은? (단, B^+-트리의 차수는 3이다)



① 

② 

③ 

④ 

정답 4번

간단한 해설

리프가 가득 차면 분할하고, **오른쪽 리프의 첫 키를 부모에 복사**한다. 8 삽입 후에는 기존 루트까지 분할되어 새 루트가 [5]가 되며, 7을 넣은 뒤 리프는 왼쪽부터 [2], [3], [5, 7], [8, 9]가 된다.

NoteBase

계산 과정

삽입 규칙: 차수가 3이므로 내부 노드의 자식은 최대 3개이고, 키는 최대 3 − 1 = 2개이다. 주어진 그림에서 리프도 최대 2개의 키를 담으며, 처음 [2, 3, 5]가 [2]와 [3, 5]로 나뉘어 있다. 이 문제에서는 리프에 키가 3개가 되면 왼쪽에 1개, 오른쪽에 2개를 두고 분할한다. 부모에는 **오른쪽 리프의 첫 키를 복사**하므로 그 키는 리프에도 남는다. 내부 노드가 넘치면 가운데 키를 부모로 올리고, 나머지 키와 자식들을 양쪽으로 나눈다.

9 삽입: 초기 루트는 [3]이고 리프는 [2], [3, 5]이다. 9는 오른쪽 리프에 들어가 [3, 5, 9]가 된다. 이를 [3]과 [5, 9]로 나누고 5를 부모에 복사하면 루트는 [3, 5], 리프는 왼쪽부터 [2], [3], [5, 9]가 된다.

8 삽입: [5, 9]에 8을 넣으면 [5, 8, 9]가 되므로 [5]와 [8, 9]로 나누고 8을 부모에 복사한다. 기존 루트는 [3, 5, 8]이 되어 허용된 키 2개를 초과한다. 따라서 **가운데 키 5를 새 루트로 올리고**, 왼쪽 내부 노드는 [3], 오른쪽 내부 노드는 [8]로 나눈다. 왼쪽 내부 노드의 자식은 [2]와 [3], 오른쪽 내부 노드의 자식은 [5]와 [8, 9]이다.

7 삽입: 루트의 5를 기준으로 오른쪽으로 이동하고, 내부 노드의 8보다 작으므로 리프 [5]에 넣는다. [5, 7]은 키가 2개이므로 더 분할하지 않는다.

최종 구조: 루트는 [5], 그 아래 내부 노드는 왼쪽 [3]과 오른쪽 [8]이다. 리프는 왼쪽부터 [2], [3], [5, 7], [8, 9]이며 이 순서로 서로 연결된다. 이 구조가 ④의 그림과 일치한다.

NoteBase