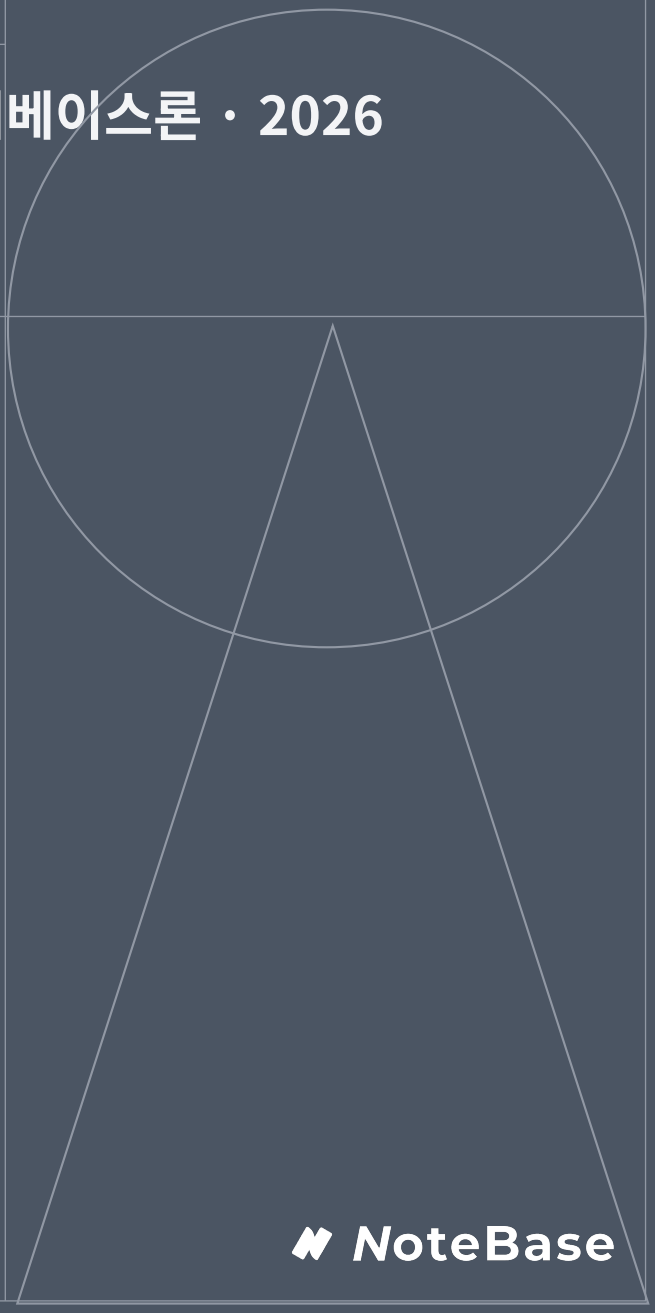
기출OX 해설집

공무원/국가직/9급 · 데이터베이스론 · 2026



NoteBase

기출OX 해설집

공무원/국가직/9급 · 데이터베이스론 · 2026

공무원/국가직/9급 · 데이터베이스론 · 2026

문 1. 데이터베이스 정의에 대한 설명으로 옳지 않은 것은?

① 데이터 중복을 최소화하고 통제하는 통합 데이터의 집합이다.

② 여러 사용자가 함께 보유하고 이용하는 공용 데이터의 집합이다.

③ 컴퓨터가 접근하는 매체에 데이터를 보관하는 저장 데이터의 집합이다.

④ 데이터의 저장 위치를 통해 내용을 참조하는 운영 데이터의 집합이다.

정답 4번

간단한 해설

④의 '저장 위치를 통해 내용을 참조한다'는 설명은 운영 데이터의 뜻이 아니다. **운영 데이터**는 조직의 업무 수행에 필요한 데이터이며, 데이터베이스에서는 저장 위치보다 **데이터의 내용으로 참조**한다.

선지별 해설

① O: 통합 데이터는 여러 곳에 흩어진 데이터를 하나의 체계로 모아 관리하는 데이터이다. 동일한 데이터를 불필요하게 중복 보관하지 않도록 **중복을 최소화하고 통제**한다는 뜻이며, 모든 중복을 반드시 없앤다는 뜻은 아니다.

② O: 공용 데이터는 여러 사용자나 응용 프로그램이 각자의 목적에 맞게 **함께 이용**하는 데이터이다. 사용자마다 별도의 데이터 집합을 독립적으로 관리하는 것과 구별된다.

③ O: 저장 데이터는 컴퓨터가 접근할 수 있는 **저장 매체에 보관**된 데이터이다. 사람이 기억하거나 종이에만 기록한 자료와 달리 컴퓨터로 처리할 수 있도록 저장되어 있다는 의미이다.

④ X: 운영 데이터는 조직이 고유한 업무를 수행하는 데 **필요한 데이터**이다. '저장 위치를 통해 내용을 참조한다'는 접근 방식으로 정의하지 않는다. 데이터베이스의 내용에 의한 참조는 물리적 저장 위치를 직접 지정하는 대신 **데이터의 값이나 조건으로 찾는 것**이다. 예를 들어 학생 정보에서 학번을 조건으로 학생을 조회하는 방식이다.

NoteBase

문 2. 개체－관계 모델에 대한 설명으로 옳지 않은 것은?

① 개체는 강한 개체와 약한 개체로 구분할 수 있다.

② 널(NULL) 값은 아직 결정되지 않았거나 모르는 값을 의미한다.

③ 관계는 개체 간의 의미 있는 연관성으로 개체와 달리 속성을 갖지 않는다.

④ 속성은 의미의 분해 가능성에 따라 단순 속성과 복합 속성으로 구분할 수 있다.

정답 3번

간단한 해설

관계는 개체 사이의 의미 있는 연관성이며, **관계 자체도 속성을 가질 수 있다**. 예를 들어 학생과 과목 사이의 '수강' 관계에는 성적을 속성으로 둘 수 있다.

선지별 해설

① O: 강한 개체는 자신의 속성으로 이루어진 키로 독립적으로 식별할 수 있다. 약한 개체는 자신의 속성만으로는 완전히 식별할 수 없어 **소유 개체의 키와 자신의 부분키를 함께 사용**하여 식별한다.

② O: 널(NULL)은 속성값이 아직 결정되지 않았거나 알려져 있지 않은 상태 등을 나타낸다. 실제 값이 없거나 알려지지 않았음을 표시하므로, **값이 정해진 숫자 0이나 빈 문자열과 구별**한다.

③ X: 관계는 개체 사이의 의미 있는 연관성을 나타내며, **관계에도 속성을 정의할 수 있다**. 예를 들어 학생과 과목 사이의 '수강' 관계에서 성적은 특정 학생이 특정 과목을 수강한 결과를 나타내므로 수강 관계의 속성이 된다. 따라서 '속성을 갖지 않는다'는 '속성을 가질 수 있다'로 고쳐야 한다.

④ O: 단순 속성은 의미상 더 작은 구성 요소로 나눌 수 없는 속성이고, 복합 속성은 **의미 있는 하위 속성으로 나눌 수 있는** 속성이다. 예를 들어 주소를 시·구·도로명 등으로 나누어 표현한다면 주소는 복합 속성이다.

NoteBase

문 3. 관계 데이터 모델에서 키(key)에 대한 설명으로 옳지 않은 것은?

① 슈퍼키(super key)는 널 값을 포함할 수 없다.

② 기본키(primary key)는 유일성과 최소성을 만족해야 한다.

③ 대체키(alternate key)는 기본키가 아닌 후보키를 의미한다.

④ 후보키(candidate key)는 유일성과 최소성을 만족해야 한다.

정답 1번

간단한 해설

슈퍼키는 튜플을 유일하게 식별하면 되므로, 추가로 포함한 속성에 **널 값이 있어도 된다**. 키를 구성하는 모든 속성에 널 값을 금지하는 제약은 **기본키**에 적용된다.

선지별 해설

① X: 슈퍼키는 튜플을 유일하게 식별하는 속성 또는 속성 집합이며, 최소성을 요구하지 않는다. 따라서 식별에 불필요한 속성을 추가할 수 있고, 그 속성에는 **널 값이 있어도 된다**. 예를 들어 학번이 기본키이면 {학번, 전화번호}도 슈퍼키이며, 전화번호가 널이어도 학번으로 학생을 구별할 수 있다. 널 값을 포함할 수 없다는 제약은 기본키를 구성하는 모든 속성에 적용된다.

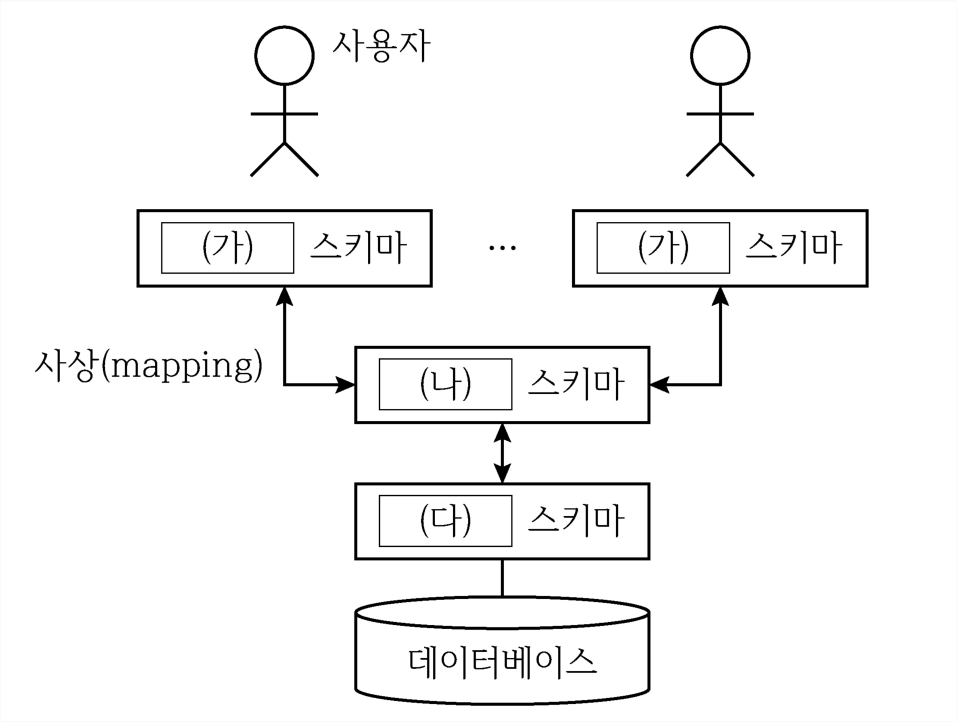
② O: 기본키는 후보키 중에서 대표 식별자로 선택한 키이다. 후보키의 성질을 가지므로 서로 다른 튜플을 구별하는 유일성과, 식별에 불필요한 속성이 없는 최소성을 모두 만족한다. 또한 기본키의 모든 속성은 널 값을 가질 수 없다.

③ O: 후보키가 여러 개일 때 그중 하나를 기본키로 선택하면, **선택되지 않은 나머지 후보키**가 대체키이다. 대체키도 후보키이므로 유일성과 최소성을 만족한다.

④ O: 후보키는 유일성과 최소성을 모두 만족하는 슈퍼키이다. 최소성이란 해당 속성 집합에서 **속성을 하나라도 빼면 유일성을 잃는다**는 뜻이며, 모든 후보키 중 속성 개수가 가장 작아야 한다는 뜻은 아니다.

NoteBase

문 4. 다음 3단계 데이터베이스 구조 스키마의 (가)~(다)에 들어갈 용어를 바르게 연결한 것은?



① (가): 개념, (나): 내부, (다): 외부

② (가): 개념, (나): 외부, (다): 내부

③ (가): 외부, (나): 개념, (다): 내부

④ (가): 외부, (나): 내부, (다): 개념

정답 3번

간단한 해설

사용자마다 따로 제시된 단계는 **외부 스키마**, 이들이 공통으로 연결되는 가운데 단계는 **개념 스키마**이다.
사용자별로 필요한 데이터의 모습과 데이터베이스 전체의 논리 구조를 구분한 것이다.

NoteBase

빈칸별 해설

(가) 외부 스키마: 개별 사용자나 응용 프로그램이 필요로 하는 데이터의 논리적 모습을 정의한다. 그림에서 사용자 아래에 (가) 스키마가 여러 개 놓인 것은 **사용자별로 서로 다른 관점**을 가질 수 있기 때문이다. 각 외부 스키마는 외부·개념 사상을 통해 전체 논리 구조와 연결된다.

(나) 개념 스키마: 데이터베이스 전체의 데이터 구조와 관계, 제약조건 등을 논리적으로 정의한다. 그림에서 여러 (가) 스키마가 하나의 (나) 스키마에 연결되므로, (나)는 **여러 사용자 관점을 통합한 전체 논리 구조**에 해당한다. 특정 사용자가 보는 일부 데이터나 실제 디스크 저장 방식을 뜻하지 않는다.

(다) 내부 스키마: 데이터가 저장 장치에 어떻게 저장되는지를 정의한다. 레코드의 저장 형식이나 접근 경로 등 **물리적 저장 구조**를 다루므로, 그림에서 데이터베이스와 직접 연결된 가장 아래 단계에 해당한다. 개념·내부 사상은 전체 논리 구조와 이러한 저장 구조를 연결한다.

따라서 (가)는 외부, (나)는 개념, (다)는 내부이며, 이 조합은 ③과 일치한다.

NoteBase

문 5. 관계 대수 연산자 중 두 릴레이션의 차수가 같지 않거나 대응 속성 간 도메인이 일치하지 않아도 수행이 가능한 연산자는?

① 합집합(union)

② 차집합(difference)

③ 교집합(intersection)

④ 카티션 프로덕트(Cartesian product)

정답 4번

간단한 해설

카티션 프로덕트는 두 릴레이션의 튜플을 하나씩 짝지어 이어 붙이는 연산이므로 **차수나 대응 속성의 도메인이 같을 필요가 없다**. 튜플의 구조를 맞추어 집합 연산을 하는 것이 아니기 때문이다.

공통 해설

릴레이션의 차수는 속성의 개수이고, 도메인은 각 속성이 가질 수 있는 값의 범위이다. 두 릴레이션의 차수가 같고 대응 속성의 도메인이 일치하는 조건을 **합병 가능 조건**이라고 한다. 합집합·차집합·교집합은 같은 구조의 튜플을 합치거나 포함 여부를 비교해 결과를 구하므로 모두 이 조건을 충족해야 한다. 따라서 차수가 다르거나 대응 속성의 도메인이 일치하지 않으면 이 세 연산은 수행할 수 없다.

카티션 프로덕트는 한 릴레이션의 각 튜플을 다른 릴레이션의 모든 튜플과 짝지어, 두 튜플의 속성값을 이어 붙인다. 대응 위치의 값을 비교하는 대신 두 튜플을 결합하므로 **합병 가능 조건이 필요하지 않다**. 따라서 문제에서 제시한 것처럼 차수가 다르거나 대응 속성의 도메인이 일치하지 않아도 수행할 수 있다.

NoteBase

문 6. 다음 '사원'과 '부서' 릴레이션에 대해 SQL문을 실행할 경우 위반하는 제약조건은? (단, 밑줄 친 속성은 기본키, '사원' 릴레이션의 부서코드는 '부서' 릴레이션의 부서코드에 대한 외래키이다)

<릴레이션>

사원

사번	이름	부서코드	계약액
16031	이순신	190	820000
18039	유관순	240	950000
20024	안중근	190	935000
21098	이봉창	160	900000

부서

부서코드	부서명
190	판매부
160	생산부
230	개발부
240	회계부

<SQL문>

```
INSERT INTO 사원(사번, 이름, 부서코드, 계약액)
VALUES (25002, '장보고', 180, 850000);
```

① 기본키 제약조건

② 개체 무결성 제약조건

③ 참조 무결성 제약조건

④ 도메인 무결성 제약조건

정답 3번

간단한 해설

삽입하려는 사원의 부서코드 180이 '부서' 릴레이션에 존재하지 않는다. 외래키의 NULL이 아닌 값은 참조 대상 키에 존재해야 하므로 **참조 무결성 제약조건**을 위반한다.

NoteBase

핵심 풀이

'사원'의 부서코드는 '부서'의 부서코드를 참조하는 외래키이다. 참조 무결성 제약조건에 따라 외래키에 NULL이 아닌 값을 넣으려면 **참조 대상 릴레이션에 같은 키 값이 존재해야 한다**. 이는 사원이 실제로 존재하는 부서에 소속되도록 하는 조건이다.

SQL문은 사번 25002, 이름 '장보고', 부서코드 180, 계약액 850000인 행을 삽입하려 한다. 그런데 '부서' 릴레이션의 부서코드는 190, 160, 230, 240이며, **180에 대응하는 부서가 없다**. 따라서 이 삽입은 참조 무결성 제약조건을 위반한다.

기본키 제약조건은 기본키 값의 중복과 NULL을 허용하지 않으며, 개체 무결성은 기본키가 NULL일 수 없다는 조건이다. 새 사번 25002는 기존 사번과 중복되지 않고 NULL도 아니다. 도메인 무결성은 속성에 허용된 자료형·값의 범위를 지키는 조건이다. 이 사례에서 확인되는 위반은 부서코드의 숫자 형식이 아니라 **외래키 값에 대응하는 참조 대상의 부재**에서 발생한다.

NoteBase

문 7. 시스템 카탈로그에 대한 설명으로 옳지 않은 것은?

① 질의 최적화를 위해 사용할 수 있다.

② 사용자 계정과 접근 권한에 대한 정보도 포함한다.

③ 데이터베이스에 저장된 데이터에 대한 메타데이터를 저장하는 테이블로 일반 사용자가 직접 수정해야 한다.

④ 데이터베이스 객체(릴레이션, 뷰 등)의 구조에 대한 데이터를 저장하며 DBMS가 이를 스스로 생성하고 유지한다.

정답 3번

간단한 해설

시스템 카탈로그의 메타데이터는 **DBMS가 자동으로 생성하고 갱신**한다. 일반 사용자가 직접 수정해야 하는 테이블이 아니며, 테이블 생성이나 권한 부여 등의 명령을 실행하면 DBMS가 관련 정보를 반영한다.

선지별 해설

① O: 시스템 카탈로그에는 테이블과 인덱스의 정보, 데이터 분포 등의 통계 정보가 저장된다. 질의 최적화기는 이를 이용해 실행 방법별 비용을 추정하고 **효율적인 실행 계획을 선택**한다.

② O: 시스템 카탈로그에는 데이터베이스 객체의 구조뿐 아니라 **사용자 계정과 접근 권한**에 관한 정보도 포함된다. DBMS는 이 정보를 이용해 사용자가 특정 테이블을 조회하거나 수정할 권한이 있는지 확인한다.

③ X: 시스템 카탈로그가 데이터에 대한 메타데이터를 저장한다는 부분은 맞지만, '일반 사용자가 직접 수정해야 한다'는 부분이 틀리다. 카탈로그의 내용은 **DBMS가 관리하고 갱신**한다. 사용자가 테이블 생성·변경이나 권한 부여 명령을 실행하면 DBMS가 해당 메타데이터를 반영하며, 일반 사용자가 카탈로그 테이블을 직접 수정하는 것은 원칙적으로 허용되지 않는다.

④ O: 시스템 카탈로그는 릴레이션의 이름, 속성의 이름과 자료형, 뷰의 정의 등 **데이터베이스 객체의 구조**를 저장한다. 예를 들어 사용자가 새 테이블을 생성하면 DBMS가 그 테이블과 속성에 대한 정보를 카탈로그에 기록하여 유지한다.

NoteBase

문 8. 다음 <조건>을 고려한 개체-관계 다이어그램(ERD, E-R Diagram)의 관계형 데이터베이스 스키마는?

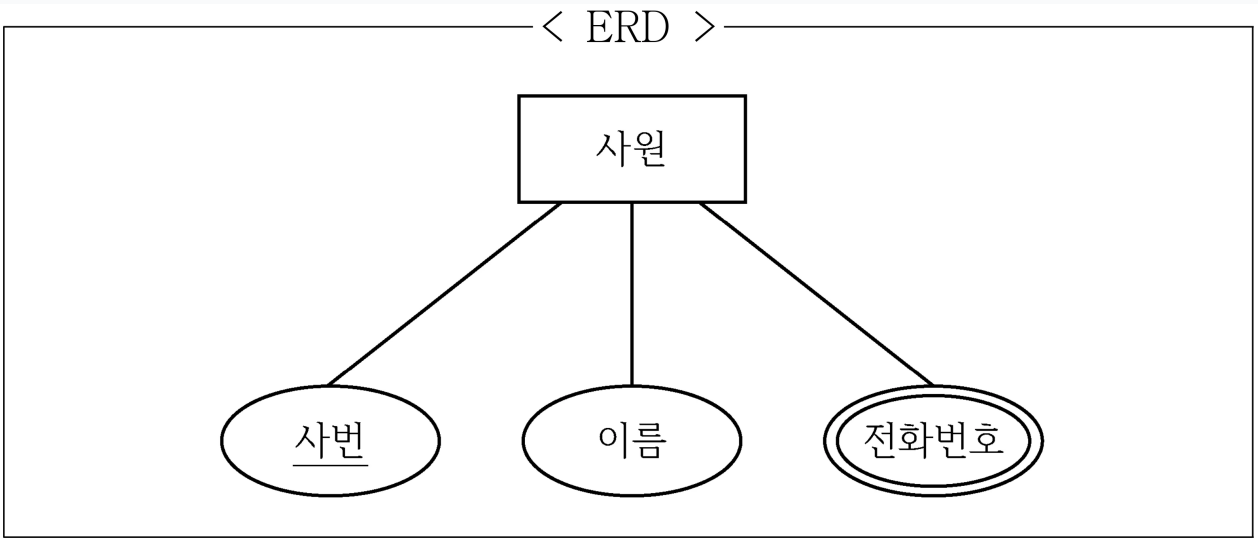
<조건>

◯ 밑줄 친 속성은 기본키이다.

◯ 같은 이름의 사원들이 존재할 수 있다.

◯ 한 명의 사원이 여러 개의 전화번호를 가질 수 있다.

◯ 전화번호 속성은 하나의 전화번호만 저장할 수 있는 크기이다.



① 사원(<u>사번</u>, 이름), 연락처(<u>사번</u>, 전화번호)

② 사원(<u>사번</u>, 이름), 연락처(<u>사번</u>, <u>전화번호</u>)

③ 사원(<u>사번</u>, 전화번호), 연락처(<u>이름</u>, 전화번호)

④ 사원(<u>사번</u>, 전화번호), 연락처(<u>이름</u>, <u>전화번호</u>)

정답 2번

간단한 해설

전화번호는 한 사원이 여러 값을 가질 수 있는 **다중값 속성**이므로 별도의 연락처 릴레이션으로 분리한다.
연락처에서는 같은 사번으로 여러 전화번호를 저장할 수 있도록 **사번과 전화번호를 함께 기본키**로 사용한다.

NoteBase

공통 해설

ERD에서 이중 타원으로 표시된 전화번호는 다중값 속성이다. 이를 관계형 스키마로 변환할 때는 사원 릴레이션에 사번과 이름을 남기고, 다중값 속성을 위한 연락처(사번, 전화번호)를 별도로 만든다. 연락처의 사번은 해당 전화번호의 소유 사원을 가리키며, 사원 릴레이션의 기본키인 사번을 참조하는 외래키가 된다.

연락처의 기본키는 **소유 사원의 기본키와 다중값 속성의 조합**인 (사번, 전화번호)이다. 예를 들어 사번이 100인 사원이 서로 다른 두 전화번호를 가지면 연락처에는 사번 100이 같은 두 행을 저장한다. 사번만 기본키로 삼으면 기본키의 중복을 허용하지 않으므로 두 행을 저장할 수 없다. 사번과 전화번호를 함께 사용하면 같은 사원의 서로 다른 전화번호를 각각 구별할 수 있다.

따라서 사원(사번, 이름)의 기본키는 사번이고, 연락처(사번, 전화번호)의 기본키는 두 속성 전체이다. 같은 이름의 사원이 있을 수 있으므로 연락처에서 이름으로 소유 사원을 식별할 수 없다. 또한 사원(사번, 전화번호)처럼 전화번호를 사원 릴레이션에 두면, 사번이 기본키이고 전화번호 칸에는 하나의 번호만 들어가므로 한 사원의 여러 번호를 표현할 수 없다.

NoteBase

문 9. 함수적 종속(functional dependency)을 표현한 X → Y에 대한 설명으로 옳지 않은 것은? (단, X와 Y는 동일 릴레이션에 존재하는 속성들의 부분 집합이다)

① X는 결정자이고 Y는 종속자라 한다.

② X 값에 대응하는 Y 값이 단 하나만 존재한다.

③ Y는 X에 함수적으로 종속되어 있음을 의미한다.

④ Y가 X에 부분 함수 종속일 때, X는 단일 속성이다.

정답 4번

간단한 해설

부분 함수 종속은 X 전체가 아니라 **X의 일부 속성만으로도 Y가 결정**되는 경우이다. 따라서 이 경우 X는 단일 속성이 아니라 둘 이상의 속성으로 이루어진 복합 속성 집합이다.

선지별 해설

① O: X → Y에서 X는 Y의 값을 결정하는 **결정자**이고, Y는 X의 값에 따라 값이 정해지는 종속자이다. X와 Y는 각각 하나의 속성일 수도 있고 여러 속성의 집합일 수도 있다.

② O: 함수적 종속은 릴레이션의 어떤 두 튜플에서든 **X 값이 같으면 Y 값도 같아야 한다**는 제약이다. 따라서 같은 X 값에 서로 다른 Y 값이 대응할 수 없다. 다만 서로 다른 X 값에 같은 Y 값이 대응하는 것은 가능하다.

③ O: X → Y는 'X가 Y를 함수적으로 결정한다' 또는 'Y가 X에 함수적으로 종속된다'는 뜻이다. 화살표는 **결정자에서 종속자**로 향한다.

④ X: 부분 함수 종속은 X → Y가 성립하면서 **X의 진부분집합만으로도 Y를 결정**할 수 있는 경우이다. 이때 X는 여러 속성으로 이루어진 복합 속성 집합이다. 예를 들어 {학번, 과목코드} → 학생이름이 성립하고 학번만으로도 학생이름이 결정된다면, 학생이름은 {학번, 과목코드}에 부분 함수 종속된다. 반대로 X에서 어떤 속성이라도 빼면 Y를 결정할 수 없어 X 전체가 필요한 경우는 완전 함수 종속이다.

NoteBase

문 10. 논리적 명칭을 통해 데이터에 접근을 보장하는 분산 데이터베이스의 투명성은?

① 병행 투명성

② 위치 투명성

③ 장애 투명성

④ 중복 투명성

정답 2번

간단한 해설

논리적 명칭만으로 데이터에 접근하고 실제 저장 위치를 알 필요가 없게 하는 성질이 **위치 투명성**이다.

공통 해설

위치 투명성은 분산 데이터베이스에서 사용자가 **데이터의 물리적 저장 위치를 몰라도** 논리적 명칭으로 데이터에 접근할 수 있는 성질이다. 논리적 명칭과 실제 저장 위치의 대응을 시스템이 처리하므로, 사용자는 접근할 데이터의 이름을 지정하면 된다. 문항의 '논리적 명칭을 통해 데이터에 접근'한다는 설명은 이러한 위치 투명성에 해당한다.

예를 들어 '고객' 테이블이 어느 지역의 서버에 저장되어 있는지 몰라도, 사용자는 '고객'이라는 테이블 이름으로 데이터를 조회할 수 있다.

NoteBase

문 11. 다음 '주문' 릴레이션에서 아래 SQL문을 실행한 결과는?

<릴레이션>

주문

고객번호	도서번호	판매가격	주문일자
1	1	7000	2026-01-08
2	6	15000	2026-02-22
1	2	13000	2026-02-16
2	5	8000	2026-01-29
3	2	13000	2026-03-21
4	4	35000	2026-02-25
1	3	22000	2026-02-26
4	3	22000	2026-03-14

<SQL문>

```
SELECT 고객번호, SUM(판매가격) AS 총금액
FROM 주문
WHERE 주문일자 >= '2026-02-01'
GROUP BY 고객번호
HAVING COUNT(도서번호) >= 2
ORDER BY 고객번호;
```

①

고객번호	총금액
1	35000
4	57000

NoteBase

②

고객번호	총금액
1	42000
4	57000

③

고객번호	총금액
1	35000
2	15000
4	57000

④

고객번호	총금액
1	42000
2	23000
4	57000

정답 1번

간단한 해설

집계에는 **2026년 2월 1일을 포함하여 그 이후의 주문만** 사용한다. 이 주문이 2건 이상인 고객은 1과 4이며, 판매가격 합계는 각각 35000과 57000이다.

NoteBase

계산 과정

행 선택: WHERE의 날짜 조건을 먼저 적용한다. 주문일자가 2026-02-01보다 이른 고객 1의 1월 8일 주문(7000)과 고객 2의 1월 29일 주문(8000)은 제외된다. 따라서 **이 두 행은 합계와 건수 계산 모두에서 빠진다.**

집계 기준: GROUP BY 고객번호는 날짜 조건을 통과한 행을 고객별로 묶는다. SUM(판매가격)은 각 그룹의 판매가격 합계이고, COUNT(도서번호)는 도서번호가 NULL이 아닌 행의 수이다. 주어진 자료에는 도서번호가 모두 있으므로 이 값은 남은 주문 건수와 같다.

고객 1: 2월 16일과 2월 26일 주문이 남는다. 주문 건수는 2건이고, 총금액은 13000 + 22000 = 35000이다.

고객 2: 1월 주문은 제외되고 2월 22일 주문만 남는다. 주문 건수는 1건이고, 총금액은 15000이다.

고객 3: 3월 21일 주문 1건이 남으며, 총금액은 13000이다.

고객 4: 2월 25일과 3월 14일 주문이 모두 남는다. 주문 건수는 2건이고, 총금액은 35000 + 22000 = 57000이다.

그룹 선택: HAVING COUNT(도서번호) >= 2는 **WHERE를 통과한 행이 2건 이상인 고객 그룹**을 남긴다. 이에 따라 고객 1과 4가 남고, 각각 1건인 고객 2와 3은 제외된다. 고객 2의 조건 충족 여부는 원래 주문 2건이 아니라 날짜 조건 적용 후 남은 1건을 기준으로 판단한다.

출력과 정렬: SELECT는 고객번호와 판매가격 합계를 출력하며, AS 총금액은 합계 열에 붙이는 이름이다. ORDER BY 고객번호에는 DESC가 없으므로 고객번호의 오름차순으로 정렬한다.

최종 결과는 고객번호 1·총금액 35000, 고객번호 4·총금액 57000의 두 행이며, 이 순서로 제시된 ①과 일치한다.

NoteBase

문 12. DBMS의 오류 복구를 위해 주기적으로 덤프를 수행할 경우, 덤프의 수행 단계를 순서대로 바르게 나열한 것은?

(가) 변경된 버퍼 블록을 디스크에 기록한다.
(나) 로그 레코드를 안정 저장 장치에 기록한다.
(다) 데이터베이스 내용을 안정 저장 장치에 기록한다.
(라) 메인 메모리의 모든 로그 레코드를 안정 저장 장치에 기록한다.

① (가) → (나) → (다) → (라)
② (가) → (라) → (나) → (다)
③ (라) → (가) → (다) → (나)
④ (라) → (다) → (나) → (가)

정답 3번

간단한 해설

변경된 데이터를 디스크에 쓰기 전에 **복구에 필요한 로그를 먼저 보존**해야 한다. 버퍼의 변경 내용을 디스크에 반영한 뒤 데이터베이스를 복사하고, **복사가 끝난 후** 덤프 완료를 나타내는 로그 레코드를 기록한다.

올바른 순서

(라) → (가) → (다) → (나)

(라) 메인 메모리의 모든 로그 레코드를 안정 저장 장치에 기록한다. 로그는 데이터 변경 내역을 담는 기록이다. 변경된 데이터 블록을 디스크에 쓰기 전에 관련 로그를 보존하는 **로그 우선 기록 원칙(WAL)**을 지켜야 장애가 나도 복구할 근거가 남는다.

(가) 변경된 버퍼 블록을 디스크에 기록한다. 버퍼는 디스크의 데이터 블록을 메인 메모리에 임시로 보관하는 공간이다. 버퍼에만 남은 변경 사항을 디스크에 반영해야 다음 단계에서 **변경 내용이 반영된 데이터베이스**를 복사할 수 있다.

(다) 데이터베이스 내용을 안정 저장 장치에 기록한다. 디스크에 반영된 데이터베이스 전체를 복사하여 디스크 손상 때 사용할 복구용 사본을 만든다. 이 기본 덤프 절차에서는 **덤프 중 트랜잭션이 실행되지 않는 상태**를 유지하여 복사 도중 데이터가 바뀌지 않게 한다.

(나) 로그 레코드를 안정 저장 장치에 기록한다. 이 단계에서는 **덤프가 완료되었음을 나타내는 로그 레코드**를 남긴다. 데이터베이스 복사가 끝난 뒤 기록해야 아직 완성되지 않은 사본을 완료된 덤프로 취급하지 않는다.

NoteBase

문 13. 다음 '제품', '주문', '고객' 릴레이션에서 아래 SQL문을 실행한 결과는?

<릴레이션>

제품

제품번호	제품명
1	노트북
2	태블릿
3	휴대전화

주문

주문번호	제품번호	고객번호
101	1	1
102	2	2
103	2	1

고객

고객번호	고객명
1	이홍위
2	한명회
3	엄흥도

<SQL문>

```
SELECT p.제품번호, p.제품명, o.주문번호, c.고객명
FROM 제품 p
        LEFT JOIN 주문 o ON p.제품번호 = o.제품번호
        LEFT JOIN 고객 c ON o.고객번호 = c.고객번호
ORDER BY p.제품번호, o.주문번호;
```

①

제품번호	제품명	주문번호	고객명
1	노트북	101	이홍위
2	태블릿	102	한명회
2	태블릿	103	이홍위
3	휴대전화	NULL	NULL

②

제품번호	제품명	주문번호	고객명
NULL	NULL	NULL	엄흥도
1	노트북	101	이홍위
2	태블릿	102	한명회
2	태블릿	103	이홍위

③

제품번호	제품명	주문번호	고객명
1	노트북	101	이홍위
2	태블릿	102	한명회
2	태블릿	103	이홍위
3	NULL	NULL	엄흥도

④

제품번호	제품명	주문번호	고객명
NULL	휴대전화	NULL	NULL
1	노트북	101	이홍위
2	태블릿	102	한명회
2	태블릿	103	이홍위

NoteBase

정답 1번

간단한 해설

LEFT JOIN은 **왼쪽 행을 모두 유지**하므로 주문이 없는 휴대전화도 결과에 남고, 주문번호와 고객명은 NULL이 된다. 태블릿은 연결되는 주문이 2건이므로 두 행으로 나타나며, 전체 결과는 4행이다.

계산 과정

제품과 주문 결합: p는 제품, o는 주문, c는 고객의 별칭이다. 첫 LEFT JOIN은 제품번호가 같은 제품과 주문을 연결하되, **연결되는 주문이 없어도 제품 행을 유지**한다. 제품번호 1의 노트북에는 주문 101이 연결되고, 제품번호 2의 태블릿에는 주문 102와 103이 각각 연결된다. 제품번호 3의 휴대전화에는 주문이 없으므로 주문 쪽 열을 NULL로 채운 한 행이 생긴다. 첫 조인 결과는 1 + 2 + 1 = 4행이다.

고객 결합: 두 번째 LEFT JOIN은 앞서 만든 각 행의 o.고객번호와 c.고객번호를 비교한다. 주문 101과 103의 고객번호는 1이므로 고객명은 이홍위이고, 주문 102의 고객번호는 2이므로 고객명은 한명회이다. 휴대전화 행의 o.고객번호는 NULL이어서 연결되는 고객이 없지만, **앞선 조인 결과의 행은 유지**되므로 고객명이 NULL로 채워진다. 오른쪽 고객 릴레이션에만 있는 엄흥도를 위한 행은 추가되지 않는다.

출력 열과 정렬: SELECT는 제품번호와 제품명을 제품 p에서, 주문번호를 주문 o에서, 고객명을 고객 c에서 가져온다. 따라서 주문이 없는 행도 p.제품번호와 p.제품명은 각각 3과 휴대전화로 유지된다. ORDER BY에 따라 제품번호 오름차순으로 정렬하고, 제품번호가 같으면 주문번호 오름차순으로 정렬하므로 태블릿의 주문 102가 103보다 먼저 나온다.

실행 결과: 열 순서는 (제품번호, 제품명, 주문번호, 고객명)이며, 행은 다음과 같다.
(1, 노트북, 101, 이홍위)
(2, 태블릿, 102, 한명회)
(2, 태블릿, 103, 이홍위)
(3, 휴대전화, NULL, NULL)

이 네 행의 값과 순서가 ①의 결과와 일치한다.

NoteBase

문 14. 2단계 로킹 규약(two-phase locking protocol)에 대한 설명으로 옳지 않은 것은?

① 확장 단계에서는 로크 연산만 수행해야 한다.

② 축소 단계에서는 언로크 연산만 수행해야 한다.

③ 트랜잭션은 확장 단계 후 축소 단계를 수행해야 한다.

④ 모든 직렬가능 스케줄은 2단계 로킹 규약을 준수한다.

정답 4번

간단한 해설

2단계 로킹 규약을 지키면 **충돌 직렬가능성**이 보장되지만, 직렬가능한 모든 스케줄이 이 규약을 지키는 것은 아니다. 즉, 이 규약은 직렬가능성을 보장하는 **충분조건이며 필요조건은 아니다**.

선지별 해설

① O: 확장 단계에서는 새로운 로크를 획득할 수 있지만 **보유한 로크를 해제할 수 없다**. 따라서 로크 관리 연산을 기준으로 로크 획득만 허용되는 단계이다. 이때 데이터의 읽기·쓰기까지 금지된다는 뜻은 아니다.

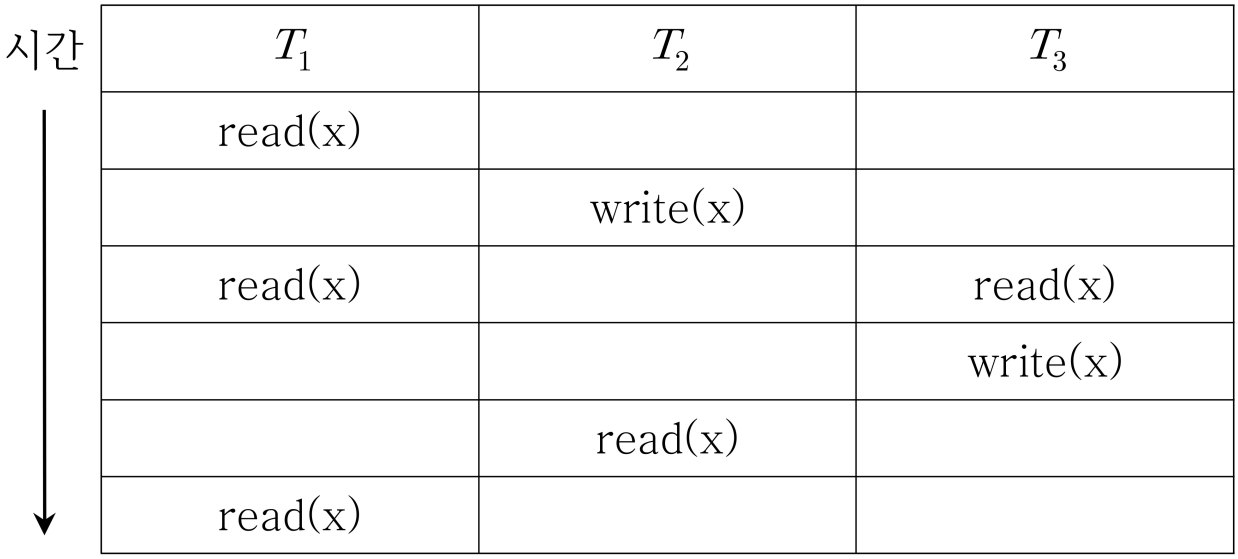
② O: 축소 단계에서는 보유한 로크를 해제할 수 있지만 **새로운 로크를 획득할 수 없다**. 한 번 해제한 데이터 항목에 다시 로크를 거는 것도 허용되지 않는다.

③ O: 트랜잭션은 먼저 확장 단계를 거친 뒤 축소 단계로 진행한다. **최초의 언로크 연산**이 축소 단계의 시작이며, 그 이후에는 확장 단계로 돌아갈 수 없다. 예를 들어 A의 로크를 획득했다가 해제한 뒤 B의 로크를 새로 획득하면 이 규약을 위반한다.

④ X: 직렬가능한 스케줄은 여러 트랜잭션을 하나씩 차례로 실행한 어떤 직렬 스케줄과 동등한 스케줄이다. 모든 트랜잭션이 2단계 로킹 규약을 따르면 충돌 직렬가능성이 보장되지만, **규약을 따르지 않아도 직렬가능한 스케줄은 존재한다**. 따라서 '모든 직렬가능 스케줄은 규약을 준수한다'가 아니라 '규약을 준수하여 생성된 스케줄은 충돌 직렬가능하다'로 고쳐야 한다.

NoteBase

문 15. 다음 트랜잭션 실행에 따라 발생하는 문제를 해결하기 위해 SQL을 이용하여 설정할 수 있는 고립도 중 가장 낮은 등급은?



① SERIALIZABLE

② READ COMMITTED

③ REPEATABLE READ

④ READ UNCOMMITTED

정답 3번

간단한 해설

같은 트랜잭션이 x를 반복해서 읽는 사이에 다른 트랜잭션이 x를 변경하면 읽는 값이 달라지는 **반복 불가능 읽기**가 발생할 수 있다. 이를 방지하는 최소 고립도는 **REPEATABLE READ**이다.

NoteBase

선지별 해설

그림의 시간은 위에서 아래로 흐른다. T_1은 x를 세 번 읽으며, 그 사이에 T_2와 T_3의 write(x)가 끼어 있다. 따라서 비교 기준은 같은 트랜잭션이 이미 읽은 항목을 다시 읽을 때 다른 트랜잭션의 변경 때문에 값이 달라지는 반복 불가능 읽기를 막는지이다. SQL 표준의 고립도는 낮은 순서대로 READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE이다.

① SERIALIZABLE: 여러 트랜잭션을 하나씩 순서대로 실행한 것과 같은 결과를 보장하는 가장 높은 등급이다. **반복 불가능 읽기를 방지**하지만, 이 문제를 해결하는 데 필요한 최소 등급보다 높다.

② READ COMMITTED: 다른 트랜잭션이 커밋하지 않은 값을 읽는 오손 읽기를 막는다. 그러나 첫 번째 읽기 이후 다른 트랜잭션이 변경한 값을 커밋하면 다음 읽기에서 그 값을 볼 수 있어, **반복 불가능 읽기는 허용**한다. 따라서 그림의 문제를 해결하기에 부족하다.

③ REPEATABLE READ: 같은 트랜잭션이 이미 읽은 항목을 다시 읽을 때 **다른 트랜잭션의 변경으로 값이 달라지는 것을 방지**한다. 그림에서 x를 반복해서 읽는 동안 읽기 결과가 바뀌는 문제를 막으며, 바로 아래 등급인 READ COMMITTED는 이를 보장하지 않는다.

④ READ UNCOMMITTED: SQL 표준상 다른 트랜잭션이 **커밋하지 않은 값도 읽을 수 있는** 가장 낮은 등급이다. 오손 읽기와 반복 불가능 읽기를 모두 허용하므로 그림의 문제를 해결하지 못한다.

반복 불가능 읽기를 막는 등급은 REPEATABLE READ와 SERIALIZABLE이며, 그중 더 낮은 등급은 ③의 REPEATABLE READ이다.

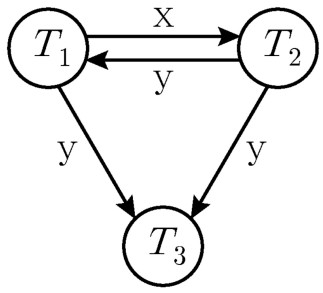
NoteBase

문 16. 트랜잭션 T_1, T_2, T_3에서 데이터 x, y에 대한 읽기(read)와 쓰기(write) 작업이 다음 순서로 스케줄링 될 경우, 스케줄 S의 선행 그래프로 옳은 것은?

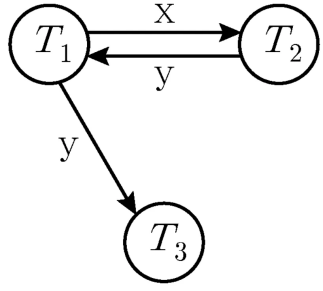
S:

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

①

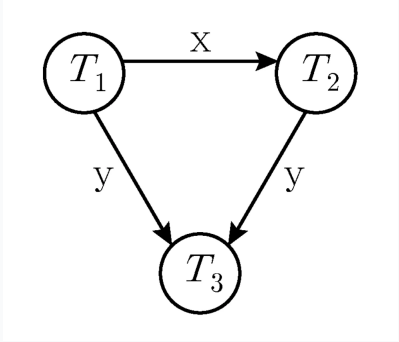


②

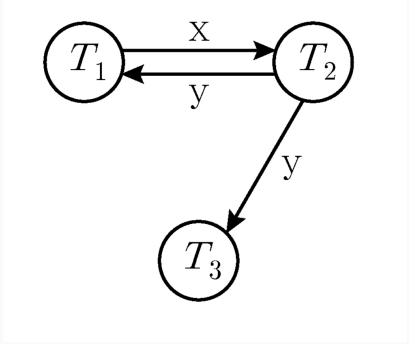


NoteBase

③



④



정답 1번

간단한 해설

선행 그래프의 간선은 **같은 데이터에 대한 충돌 연산의 실행 순서**로 정한다. x에서는 T_1이 T_2보다 앞서지만, y에서는 T_2가 T_1보다 앞서므로 두 트랜잭션 사이에 양방향 간선이 생기며, 두 트랜잭션에서 T_3으로 향하는 간선도 필요하다.

NoteBase

계산 과정

간선 생성 기준: 각 트랜잭션을 정점으로 놓고, 서로 다른 트랜잭션의 연산이 **같은 데이터에 접근하며 적어도 하나가 쓰기 연산**이면 충돌한다고 본다. 먼저 실행된 연산의 트랜잭션에서 나중 연산의 트랜잭션으로 간선을 그린다. 읽기끼리는 충돌하지 않으며, 같은 방향의 간선이 여러 번 도출되어도 하나로 표시한다.

x의 충돌: 표를 위에서 아래로 읽으면 T_1의 read(x), write(x) 다음에 T_2의 read(x)가 실행된다. T_1의 write(x)와 T_2의 read(x)가 충돌하므로 x에 의한 간선은 $T_1 \rightarrow T_2$이다. 두 read(x)는 모두 읽기 연산이므로 그 읽기 연산 쌍 자체는 간선을 만들지 않는다.

y에서 T_2와 T_1의 관계: T_2의 write(y)가 T_1의 read(y)보다 먼저 실행되므로 $T_2 \rightarrow T_1$을 추가한다. x에서 도출한 간선과 방향이 반대이므로 **두 트랜잭션 사이의 양방향 간선을 모두 유지**해야 한다.

y에서 T_2와 T_3의 관계: T_2의 write(y)가 T_3의 read(y)와 write(y)보다 먼저 실행된다. 또한 T_2의 read(y)도 T_3의 write(y)보다 먼저 실행된다. 이 충돌들은 모두 같은 방향이므로 $T_2 \rightarrow T_3$ 하나로 표시한다.

y에서 T_1과 T_3의 관계: T_3의 read(y)는 T_1의 read(y)보다 먼저지만, **읽기끼리는 충돌하지 않는다**. 이후 T_3의 write(y)가 실행되므로 T_1의 read(y)와 충돌하여 $T_1 \rightarrow T_3$이 생긴다. 따라서 이 두 트랜잭션 사이에는 반대 방향의 간선이 생기지 않는다.

도출된 간선은 x에 의한 $T_1 \rightarrow T_2$, y에 의한 $T_2 \rightarrow T_1$, $T_2 \rightarrow T_3$, $T_1 \rightarrow T_3$의 네 개이다. 이 방향과 데이터 표기를 모두 나타낸 그래프가 ①이다.

NoteBase

문 17. 다음 SQL문을 통해 ‘department’와 ‘student’ 테이블을 생성하고 투플을 추가한 후 (가), (나) 문을 실행한 결과에 대한 설명으로 옳은 것은?

```
CREATE TABLE department (
        dcode CHAR(5) NOT NULL,
        dename VARCHAR(20),
        PRIMARY KEY(dcode));

CREATE TABLE student (
        sno INT NOT NULL,
        dept CHAR(5),
        PRIMARY KEY(sno),
        FOREIGN KEY(dept) REFERENCES department(dcode)
                                ON DELETE CASCADE);

INSERT INTO department VALUES ('D1', 'Computer');
INSERT INTO department VALUES ('D2', 'AI');
INSERT INTO department VALUES ('D3', 'Electronics');
INSERT INTO student VALUES (100, 'D1');
INSERT INTO student VALUES (200, 'D1');
INSERT INTO student VALUES (300, 'D2');
INSERT INTO student VALUES (400, 'D3');

DELETE FROM student WHERE dept = 'D1'; -- (가)
DELETE FROM department WHERE dcode = 'D3'; -- (나)
```

① (가) 문장은 정상적으로 실행되나 (나) 문장은 오류가 발생하고 실행되지 않는다.

② (나) 문장은 정상적으로 실행되나 (가) 문장은 오류가 발생하고 실행되지 않는다.

③ (가), (나) 문장은 실행되며 그 결과 ‘department’와 ‘student’ 테이블의 투플 수는 각각 2개, 1개이다.

④ (가), (나) 문장은 실행되며 그 결과 ‘department’와 ‘student’ 테이블의 투플 수는 각각 2개, 2개이다.

정답 3번

간단한 해설

(가)는 D1 소속 학생 2명만 삭제하고, (나)는 D3 부서를 삭제하면서 **ON DELETE CASCADE**에 따라 그 소속 학생 1명도 함께 삭제한다. 두 문장은 모두 실행되며, 남은 투플 수는 **department 2개, student 1개**이다.

NoteBase

계산 과정

초기 상태: department에는 D1·D2·D3의 투플 3개가 있고, student에는 (100, D1), (200, D1), (300, D2), (400, D3)의 투플 4개가 있다. student의 dept는 department의 dcode를 참조하는 외래 키이다. ON DELETE CASCADE는 **참조되는 부서 투플이 삭제되면 이를 참조하는 학생 투플도 삭제**하도록 지정한 것이다.

(가) 실행: student에서 dept가 'D1'인 투플, 즉 학번 100과 200의 투플 2개를 삭제한다. 참조하는 쪽인 **학생을 삭제해도 부서는 삭제되지 않는다**. 따라서 department는 3개 그대로이고, student는 4 − 2 = 2개가 되어 (300, D2), (400, D3)가 남는다. 외래 키 제약을 위반하지 않으므로 정상 실행된다.

(나) 실행: department에서 dcode가 'D3'인 투플 1개를 삭제한다. 이 부서를 참조하는 (400, D3)도 **연쇄 삭제**되므로 참조 무결성을 유지하며 정상 실행된다. department는 3 − 1 = 2개, student는 (가) 실행 후의 2개에서 1개가 더 삭제되어 2 − 1 = 1개가 된다.

최종적으로 department에는 D1·D2가, student에는 (300, D2)만 남는다. 두 문장이 모두 실행되고 투플 수가 각각 2개, 1개가 되므로 ③의 설명과 일치한다.

NoteBase

문 18. 데이터베이스 보안에 대한 설명으로 옳지 않은 것은?

① 역할에 부여된 권한에 변화가 생기면 그 역할을 허가받은 모든 사용자에게 영향을 미친다.

② 강제보안기법은 데이터와 사용자의 보안 등급에 따라 해당 보안 정책을 적용하는 다단계 보안 기법이다.

③ 임의보안기법은 사용자들에게 객체에 대한 권한을 기반으로 접근할 수 있도록 권한을 부여하는 기법이다.

④ 릴레이션의 생성자라 하더라도 자신이 생성한 릴레이션을 관리하기 위한 권한을 자동으로 부여받지 않는다.

정답 4번

간단한 해설

릴레이션을 생성한 사용자는 그 릴레이션의 **소유자로서 관리에 필요한 권한을 자동으로 부여받는다**. 따라서 ④의 '자동으로 부여받지 않는다'는 설명은 잘못되었다.

선지별 해설

① O: 역할 기반 접근 제어는 사용자마다 권한을 개별적으로 지정하는 대신, 역할에 권한을 부여하고 사용자에게 그 역할을 부여한다. 사용자가 **역할을 통해 권한을 얻으므로** 역할의 권한이 바뀌면 그 역할을 부여받은 모든 사용자에게 영향을 미친다. 예를 들어 담당자 역할에 조회 권한을 추가하면 그 역할을 가진 사용자들은 해당 권한을 이용할 수 있다.

② O: 강제보안기법은 데이터의 보안 등급과 사용자의 보안 인가 등급을 비교하여 시스템이 정한 정책에 따라 접근을 통제한다. 여러 보안 등급을 구분하여 적용하므로 다단계 보안 기법에 해당한다. 객체 소유자의 임의적인 허가보다 **시스템의 보안 등급 정책**이 우선한다.

③ O: 임의보안기법은 사용자에게 부여된 객체별 접근 권한을 바탕으로 접근을 허용하거나 제한한다. 예를 들어 특정 릴레이션에 조회 권한만 부여받은 사용자는 그 권한으로 데이터를 조회할 수 있지만 수정할 수는 없다. 객체 소유자가 다른 사용자에게 권한을 부여할 수 있다는 점에서 **소유자의 재량에 따른 권한 부여**가 특징이다.

④ X: 릴레이션을 생성한 사용자는 그 릴레이션의 소유자가 되며, 이를 **관리하는 데 필요한 권한을 자동으로 부여받는다**. 생성자가 자신의 릴레이션을 관리하려고 다른 사용자에게서 별도로 권한을 받아야 하는 것은 아니다. 따라서 '자동으로 부여받지 않는다'를 '자동으로 부여받는다'로 고쳐야 한다.

NoteBase

문 19. 다음 <상황>에서 교착상태 예방 방법별 동작에 대한 설명으로 옳지 않은 것은?

<상황>
트랜잭션 T에 대한 타임스탬프 TS(T)는 트랜잭션의 시작 시각을 표현하며, TS(T_j) < TS(T_i)는 T_j가 T_i보다 먼저 작업을 시작했다는 의미이다. 트랜잭션 T_j가 항목 x에 대해 배타 로크 (exclusive lock)를 획득하였으며, 트랜잭션 T_i가 항목 x에 대해 배타 로크를 획득하고자 한다.

① TS(T_i) < TS(T_j)일 때, wound-wait 기법에서 T_j는 철회하고 동일 타임스탬프를 사용하여 재시작한다.
② TS(T_j) < TS(T_i)일 때, wound-wait 기법에서 T_j는 대기한다.
③ TS(T_j) < TS(T_i)일 때, wait-die 기법에서 T_i는 철회하고 동일 타임스탬프를 사용하여 재시작한다.
④ TS(T_i) < TS(T_j)일 때, wait-die 기법에서 T_i는 대기한다.

정답 2번

간단한 해설

②의 조건에서는 로크를 보유한 T_j가 먼저 시작한 트랜잭션이다. wound-wait에서는 나중에 시작한 **요청자 T_i가 대기**하고, T_j는 로크를 유지하며 계속 수행한다.

NoteBase

선지별 해설

타임스탬프가 작을수록 먼저 시작한 트랜잭션이다. 이 상황에서 T_j는 항목 x의 배타 로크를 보유한 트랜잭션이고, T_i는 같은 항목에 배타 로크를 요청하는 트랜잭션이다. 두 트랜잭션이 같은 항목의 배타 로크를 동시에 보유할 수 없으므로, 각 예방 기법의 규칙에 따라 대기하거나 철회한다.

① O: $TS(T_i) < TS(T_j)$이면 요청자 T_i가 먼저 시작했다. wound-wait에서는 먼저 시작한 요청자가 나중에 시작한 **로크 보유자를 철회**시켜 필요한 로크를 확보한다. 따라서 T_j가 철회되며, 이후 기존과 동일한 타임스탬프를 사용하여 재시작한다.

② X: $TS(T_j) < TS(T_i)$이면 로크 보유자 T_j가 먼저 시작했다. wound-wait에서는 나중에 시작한 요청자가 먼저 시작한 보유자의 로크가 해제되기를 기다린다. 따라서 **대기하는 트랜잭션은** T_i이며, T_j는 로크를 유지한 채 계속 수행한다.

③ O: $TS(T_j) < TS(T_i)$이면 요청자 T_i가 나중에 시작했다. wait-die에서는 나중에 시작한 요청자가 먼저 시작한 보유자의 로크를 요청할 때 **기다리지 않고 철회**한다. 따라서 T_i가 철회되며, 이후 기존과 동일한 타임스탬프를 사용하여 재시작한다.

④ O: $TS(T_i) < TS(T_j)$이면 요청자 T_i가 먼저 시작했다. wait-die에서는 **먼저 시작한 요청자의 대기를 허용**하므로, T_i는 T_j가 항목 x의 로크를 해제할 때까지 기다린다. 이 경우 로크 보유자인 T_j를 강제로 철회시키지 않는다.

NoteBase

문 20. 비트맵 인덱스에 대한 설명으로 옳지 않은 것은?

① 한 가지 특정 속성값에 대해 하나의 비트맵 인덱스가 만들어진다.

② 릴레이션의 카디널리티가 n일 경우, 비트맵 인덱스의 크기는 $\log_2 n$비트이다.

③ 탐색 키의 중복 비율이 높은 속성을 대상으로 수행되는 질의에 효율적이다.

④ 다중 키를 가진 질의를 쉽게 처리하기 위해 고안된 특수한 형태의 인덱스이다.

정답 2번

간단한 해설

비트맵은 **튜플마다 1비트**를 대응시켜 특정 속성값의 존재 여부를 표시한다. 따라서 튜플이 n개이면 속성값 하나에 대한 비트맵의 길이는 압축 전 기준으로 n**비트**이며, $\log_2 n$비트가 아니다.

선지별 해설

① O: 서로 다른 속성값마다 하나의 비트맵을 만들고, 각 튜플이 그 값을 가지면 해당 위치를 1, 그렇지 않으면 0으로 표시한다. 예를 들어 세 튜플의 속성값이 차례로 A, B, A라면 A의 비트맵은 '101', B의 비트맵은 '010'이다. 선지의 비트맵 인덱스는 이처럼 **특정 속성값에 대응하는 비트맵**을 뜻한다.

② X: 릴레이션의 카디널리티는 **튜플의 수**이다. 튜플이 n개이면 각 튜플에 1비트씩 대응하므로, 압축과 부가 정보를 제외한 속성값 하나의 비트맵 길이는 n비트이다. 따라서 '$\log_2 n$비트'를 'n**비트**'로 고쳐야 한다. 인덱스 전체에는 서로 다른 속성값에 대응하는 여러 비트맵이 포함된다.

③ O: 탐색 키의 중복 비율이 높다는 것은 전체 튜플 수에 비해 **서로 다른 키 값의 수가 적다**는 뜻이다. 이때 적은 수의 비트맵으로 각 값에 해당하는 많은 튜플을 나타낼 수 있고, 비트 연산으로 조건에 맞는 튜플을 찾아 질의를 효율적으로 처리할 수 있다.

④ O: 여러 속성의 조건을 함께 사용하는 질의는 각 조건에 대응하는 비트맵을 **AND·OR 비트 연산으로 결합**하여 처리할 수 있다. 두 조건을 모두 만족하는 튜플은 AND 연산으로, 둘 중 하나 이상을 만족하는 튜플은 OR 연산으로 찾는다. 따라서 여러 탐색 키의 조건을 조합하는 질의에 유용하다.

NoteBase