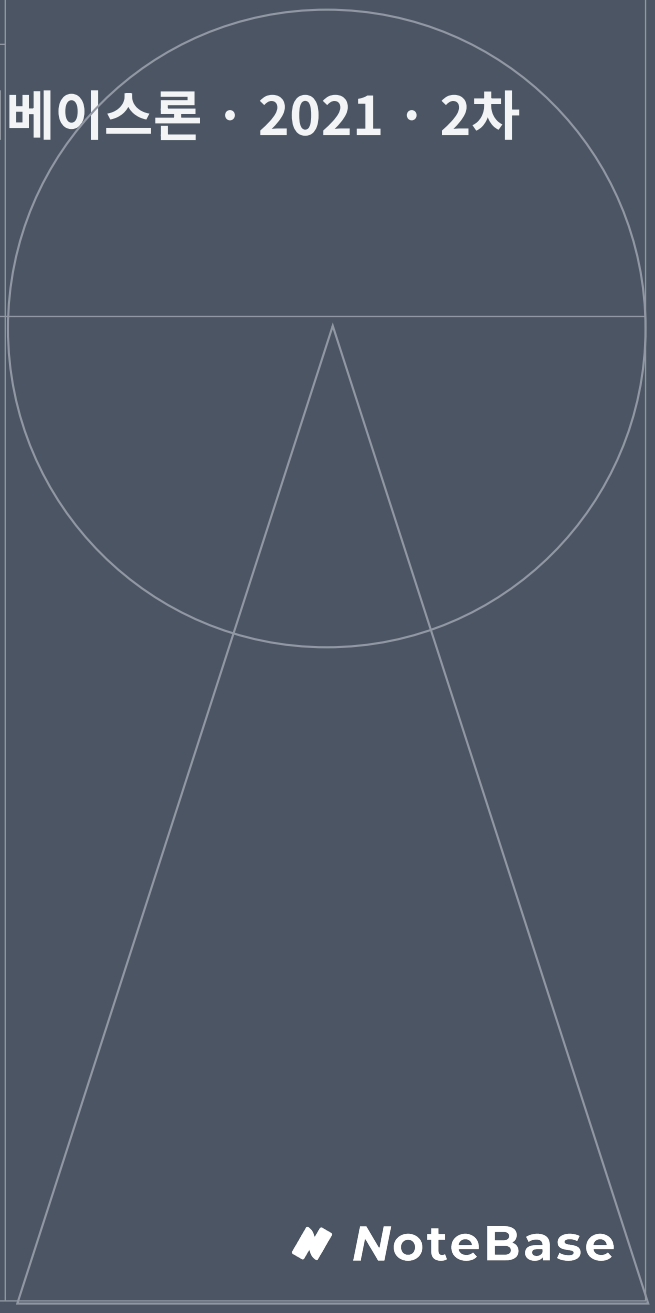
기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2021 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2021 · 2차

공무원/국가직/7급 · 데이터베이스론 · 2021 · 2차

문 1. SQL 언어에 대한 설명으로 옳지 않은 것은?

① SQL은 관계 데이터베이스 시스템의 표준 언어이다.

② SQL은 포괄적인 데이터베이스 언어로서 데이터 정의, 질의, 갱신을 위한 문들을 가지고 있다.

③ 트랜잭션의 시작, 철회, 완료 등을 표현하기 위해 SQL에서는 COMMIT, ROLLBACK 등을 사용한다.

④ 데이터 조작어는 데이터베이스에 데이터를 검색하여 추가하고 삭제하는 데 사용하며 SELECT, REVOKE가 이에 해당된다.

정답 4번

출제 포인트

SQL 문장을 기능별로 데이터 정의어(DDL), 데이터 조작어(DML), 데이터 제어어(DCL)로 분류할 수 있는지, 그리고 각 범주에 어떤 명령어가 속하는지를 정확히 구분할 수 있는지를 묻는다. 특히 SELECT, INSERT, UPDATE, DELETE는 DML이고 GRANT, REVOKE는 DCL이라는 점을 혼동하지 않아야 한다.

핵심 개념

데이터 정의어(DDL)는 테이블·뷰·인덱스 등 스키마 객체를 생성·변경·삭제하는 CREATE, ALTER, DROP 등을 포함한다. 데이터 조작어(DML)는 저장된 데이터를 검색, 삽입, 수정, 삭제하는 SELECT, INSERT, UPDATE, DELETE로 구성된다. 데이터 제어어(DCL)는 사용자 권한을 부여하거나 회수하는 GRANT, REVOKE와 트랜잭션을 완료·취소하는 COMMIT, ROLLBACK을 포함한다. REVOKE는 데이터 검색·추가·삭제와 무관하게 접근 권한을 회수하는 명령이므로 DML이 아니라 DCL로 분류된다.

NoteBase

선지별 해설

① SQL은 관계 데이터베이스 시스템의 표준 언어이다 → O. SQL은 관계형 데이터베이스를 다루기 위한 표준 질의어로 국제 표준(ISO/ANSI)으로 제정되어 대부분의 관계형 DBMS가 지원한다.
② SQL은 포괄적인 데이터베이스 언어로서 데이터 정의, 질의, 갱신을 위한 문들을 가지고 있다 → O. SQL은 DDL(정의), DML(질의·갱신), DCL(제어)을 모두 포함하는 종합적인 언어로 설계되었다.
③ 트랜잭션의 시작, 철회, 완료 등을 표현하기 위해 SQL에서는 COMMIT, ROLLBACK 등을 사용한다 → O. 트랜잭션을 정상적으로 완료할 때는 COMMIT을, 변경 사항을 취소하고 이전 상태로 되돌릴 때는 ROLLBACK을 사용하며 이는 SQL의 트랜잭션 제어 기능에 해당한다.
④ 데이터 조작어는 데이터베이스에 데이터를 검색하여 추가하고 삭제하는 데 사용하며 SELECT, REVOKE가 이에 해당된다 → X. 데이터 조작어(DML)에는 SELECT, INSERT, UPDATE, DELETE가 해당하며, REVOKE는 사용자에게 부여된 접근 권한을 회수하는 데이터 제어어(DCL)이다. 틀린 표현 → 올바른 표현: "SELECT, REVOKE" → "SELECT, INSERT, UPDATE, DELETE"(REVOKE는 DCL).

오답 포인트

REVOKE라는 명령어 이름만 보고 SQL 명령이니 DML이라고 넓게 묶어버리는 실수를 주의해야 한다. REVOKE는 데이터 자체를 다루는 명령이 아니라 사용자 권한을 다루는 명령이므로 DML이 아니라 DCL에 속한다는 점을 명확히 구분해야 한다.

다시 나오면 이렇게 판단

선지에 SQL 명령어가 나열되면 먼저 각 명령어를 DDL(CREATE, ALTER, DROP)인지, DML(SELECT, INSERT, UPDATE, DELETE)인지, DCL(GRANT, REVOKE, COMMIT, ROLLBACK)인지 분류표대로 나눠본 뒤 선지의 분류 주장과 대조한다. 한 선지 안에 서로 다른 범주의 명령어가 섞여 나열되어 있으면 그 선지를 우선 의심한다.

암기 포인트

DDL(정의)=CREATE·ALTER·DROP, DML(조작)=SELECT·INSERT·UPDATE·DELETE, DCL(제어)=GRANT·REVOKE·COMMIT·ROLLBACK으로 세 갈래를 짝지어 외워 둔다.

NoteBase

문 2. 다음은 4개의 트랜잭션 T1, T2, T3, T4에 대하여 시스템 고장(crash) 시점에 특정 스케줄에 대응하는 로그를 나타낸 것이다. 이 시스템은 로그를 이용한 회복기법으로 검사점(checkpoint)을 가진 즉시갱신규약(immediate update protocol)을 사용한다고 가정한다. 시스템 고장으로부터 회복(recovery)하는 과정에서 undo와 redo 연산들을 수행하게 된다. 회복 후 데이터 항목 A, B, C, D 값을 바르게 연결한 것은? (단, 로그 레코드 구조는 <트랜잭션, 데이터 항목, 현재 값, 변경 값>이다)

로그 번호	로그 레코드
1	<T1 start>
2	<T1, D, 20, 25>
3	<T1 commit>
4	<checkpoint {}>
5	<T2 start>
6	<T2, B, 12, 18>
7	<T4 start>
8	<T4, D, 25, 15>
9	<T3 start>
10	<T3, C, 30, 40>
11	<T4, A, 30, 20>
12	<T4 commit>
13	<T2, D, 15, 25>
	시스템 고장

① A: 30, B: 12, C: 30, D: 20

② A: 20, B: 18, C: 40, D: 25

③ A: 30, B: 18, C: 40, D: 15

④ A: 20, B: 12, C: 30, D: 15

정답 4번

출제 포인트

검사점(checkpoint)을 가진 즉시갱신규약(immediate update) 환경에서 로그 기반 회복을 수행할 때, 어떤 트랜잭션을 redo하고 어떤 트랜잭션을 undo하는지 구분하고 그 결과로 각 데이터 항목의 최종 값을 계산하는 문제이다. 즉시갱신규약에서는 트랜잭션 실행 중에도 변경 값이 데이터베이스에 바로 반영될 수 있으므로, 커밋된 트랜잭션은 redo로 변경을 확정하고 커밋되지 않은 트랜잭션은 undo로 변경 이전 값으로 되돌려야 한다.

NoteBase

조건 정리

로그 번호 4에서 <checkpoint {}>가 기록되었고 활성 트랜잭션 목록이 비어 있다. 이는 검사점 시점에는 어떤 트랜잭션도 실행 중이 아니었음을 뜻한다(T1은 로그 번호 3에서 이미 commit). 검사점 이후 T2(로그 번호 5 start), T4(로그 번호 7 start), T3(로그 번호 9 start)가 차례로 시작된다. 이 중 T4는 로그 번호 12에서 commit되었고, T2와 T3는 시스템 고장 시점까지 commit도 abort도 되지 않은 채 종료되지 않은 상태로 남아 있다.

사용할 공식

회복 절차는 다음과 같다. 검사점 이후 로그를 검사하여, 고장 시점까지 commit 기록이 있는 트랜잭션은 redo-list에, commit 기록 없이 활성 상태로 남아 있는 트랜잭션은 undo-list에 넣는다. redo는 로그를 앞에서 뒤로(오래된 것부터) 진행하면서 각 레코드의 변경 값(after-image)을 데이터 항목에 다시 적용한다. undo는 로그를 뒤에서 앞으로(최근 것부터) 진행하면서 각 레코드의 현재 값(before-image)으로 데이터 항목을 되돌린다.

계산 과정

T1은 검사점 이전에 이미 commit되었으므로 별도의 redo/undo 대상이 아니다.
T4는 commit되었으므로 redo-list에 포함: 로그 번호 8 <T4, D, 25, 15>를 redo하여 D=15로 확정하고, 로그 번호 11 <T4, A, 30, 20>을 redo하여 A=20으로 확정한다.
T2는 시스템 고장까지 commit되지 않았으므로 undo-list에 포함: 로그를 뒤에서부터 적용하여 먼저 로그 번호 13 <T2, D, 15, 25>를 undo하여 D를 현재 값인 15로 되돌리고, 다음으로 로그 번호 6 <T2, B, 12, 18>을 undo하여 B를 현재 값인 12로 되돌린다.
T3도 commit되지 않았으므로 undo-list에 포함: 로그 번호 10 <T3, C, 30, 40>을 undo하여 C를 현재 값인 30으로 되돌린다.
정리하면 A=20(T4 redo), B=12(T2 undo), C=30(T3 undo), D=15(T4 redo의 결과와 T2 undo의 결과가 모두 15로 일치)이다.

선지 판단

① A=30, B=12, C=30, D=20 — X: A는 T4가 commit되어 redo로 20이 확정되어야 하는데 30(변경 전 값)으로 남아 있어 틀렸고, D도 15가 아닌 20으로 되어 있어 틀렸다.
② A=20, B=18, C=40, D=25 — X: B는 T2가 commit되지 않아 undo로 12가 되어야 하는데 18(변경 후 값)로 남아 있고, C도 T3가 commit되지 않아 undo로 30이 되어야 하는데 40으로 남아 있으며, D도 25가 아니라 15여야 한다.
③ A=30, B=18, C=40, D=15 — X: A가 T4의 redo 결과인 20이 아니라 30으로 되어 있고, B와 C도 undo가 반영되지 않은 상태(18, 40)로 남아 있어 틀렸다.
④ A=20, B=12, C=30, D=15 — O: T4의 redo로 A=20, D=15가 확정되고, T2와 T3의 undo로 B=12, C=30으로 되돌려진 결과와 정확히 일치한다.

NoteBase

오답 포인트

가장 흔한 실수는 시스템 고장 시점에 commit 기록이 없는 트랜잭션(T2, T3)까지 값을 '변경 후 값'으로 그대로 두는 것이다. 즉시갱신규약에서는 커밋 전에도 값이 디스크에 반영될 수 있으므로, commit되지 않은 트랜잭션의 변경은 반드시 undo로 되돌려야 한다. 또한 검사점의 활성 트랜잭션 목록이 비어 있다는 것을 검사점 이전 트랜잭션에 대한 정보가 없다는 뜻으로 오해하면 안 되며, 이는 오히려 검사점 시점에 실행 중인 트랜잭션이 하나도 없었다는 뜻으로 T1의 처리가 이미 끝났음을 보여 준다.

다시 나오면 이렇게 판단

로그를 보면 먼저 각 트랜잭션이 고장 시점까지 commit 기록을 가지는지부터 표시한다. commit이 있는 트랜잭션은 redo-list, 없는 트랜잭션은 undo-list로 분류한 뒤, redo는 로그 순서대로 변경 후 값을 적용하고 undo는 로그 역순으로 변경 전 값을 적용한다. 같은 데이터 항목에 redo와 undo가 모두 관련되면 마지막에 적용되는 값(보통 서로 일치하도록 설계됨)을 확인한다.

암기 포인트

즉시갱신규약 + 검사점 회복 = commit된 트랜잭션은 redo(앞→뒤, 변경 후 값), commit되지 않은 트랜잭션은 undo(뒤→앞, 변경 전 값). 검사점의 활성 트랜잭션 목록이 비어 있으면 그 시점에는 실행 중인 트랜잭션이 없었다는 뜻이다.

NoteBase

문 3. 뷰(view)에 대한 설명으로 옳지 않은 것은?

① 생성된 뷰는 새로운 독립적인 테이블로 물리 저장소에 저장된다.

② 데이터베이스 질의 시 사용자 편의성 및 재사용성, 보안성을 높이기 위한 기술이다.

③ 뷰의 질의연산은 제한을 받지 않지만 갱신연산은 제한을 받는다.

④ 하나의 테이블로 여러 개의 상이한 뷰를 정의하여 사용자의 요구에 따라 활용할 수 있다.

정답 1번

출제 포인트

뷰(view)의 정의와 특성을 정확히 알고 있는지 묻는 문제이다. 뷰가 실제로 어떻게 저장되고, 어떤 목적으로 쓰이며, 질의·갱신 연산에서 어떤 제약을 받는지를 구분할 수 있어야 한다.

핵심 개념

뷰는 하나 이상의 기본 테이블로부터 유도되는 가상 테이블이다. 실제 데이터를 별도로 저장하지 않고, 질의가 실행될 때마다 정의된 SQL(뷰 정의문)을 통해 기본 테이블에서 데이터를 가져와 보여준다. 뷰를 사용하는 목적은 복잡한 질의를 단순화하여 사용자 편의성을 높이고, 동일한 테이블에서 필요한 부분만 여러 관점으로 재사용할 수 있게 하며, 특정 열이나 행만 노출하여 보안성을 높이는 데 있다. 조회(SELECT) 연산은 기본 테이블에 대한 질의로 그대로 변환되므로 비교적 자유롭게 수행되지만, 삽입·삭제·수정과 같은 갱신 연산은 여러 테이블을 조인한 뷰, 그룹 함수나 DISTINCT를 포함한 뷰 등에서는 기본 테이블의 기본 키가 포함되지 않거나 애매성이 발생하여 제한되거나 금지된다.

선지별 해설

① 생성된 뷰는 새로운 독립적인 테이블로 물리 저장소에 저장된다. → 틀린 표현. 뷰는 물리적으로 별도 저장 공간에 데이터를 저장하지 않는 가상 테이블이며, 저장되는 것은 데이터 자체가 아니라 뷰를 정의하는 SQL문(뷰 정의)뿐이다. 조회할 때마다 기본 테이블에서 데이터를 가져오므로 '새로운 독립적인 테이블로 물리 저장소에 저장된다'는 설명은 옳지 않다.

② 데이터베이스 질의 시 사용자 편의성 및 재사용성, 보안성을 높이기 위한 기술이다. → 옳은 설명이다. 뷰는 복잡한 질의를 단순한 이름으로 재사용하게 해 편의성과 재사용성을 높이고, 필요한 열·행만 노출해 보안성도 높인다.

③ 뷰의 질의연산은 제한을 받지 않지만 갱신연산은 제한을 받는다. → 옳은 설명이다. 뷰에 대한 SELECT는 기본 테이블 질의로 그대로 변환되어 자유롭게 수행되지만, 삽입·삭제·수정 등 갱신 연산은 조인 뷰, 그룹 함수·집계 함수를 포함한 뷰 등에서 기본 키 결정이 모호해져 제한을 받는다.

④ 하나의 테이블로 여러 개의 상이한 뷰를 정의하여 사용자의 요구에 따라 활용할 수 있다. → 옳은 설명이다. 하나의 기본 테이블에서 필요한 열이나 행 조건을 달리하여 여러 개의 뷰를 만들고, 사용자마다 필요한 뷰를 선택해 활용할 수 있다.

NoteBase

오답 포인트

뷰를 '저장된 테이블'로 오해하면 안 된다. 저장되는 것은 데이터가 아니라 뷰를 정의하는 질의문이며, 실제 데이터는 여전히 기본 테이블에만 존재한다. 이 문항처럼 뷰의 물리적 저장 여부를 묻는 선지가 나오면 '가상 테이블'이라는 키워드로 바로 오답을 가려낼 수 있다.

다시 나오면 이렇게 판단

선지에 '물리적으로 저장', '별도의 테이블로 저장', '실제 데이터를 저장' 같은 표현이 뷰에 대해 등장하면 우선 의심한다. 뷰는 정의(SQL)만 저장되고 데이터는 저장되지 않는다는 원칙을 기준으로 삼아 판단한다.

암기 포인트

뷰=가상 테이블, 저장되는 것은 데이터가 아니라 정의(SQL)이다. 질의(SELECT)는 자유,
갱신(삽입·삭제·수정)은 제한적이다. 목적은 편의성·재사용성·보안성이다.

NoteBase

문 4. 관계모델에 대한 설명으로 옳지 않은 것은?

① 관계모델에서 행은 투플(tuple), 열은 애트리뷰트(attribute)로 부른다.

② 투플 내의 각 값은 더 이상 나누어지지 않는 원자값이어야 한다.

③ 투플 내 어떤 애트리뷰트의 값을 알 수 없거나 값이 지정되지 않을 때는 NULL이라는 특수한 값을 사용한다.

④ 릴레이션은 투플들의 집합이기 때문에 릴레이션에서 투플이 순서대로만 나타나야 한다.

정답 4번

출제 포인트

관계모델(relational model)의 기본 구성요소와 성질을 정확히 아는지 묻는 문제다. 릴레이션, 투플, 애트리뷰트, 도메인, NULL 등 관계모델의 핵심 개념과 릴레이션이 집합이라는 성질에서 파생되는 특성(무순서성, 중복 불허)을 구분할 수 있어야 한다.

핵심 개념

릴레이션(relation)은 하나의 테이블에 해당하며 투플(tuple, 행)들의 집합으로 정의된다.
애트리뷰트(attribute, 열)는 릴레이션을 구성하는 각 칼럼의 이름이다. 원자성(atomicity)은 투플 내 각 값이 더 이상 분해되지 않는 단일 값이어야 한다는 관계모델의 기본 제약이다. NULL은 값이 존재하지 않거나 알려지지 않은 경우를 나타내기 위한 특수한 표식 값이다. 릴레이션이 수학적 집합(set)으로 정의되므로, 집합의 성질상 원소(투플) 사이에는 순서 개념이 없고 동일한 투플이 중복되어 존재할 수 없다.

선지별 해설

① O — 관계모델에서 행(row)은 투플, 열(column)은 애트리뷰트로 부르는 것이 맞는 설명이다.
② O — 관계모델은 제1정규형의 전제인 원자값 제약을 따르므로, 투플 내 각 값은 더 이상 나누어지지 않는 원자값이어야 한다는 설명은 맞다.
③ O — 관계모델에서 투플 내 애트리뷰트의 값을 모르거나 값이 존재하지 않을 때 NULL이라는 특수한 값을 사용한다는 설명은 맞다.
④ X — 릴레이션은 투플들의 집합이므로 집합의 정의상 원소 간에 순서가 없다. 따라서 릴레이션에서 투플은 순서에 무관하게 나타날 수 있으며, '순서대로만 나타나야 한다'는 설명은 틀렸다.

오답 포인트

릴레이션이 '집합(set)'으로 정의된다는 사실을 놓치면 순서가 있다고 착각하기 쉽다. 집합은 원소의 순서를 따지지 않고 중복 원소를 허용하지 않는 자료구조이므로, 릴레이션에 저장된 투플들도 특정 순서로 나열되어야 할 필요가 없고 동일한 투플이 두 번 이상 존재할 수도 없다. 이 문제는 이 성질을 반대로 뒤집어 '순서대로만 나타나야 한다'고 서술해 오답을 유도한다.

NoteBase

다시 나오면 이렇게 판단

선지에 '릴레이션은 투플들의 집합'이라는 전제가 나오면 이어지는 서술이 집합의 성질(무순서성, 중복 불허)과 일치하는지부터 확인한다. '순서대로', '순서가 정해져', '먼저 나온 투플이 우선' 같은 표현이 붙으면 집합의 정의에 어긋나므로 옳지 않은 설명일 가능성이 높다.

암기 포인트

행=투플, 열=애트리뷰트. 투플 내 값은 원자값이어야 한다(원자성). 값이 없거나 모르면 NULL. 릴레이션=투플의 집합 → 투플 간 순서 없음, 중복 투플 없음.

NoteBase

문 5. 다음 사원 테이블에 대해 뷰_사원1과 같은 뷰를 생성하는 SQL 질의어를 작성할 때, ㉠, ㉡에 들어갈 내용을 바르게 연결한 것은?

사원

사번	이름	직급	부서
108002	김진수	부장	인사
105123	이수진	대리	연구
128372	박지훈	과장	영업
126721	김지수	부장	영업
132122	홍성대	대리	인사

뷰_사원1

사번	이름
105123	이수진

```
CREATE VIEW      뷰_사원1(사번, 이름)
( ㉠ )SELECT     사번, 이름
      FROM       사원
      WHERE      ( ㉡ );
```

① ㉠: IN, ㉡: 직급 = '대리'

② ㉠: AS, ㉡: 부서 = '연구'

③ ㉠: AS, ㉡: 직급 = '대리'

④ ㉠: IN, ㉡: 부서 = '연구'

정답 2번

출제 포인트

뷰(view) 생성 SQL 문법에서 CREATE VIEW와 SELECT 사이에 들어가는 키워드가 무엇인지, 그리고 WHERE 절 조건이 사원 테이블의 실제 데이터와 맞아떨어져 뷰_사원1의 결과(105123, 이수진 단 한 행)를 만들어내는지를 함께 판단하는 문제이다.

NoteBase

핵심 개념

뷰를 생성하는 표준 SQL 구문은 CREATE VIEW 뷰이름(속성목록) AS SELECT … FROM … [WHERE …]이다. 즉 ㉠에는 AS가 들어가야 하며, IN은 부속 질의나 값 목록 비교에 쓰이는 연산자일 뿐 CREATE VIEW 구문에는 쓰이지 않는다. ㉡은 WHERE 절의 조건으로, 사원 테이블에서 실행했을 때 결과가 정확히 105123(이수진) 한 행만 나와야 한다. 직급='대리' 조건으로 사원 테이블을 걸러보면 105123(이수진)과 132122(홍성대) 두 행이 모두 대리이므로 두 행이 나와 뷰_사원1과 일치하지 않는다. 반면 부서='연구' 조건으로 걸러보면 사원 테이블에서 연구 부서는 105123(이수진) 한 명뿐이므로 정확히 뷰_사원1과 같은 한 행이 나온다.

선지별 해설

① IN, 직급='대리' → X. IN은 CREATE VIEW 구문의 키워드가 아니며, 직급='대리' 조건은 이수진과 홍성대 두 행을 반환해 뷰_사원1(1행)과 맞지 않는다.
② AS, 부서='연구' → O. AS가 CREATE VIEW의 올바른 키워드이고, 부서='연구' 조건은 사원 테이블에서 이수진 한 행만 반환해 뷰_사원1과 정확히 일치한다.
③ AS, 직급='대리' → X. AS는 맞지만 직급='대리' 조건은 이수진과 홍성대 두 행이 반환되어 뷰_사원1(1행)과 맞지 않는다.
④ IN, 부서='연구' → X. 부서='연구' 조건 자체는 결과가 맞지만, IN은 CREATE VIEW 구문의 키워드로 쓰이지 않는다.

오답 포인트

①과 ④는 CREATE VIEW … AS SELECT라는 기본 문법을 IN으로 잘못 알고 있을 때 고르기 쉽다. ③은 문법(AS)은 맞게 골랐지만 WHERE 조건을 표에서 직접 대입해 결과 행 수를 확인하지 않고 직급='대리'를 그럴듯해 보인다는 이유만으로 고른 경우이다. WHERE 조건은 반드시 원본 테이블에 대입해 실제로 몇 행이 나오는지 확인해야 한다.

다시 나오면 이렇게 판단

뷰 생성 문법 문제는 먼저 CREATE VIEW 뷰이름(속성목록) AS SELECT 구조를 떠올려 키워드 자리를 채우고, 그다음 WHERE 조건 후보는 반드시 원본 테이블 데이터에 하나씩 대입해 결과 행이 목표 뷰와 정확히 일치하는지 대조한다. 조건에 부합하는 값이 여러 행이면 오답이다.

암기 포인트

뷰 생성 구문: CREATE VIEW 뷰이름(속성목록) AS SELECT … FROM … [WHERE 조건]. IN은 뷰 생성 키워드가 아니다.

NoteBase

문 6. 트랜잭션 T1과 T2가 아래와 같이 수행될 때, T1의 ㉠, ㉡의 출력 값은? (단, 팬텀(phantom)을 초래하는 위반들을 허용하지 않는다)

sale

no	name	price
1	a1	200
2	a2	150
3	b1	100

T1	T2
set transaction isolation level repeatable read;	
	set transaction isolation level repeatable read;
start transaction;	
	start transaction;
	insert into sale values(4, 'b2', 50);
select ㉠ sum(price) from sale;	
	commit
update sale set price=price+100 where no=3;	
select ㉡ sum(price) from sale;	
commit	

① ㉠: 450, ㉡: 550

② ㉠: 450, ㉡: 600

③ ㉠: 500, ㉡: 600

④ ㉠: 500, ㉡: 500

정답 1번

출제 포인트

이 문제는 REPEATABLE READ 격리수준에서 두 트랜잭션이 겹쳐 실행될 때 sum(price) 집계 결과가 어떻게 나오는지를 묻는다. 핵심 조건은 두 가지다. 첫째, T2가 커밋하기 전의 데이터는 T1이 읽을 수 없다(더티 리드 금지). 둘째, 발문의 '팬텀을 초래하는 위반들을 허용하지 않는다'는 단서는 T2가 이미 커밋을 한 뒤라도, T1이 같은 조건의 집계 쿼리를 두 번 실행했을 때 그 사이에 새로 삽입된 행(팬텀 행)이 두 번째 실행 결과에 나타나서는 안 된다는 뜻이다. 즉 T1 입장에서는 트랜잭션 시작 시점에 존재하던 행 집합(no=1,2,3)만을 기준으로 두 번의 집계가 항상 같은 대상 행 위에서 이루어져야 한다.

NoteBase

조건 정리

sale 테이블의 초기 값은 no=1(a1,200), no=2(a2,150), no=3(b1,100)이다. 실행 순서는 다음과 같다. ① T1, T2 모두 격리수준을 repeatable read로 설정하고 트랜잭션을 시작한다. ② T2가 sale에 (4,'b2',50) 행을 삽입한다(아직 커밋 전). ③ T1이 sum(price)를 조회한다(㉠). ④ T2가 커밋한다. ⑤ T1이 no=3의 price를 100 증가시킨다. ⑥ T1이 다시 sum(price)를 조회한다(㉡). ⑦ T1이 커밋한다.

사용할 공식

$\text{sum(price)} = \sum_{i} \text{price}_i$, 단 집계 대상 행 집합은 T1 트랜잭션이 처음 관찰한 행 집합으로 고정된다(팬텀 행 제외).

계산 과정

㉠ 계산: T1이 첫 번째 sum(price)를 조회하는 시점(③)에서 T2가 삽입한 (4,'b2',50) 행은 아직 커밋되지 않은 상태다. 더티 리드는 어떤 격리수준에서도 허용되지 않으므로 T1은 이 행을 볼 수 없다. 따라서 ㉠ = 200 + 150 + 100 = 450이다.
㉡ 계산: T1이 두 번째 sum(price)를 조회하는 시점(⑥)에서는 T2가 이미 커밋(④)을 마쳐 (4,'b2',50) 행이 테이블에 정식으로 존재한다. 하지만 발문의 단서에 따라 팬텀을 초래하는 위반은 허용하지 않으므로, T1은 자신이 트랜잭션 시작 후 처음 관찰했던 행 집합(no=1,2,3)에 한정하여 반복 가능한 읽기를 보장받는다. 즉 T2가 나중에 커밋한 삽입 행은 T1의 두 번째 집계에도 나타나지 않는다. 한편 T1 자신이 수행한 update(no=3의 price를 100+100=200으로 변경, ⑤)는 T1 자신의 변경이므로 그대로 반영된다. 따라서 ㉡ = 200(no=1) + 150(no=2) + 200(no=3, 갱신 후) = 550이다.

선지 판단

① ㉠=450, ㉡=550 → O. 계산 과정과 정확히 일치한다.
② ㉠=450, ㉡=600 → X. ㉡을 600으로 보려면 T2가 삽입한 행의 price 50까지 합산해야 하는데(150+100+200+50=500이 아니라 200+150+200+50=600), 이는 팬텀 행을 반영한 값이라 발문의 전제(팬텀 위반 불허)와 어긋난다.
③ ㉠=500, ㉡=600 → X. ㉠을 500으로 보면 T2가 삽입한 행을 더티 리드로 미리 반영(200+150+100+50=500)한 셈이 되어 커밋 전 데이터를 읽지 않는다는 원칙에 위배된다.
④ ㉠=500, ㉡=500 → X. ㉠에서 더티 리드 오류가 있고, ㉡에서도 T1 자신의 update가 반영되지 않은 값(200+150+100=450, 또는 팬텀 행만 더한 다른 값)으로 계산 과정과 맞지 않는다.

NoteBase

오답 포인트

가장 흔한 실수는 두 가지다. 첫째, T2의 insert가 T1의 첫 select보다 먼저 실행됐다고 해서 그 값을 바로 반영해 버리는 것이다. 삽입 순서가 먼저라도 T2가 커밋하기 전까지는 어떤 격리수준에서도 그 값을 읽을 수 없다. 둘째, T2가 커밋을 마친 뒤에는 당연히 새 행이 보일 것이라고 판단하는 것이다. 이 문제는 발문에서 '팬텀을 초래하는 위반을 허용하지 않는다'고 명시했으므로, 커밋 여부와 무관하게 T1의 반복 조회는 처음 관찰한 행 집합 기준으로 일관성을 유지해야 한다. T1 자신이 수행한 update는 팬텀이 아니라 자기 트랜잭션 내의 정상적인 변경이므로 반영되는 점과 혼동하지 않는다.

다시 나오면 이렇게 판단

격리수준 문제가 나오면 먼저 각 문장이 커밋 전 시점인지 커밋 후 시점인지를 표시하고, 더티 리드 금지(어느 격리수준이든 공통)와 반복 가능한 읽기(같은 트랜잭션 내 동일 조건 재조회는 같은 결과) 원칙을 각각 적용한다. 발문에 팬텀 허용 여부가 명시되어 있으면, 다른 트랜잭션이 삽입한 행이 커밋되었더라도 이미 시작된 트랜잭션의 반복 집계에는 반영하지 않는다고 이해하고 계산한다. 자기 트랜잭션이 실행한 갱신은 항상 자신의 후속 조회에 반영됨을 놓치지 않는다.

암기 포인트

더티 리드는 커밋 전 데이터를 절대 읽지 않는다는 규칙, 팬텀 방지는 이미 시작된 트랜잭션의 반복 집계에서 남이 나중에 커밋한 삽입 행을 배제한다는 규칙, 그리고 자기 자신의 변경은 항상 자신에게 즉시 반영된다는 규칙, 이 세 가지를 짝지어 기억해 둔다.

NoteBase

문 7. 테이블 R과 S에 대해 아래의 SQL 질의어를 수행한 결과로 나오는 RSID 값은?

R	
RSID	CName
1001	국어
1002	영어
1003	국어
1003	수학
1004	국어
1004	영어
1004	수학
1005	과학

S
CName
영어
국어

```
SELECT distinct(RSID) FROM R as x
WHERE NOT EXISTS (
        (SELECT p.CName FROM S as p)
        EXCEPT
        (SELECT y.CName FROM  R as y WHERE
        x.RSID = y.RSID));
```

① 1001

② 1002

③ 1003

④ 1004

정답 4번

출제 포인트

중첩 질의에서 EXCEPT(차집합)와 NOT EXISTS를 결합해 '모든 값을 포함하는지'를 판정하는 관계 나눗셈(전칭 조건) 패턴을 이해하는지 묻는다. 안쪽의 (S의 CName 전체) EXCEPT (해당 RSID의 CName 전체) 결과가 빈 집합이면, S의 모든 과목명을 그 RSID가 다 가지고 있다는 뜻이며, 이때 NOT EXISTS가 참이 되어 해당 RSID가 결과에 포함된다.

조건 정리

테이블 R은 (RSID, CName) 튜플의 집합이다. 1001=국어, 1002=영어, 1003=국어·수학, 1004=국어·영어·수학, 1005=과학이다. 테이블 S는 CName만 가지며 영어, 국어 두 값을 갖는다. 질의는 R의 각 RSID(x)에 대해, S의 CName 전체 집합에서 그 RSID가 R에서 갖는 CName 전체 집합을 뺀 차집합이 비어 있는지를 확인한다.

NoteBase

사용할 공식

조건: (S의 CName 집합) (RSID=x인 R의 CName 집합) = 이면 해당 RSID를 결과에 포함한다. 이는 S ⊆ (RSID=x인 R의 CName 집합) 과 동치이며, EXCEPT 결과가 빈 집합일 때 NOT EXISTS가 참이 되는 원리를 이용한다.

계산 과정

S의 CName 집합은 {영어, 국어}로 고정이다. 각 RSID별로 R에서의 CName 집합을 구해 S와 비교한다.
RSID 1001: R의 CName 집합 = {국어}. {영어,국어} - {국어} = {영어} (비어있지 않음) → 조건 불성립, 제외.
RSID 1002: R의 CName 집합 = {영어}. {영어,국어} - {영어} = {국어} (비어있지 않음) → 조건 불성립, 제외.
RSID 1003: R의 CName 집합 = {국어, 수학}. {영어,국어} - {국어,수학} = {영어} (비어있지 않음) → 조건 불성립, 제외.
RSID 1004: R의 CName 집합 = {국어, 영어, 수학}. {영어,국어} - {국어,영어,수학} = {} (빈 집합) → 조건 성립, 결과에 포함.
RSID 1005: R의 CName 집합 = {과학}. {영어,국어} - {과학} = {영어,국어} (비어있지 않음) → 조건 불성립, 제외.
따라서 조건을 만족하는 RSID는 1004 하나뿐이다.

선지 판단

① 1001 → X. R에서 1001은 국어만 가지고 있어 S의 영어를 포함하지 못하므로 차집합이 비지 않아 제외된다.
② 1002 → X. R에서 1002는 영어만 가지고 있어 S의 국어를 포함하지 못하므로 제외된다.
③ 1003 → X. R에서 1003은 국어, 수학을 가지지만 S의 영어가 없어 차집합이 {영어}로 비지 않으므로 제외된다.
④ 1004 → O. R에서 1004는 국어, 영어, 수학을 모두 가지고 있어 S의 {영어, 국어}를 전부 포함하므로 차집합이 빈 집합이 되어 NOT EXISTS가 참이 되고, 결과에 포함된다.

오답 포인트

①②③은 모두 R에서의 CName 집합이 S의 CName 집합 {영어, 국어}를 부분적으로만 포함하고 있어 EXCEPT 결과가 비지 않는다는 점을 놓치면 잘못 고르기 쉽다. NOT EXISTS(EXCEPT 결과)가 참이 되려면 EXCEPT의 결과 집합이 완전히 비어야 한다는 점, 즉 바깥쪽 집합(S)의 원소가 안쪽 집합(해당 RSID의 R)에 하나도 빠짐없이 포함되어야 한다는 점을 혼동하면 안 된다.

다시 나오면 이렇게 판단

EXCEPT와 NOT EXISTS가 결합된 질의를 보면 먼저 EXCEPT의 두 피연산자가 각각 어떤 집합인지(고정 집합인지, 상관 서브쿼리로 매 튜플마다 달라지는 집합인지)를 구분한다. 그 다음 각 후보값(x)마다 안쪽 집합을 구체적으로 나열하고, 바깥쪽 고정 집합에서 그 안쪽 집합을 뺐을 때 남는 원소가 있는지 하나씩 확인하면 실수를 줄일 수 있다. 남는 원소가 하나도 없을 때만 그 x가 결과에 포함된다는 점을 기억한다.

NoteBase

암기 포인트

A EXCEPT B가 빈 집합이라는 것은 A의 모든 원소가 B에 포함된다는 뜻이며, 이를 NOT EXISTS로 감싸면 'B가 A를 전부 포함하는 x만 골라낸다'는 전칭 조건(관계 나눗셈)이 된다.

NoteBase

문 8. 다음과 같은 student 테이블의 스키마가 주어졌을 때, 주어진 요구에 대한 SQL 질의로 옳지 않은 것은?

student(studno, name, grade, score, deptno)
(단, studno는 학번, name은 학생 이름, grade는 학년, score는 성적, deptno는 학생이 속한 학과 번호를 의미하며, studno는 기본키이다)

① student 테이블에서 각 학과별 평균 성적, 최고 성적, 최저 성적 검색

```
SELECT deptno, AVG(score), MAX(score), MIN(score)
FROM student
GROUP BY deptno
```

② student 테이블에서 각 학과의 각 학년별 인원수와 평균 성적 검색

```
SELECT deptno, grade, COUNT(*), AVG(score)
FROM student
GROUP BY deptno, grade
```

③ student 테이블에서 학과 번호가 100번 이상인 학과들의 평균 성적 검색

```
SELECT deptno, AVG(score)
FROM student
WHERE deptno >= 100
GROUP BY deptno
```

④ student 테이블에서 각 학년별로 학생 수가 10명 이상인 학년, 학생 수, 평균 성적 검색

```
SELECT grade, COUNT(*), AVG(score)
FROM student
WHERE COUNT(*) >= 10
GROUP BY grade
```

정답 4번

출제 포인트

GROUP BY와 집계함수(COUNT, AVG, MAX, MIN)를 활용한 SQL 질의를 작성할 때, 그룹별 조건을 걸러내는 위치가 WHERE절인지 HAVING절인지를 구분할 수 있는지를 묻는다. WHERE절은 그룹으로 묶이기 전의 개별 행을 걸러내는 조건이고, HAVING절은 GROUP BY로 묶인 그룹 단위의 집계 결과를 걸러내는 조건이라는 점이 핵심이다.

조건 정리

student(studno, name, grade, score, deptno)이며 studno가 기본키이다. ①은 학과별 성적 통계, ②는 학과·학년별 인원수와 평균 성적, ③은 학과 번호 100번 이상인 학과의 평균 성적, ④는 학년별로 인원이 10명 이상인 학년의 인원수와 평균 성적을 구하는 요구조건이다.

NoteBase

사용할 공식

그룹화 질의의 처리 순서는 FROM → WHERE(개별 행 필터링) → GROUP BY(그룹 형성) → HAVING(그룹 단위 필터링) → SELECT 순이다. 따라서 COUNT(*), AVG(score)처럼 그룹별로 계산되는 집계함수 값에 대한 조건은 WHERE절이 아니라 HAVING절에 와야 한다. WHERE절에는 아직 그룹이 형성되지 않았으므로 집계함수를 조건으로 쓸 수 없다.

계산 과정

④의 질의를 순서대로 보면 FROM student로 전체 학생 행을 가져온 뒤, WHERE COUNT(*) >= 10을 그룹화 이전 단계에서 적용하려고 한다. 이 시점에서는 아직 grade별로 묶이지 않았으므로 COUNT(*)라는 집계값 자체가 존재하지 않는다. 집계함수는 GROUP BY grade로 그룹이 만들어진 이후에만 계산되므로, 그룹별 인원수 조건을 걸려면 GROUP BY grade 다음에 HAVING COUNT(*) >= 10을 써야 한다. 즉 올바른 질의는 SELECT grade, COUNT(*), AVG(score) FROM student GROUP BY grade HAVING COUNT(*) >= 10 이다.

선지 판단

① SELECT deptno, AVG(score), MAX(score), MIN(score) FROM student GROUP BY deptno — O, deptno로 그룹을 묶고 각 그룹의 평균·최고·최저 성적을 집계함수로 구하는 정상적인 질의이다.
② SELECT deptno, grade, COUNT(*), AVG(score) FROM student GROUP BY deptno, grade — O, deptno와 grade 두 컬럼으로 그룹을 세분화하여 학과의 학년별 인원수와 평균 성적을 구하는 정상적인 질의이다.
③ SELECT deptno, AVG(score) FROM student WHERE deptno >= 100 GROUP BY deptno — O, deptno >= 100은 그룹화 전 개별 행에 대한 단순 조건이므로 WHERE절에 쓰는 것이 옳다.
④ SELECT grade, COUNT(*), AVG(score) FROM student WHERE COUNT(*) >= 10 GROUP BY grade — X, COUNT(*)는 그룹별 집계값이므로 WHERE절이 아니라 HAVING절에 와야 한다. 틀린 표현 WHERE COUNT(*) >= 10 → 올바른 표현 GROUP BY grade HAVING COUNT(*) >= 10 (HAVING절을 GROUP BY 뒤에 추가).

오답 포인트

이 문제에서 실수하기 쉬운 지점은 ③과 ④를 같은 유형으로 착각하는 것이다. ③의 deptno >= 100은 원본 테이블의 개별 행이 가진 값(학과 번호)에 대한 조건이라 WHERE절이 맞지만, ④의 COUNT(*) >= 10은 GROUP BY로 묶인 이후에만 계산되는 집계값에 대한 조건이라 WHERE절에 쓸 수 없다. 조건에 등장하는 값이 원본 컬럼 값인지 집계함수 결과인지를 먼저 구분하지 않으면 이 둘을 혼동하게 된다.

NoteBase

다시 나오면 이렇게 판단

SQL 질의에서 WHERE와 HAVING 중 어느 것을 써야 하는지 헷갈리면, 조건절 안의 값이 집계함수(COUNT, SUM, AVG, MAX, MIN)로 계산된 값인지를 먼저 확인한다. 집계함수가 조건에 들어가면 무조건 HAVING이고, 원본 컬럼 값 그대로의 조건이면 WHERE이다.

암기 포인트

WHERE는 그룹화 전 개별 행 필터, HAVING은 그룹화 후 집계값 필터라는 원칙을 공식처럼 기억해 둔다.
집계함수가 조건식에 보이면 그 즉시 HAVING절을 떠올리는 습관을 들여 둔다.

NoteBase

문 9. 정규화(normalization)에 대한 설명 중 옳은 것만을 모두 고르면?

ㄱ. 데이터의 정규화는 중복을 최소화하고 삽입, 삭제, 수정 이상을 최소화하기 위해서 함수적 종속성과 기본키를 기반으로, 주어진 릴레이션 스키마를 분석하는 과정이다.
ㄴ. 릴레이션 스키마 R의 모든 원소들의 도메인(domain)이 나눌 수 있는 단위로 되어있을 때, R이 제1정규형에 속한다.
ㄷ. 제2정규형이 되기 위해서는 릴레이션 R이 제1정규형이고 기본키가 아닌 속성이 기본키에 부분함수 종속이어야 한다.
ㄹ. 제3정규형이 되기 위해서는 릴레이션 R이 제2정규형이고, 릴레이션 R의 함수 종속 관계에서 이행적 함수 종속을 제거해야 한다.
ㅁ. 제4정규형이 되기 위해서는 릴레이션 R이 제3정규형이고, 함수 종속 관계에서 모든 결정자가 후보키이면 된다.

① ㄱ, ㄹ
② ㄴ, ㄷ
③ ㄱ, ㄹ, ㅁ
④ ㄷ, ㄹ, ㅁ

정답 1번

출제 포인트

제1정규형부터 제4정규형(및 보이스-코드 정규형)까지 각 단계의 정의를 정확히 구분하는 문제이다. 특히 완전함수종속과 부분함수종속의 차이, 이행적 함수종속의 제거 시점, 그리고 보이스-코드 정규형과 제4정규형의 조건을 서로 뒤섞어 놓았을 때 이를 가려낼 수 있는지를 묻는다.

핵심 개념

제1정규형(1NF)은 릴레이션의 모든 속성값이 더 이상 나눌 수 없는 원자값(atomic value)으로만 되어 있을 때 성립한다.
제2정규형(2NF)은 릴레이션이 1NF이면서, 기본키가 아닌 모든 속성이 기본키에 완전함수종속일 때(부분함수종속이 존재하지 않을 때) 성립한다.
제3정규형(3NF)은 릴레이션이 2NF이면서, 기본키가 아닌 속성 간에 이행적 함수종속이 존재하지 않을 때 성립한다.
보이스-코드 정규형(BCNF)은 릴레이션이 3NF이면서, 모든 결정자(determinant)가 후보키일 때 성립한다.
제4정규형(4NF)은 릴레이션이 BCNF이면서, 자명하지 않은 다치종속(multi-valued dependency)이 있는 경우 그 다치종속이 후보키를 통해서만 성립할 때 성립한다.

NoteBase

선지별 해설

ㄱ. 정규화는 함수적 종속성과 기본키를 기반으로 삽입·삭제·수정 이상을 줄이기 위해 릴레이션 스키마를 분석하고 분해하는 과정이라는 설명이다 → O, 정규화의 정의를 그대로 서술한 것으로 옳다.
ㄴ. 릴레이션 스키마 R의 모든 원소의 도메인이 "나눌 수 있는" 단위일 때 제1정규형이라는 설명이다 → X, 제1정규형은 도메인이 더 이상 나눌 수 없는(원자값) 단위일 때 성립하므로 "나눌 수 있는"이 아니라 "나눌 수 없는"으로 바꾸어야 옳은 설명이 된다.
ㄷ. 제2정규형이 되려면 R이 제1정규형이고 기본키가 아닌 속성이 기본키에 "부분함수 종속"이어야 한다는 설명이다 → X, 제2정규형은 기본키가 아닌 속성이 기본키에 부분함수종속이 아니라 완전함수종속이어야 성립하므로 "부분함수 종속"을 "완전함수 종속"으로 바꾸어야 옳은 설명이 된다.
ㄹ. 제3정규형이 되려면 R이 제2정규형이고, 함수종속 관계에서 이행적 함수종속을 제거해야 한다는 설명이다 → O, 제3정규형의 정의를 그대로 서술한 것으로 옳다.
ㅁ. 제4정규형이 되려면 R이 제3정규형이고, 함수종속 관계에서 모든 결정자가 후보키이면 된다는 설명이다 → X, 이는 제4정규형이 아니라 보이스-코드 정규형(BCNF)의 조건이다. 제4정규형이 되려면 R이 보이스-코드 정규형이면서 다치종속에 대한 조건까지 만족해야 하므로 이 설명은 제4정규형의 정의로는 옳지 않다.
따라서 옳은 설명은 ㄱ, ㄹ이므로 ① ㄱ, ㄹ이 정답이고, ② ㄴ, ㄷ / ③ ㄱ, ㄹ, ㅁ / ④ ㄷ, ㄹ, ㅁ은 모두 옳지 않은 항목(ㄴ, ㄷ, ㅁ)을 포함하므로 답이 될 수 없다.

오답 포인트

ㄴ을 옳다고 착각하는 경우는 "나눌 수 있는"과 "나눌 수 없는"을 순간적으로 혼동하는 경우이다. 1NF는 값을 더 쪼갤 수 없어야 한다는 점을 기억해야 한다.
ㄷ을 옳다고 착각하는 경우는 부분함수종속과 완전함수종속을 헷갈리는 경우이다. 2NF는 부분함수종속을 없애는 단계, 즉 완전함수종속만 남기는 단계임을 구분해야 한다.
ㅁ을 옳다고 착각하는 경우는 "모든 결정자가 후보키"라는 조건을 4NF의 조건으로 오인하는 경우이다. 이 조건은 BCNF의 조건이며, 4NF는 여기에 다치종속 조건이 추가로 붙는다는 점을 구분해야 한다.

다시 나오면 이렇게 판단

정규형을 묻는 지문에서는 먼저 "완전/부분함수종속", "이행적 함수종속", "모든 결정자가 후보키" 같은 키워드가 각각 몇 정규형의 조건인지부터 짝을 맞춘다. 부분함수종속이 나오면 2NF가 아니라 2NF 이전 상태(1NF에 머무는 이유)를 가리키는 표현이라는 점, "모든 결정자가 후보키"는 BCNF의 정의라는 점을 우선 확인한 뒤 선지를 O/X로 판정한다.

NoteBase

암기 포인트

1NF: 모든 속성값이 원자값(더 이상 나눌 수 없음).
2NF: 1NF + 완전함수종속(부분함수종속 제거).
3NF: 2NF + 이행적 함수종속 제거.
BCNF: 3NF + 모든 결정자가 후보키.
4NF: BCNF + 다치종속이 후보키를 통해서만 성립.

NoteBase

문 10. NOSQL에 대한 설명으로 옳지 않은 것은?

① NOSQL은 샤딩(sharding)을 지원한다.

② BigTable, Cassandra 등이 대표적인 NOSQL이다.

③ NOSQL은 RDBMS와 같이 스키마(schema)를 필요로 한다.

④ NOSQL은 가용성(availability)과 확장성(scalability)을 중요시한다.

정답 3번

출제 포인트

NOSQL의 기본 특징을 RDBMS와 대비해 정확히 알고 있는지 묻는 문항이다. 스키마 유연성, 분산 저장 방식(샤딩), 대표 제품, 설계 목표(가용성·확장성) 네 가지 축에서 옳지 않은 설명을 골라내야 한다.

핵심 개념

NOSQL(Not Only SQL)은 고정된 스키마 없이 데이터를 저장하는 스키마리스(schema-less) 구조를 특징으로 한다. RDBMS는 테이블 생성 시 컬럼과 자료형을 미리 정의한 스키마가 반드시 필요하지만, NOSQL은 문서·키-값·컬럼-패밀리·그래프 등 다양한 모델에서 각 레코드마다 다른 구조의 데이터를 유연하게 저장할 수 있다. 또한 NOSQL은 대용량 데이터를 여러 서버로 분산 저장하는 샤딩을 기본적으로 지원하며, CAP 정리(Consistency, Availability, Partition tolerance) 관점에서 강한 일관성보다는 가용성과 수평적 확장성을 중요시하도록 설계된 경우가 많다.

선지별 해설

① NOSQL은 샤딩(sharding)을 지원한다. → O. NOSQL은 대량의 데이터를 여러 노드에 분산 저장하기 위해 샤딩(파티셔닝) 기능을 기본적으로 지원한다.

② BigTable, Cassandra 등이 대표적인 NOSQL이다. → O. Google의 BigTable과 Apache Cassandra는 각각 컬럼-패밀리 모델을 사용하는 대표적인 NOSQL 데이터베이스이다.

③ NOSQL은 RDBMS와 같이 스키마(schema)를 필요로 한다. → X. 틀린 표현 → 올바른 표현: NOSQL은 고정된 스키마를 요구하지 않는 스키마리스 구조가 핵심 특징이며, 이는 RDBMS와 정반대되는 성질이다. RDBMS처럼 스키마를 미리 정의해야 한다는 설명은 NOSQL의 본질과 어긋난다.

④ NOSQL은 가용성(availability)과 확장성(scalability)을 중요시한다. → O. NOSQL은 대규모 분산 환경에서 서비스가 끊기지 않는 가용성과 노드를 추가해 처리 능력을 늘리는 수평적 확장성을 핵심 설계 목표로 삼는다.

NoteBase

오답 포인트

'스키마'라는 단어만 보고 NOSQL에도 어느 정도 구조가 있으니 스키마가 필요하다고 착각하면 안 된다. NOSQL도 내부적으로 데이터 형태를 갖지만, RDBMS처럼 저장 전에 스키마를 강제로 정의해야 하는 것은 아니라는 점이 핵심 구분이다. 또한 샤딩과 확장성을 헷갈려 NOSQL이 분산 처리에 취약하다고 생각하는 오답도 주의한다.

다시 나오면 이렇게 판단

NOSQL 관련 문항에서는 'RDBMS와 같이 스키마가 필요하다' 혹은 '고정된 스키마가 필수이다'처럼 스키마리스 특성을 부정하는 선지가 나오면 옳지 않은 설명일 가능성이 높다고 우선 의심한다. 반대로 샤딩 지원, 가용성·확장성 강조, 대표 제품(BigTable, Cassandra, MongoDB, Redis 등) 언급은 대체로 옳은 설명으로 처리한다.

암기 포인트

NOSQL 4대 특징: 스키마리스(고정 스키마 불필요) · 샤딩을 통한 수평 분산 저장 · 가용성·확장성 중심 설계 · 대표 제품 BigTable(컬럼형), Cassandra(컬럼형), MongoDB(문서형), Redis(키-값형).

NoteBase

문 11. 함수 F의 종속성이 다음과 같을 때 구할 수 있는 함수 종속성으로 옳지 않은 것은?

$F = \{A \to B, D \to B, D \to AC, BE \to D\}$

① $E \to D$

② $E \to E$

③ $AE \to D$

④ $AC \to BC$

정답 1번

출제 포인트

주어진 함수 종속성 집합 F가 있을 때, 특정 속성 집합의 폐포(closure)를 계산해서 그 폐포에 포함되지 않는 함수 종속을 찾아내는 문제이다. 각 선지의 좌변 속성 집합에 대해 폐포를 구한 뒤, 우변 속성이 그 폐포에 포함되는지를 확인해야 한다.

조건 정리

$F = \{A \to B,\ D \to B,\ D \to AC,\ BE \to D\}$이다. 각 선지가 제시하는 함수 종속이 F로부터 논리적으로 유도되는지(F의 함수 종속 폐포 F^+에 속하는지)를 판단해야 한다.

사용할 공식

속성 집합 X의 폐포 X^+는 다음 알고리즘으로 구한다. 먼저 $X^+ = X$로 둔다. F의 각 함수 종속 $Y \to Z$에 대해 $Y \subseteq X^+$이면 $X^+ = X^+ \cup Z$로 갱신한다. 더 이상 추가되는 속성이 없을 때까지 이 과정을 반복한다. $X \to A$가 성립하려면 $A \subseteq X^+$이어야 한다.

계산 과정

① E+ 계산: 시작 집합은 $\{E\}$이다. F의 좌변(A, D, BE) 중 어느 것도 $\{E\}$의 부분집합이 아니므로 더 이상 추가되는 속성이 없다. 따라서 $E^+ = \{E\}$이고 D가 포함되지 않는다.
③ AE+ 계산: 시작 집합은 $\{A, E\}$이다. $A \to B$가 적용되어 B가 추가되면 $\{A, B, E\}$가 된다. 이제 BE가 부분집합이 되므로 $BE \to D$가 적용되어 D가 추가되어 $\{A, B, D, E\}$가 된다. 다시 $D \to AC$가 적용되어 C가 추가되어 $\{A, B, C, D, E\}$가 된다. 따라서 AE^+는 D를 포함한다.
④ AC+ 계산: 시작 집합은 $\{A, C\}$이다. $A \to B$가 적용되어 B가 추가되어 $\{A, B, C\}$가 된다. D를 포함하지 않으므로 $D \to B, D \to AC$는 적용되지 않고, BE도 부분집합이 아니므로 더 이상 추가되지 않는다. 따라서 $AC^+ = \{A, B, C\}$이고 B와 C를 모두 포함한다.

NoteBase

선지 판단

① E→D: X → 틀린 표현. $E^+ = \{E\}$이므로 D가 폐포에 포함되지 않아 $E \to D$는 F로부터 유도되지 않는다 → 옳지 않은 것이므로 이 문제의 정답이다.
② E→E: O. 임의의 속성 집합은 자기 자신을 항상 함수 종속으로 결정하는 자명한 함수 종속(reflexivity)이므로 $E \to E$는 항상 성립한다.
③ AE→D: O. 계산 과정에서 보였듯이 AE^+에 D가 포함되므로 $AE \to D$는 성립한다.
④ AC→BC: O. 계산 과정에서 보였듯이 AC^+에 B와 C가 모두 포함되므로 $AC \to BC$는 성립한다.

오답 포인트

폐포를 구할 때 시작 속성 집합 자체에 포함된 속성만 보고 다른 함수 종속의 좌변이 그 폐포의 부분집합인지 매번 다시 확인하지 않으면, 연쇄적으로 추가되는 속성(예: AE+에서 B가 추가된 뒤 BE가 성립해 D까지 연결되는 것)을 놓치기 쉽다. 또한 E+처럼 좌변이 한 개 속성뿐인 경우 F의 어떤 함수 종속도 적용되지 않으면 그 폐포는 자기 자신에서 더 이상 확장되지 않는다는 점을 놓치지 않아야 한다.

다시 나오면 이렇게 판단

함수 종속 폐포 문제가 나오면 먼저 물어보는 속성 집합을 X로 놓고 X+ = X에서 시작한 뒤, F에 있는 함수 종속들의 좌변이 현재까지 구한 폐포의 부분집합이 되는 순간마다 우변을 추가해 나가면서 더 이상 추가할 것이 없을 때까지 반복한다. 그 다음 각 선지의 우변이 좌변의 폐포에 포함되는지만 확인하면 옳고 그름을 판단할 수 있다.

암기 포인트

함수 종속 폐포 계산은 항상 시작 집합에서 출발해 적용 가능한 규칙을 반복적으로 적용하는 고정점(fixpoint) 계산이라는 점, 그리고 어떤 속성 집합이든 자기 자신을 결정하는 자명한 함수 종속(reflexivity, X→X)은 항상 성립한다는 점을 기억해 둔다.

NoteBase

문 12. 데이터베이스 트리거(trigger)에 대한 설명으로 옳지 않은 것은?

① 테이블에서 이벤트 발생 시 자동으로 반응해 실행되는 작업이다.

② 트리거는 데이터베이스의 무결성을 유지하기 위한 도구이다.

③ 행-수준 트리거(row-level trigger)는 FOR EACH STATEMENT 절을 사용하여 표시한다.

④ 이벤트가 발생한 이후 실행되는 After 트리거와 이벤트가 발생하기 전에 실행되는 Before 트리거가 있다.

정답 3번

출제 포인트

트리거(trigger)의 기본 개념과 동작 방식, 그리고 행-수준 트리거와 문장-수준 트리거를 구분하는 SQL 문법(FOR EACH ROW / FOR EACH STATEMENT)을 정확히 아는지를 묻는다.

핵심 개념

트리거는 테이블에 삽입·수정·삭제 같은 이벤트가 발생했을 때 데이터베이스가 자동으로 반응하여 실행하는 절차형 객체이다. 주로 참조 무결성이나 업무 규칙을 강제하기 위한 도구로 쓰인다.
트리거는 실행 시점에 따라 이벤트 발생 전에 동작하는 Before 트리거와 이벤트 발생 후에 동작하는 After 트리거로 나뉜다.
트리거는 처리 단위에 따라서도 구분된다. 행-수준 트리거는 영향을 받은 행 하나하나마다 반복 실행되며 FOR EACH ROW 절로 지정한다. 반면 문장-수준 트리거는 SQL 문장 하나당 한 번만 실행되며 FOR EACH STATEMENT 절로 지정한다(또는 절을 생략하면 기본이 문장-수준이 된다).

선지별 해설

① O — 트리거는 테이블에서 삽입·수정·삭제 등의 이벤트가 발생하면 데이터베이스가 자동으로 반응해 실행하는 작업이 맞다.
② O — 트리거는 참조 무결성 유지, 업무 규칙 강제 등 데이터베이스의 무결성을 지키기 위한 대표적인 도구로 사용된다.
③ X — 행-수준 트리거(row-level trigger)는 FOR EACH ROW 절을 사용하여 표시한다. FOR EACH STATEMENT 절은 문장-수준 트리거(statement-level trigger)를 표시할 때 사용하는 절이므로, 선지에 제시된 절과 트리거 종류가 서로 뒤바뀌어 있다.
④ O — 트리거는 실행 시점에 따라 이벤트 발생 후 실행되는 After 트리거와 이벤트 발생 전에 실행되는 Before 트리거로 나뉘는 것이 맞다.

NoteBase

오답 포인트

행-수준과 문장-수준을 지정하는 절의 이름을 서로 바꿔서 외우면 이런 유형의 문제에서 함정에 빠지기 쉽다. ROW는 '행'을 의미하므로 FOR EACH ROW가 행-수준, STATEMENT는 'SQL 문장' 자체를 의미하므로 FOR EACH STATEMENT가 문장-수준이라는 단어의 뜻과 절 이름을 연결해 기억해 두면 헷갈리지 않는다.

다시 나오면 이렇게 판단

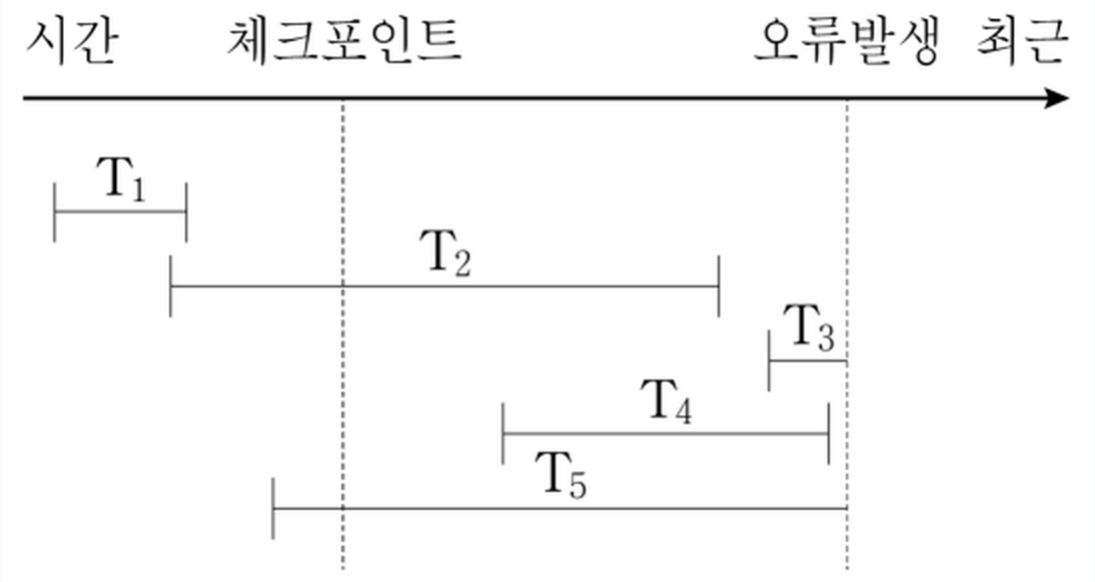
선지에 FOR EACH ROW/FOR EACH STATEMENT 같은 구체적인 절 이름이 등장하면, 먼저 그 절이 행-수준을 가리키는지 문장-수준을 가리키는지부터 확인한 뒤 선지에서 말하는 트리거 종류와 일치하는지 대조한다. 트리거의 실행 시점(Before/After)과 처리 단위(행-수준/문장-수준)는 서로 다른 분류 기준이므로 섞어서 헷갈리지 않도록 구분해 둔다.

암기 포인트

행-수준 트리거 = FOR EACH ROW(행마다 반복 실행), 문장-수준 트리거 = FOR EACH STATEMENT(문장당 한 번 실행). 실행 시점 구분은 Before(이벤트 전) / After(이벤트 후).

NoteBase

문 13. 트랜잭션 수행 과정이 다음과 같을 때, DBMS에서 데이터 복원을 위해 수행되는 작업으로 옳지 않은 것은?



① T2는 redo(재실행)된다.

② T4는 undo(실행취소)된다.

③ T1에 대해서는 어떠한 작업도 수행하지 않는다.

④ 체크포인트 시 로그 레코드 <checkpoint {T2, T5}>를 저장장치에 기록한다.

정답 2번

출제 포인트

체크포인트 기법을 이용한 장애 회복에서, 체크포인트와 시스템 장애(오류발생) 시점 사이에 각 트랜잭션이 걸쳐 있는 구간을 보고 redo 대상인지 undo 대상인지 판정할 수 있는지를 묻는다. 판정 기준은 두 가지뿐이다. 체크포인트 이전에 이미 완료(커밋)된 트랜잭션은 회복 작업이 필요 없고, 체크포인트 이후에도 계속 실행 중이던 트랜잭션 중 장애 시점 이전에 완료된 것은 redo, 장애 시점까지도 완료되지 못한 것은 undo 대상이 된다.

자료 읽는 순서

먼저 각 트랜잭션의 시작점이 체크포인트보다 앞인지 뒤인지를 본다. 다음으로 각 트랜잭션의 종료점이 오류발생(장애) 시점보다 앞(완료)인지, 장애 시점까지 이어져 있는지(미완료)를 본다. T1은 시작과 종료가 모두 체크포인트 이전이므로 이미 디스크에 반영되어 회복 대상이 아니다. T2는 체크포인트 이전에 시작해 장애 이전에 종료되었으므로 redo 대상이다. T3와 T4는 체크포인트 이후에 시작했지만 장애 발생 이전에 종료되었으므로 이 역시 redo 대상이다. T5는 체크포인트 이전에 시작했지만 장애 시점까지도 종료되지 않고 진행 중이므로 undo 대상이다. 즉 회복 대상 분류는 T1=해당 없음, T2·T3·T4=redo, T5=undo이다.

NoteBase

선지별 해설

① T2는 redo(재실행)된다. → O. T2는 체크포인트 이전에 시작해 장애 발생 이전에 완료되었으므로 로그를 이용해 재실행(redo)하는 대상이 맞다.
② T4는 undo(실행취소)된다. → X. T4는 체크포인트 이후에 시작했지만 장애 발생 이전에 이미 종료(완료)된 트랜잭션이므로 실행취소(undo) 대상이 아니라 재실행(redo) 대상이다. 틀린 표현 "T4는 undo된다" → 올바른 표현 "T4는 redo된다".
③ T1에 대해서는 어떠한 작업도 수행하지 않는다. → O. T1은 체크포인트 이전에 이미 시작과 종료가 모두 끝난 트랜잭션이므로 체크포인트 시점에 이미 그 결과가 디스크에 반영되어 있어 redo도 undo도 필요 없다.
④ 체크포인트 시 로그 레코드 <checkpoint {T2, T5}>를 저장장치에 기록한다. → O. 체크포인트 로그 레코드에는 그 시점에 아직 종료되지 않고 실행 중인(active) 트랜잭션들의 목록이 기록되는데, 체크포인트 시점에 시작은 했지만 아직 끝나지 않은 트랜잭션은 T2와 T5이므로 이 표현은 맞다.

오답 포인트

이 문제에서 가장 흔한 실수는 "체크포인트 이후에 시작한 트랜잭션은 무조건 undo"라고 단순화하는 것이다. undo와 redo를 가르는 기준은 시작 시점이 아니라 종료 시점, 즉 장애 발생 시점까지 완료되었는지 여부이다. T4처럼 체크포인트 이후에 시작했더라도 장애 발생 전에 이미 커밋되었다면 redo 대상이지 undo 대상이 아니다. 반대로 T5처럼 체크포인트 이전에 시작했더라도 장애 시점까지 끝나지 않았다면 undo 대상이 된다.

다시 나오면 이렇게 판단

타임라인 그림이 나오면 먼저 체크포인트 선과 장애발생 선 두 개의 기준선을 찾고, 각 트랜잭션 막대의 오른쪽 끝(종료점)이 장애발생 선의 왼쪽에서 끝나는지, 아니면 장애발생 선까지 이어지는지만 확인한다. 왼쪽에서 끝나 있으면(완료) redo, 장애 시점까지 이어져 있으면(미완료) undo, 체크포인트 이전에 막대 전체가 끝나 있으면 해당 없음으로 분류한다.

암기 포인트

redo/undo 판정은 시작 시점이 아니라 종료 시점 기준이다: 장애 발생 전 완료=redo, 장애 발생 시점까지 미완료=undo, 체크포인트 이전 완료=해당 없음. 체크포인트 로그 레코드에는 체크포인트 시점의 활성(active) 트랜잭션 목록이 기록된다.

NoteBase

문 14. 다음 데이터베이스가 만족하지 않는 제약조건은? (단, STUDENT 릴레이션의 Dept_No는 DEPARTMENT 릴레이션의 Dept_Number를 참조한다)

STUDENT

ID	Name	Dept_No	Birth_Data
20120011	이지연	1	1993-01-23
20130023	김병수	3	1994-11-05
20140101	김남현	3	1995-08-21
20144006	차기현	2	1995-04-19
20105789	안정민	4	1991-01-03

DEPARTMENT

Dept_Name	Dept_Number	Quota	Dean
컴퓨터공학과	1	35	김지연
전자공학과	2	40	정주현

① 유일성 제약조건

② 참조 무결성 제약조건

③ 엔티티 무결성 제약조건

④ 함수적 종속성 제약조건

정답 2번

출제 포인트

STUDENT 릴레이션과 DEPARTMENT 릴레이션의 실제 데이터 값을 비교해 어떤 무결성 제약조건이 위배되는지 찾는 문제이다. STUDENT의 Dept_No는 DEPARTMENT의 기본키인 Dept_Number를 참조하는 외래키이므로, 두 릴레이션의 값을 대조해 참조 무결성이 지켜지는지 확인하는 것이 핵심이다.

NoteBase

자료 읽는 순서

먼저 DEPARTMENT 릴레이션에서 실제로 존재하는 Dept_Number 값의 집합을 확인한다. 표에는 1(컴퓨터공학과), 2(전자공학과) 두 값만 존재한다.
다음으로 STUDENT 릴레이션의 Dept_No 값을 하나씩 대조한다. 이지연=1, 김병수=3, 김남현=3, 차기현=2, 안정민=4이다.
이 중 3과 4는 DEPARTMENT 릴레이션의 Dept_Number 어디에도 존재하지 않는 값이다. 즉 김병수·김남현·안정민의 Dept_No는 참조할 부모 행이 없는 상태이므로, 외래키가 부모 릴레이션에 존재하지 않는 값을 가리키는 참조 무결성 위반이 발생한 것이다.

선지별 해설

① 유일성 제약조건: X. STUDENT의 기본키 ID(20120011, 20130023, 20140101, 20144006, 20105789)와 DEPARTMENT의 기본키 Dept_Number(1, 2)는 각 릴레이션 안에서 모두 서로 다른 값으로 중복이 없다. 유일성은 위배되지 않는다.
② 참조 무결성 제약조건: O. STUDENT.Dept_No 값 중 3(김병수, 김남현)과 4(안정민)는 DEPARTMENT.Dept_Number에 존재하지 않는 값이다. 외래키 값이 참조 대상 릴레이션의 기본키 값 중 하나와 일치해야 한다는 참조 무결성이 지켜지지 않는다.
③ 엔티티 무결성 제약조건: X. 엔티티 무결성은 기본키가 널(NULL) 값을 가질 수 없다는 제약이다. STUDENT의 ID, DEPARTMENT의 Dept_Number 모두 모든 행에서 값이 채워져 있어 널이 없다. 위배되지 않는다.
④ 함수적 종속성 제약조건: X. 함수적 종속성은 한 릴레이션 내에서 특정 속성 값이 다른 속성 값을 유일하게 결정하는 관계로, 이 표의 데이터만으로는 별도의 함수적 종속 규칙이 깨졌다고 볼 근거가 없다(ID가 기본키로서 나머지 속성을 결정하는 구조는 정상적으로 유지된다).

오답 포인트

①과 ③을 헷갈리는 경우가 많은데, 두 제약 모두 '기본키'에 관한 것이지만 대상이 다르다. 유일성은 값의 중복 여부, 엔티티 무결성은 값의 널 여부를 다룬다. 이 문제의 표에는 중복도 널도 없으므로 둘 다 해당하지 않는다. 반면 참조 무결성은 기본키가 아니라 외래키(FK)가 부모 릴레이션의 기본키 값 범위 안에 있는지를 보는 제약이라는 점에서 완전히 다른 축의 제약조건임을 구분해야 한다.

다시 나오면 이렇게 판단

두 릴레이션이 제시되고 한쪽에 외래키 참조 관계가 명시되면, 가장 먼저 부모 릴레이션(참조되는 쪽)의 기본키 값 집합을 적어두고, 자식 릴레이션(참조하는 쪽)의 외래키 값을 하나씩 그 집합과 대조한다. 대조 중 집합에 없는 값이 하나라도 나오면 참조 무결성 위반으로 즉시 판단한다. 기본키 열에서 중복이나 빈 칸이 있는지는 별도로 확인해 유일성·엔티티 무결성과 혼동하지 않는다.

NoteBase

암기 포인트

유일성 제약조건=기본키 값의 중복 금지, 엔티티 무결성 제약조건=기본키 값의 널(NULL) 금지, 참조 무결성 제약조건=외래키 값은 반드시 부모 릴레이션의 기본키 값 중 하나이거나 널이어야 함, 함수적 종속성=속성 간 결정 관계로 나누어 기억해 둔다.

NoteBase

문 15. 외부, 개념, 내부 스키마로 구성되는 3-레벨 스키마 구조와 데이터 독립성에 대한 설명으로 옳지 않은 것은?

① 내부 레벨 또는 내부 스키마는 데이터베이스의 물리적 스토리지 구조를 기술한다.

② 논리적 데이터 독립성은 응용 프로그램이나 데이터베이스의 논리적 구조에 영향을 주지 않고 DBMS가 데이터의 물리적 구조를 변경할 수 있는 것을 말한다.

③ 외부 스키마는 특정 사용자 그룹이 관심을 가지고 있는 데이터베이스의 일부분을 기술하며 데이터베이스의 나머지 부분을 그 사용자 그룹으로부터 숨겨 준다.

④ SQL 뷰가 정의된 기본 테이블이 확장된다든지 뷰가 속해 있는 테이블이 더 늘어난다고 하더라도 기존의 뷰를 사용하는 프로그램이나 사용자는 영향을 받지 않으므로, SQL 뷰는 데이터의 논리적 독립성을 어느 정도 제공할 수 있다.

정답 2번

출제 포인트

3-레벨 스키마 구조(외부·개념·내부 스키마)의 각 층이 기술하는 대상과, 논리적 데이터 독립성·물리적 데이터 독립성의 정의를 서로 바꿔치기해도 알아채는지를 묻는다. 특히 '논리적 데이터 독립성'과 '물리적 데이터 독립성'의 정의를 정확히 구분하는 것이 핵심이다.

핵심 개념

물리적 데이터 독립성=내부(물리적) 스키마를 변경해도 개념 스키마와 그 위의 응용 프로그램·외부 스키마가 영향을 받지 않는 성질이다. 논리적 데이터 독립성=개념 스키마(논리적 구조)를 변경해도 외부 스키마와 그것을 사용하는 응용 프로그램이 영향을 받지 않는 성질이다. 즉 물리적 독립성은 '내부 변경 → 개념/논리 불변'을 말하고, 논리적 독립성은 '개념/논리 변경 → 외부/응용 불변'을 말한다.

NoteBase

선지별 해설

① O — 내부 레벨(내부 스키마)은 데이터베이스가 실제로 저장되는 방식, 즉 물리적 스토리지 구조(파일 조직, 인덱스, 저장 방식 등)를 기술하는 층이므로 옳은 설명이다.
② X — 이 선지는 '논리적 데이터 독립성'을 정의한다면서 실제로는 물리적 데이터 독립성의 정의를 가져다 썼다. 논리적 데이터 독립성은 개념(논리적) 스키마를 변경해도 응용 프로그램이나 외부 스키마가 영향을 받지 않는 것을 말하는데, 이 선지는 반대로 '논리적 구조에 영향을 주지 않고 물리적 구조를 변경할 수 있는 것'이라고 서술했다. 이는 물리적 데이터 독립성의 정의이므로 틀린 표현이다. 틀린 표현 → 올바른 표현: '응용 프로그램이나 데이터베이스의 논리적 구조에 영향을 주지 않고 DBMS가 물리적 구조를 변경할 수 있는 것'은 물리적 데이터 독립성이며, 논리적 데이터 독립성은 '개념 스키마가 변경되어도 외부 스키마와 응용 프로그램이 영향을 받지 않는 것'이다.
③ O — 외부 스키마는 개별 사용자나 사용자 그룹의 관점에서 필요한 데이터베이스의 일부만을 정의하고, 그 나머지 부분은 해당 사용자 그룹에게 노출하지 않으므로 옳은 설명이다.
④ O — SQL 뷰는 개념 스키마(기본 테이블)가 확장되거나 변경되어도 기존 뷰의 정의만 그대로 유지되면 그 뷰를 사용하는 응용 프로그램이나 사용자는 영향을 받지 않는다. 이는 뷰가 논리적 데이터 독립성을 어느 정도 제공하는 대표적인 예이므로 옳은 설명이다.

오답 포인트

'논리적'과 '물리적'이라는 단어만 보고 넘어가면 ②를 옳은 설명으로 착각하기 쉽다. 정의 문장 안에서 '무엇이 변하고 무엇이 영향을 받지 않는지'의 주체를 뒤집어 놓았는지를 반드시 확인해야 한다. 물리적 독립성은 내부(물리적) 스키마의 변경이 원인이고, 논리적 독립성은 개념(논리적) 스키마의 변경이 원인이라는 인과관계를 헷갈리면 안 된다.

다시 나오면 이렇게 판단

데이터 독립성 관련 선지가 나오면 먼저 '무엇이 바뀌었을 때(원인)' '무엇이 안 바뀌는지(결과)'를 표로 그리듯 정리한다. 원인이 내부/물리적 스키마 변경이면 물리적 데이터 독립성, 원인이 개념/논리적 스키마 변경이면 논리적 데이터 독립성이다. 이름과 정의의 원인-결과 방향이 서로 맞바뀐 함정 선지를 의심하며 읽는다.

암기 포인트

물리적 데이터 독립성: 내부 스키마 변경 → 개념 스키마·응용 프로그램 불변. 논리적 데이터 독립성: 개념 스키마 변경 → 외부 스키마·응용 프로그램 불변. 외부 스키마=사용자 관점의 일부 뷰, 개념 스키마=전체 논리적 구조, 내부 스키마=물리적 저장 구조. SQL 뷰는 논리적 데이터 독립성의 대표 사례이다.

NoteBase

문 16. 다음 SQL 명령어 중 데이터 정의어(DDL)만을 고르면?

ㄱ. ALTER
ㄴ. DROP
ㄷ. TRUNCATE
ㄹ. GRANT
ㅁ. COMMIT

① ㄱ, ㅁ
② ㄴ, ㄷ
③ ㄷ, ㄹ
④ ㄹ, ㅁ

정답 2번

출제 포인트

SQL 명령어를 DDL(데이터 정의어), DML(데이터 조작어), DCL(데이터 제어어), TCL(트랜잭션 제어어)로 분류할 수 있는지를 묻는다. 보기로 주어진 다섯 개 명령어(ALTER, DROP, TRUNCATE, GRANT, COMMIT)가 각각 어느 범주에 속하는지 정확히 구분해야 한다.

핵심 개념

DDL(Data Definition Language)은 테이블·스키마 등 데이터베이스 객체의 구조를 정의·변경·삭제하는 명령어로 CREATE, ALTER, DROP, TRUNCATE, RENAME이 해당한다.
DML(Data Manipulation Language)은 데이터를 조회·삽입·수정·삭제하는 명령어로 SELECT, INSERT, UPDATE, DELETE가 해당한다.
DCL(Data Control Language)은 권한을 부여·회수하는 명령어로 GRANT, REVOKE가 해당한다.
TCL(Transaction Control Language)은 트랜잭션을 확정·취소하는 명령어로 COMMIT, ROLLBACK, SAVEPOINT가 해당한다.
TRUNCATE는 DELETE와 달리 테이블의 모든 행을 삭제하면서 저장공간을 초기화하고 롤백이 불가능한 특성을 가지는데, 이는 데이터를 정의하는 테이블 구조 차원의 명령으로 취급되어 DDL로 분류되고 별도의 WHERE 조건도 사용할 수 없다.

NoteBase

선지별 해설

ㄱ(ALTER): DDL이다. 테이블의 열 추가·삭제·변경 등 구조를 수정하는 명령어이다.
ㄴ(DROP): DDL이다. 테이블이나 스키마 등 객체 자체를 삭제하는 명령어이다.
ㄷ(TRUNCATE): DDL이다. 테이블의 모든 데이터를 삭제하고 저장공간을 초기화하는 명령어로 구조 관련 명령으로 분류된다.
ㄹ(GRANT): DCL이다. 사용자에게 데이터베이스 접근·조작 권한을 부여하는 명령어이므로 DDL이 아니다.
ㅁ(COMMIT): TCL이다. 트랜잭션의 변경사항을 확정하는 명령어이므로 DDL이 아니다.
① ㄱ, ㅁ → ALTER는 DDL이 맞지만 COMMIT은 TCL이므로 틀린 조합이다.
② ㄴ, ㄷ → DROP과 TRUNCATE 모두 DDL이므로 옳은 조합이다.
③ ㄷ, ㄹ → TRUNCATE는 DDL이 맞지만 GRANT는 DCL이므로 틀린 조합이다.
④ ㄹ, ㅁ → GRANT는 DCL, COMMIT은 TCL이므로 DDL이 하나도 없는 틀린 조합이다.

오답 포인트

GRANT를 DDL로 착각하는 경우가 많은데, 권한을 다루는 명령어는 DCL로 분류된다. 또한 COMMIT을 데이터 조작과 관련된 명령으로 오인해 DML이나 DDL로 착각하기 쉬운데, COMMIT은 트랜잭션을 확정하는 TCL 명령어이다. TRUNCATE는 DELETE(DML)와 기능이 비슷해 DML로 착각하기 쉬우나, 테이블 구조 차원에서 저장공간을 초기화하고 롤백이 불가능하다는 점에서 DDL로 분류된다는 점을 구분해야 한다.

다시 나오면 이렇게 판단

명령어를 하나씩 DDL·DML·DCL·TCL 네 범주 중 어디에 속하는지 표시한 뒤, 문제에서 요구하는 범주(이 문제는 DDL)에 해당하는 것만 남기고 나머지를 제거하는 방식으로 접근한다. TRUNCATE와 GRANT, COMMIT의 소속 범주를 헷갈리지 않는 것이 핵심이다.

암기 포인트

DDL: CREATE, ALTER, DROP, TRUNCATE, RENAME(구조 정의).
DML: SELECT, INSERT, UPDATE, DELETE(데이터 조작).
DCL: GRANT, REVOKE(권한 제어).
TCL: COMMIT, ROLLBACK, SAVEPOINT(트랜잭션 제어).

NoteBase

문 17. 다음 데이터베이스 스키마를 보고, 아래 SQL 질의문과 동등한 관계대수식은?

○ 스키마

사원(사원번호, 부서번호, 급여, 취미)
부서(부서번호, 부서명, 건물층, 전화번호)
재정(부서번호, 예산, 매출, 비용)

○ 질의문

```
Select D.부서명, F.예산
From 사원 E, 부서 D, 재정 F
Where E.부서번호=D.부서번호 and D.부서번호=F.부서번호
      and D.건물층=1 and E.급여>=3000000
      and E.취미='테니스'
```

① $\pi_{D.부서명,\ F.예산}((\pi_{E.부서번호}(\sigma_{E.급여>=3000000\ and\ E.취미='테니스'}(E)) \bowtie \pi_{D.부서번호,\ D.부서명}(\sigma_{D.건물층=1}(D))) \bowtie \pi_{F.예산,\ F.부서번호}(F))$

② $\pi_{E.부서번호}((\pi_{D.부서명,\ F.예산}(\sigma_{E.급여>=3000000\ and\ E.취미='테니스'}(E)) \bowtie \pi_{D.부서번호,\ D.부서명}(\sigma_{D.건물층=1}(D))) \bowtie \pi_{F.예산,\ F.부서번호}(F))$

③ $\pi_{D.부서명,\ F.예산}((\pi_{F.부서번호,\ D.부서명}(\sigma_{D.건물층=1}(D)) \bowtie \pi_{E.부서번호}(\sigma_{E.급여>=3000000\ and\ E.취미='테니스'}(E))) \bowtie \pi_{F.예산}(F))$

④ $\pi_{D.부서명,\ F.예산}((\pi_{E.부서번호}(\sigma_{F.예산,\ F.부서번호}(F)) \bowtie \pi_{D.부서번호,\ D.부서명}(\sigma_{D.건물층=1}(D))) \bowtie \pi_{E.급여><3000000\ and\ E.취미='테니스'}(E))$

정답 1번

출제 포인트

SQL 질의문을 관계대수식으로 옮길 때, 선택(σ)과 사영(π)을 조인보다 먼저 적용해 중간 결과의 크기를 줄이는 최적화된 형태를 찾는 문제이다. 조인 순서나 조건 자체가 아니라 '어디에 어떤 속성을 사영·선택으로 남겨야 조인이 성립하는가'가 관건이다.

조건 정리

조인 조건은 E.부서번호=D.부서번호, D.부서번호=F.부서번호 두 개이다. 선택 조건은 D.건물층=1, E.급여≥3000000, E.취미='테니스' 세 개이며, 급여·취미 조건은 사원(E)에, 건물층 조건은 부서(D)에 걸린다. 재정(F)에는 선택 조건이 없다. 최종 출력 속성은 D.부서명, F.예산이다.

NoteBase

사용할 공식

자연조인(⋈)이 성립하려면 조인하는 두 릴레이션에 공통 속성(조인키)이 남아 있어야 한다. 조인 전에 사영을 적용할 때는 (1) 조인키가 되는 속성, (2) 최종 출력에 필요한 속성만 남기고 나머지는 제거해야 중간 결과 크기가 줄면서도 결과가 원래 질의와 같아진다. 또한 선택·사영은 그 속성이 실제로 존재하는 릴레이션에만 적용할 수 있다.

계산 과정

사원(E)에는 급여·취미 조건으로 먼저 σ를 적용하고, 이후 조인에만 쓰이므로 부서번호만 π로 남긴다: π E.부서번호(σ E.급여≥3000000 and E.취미='테니스'(E)).
부서(D)에는 건물층=1 조건으로 σ를 적용하고, 조인키인 부서번호와 최종 출력에 필요한 부서명을 π로 남긴다: π D.부서번호, D.부서명(σ D.건물층=1(D)).
재정(F)에는 선택 조건이 없으므로 조인키 부서번호와 최종 출력에 필요한 예산만 π로 남긴다: π F.예산, F.부서번호(F).
세 결과를 부서번호를 기준으로 자연조인한 뒤, 바깥에서 π D.부서명, F.예산으로 최종 사영한다. 이 전체 형태가 1번과 정확히 일치한다.

선지 판단

① O: 세 릴레이션 모두에서 조인키(부서번호)와 최종 출력 속성만 남기고, σ를 조인보다 먼저 적용한 올바른 최적화 형태이다.
② X: 바깥쪽 사영이 π E.부서번호 하나뿐이고, 안쪽에서 사원(E)을 사영할 때 D.부서명, F.예산처럼 E에 존재하지 않는 다른 릴레이션의 속성을 사영 대상으로 지정하고 있어 사영 대상 릴레이션과 속성이 맞지 않는다.
③ X: 재정(F)을 마지막에 π F.예산(F)으로만 사영해 부서번호를 빼버렸기 때문에, D와 조인할 공통 속성(부서번호)이 사라져 자연조인이 성립하지 않는다.
④ X: σ의 조건 자리에 F.예산, F.부서번호처럼 속성 목록을 넣어 선택조건과 사영 목록이 뒤바뀌어 있고, 정작 급여·취미 조건이 걸려야 할 사원(E)에는 조건 없는 사영만 적용되어 있어 조건과 대상이 서로 어긋난다.

오답 포인트

조인 전에 사영을 적용해 중간 결과를 줄이려면 반드시 두 가지를 동시에 지켜야 한다. 첫째, 조인에 쓰일 공통 속성(부서번호)은 어느 릴레이션에서도 빠뜨리면 안 된다. 둘째, 사영·선택 대상 속성은 그 릴레이션에 실제로 존재하는 속성이어야 한다. ②는 두 번째 원칙을, ③은 첫 번째 원칙을, ④는 선택조건과 사영 대상의 배치 자체를 어긴 경우이다.

다시 나오면 이렇게 판단

선지들이 비슷해 보일 때는 먼저 각 σ와 π 안에 적힌 속성이 그 괄호 안 릴레이션에 실제로 존재하는 속성인지부터 확인한다. 그다음 조인마다 양쪽에 공통 속성(조인키)이 남아 있는지 확인하면 오답을 빠르게 걸러낼 수 있다.

NoteBase

암기 포인트

선택·사영 밀어넣기(push-down) 최적화 = 조인 전에 선택과 사영을 먼저 적용해 중간 결과를 줄이되, 조인키와 최종 출력 속성은 반드시 보존한다.

NoteBase

문 18. 다음 릴레이션 R과 S에 대한 조인 연산 결과가 옳지 않은 것은? (단, ⋈ N은 자연 조인, ⟕은 왼쪽 외부 조인, ⟖은 오른쪽 외부 조인, ⋉은 세미 조인을 의미한다)

R		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b3	c3
a5	b5	c5

S		
B	C	D
b1	c1	d1
b2	c2	d2
b2	c2	d3
b3	c3	d4
b4	c4	d5

① R ⋈ N S

A B C D

a1 b1 c1 d1

a2 b2 c2 d2

a2 b2 c2 d3

a3 b3 c3 d4

NoteBase

a4 b3 c3 d4

② $R \underset{R.B=S.B \text{ and } R.C=S.C}{\bowtie} S$

A B C D

a1 b1 c1 d1

a2 b2 c2 d2

a2 b2 c2 d3

a3 b3 c3 d4

a4 b3 c3 d4

a5 b5 c5 NULL

③ $R \underset{R.B=S.B \text{ and } R.C=S.C}{\bowtie} S$

A B C D

a1 b1 c1 d1

a2 b2 c2 d2

a2 b2 c2 d3

a3 b3 c3 d4

a4 b3 c3 d4

NULL b4 c4 d5

④ $R \underset{R.B=S.B \text{ and } R.C=S.C}{\ltimes} S$

A B C

a1 b1 c1

a2 b2 c2

a3 b3 c3

정답 4번

출제 포인트

자연 조인·왼쪽 외부 조인·오른쪽 외부 조인·세미 조인 네 가지 조인 연산의 정의를 정확히 알고, 주어진 릴레이션 R(A,B,C)과 S(B,C,D)에 각 연산을 적용했을 때 나오는 실제 결과 튜플과 스키마(포함되는 속성)를 직접 계산해 선지의 표와 대조할 수 있는지를 묻는다.

조건 정리

R의 튜플: (a1,b1,c1), (a2,b2,c2), (a3,b3,c3), (a4,b3,c3), (a5,b5,c5). S의 튜플: (b1,c1,d1), (b2,c2,d2), (b2,c2,d3), (b3,c3,d4), (b4,c4,d5). R과 S의 공통 속성은 B, C이다. R.B=S.B and R.C=S.C 조건으로 짝을 지으면 a1↔(b1,c1,d1), a2↔(b2,c2,d2)와 (b2,c2,d3) 두 건, a3↔(b3,c3,d4), a4↔(b3,c3,d4)가 매칭되고, a5(b5,c5)는 S에 대응 튜플이 없다. 반대로 S의 b4,c4는 R에 대응 튜플이 없다.

NoteBase

사용할 공식

자연 조인 $R \bowtie_N S$: 공통 속성(B,C) 값이 같은 튜플만 결합하고 공통 속성은 한 번만 남긴다(양쪽 모두 일치해야 결과에 포함, 즉 내부 조인). 왼쪽 외부 조인 $R ⟕ S$: 조인 조건으로 결합하되 R의 모든 튜플을 보존하며, 대응되는 S 튜플이 없으면 S쪽 속성을 NULL로 채운다. 오른쪽 외부 조인 $R ⟖ S$: 반대로 S의 모든 튜플을 보존하며, 대응되는 R 튜플이 없으면 R쪽 속성을 NULL로 채운다. 세미 조인 $R \ltimes S$: 조인 조건을 만족하는 R의 튜플만을, R의 속성(A,B,C)만 사용해 골라내는 연산이다 — 결과 스키마는 R의 속성이고, S와 여러 번 매칭되어도 중복 없이 R의 원래 튜플 수만큼만 남긴다.

계산 과정

① 자연 조인: a1,a2(2건),a3,a4가 매칭되어 (a1,b1,c1,d1),(a2,b2,c2,d2),(a2,b2,c2,d3),(a3,b3,c3,d4),(a4,b3,c3,d4) 5개 튜플이 나오고 a5는 매칭이 없으므로 제외된다. 이는 선지 ①의 결과와 정확히 일치한다.
② 왼쪽 외부 조인: 조인 결과(위 5개 튜플)에 R의 미매칭 튜플 a5를 D=NULL로 채워 추가한 (a5,b5,c5,NULL)까지 6개 튜플이 되어야 한다. 이는 선지 ②의 결과와 정확히 일치한다.
③ 오른쪽 외부 조인: 조인 결과(위 5개 튜플)에 S의 미매칭 튜플 (b4,c4,d5)를 A=NULL로 채워 추가한 (NULL,b4,c4,d5)까지 6개 튜플이 되어야 한다. 이는 선지 ③의 결과와 정확히 일치한다.
④ 세미 조인: R의 튜플 중 조인 조건을 만족하는 것은 a1,a2,a3,a4 네 개이며(a2는 S와 두 번 매칭되어도 세미 조인은 R의 튜플 자체만 남기므로 중복 없이 한 번만 포함), 속성은 A,B,C만 남는다. 따라서 옳은 결과는 (a1,b1,c1),(a2,b2,c2),(a3,b3,c3),(a4,b3,c3) 4개 튜플이어야 한다. 그런데 선지 ④의 표에는 (a1,b1,c1),(a2,b2,c2),(a3,b3,c3) 3개 튜플만 제시되어 있고, 조인 조건을 만족하는 a4(b3,c3)가 누락되어 있다. 이는 세미 조인의 정의에 맞지 않는 잘못된 결과이다.

선지 판단

① R ⨝N S: O — 계산 결과와 정확히 일치한다.
② R ⨝(R.B=S.B and R.C=S.C) S: O — 계산 결과와 정확히 일치한다.
③ R ⨝(R.B=S.B and R.C=S.C) S: O — 계산 결과와 정확히 일치한다.
④ R ⨝(R.B=S.B and R.C=S.C) S: X → a4(b3,c3)는 S의 (b3,c3,d4)와 조인 조건을 만족하므로 세미 조인 결과에 포함되어야 하는데 표에서 빠져 있다. 옳은 결과는 (a1,b1,c1),(a2,b2,c2),(a3,b3,c3),(a4,b3,c3) 4개 튜플이다.

오답 포인트

세미 조인은 조인 조건을 만족하는 R의 튜플을 '있는 그대로, 중복 없이' 골라내는 연산이라는 점을 놓치면 안 된다. a2가 S의 두 튜플과 매칭된다고 해서 결과에 a2가 두 번 나오지 않으며(자연 조인·외부 조인과 다른 점), 반대로 조건을 만족하는 R의 튜플(a4)을 빠뜨려서도 안 된다. 이 문제는 후자, 즉 결과 표에서 매칭되는 a4 튜플이 누락된 경우가 오류로 제시되었다.

NoteBase

다시 나오면 이렇게 판단

조인 결과표가 주어지면 먼저 R과 S의 튜플을 조인 조건(공통 속성 값 일치)에 따라 손으로 짝지어 본다. 자연 조인은 매칭되는 것만, 왼쪽 외부 조인은 R 전체 보존+미매칭 S쪽 NULL, 오른쪽 외부 조인은 S 전체 보존+미매칭 R쪽 NULL, 세미 조인은 매칭되는 R의 튜플만(R의 속성만, 중복 없이) 남는다는 네 가지 정의를 표의 튜플 개수·속성 개수·NULL 위치와 하나씩 대조하면 빠진 튜플이나 잘못 채워진 값을 바로 찾아낼 수 있다.

암기 포인트

자연 조인=매칭만, 왼쪽 외부=R 전체 보존, 오른쪽 외부=S 전체 보존, 세미 조인=매칭되는 R 튜플만(R 속성만, 중복 없이 원래 개수만큼). 세미 조인 결과의 튜플 수는 항상 원래 R에서 조인 조건을 만족하는 튜플 수와 같아야 한다는 것을 기준으로 삼아 개수를 검산한다.

NoteBase

문 19. 분산 데이터베이스 관리시스템에서 복수의 분할된 물리적 데이터베이스를 논리적으로 단일화된 데이터베이스처럼 인식하게 하기 위한 투명성(transparency)에 대한 설명으로 옳지 않은 것은?

① 단편화(fragmentation) 투명성은 사용자들이 릴레이션이 어떻게 단편화되어 있는지 알 필요가 없다.

② 장애(failure) 투명성은 다수의 트랜잭션이 동시에 수행되는 경우에도 오류 없이 결과의 일관성이 유지되어야 하는 성질이다.

③ 중복(replication) 투명성은 중복된 데이터가 무엇인지 저장 위치 등에 대한 정보를 사용자가 별도로 인지할 필요가 없어야 하는 성질이다.

④ 위치(location) 투명성은 사용자나 애플리케이션에서 작업을 수행하기 위해 데이터의 물리적인 위치도 알 필요가 없어야 하는 성질이다.

정답 2번

출제 포인트

분산 데이터베이스 관리시스템(DDBMS)이 여러 사이트에 흩어진 물리적 데이터베이스를 사용자에게 하나의 논리적 데이터베이스처럼 보이게 하는 여러 투명성 개념 중, 각 투명성이 정확히 어떤 대상(단편화·중복·위치·장애)을 사용자로부터 감추는지 구분하는 문제이다. 특히 장애 투명성과 병행(동시성) 투명성을 혼동하지 않는 것이 핵심이다.

핵심 개념

단편화 투명성=릴레이션이 어떤 기준으로 여러 조각(단편)으로 나뉘어 저장되는지를 사용자가 몰라도 되는 성질. 중복(복제) 투명성=동일 데이터의 사본이 어디에 몇 개 존재하는지를 사용자가 몰라도 되는 성질. 위치 투명성=데이터(또는 그 단편·사본)가 물리적으로 어느 사이트에 있는지를 사용자가 몰라도 되는 성질. 장애 투명성=일부 사이트나 통신 링크에 장애가 발생해도 트랜잭션의 원자성과 일관성이 깨지지 않도록 시스템이 이를 감추어 처리하는 성질이며, 이는 장애 상황에 한정된 개념이다. 이와 달리 여러 트랜잭션이 동시에 수행될 때 결과의 일관성을 보장하는 것은 병행(동시성) 투명성의 역할이다.

NoteBase

선지별 해설

① O — 단편화 투명성은 사용자가 릴레이션이 수평·수직 등으로 어떻게 단편화되어 저장되는지 알 필요가 없다는 설명으로 정의에 부합한다.
② X — 장애 투명성을 "다수의 트랜잭션이 동시에 수행되는 경우에도 오류 없이 결과의 일관성이 유지되어야 하는 성질"이라고 설명하고 있으나, 이는 장애(failure) 투명성이 아니라 병행(동시성) 투명성에 대한 설명이다. 장애 투명성은 사이트나 링크의 장애가 발생했을 때 트랜잭션의 원자성이 보장되도록 감추는 성질이며, 동시 수행 트랜잭션 간 일관성 유지와는 별개의 개념이므로 틀린 표현이다. 틀린 표현 → 올바른 표현: "장애 투명성은 사이트나 통신 링크에 장애가 발생해도 트랜잭션의 원자성이 유지되도록 하는 성질이다."
③ O — 중복(복제) 투명성은 중복 저장된 데이터가 무엇인지, 어느 위치에 저장되는지를 사용자가 별도로 인지하지 않아도 되는 성질이라는 설명으로 정의에 부합한다.
④ O — 위치 투명성은 사용자나 애플리케이션이 작업 수행을 위해 데이터의 물리적 위치를 알 필요가 없어야 하는 성질이라는 설명으로 정의에 부합한다.

오답 포인트

장애 투명성과 병행(동시성) 투명성은 둘 다 "오류 없는 일관된 결과"라는 말이 등장해 헷갈리기 쉬우나, 장애 투명성은 장애 발생이라는 상황을 전제로 한 개념이고 병행 투명성은 여러 트랜잭션이 동시에 실행되는 상황을 전제로 한 개념이라는 점에서 구분된다. 단편화·중복·위치 투명성은 모두 "사용자가 무엇을 몰라도 되는가"를 묻는 형태로 서로 대상만 다르므로, 각 선지의 대상 명사(단편화 방식/중복 사본 위치/물리적 위치)를 정확히 짚어 읽어야 한다.

다시 나오면 이렇게 판단

선지에 '장애(failure)'라는 용어가 나오면 그 뒤에 이어지는 설명이 '장애 발생 시에도 원자성·일관성 유지'인지, 아니면 '동시 수행 트랜잭션 간 일관성 유지'인지부터 구분한다. 후자라면 이는 장애 투명성이 아니라 병행 투명성의 정의이므로 옳지 않은 설명으로 판단한다.

암기 포인트

단편화 투명성=조각내는 방식을 숨김, 중복 투명성=복사본의 존재·위치를 숨김, 위치 투명성=물리적 저장 위치를 숨김, 장애 투명성=장애 발생 시 원자성 보장, 병행 투명성=동시 실행 트랜잭션 간 일관성 보장으로 다섯 가지를 대상 중심으로 짝지어 외워 둔다.

NoteBase

문 20. 다음 사원 테이블에서 차수와 카디날리티 값을 바르게 연결한 것은?

사원			
사번	이름	직급	부서
108002	김진수	부장	인사
105123	이수진	대리	연구
128372	박지훈	과장	영업

① 차수: 3, 카디날리티: 3

② 차수: 3, 카디날리티: 4

③ 차수: 4, 카디날리티: 3

④ 차수: 4, 카디날리티: 4

정답 3번

출제 포인트

릴레이션(테이블)의 차수(degree)와 카디날리티(cardinality) 개념을 구분해 실제 테이블에 적용할 수 있는지를 묻는다. 차수는 하나의 릴레이션이 가진 속성(열)의 개수이고, 카디날리티는 그 릴레이션이 가진 튜플(행)의 개수이다.

핵심 개념

차수(degree) = 애트리뷰트(열)의 수. 카디날리티(cardinality) = 튜플(행)의 수. 사원 테이블은 사번, 이름, 직급, 부서 4개의 애트리뷰트로 구성되어 차수는 4이다. 데이터는 108002/김진수, 105123/이수진, 128372/박지훈 3개의 튜플로 구성되어 카디날리티는 3이다.

NoteBase

선지별 해설

① 차수 3, 카디날리티 3 → X: 차수를 3으로 본 것이 틀렸다. 애트리뷰트는 사번, 이름, 직급, 부서 4개이므로 차수는 4이다.
② 차수 3, 카디날리티 4 → X: 차수와 카디날리티 값이 모두 실제 테이블 구성과 맞지 않는다(애트리뷰트 4개, 튜플 3개).
③ 차수 4, 카디날리티 3 → O: 애트리뷰트(열) 4개(사번, 이름, 직급, 부서)이므로 차수 4, 튜플(행) 3개(김진수, 이수진, 박지훈)이므로 카디날리티 3이다.
④ 차수 4, 카디날리티 4 → X: 차수 4는 맞지만 카디날리티를 4로 본 것이 틀렸다. 튜플은 3개뿐이다.

오답 포인트

차수와 카디날리티를 서로 반대로 외우거나, 열과 행을 세는 기준을 혼동해 틀리는 경우가 많다. 표를 볼 때 세로로 나열된 항목(속성 이름이 적힌 머리글 칸의 개수)을 세면 차수, 실제 데이터가 채워진 가로 줄의 개수를 세면 카디날리티라고 순서대로 확인한다.

다시 나오면 이렇게 판단

테이블이 주어지면 먼저 머리글에 있는 속성명의 개수를 세어 차수를 정하고, 그다음 데이터 행의 개수를 세어 카디날리티를 정한다. 두 값을 헷갈리지 않도록 항상 '차수는 열, 카디날리티는 행' 순서로 확인한다.

암기 포인트

차수(degree)=애트리뷰트(열) 개수, 카디날리티(cardinality)=튜플(행) 개수. '차수는 세로 항목 수, 카디날리티는 데이터 줄 수'로 기억해 둔다.

NoteBase

문 21. 데이터베이스 암호화 방식 중 API 방식에 대한 설명으로 옳지 않은 것은?

① API 방식은 암·복호화 모듈을 어플리케이션 서버에 설치한다.

② 플러그인 방식보다 암·복호화 속도가 상대적으로 빠르다.

③ 플러그인 방식보다 응용 프로그램의 수정이 필요 없어 사용이 편리하다.

④ 플러그인 방식보다 데이터베이스 서버의 부하가 거의 발생하지 않는다.

정답 3번

출제 포인트

데이터베이스 암호화 방식은 크게 API 방식과 플러그인(Plug-in) 방식으로 나뉜다. 이 문항은 API 방식의 특징을 플러그인 방식과 비교하여 정확히 알고 있는지를 묻는다. API 방식은 암·복호화 모듈을 애플리케이션 서버(응용 프로그램 영역)에 설치하고 API 호출을 통해 암·복호화를 수행하는 방식이고, 플러그인 방식은 데이터베이스 서버 내부에 암·복호화 모듈을 설치하는 방식이다. 두 방식은 설치 위치가 다르기 때문에 성능, 서버 부하, 응용 프로그램 수정 필요성 등에서 서로 반대되는 장단점을 가진다.

핵심 개념

API 방식: 암·복호화 모듈을 애플리케이션 서버에 설치하고 API 호출로 암·복호화를 수행한다. 데이터베이스 서버에는 암호화 모듈이 없으므로 데이터베이스 서버의 부하가 거의 발생하지 않고, 애플리케이션 서버에서 직접 처리하므로 암·복호화 속도가 플러그인 방식보다 상대적으로 빠르다. 다만 API를 호출하도록 응용 프로그램의 소스 코드를 수정해야 하므로 도입과 유지보수에 응용 프로그램 수정이 필요하다.
플러그인 방식: 암·복호화 모듈을 데이터베이스 서버에 설치하여 응용 프로그램의 소스 코드 수정 없이 적용할 수 있어 도입이 상대적으로 간편하다. 다만 암·복호화 처리를 데이터베이스 서버가 직접 수행하므로 데이터베이스 서버에 부하가 발생하고 속도도 API 방식보다 상대적으로 느리다.

선지별 해설

① API 방식은 암·복호화 모듈을 어플리케이션 서버에 설치한다. → O. API 방식의 정의 그대로이다. 플러그인 방식이 데이터베이스 서버에 모듈을 설치하는 것과 대비된다.
② 플러그인 방식보다 암·복호화 속도가 상대적으로 빠르다. → O. API 방식은 애플리케이션 서버에서 직접 암·복호화를 처리하므로, 데이터베이스 서버가 처리를 전담하는 플러그인 방식보다 속도가 빠르다.
③ 플러그인 방식보다 응용 프로그램의 수정이 필요 없어 사용이 편리하다. → X. 응용 프로그램의 소스 코드 수정 없이 사용이 편리한 쪽은 플러그인 방식이다. API 방식은 오히려 API를 호출하도록 응용 프로그램을 수정해야 하므로, '플러그인 방식보다 수정이 필요 없다'는 설명은 사실과 반대이다. → 올바른 표현: API 방식은 플러그인 방식보다 응용 프로그램의 수정이 더 필요하며, 플러그인 방식이 응용 프로그램 수정 없이 사용이 편리하다.
④ 플러그인 방식보다 데이터베이스 서버의 부하가 거의 발생하지 않는다. → O. API 방식은 암·복호화를 애플리케이션 서버에서 처리하므로 데이터베이스 서버 자체에는 부하가 거의 발생하지 않는다.

NoteBase

오답 포인트

API 방식과 플러그인 방식의 장단점을 짝지어 외우지 않으면 ②·④처럼 옳은 설명을 옳지 않다고 착각하기 쉽다. 두 방식은 암·복호화 모듈의 설치 위치(애플리케이션 서버 vs 데이터베이스 서버)가 다르다는 한 가지 기준에서 속도, 서버 부하, 응용 프로그램 수정 필요성이 모두 반대 방향으로 갈린다는 점을 함께 기억해야 한다.

다시 나오면 이렇게 판단

선지에 '~보다 빠르다/느리다', '~이 필요 없다/필요하다', '부하가 발생한다/발생하지 않는다'처럼 두 방식을 비교하는 표현이 나오면, 먼저 암·복호화 모듈이 애플리케이션 서버에 있는지(API) 데이터베이스 서버에 있는지(플러그인)를 확인한다. 모듈이 설치된 쪽 서버가 처리 부담과 속도 저하를 떠안고, 모듈이 설치되지 않은 응용 프로그램 쪽은 소스 코드 수정이 필요해진다는 대응 관계로 각 선지를 검증한다.

암기 포인트

API 방식 = 모듈이 애플리케이션 서버에 위치 → 속도 빠름, DB 서버 부하 적음, 응용 프로그램 수정 필요.
플러그인 방식 = 모듈이 데이터베이스 서버에 위치 → 속도 느림, DB 서버 부하 발생, 응용 프로그램 수정 불필요(도입 간편).

NoteBase

문 22. 여러 트랜잭션을 병행수행(concurrency)하였을 때, 발생할 수 있는 문제 가운데 갱신분실(lost update)에 대한 설명으로 옳은 것은?

① 두 개의 트랜잭션은 서로 상대 트랜잭션이 완료되기만을 기다리고 있는 상태로, 두 트랜잭션은 영원히 완료될 수 없는 문제이다.

② 장애가 발생한 트랜잭션에 대한 롤백(rollback)이 수행되기 전에, 변경된 데이터를 가져가 사용하는 다른 트랜잭션의 수행이 완료되어 롤백이 실행될 수 없는 문제이다.

③ 하나의 트랜잭션이 수행한 데이터 변경 연산의 결과를 다른 트랜잭션이 덮어써서 기존 트랜잭션의 변경 연산 결과가 손실되는 것이다.

④ 하나의 트랜잭션이 여러 개의 데이터에 대해 변경 연산을 실행했을 때, 일관성이 유지되지 않는 데이터베이스로부터 데이터를 가져와 연산을 실행함으로 발생될 수 있다.

정답 3번

출제 포인트

병행수행(concurrency control)에서 발생하는 여러 이상 현상(갱신분실, 교착상태, 비완료 의존성, 비일관성 분석) 가운데 갱신분실(lost update)의 정의를 다른 이상 현상과 구별할 수 있는지를 묻는다. 네 선지가 각각 서로 다른 병행제어 문제 상황을 설명하고 있으므로, 그중 갱신분실에 해당하는 설명 하나만 골라야 한다.

핵심 개념

갱신분실(lost update)은 두 개 이상의 트랜잭션이 같은 데이터를 동시에 읽고 각자 변경 연산을 수행할 때, 나중에 완료된 트랜잭션의 갱신 결과가 먼저 완료된 트랜잭션의 갱신 결과를 덮어써서 앞선 갱신이 반영되지 않은 채 사라지는 현상이다. 즉 하나의 갱신 결과가 다른 트랜잭션의 갱신에 의해 무효화되어 손실되는 것이 핵심이다. 이는 교착상태(상호 대기), 비완료 의존성(롤백 전파 실패), 비일관성 분석(읽는 도중 다른 트랜잭션이 갱신하여 일관성 없는 값을 읽는 문제)과는 서로 다른 현상이다.

NoteBase

선지별 해설

① X — 두 트랜잭션이 서로 상대의 완료만을 무한정 기다려 어느 쪽도 끝날 수 없는 상태는 교착상태(deadlock)에 대한 설명이며 갱신분실이 아니다.
② X — 장애 트랜잭션의 롤백이 수행되기 전에 그 트랜잭션이 변경한 데이터를 다른 트랜잭션이 이미 읽어 사용해버려 롤백을 실행할 수 없게 되는 문제는 비완료(연쇄) 의존성(uncommitted dependency, cascading rollback) 문제이며 갱신분실이 아니다.
③ O — 하나의 트랜잭션이 수행한 데이터 변경 연산의 결과를 다른 트랜잭션이 덮어써서 기존 트랜잭션의 변경 결과가 손실되는 것은 갱신분실(lost update)의 정의 그대로이다.
④ X — 하나의 트랜잭션이 여러 데이터에 대해 연산을 수행하는 도중 일관성이 유지되지 않은 상태의 데이터베이스로부터 값을 읽어 연산에 사용하게 되는 문제는 비일관성 분석(inconsistent analysis) 문제이며 갱신분실이 아니다.

오답 포인트

네 선지 모두 병행수행 시 발생하는 실제 이상 현상을 설명하고 있어 언뜻 모두 그럴듯해 보이지만, 문제가 요구하는 것은 그중 정확히 갱신분실에 해당하는 설명 하나이다. '상대를 기다리며 영원히 끝나지 않는다'는 교착상태, '롤백이 실행될 수 없다'는 비완료 의존성, '일관성 없는 값을 읽는다'는 비일관성 분석으로 각각 갱신분실과 혼동하기 쉬우므로, '덮어써서 갱신 결과 자체가 사라진다'는 표현이 있는 선지만 갱신분실임을 기준으로 구별해야 한다.

다시 나오면 이렇게 판단

선지에 '덮어써서(overwrite)', '변경 연산 결과가 손실된다'는 표현이 있으면 갱신분실이고, '서로 기다린다'는 교착상태, '롤백이 실행될 수 없다'는 비완료 의존성, '일관성이 유지되지 않은 데이터를 읽는다'는 비일관성 분석으로 즉시 매칭한다.

암기 포인트

갱신분실=덮어쓰기로 갱신 결과 손실, 교착상태=상호 대기로 영원히 미완료, 비완료 의존성=롤백 전 데이터 사용으로 롤백 불가, 비일관성 분석=일관성 없는 상태의 데이터 읽기로 4가지 병행수행 이상 현상을 키워드로 짝지어 기억해 둔다.

NoteBase

문 23. 다중 데이터베이스 시스템에서 2단계 완료 프로토콜(two-phase commit protocol) 기법에 대한 설명으로 옳은 것은?

① 1단계 수행 중 고장이 발생하면, 해당 트랜잭션은 회복되거나 완료된다.

② 트랜잭션 관리자(Transaction Manager: TM)는 그 지역 사이트에서 시작된 트랜잭션의 실행을 조정하는 역할을 한다.

③ 1단계에서는 조정자가 메시지[prepare T]에 대한 응답을 모든 사이트로부터 받으면 트랜잭션 T가 완료(commit)될 수 있는지 결정할 수 있다.

④ 2단계에서 조정자가 모든 참여 사이트로부터 메시지[ready T]를 받았다면 트랜잭션 T는 완료되고 그렇지 않으면 취소된다.

정답 4번

출제 포인트

2단계 완료 프로토콜(two-phase commit protocol)이 준비 단계(1단계)와 완료 단계(2단계)에서 각각 무엇을 주고받고 무엇을 결정하는지, 그리고 장애 발생 시 트랜잭션이 어떻게 처리되는지를 정확히 구분해서 알고 있는지를 묻는다.

핵심 개념

2단계 완료 프로토콜은 여러 사이트에 걸친 분산 트랜잭션을 원자적으로 완료시키기 위한 기법이다. 트랜잭션을 시작한 사이트의 트랜잭션 관리자가 조정자(coordinator) 역할을 맡고, 그 트랜잭션에 관여하는 다른 사이트들은 참여자(participant)가 된다. 1단계(준비 단계)에서 조정자는 모든 참여 사이트에 [prepare T] 메시지를 보내고, 각 참여 사이트는 자신의 하위 트랜잭션을 로그에 기록한 뒤 커밋할 준비가 되면 [ready T], 그렇지 않으면 [abort T] 메시지로 응답한다. 2단계(완료 단계)에서 조정자는 모든 참여 사이트로부터 [ready T]를 받았는지를 확인해, 전부 받았다면 트랜잭션 T의 전역 커밋을 결정해 [commit T]를 전파하고, 하나라도 [abort T]를 보냈거나 응답이 없으면 전역 취소를 결정해 [abort T]를 전파한다.

NoteBase

선지별 해설

① X — 1단계 수행 중 어느 사이트에서 고장이 발생하면 그 사이트는 [ready T]를 보낼 수 없으므로, 조정자는 해당 트랜잭션을 완료(commit)시키지 않고 전역적으로 취소(abort)한다. "회복되거나 완료된다"는 설명은 틀렸다. 1단계 장애는 항상 취소로 귀결된다.
② X — 지역 사이트에서 시작된 트랜잭션의 실행을 조정하는 것은 그 사이트의 트랜잭션 관리자가 조정자 역할을 맡아 수행하는 것이지, 트랜잭션 관리자 일반이 갖는 고유 역할이 아니다. 참여자로만 관여하는 사이트의 트랜잭션 관리자는 자신이 시작하지 않은 트랜잭션에 대해서는 조정 역할을 하지 않고 조정자의 지시(prepare/commit/abort)에 따라 지역 하위 트랜잭션만 처리한다. 조정 역할과 트랜잭션 관리자의 역할을 동일시한 설명은 부정확하다.
③ X — 1단계는 조정자가 [prepare T]를 보내고 각 사이트의 응답([ready T] 또는 [abort T])을 모으는 단계일 뿐이다. 모든 응답을 받은 뒤 트랜잭션 T를 커밋할 수 있는지 실제로 결정하는 것은 2단계(완료 단계)의 몫이다. 결정 자체를 1단계의 일로 서술한 것이 틀렸다.
④ O — 2단계에서 조정자가 모든 참여 사이트로부터 [ready T]를 받았다면 트랜잭션 T의 전역 커밋을 결정해 완료시키고, 하나라도 [ready T]를 보내지 않았다면(즉 [abort T]를 보냈거나 응답이 없다면) 전역 취소를 결정해 트랜잭션을 취소한다.

오답 포인트

1단계와 2단계의 역할을 뒤바꿔 서술하는 오답이 반복된다. 1단계는 "응답을 모으는 단계"이고, 모은 응답을 근거로 실제 커밋 여부를 결정하고 그 결과를 전파하는 것은 2단계라는 순서를 혼동하지 않아야 한다. 또한 조정자와 트랜잭션 관리자를 같은 개념으로 뭉뚱그리는 설명도 주의해야 한다. 조정자는 트랜잭션을 시작한 사이트의 트랜잭션 관리자가 그 트랜잭션에 한해 맡는 역할일 뿐이다.

다시 나오면 이렇게 판단

선지에 "1단계"와 "2단계"라는 단어가 등장하면 먼저 각 단계에서 실제로 무슨 일이 일어나는지(1단계=prepare 전송과 ready/abort 응답 수집, 2단계=수집된 응답을 근거로 한 커밋/취소 결정과 전파)를 표시해 놓고 선지와 대조한다. 장애 발생 시나리오가 나오면 "1단계 장애는 곧 취소"라는 원칙을 먼저 떠올린다.

암기 포인트

1단계(준비, voting): 조정자 prepare T 전송 → 참여자 ready T/abort T 응답. 2단계(완료, decision): 전원 ready T면 commit T 결정·전파, 하나라도 abort면 abort T 결정·전파. 1단계 중 장애는 전역 취소로 귀결된다.

NoteBase

문 24. 데이터베이스 재해 복구(disaster recovery) 기술에 대한 설명으로 옳은 것은?

① 재해적 실패가 발생하면 가장 최근의 백업 사본이 디스크에서 백업장치로 적재된다.

② 은행, 보험, 주식 등과 같은 중요한 응용에서는 데이터 전체를 주기적으로 백업하여 안전한 장소에 보관한다.

③ 시스템 로그를 백업하면 사용자는 마지막 데이터베이스 백업 이후에 수행한 모든 트랜잭션을 잃게 된다.

④ 백업된 시스템 로그에 기록되어 있는 모든 완료된 트랜잭션의 실행결과는 데이터베이스를 undo하는 데 사용될 수 있다.

정답 2번

출제 포인트

재해 복구(disaster recovery)를 위한 백업 전략의 방향과 대상을 정확히 이해하는지를 묻는다. 데이터 백업의 목적과 저장 위치, 시스템 로그(로그 백업)가 트랜잭션 복구에서 하는 역할(redo)과 undo의 차이를 구분하는 것이 핵심이다.

핵심 개념

재해 복구는 화재·지진·정전 등으로 데이터베이스가 저장된 디스크 자체가 손상되는 상황에 대비하는 것이다. 이때 복구는 안전한 장소에 보관된 백업 사본을 백업장치에서 디스크로 적재하는 방향으로 이루어진다(디스크에서 백업장치로 가는 것이 아니다). 시스템 로그(트랜잭션 로그)를 함께 백업해 두면 마지막 전체 백업 이후에 완료된 트랜잭션들도 로그에 기록된 연산을 다시 실행(redo)하여 복구할 수 있으므로, 로그를 잃지 않는 한 그 트랜잭션들이 사라지는 것은 아니다. undo는 완료되지 않은(커밋되지 않은) 트랜잭션이 이미 반영한 변경을 되돌리는 연산이므로, 완료된(커밋된) 트랜잭션의 로그 기록은 undo가 아니라 redo에 사용된다.

선지별 해설

① X: 재해적 실패가 발생하면 복구는 안전한 장소에 보관된 가장 최근 백업 사본을 백업장치에서 디스크로 적재하는 방향으로 진행된다. 지문은 방향을 반대로 서술하여(디스크에서 백업장치로) 틀렸다.

② O: 은행, 보험, 주식 등 데이터 손실이 큰 피해로 이어지는 중요한 응용에서는 데이터베이스 전체를 주기적으로 백업하여 재해에 대비해 안전한 장소(원격지 등)에 보관하는 것이 재해 복구의 기본 전략이다. 옳은 설명이다.

③ X: 시스템 로그를 함께 백업해 두면, 마지막 전체 백업 이후에 수행되어 완료된 트랜잭션이라도 로그에 기록된 연산 내용을 다시 실행(redo)하여 복구할 수 있다. 오히려 로그 백업이 없으면 그 트랜잭션들을 잃게 되는 것이므로, 지문은 로그 백업의 효과를 정반대로 서술하여 틀렸다.

④ X: 백업된 시스템 로그에 기록된 완료(커밋)된 트랜잭션의 실행결과는 데이터베이스를 다시 그 상태로 만드는 redo에 사용되는 것이지 undo에 사용되는 것이 아니다. undo는 완료되지 않은 트랜잭션이 반영한 변경을 되돌릴 때 사용하는 연산이므로, 지문은 redo와 undo의 역할을 바꾸어 서술하여 틀렸다.

NoteBase

오답 포인트

①과 ③, ④는 모두 방향이나 역할을 반대로 뒤집어 놓은 함정이다. ①은 백업 적재 방향(백업장치→디스크가 맞음)을, ③은 로그 백업의 효과(트랜잭션을 잃는 것이 아니라 복구할 수 있음)를, ④는 완료된 트랜잭션 로그의 용도(undo가 아니라 redo)를 각각 반대로 서술했다. '백업/로그'라는 단어만 보고 넘기지 말고 방향과 대상(완료/미완료, undo/redo)을 함께 확인해야 한다.

다시 나오면 이렇게 판단

재해 복구 지문이 나오면 먼저 복구 방향(백업장치→디스크)이 맞는지 확인한다. 다음으로 로그 백업이 언급되면 완료된 트랜잭션은 redo, 미완료(취소된) 트랜잭션은 undo라는 대응을 기준으로 각 선지의 방향이 뒤집혀 있지 않은지 점검한다.

암기 포인트

재해 복구 = 백업장치에서 디스크로 복원. 완료된 트랜잭션의 로그 → redo(재실행), 미완료 트랜잭션의 로그 → undo(취소). 로그 백업은 트랜잭션을 잃지 않게 해 주는 장치이지 잃게 만드는 장치가 아니다.

NoteBase

문 25. 릴레이션 R에 K 단계 합병(k-way merge) 정렬을 수행하고자 한다. 정렬이 완료될 때까지 최소 몇 번의 디스크 접근(읽기 + 쓰기 횟수)이 필요한가? (단, 문제에서 언급되지 않은 다른 기법은 고려하지 않는다)

각 디스크 블록에는 레코드 200개씩 저장되어 있음
주 기억장치의 최대 사용가능한 크기: 레코드 1,000개 크기
전체 정렬해야 할 레코드의 개수: 3,200개

① 32회
② 64회
③ 96회
④ 128회

정답 3번

출제 포인트

외부 정렬(external sort-merge)에서 총 디스크 접근 횟수를 구하는 문제이다. 전체 블록 수와 주기억장치에 올릴 수 있는 블록 수를 구해 초기 정렬 실행(run) 개수를 정한 뒤, 그 run들을 하나로 합치는 합병 단계 수를 세어 합산한다. 한 단계(pass)마다 전체 블록을 한 번 읽고 한 번 쓰므로 접근 횟수는 (전체 단계 수)×(전체 블록 수)×2이다. 이 문항은 합병을 한 번에 몇 개씩 묶어 진행하는지(합병 차수)에 따라 답이 갈리며, 공식 정답은 2개씩 묶는 2-way 합병을 전제로 한 값이다.

조건 정리

전체 레코드 3,200개, 블록당 200개이므로 전체 블록 수는 $3200/200 = 16$블록이다. 주기억장치에는 최대 1,000개, 곧 $1000/200 = 5$블록을 올릴 수 있다. 즉 한 번에 정렬할 수 있는 블록 수는 5블록이며, 16블록을 5블록씩 나누면 초기 run은 $\lceil 16/5 \rceil = 4$개(5, 5, 5, 1블록)가 만들어진다.

사용할 공식

초기 run 생성(1단계)에서 16블록을 읽고 16블록을 써서 4개의 정렬된 run을 만든다(디스크 접근 32회). 이후 합병 단계 수는 한 번에 몇 개의 run을 묶느냐(합병 차수)에 따라 달라진다. 버퍼를 최대한 활용하는 표준 다중 합병에서는 입력 버퍼를 (주기억 블록 수−1)개, 곧 $5 - 1 = 4$개까지 써서 4개 run을 단 1단계에 1개로 합칠 수 있다(4-way). 반면 2개씩 묶는 2-way 합병에서는 4개→2개→1개로 2단계가 필요하다. 어느 경우든 합병 단계마다 전체 16블록을 한 번 읽고 한 번 써서 32회가 든다.

NoteBase

계산 과정

초기 run 생성은 16블록 읽기 + 16블록 쓰기 = 32회이다. 이 문항의 공식 정답은 2-way 합병을 전제로 하므로, 1차 합병(4개→2개) 32회와 2차 합병(2개→1개) 32회가 더해져 총 32 + 32 + 32 = 96회가 된다. 참고로 버퍼를 최대한 활용하는 표준 4-way 합병으로 보면 합병이 1단계로 끝나 32 + 32 = 64회가 된다.

선지 판단

① 32회(X): 초기 run 생성 단계만 계산하고 합병 과정을 통째로 빠뜨린 값이라 옳지 않다.
② 64회(X, 관례에 따라 성립): 주기억 버퍼를 최대한 활용하는 표준 (블록 수−1)-way, 곧 4-way 합병으로 계산한 값이다(4개 run을 1단계에 합병, 32+32). 알고리즘 관점에서는 타당한 값이지만, 이 문항은 2-way 합병을 전제로 출제되어 공식 정답에서는 제외되었다.
③ 96회(O): 2개씩 합병하는 2-way 합병으로 계산한 값이다. 초기 run 생성 32회 + 합병 2단계(각 32회)로 96회이며, 출제가 전제한 공식 정답이다.
④ 128회(X): 합병을 3단계로 잡은 경우의 값으로, 4개 run은 2개씩 합쳐도 2단계면 충분하므로 과도하게 계산한 오답이다.

오답 포인트

이 문항의 쟁점은 합병 차수이다. 주기억 5블록이면 표준 외부 정렬은 (5−1)=4-way 합병이 가능해 4개 run을 1단계에 합쳐 총 64회(②)가 되지만, 공식 정답은 2개씩 묶는 2-way 합병을 가정해 합병 2단계·총 96회(③)로 처리했다. 두 값 모두 계산 절차 자체는 옳으므로, '합병을 몇 개씩 하는가'라는 전제를 먼저 확인해야 한다. 2-way를 전제하면 96, (M−1)-way를 전제하면 64가 답이 된다.

다시 나오면 이렇게 판단

전체 블록 수(레코드÷블록크기)와 한 번에 올릴 수 있는 블록 수(주기억÷블록크기)를 구해 초기 run 개수를 정한다. 그다음 합병 차수를 확인한다. 2개씩 묶는 2-way를 전제하면 run이 1개가 될 때까지 _2(run 수)단계, 버퍼를 최대 활용하는 (M−1)-way를 전제하면 _M-1(run 수)단계로 센다. 총 디스크 접근 = (1 + 합병 단계 수)×(전체 블록 수)×2이다.

암기 포인트

외부 정렬 디스크 접근 = 2 × b × (전체 단계 수), 전체 단계 수 = 1(초기 run 생성) + 합병 단계 수. 합병 단계 수는 2-way면 _2(run 수), 표준 (M−1)-way면 _M-1(run 수)이다. run 4개인 이 문항은 2-way로 2단계면 96, 4-way로 1단계면 64.

NoteBase