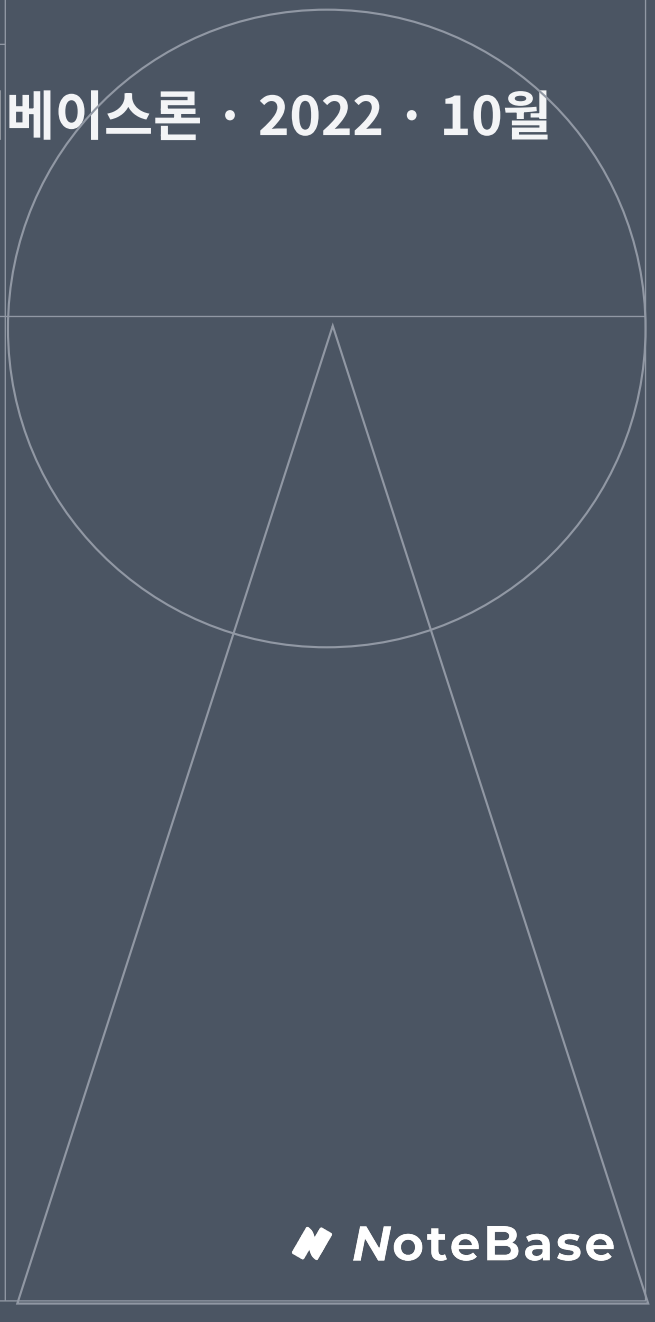
기출OX 해설집

공무원/서울시/7급 · 데이터베이스론 · 2022 · 10월



NoteBase

기출OX 해설집

공무원/서울시/7급 · 데이터베이스론 · 2022 · 10월

공무원/서울시/7급 · 데이터베이스론 · 2022 · 10월

문 1. 데이터 웨어하우스(data warehouse)에 대한 설명으로 가장 옳지 않은 것은?

① 데이터가 빈번하게 갱신되는 환경에 최적화된 시스템이다.

② ETL(extract, transform, load)은 데이터 수집 및 저장 과정에 사용된다.

③ 데이터를 데이터 큐브(data cube)의 형태로 저장할 수 있다.

④ 스타 스키마(star schema)는 1개의 사실 테이블(fact table)과 여러 개의 차원 테이블(dimension table)들로 구성된다.

정답 1번

출제 포인트

이 문제는 데이터 웨어하우스의 목적과 OLTP 환경의 차이, 그리고 ETL·데이터 큐브·스타 스키마의 기본 개념을 구분할 수 있는지를 묻는다.

핵심 개념

데이터 웨어하우스는 여러 운영 시스템의 데이터를 통합해 분석과 의사결정에 활용하는 저장소이다.
OLTP는 빈번한 삽입·수정·삭제가 일어나는 거래 처리 중심 시스템이다.
ETL은 원천 데이터를 추출하고 변환한 뒤 웨어하우스로 적재하는 과정이다.
데이터 큐브는 다차원 분석을 위한 OLAP 구조이고, 스타 스키마는 1개의 사실 테이블과 여러 차원 테이블로 이루어진 전형적인 모델이다.

선지별 해설

① X: 데이터 웨어하우스는 빈번한 갱신 처리보다 분석 질의와 이력 데이터 조회에 최적화된 시스템이다. 틀린 표현은 "빈번하게 갱신되는 환경에 최적화"이다. 올바르게는 "주기적으로 적재된 통합 데이터를 분석하는 환경에 최적화"이다.
② O: ETL은 원천 시스템에서 데이터를 추출하고 필요한 형태로 변환한 뒤 저장소에 적재하는 과정이므로 데이터 웨어하우스의 데이터 수집 및 저장 과정에 사용된다.
③ O: 데이터 웨어하우스나 OLAP 환경에서는 다차원 분석을 위해 데이터를 데이터 큐브 형태로 구성할 수 있다.
④ O: 스타 스키마는 중앙의 사실 테이블과 이를 둘러싼 여러 차원 테이블로 구성되는 전형적인 데이터 웨어하우스 스키마이다.

NoteBase

오답 포인트

데이터 웨어하우스와 OLTP의 역할을 뒤바꾸면 틀리기 쉽다. "빈번한 갱신"이라는 표현은 거래 처리 시스템의 특징인데, 이를 데이터 웨어하우스의 특징처럼 바꿔 놓은 것이 함정이다.

다시 나오면 이렇게 판단

먼저 그 시스템의 목적이 거래 처리인지 분석 지원인지부터 구분하면 된다. 실시간 수정과 짧은 트랜잭션이 핵심이면 OLTP이고, 통합·이력 데이터 기반의 조회와 분석이 핵심이면 데이터 웨어하우스이다.

암기 포인트

OLTP는 자주 갱신하고, 데이터 웨어하우스는 모아 두고 분석한다.

NoteBase

문 2. <보기>는 데이터베이스 관리 시스템(DBMS)의 주요 구성 요소들에 대한 설명이다. ㉠~㉣에 들어갈 요소를 순서대로 옳게 짝지은 것은?

<보기>
㉠는 디스크에 저장되어 있는 데이터에 대한 접근을 제어한다.
㉡는 데이터 정의어(DDL)로 작성된 데이터베이스 스키마를 메타데이터로 처리하여 시스템 카탈로그에 저장한다.
㉢는 대화식으로 입력된 데이터 조작어(DML)를 파싱하여 컴파일하고, 데이터베이스에 접근하는 최적의 코드를 생성한다.
㉣는 호스트 프로그램 내에 삽입된 데이터 조작어(DML)를 추출하고 프로시저 호출로 대체시킨다.

① ㉠: 예비컴파일러(precompiler), ㉡: 질의처리기(query processor), ㉢: 데이터 정의어(DDL) 컴파일러, ㉣: 트랜잭션관리자

② ㉠: 예비컴파일러(precompiler), ㉡: 질의처리기(query processor), ㉢: 트랜잭션관리자, ㉣: 런타임 데이터베이스처리기

③ ㉠: 저장데이터관리자, ㉡: 데이터 정의어(DDL) 컴파일러, ㉢: 질의처리기(query processor), ㉣: 예비컴파일러(precompiler)

④ ㉠: 저장데이터관리자, ㉡: 트랜잭션관리자, ㉢: 데이터 정의어(DDL) 컴파일러, ㉣: 예비컴파일러(precompiler)

정답 3번

출제 포인트

이 문제는 DBMS 주요 구성 요소를 기능 설명과 정확히 대응시킬 수 있는지를 묻는다.

핵심 개념

저장데이터관리자: 디스크에 저장된 실제 데이터와 접근 경로를 관리하는 구성 요소이다. 키워드: 물리 저장, 접근 제어
DDL 컴파일러: DDL로 기술된 스키마를 메타데이터로 변환하여 시스템 카탈로그에 기록하는 구성 요소이다. 키워드: 스키마, 카탈로그
질의처리기: 대화식 DML을 파싱하고 최적화하여 실행 가능한 접근 코드를 만드는 구성 요소이다. 키워드: 파싱, 최적화
예비컴파일러: 호스트 프로그램 안에 삽입된 DML을 추출하여 데이터베이스 호출문으로 바꾸는 구성 요소이다. 키워드: 내장 DML, 추출

NoteBase

선지별 해설

1번: X. ㄱ의 '예비컴파일러'가 틀린 표현이다. 디스크에 저장된 데이터 접근 제어는 저장데이터관리자의 기능이다. ㄴ도 질의처리기가 아니라 DDL 컴파일러가 맞고, ㄷ은 DDL 컴파일러가 아니라 질의처리기이며, ㄹ은 트랜잭션관리자가 아니라 예비컴파일러가 맞다.
2번: X. ㄱ의 '예비컴파일러', ㄴ의 '질의처리기', ㄷ의 '트랜잭션관리자', ㄹ의 '런타임 데이터베이스처리기'가 모두 보기 설명과 맞지 않는다. 올바르게는 ㄱ 저장데이터관리자, ㄴ DDL 컴파일러, ㄷ 질의처리기, ㄹ 예비컴파일러이다. 런타임 데이터베이스처리기는 실행 단계의 저수준 연산 담당이지 내장 DML 추출 담당이 아니다.
3번: O. ㄱ은 저장데이터관리자, ㄴ은 DDL 컴파일러, ㄷ은 질의처리기, ㄹ은 예비컴파일러로 각각의 기능 설명과 정확히 대응한다.
4번: X. ㄱ과 ㄹ은 맞지만 ㄴ의 '트랜잭션관리자'와 ㄷ의 '데이터 정의어(DDL) 컴파일러'가 틀린 표현이다. 트랜잭션관리자는 동시성 제어와 회복을 담당하고, 대화식 DML의 파싱과 최적화는 질의처리기의 기능이다. 올바르게는 ㄴ DDL 컴파일러, ㄷ 질의처리기이다.

오답 포인트

DDL 컴파일러, 질의처리기, 예비컴파일러의 역할을 모두 '질의를 처리하는 모듈'로 뭉뚱그리면 틀린다. 특히 '대화식 DML'과 '호스트 프로그램 내 삽입 DML'을 구분하지 못하면 4번처럼 일부만 맞는 선택지에 흔들리기 쉽다.

다시 나오면 이렇게 판단

설명 문장에서 처리 대상이 무엇인지 먼저 본다. 디스크 저장 데이터면 저장데이터관리자, DDL과 시스템 카탈로그면 DDL 컴파일러, 대화식 DML의 파싱·최적화면 질의처리기, 호스트 프로그램 안의 내장 DML 추출이면 예비컴파일러로 대응하면 된다.

암기 포인트

저장 데이터는 저장데이터관리자, DDL은 DDL 컴파일러, 대화식 DML은 질의처리기, 내장 DML은 예비컴파일러이다.

NoteBase

문 3. 릴레이션 R(A, B, C, D)에 대해 <보기 1>이 성립하고, 이를 <보기 2>와 같이 분해하였을 때, (가) 릴레이션 R의 정규형과 분해 시 (나) 정보 손실 여부를 옳게 짝지은 것은?

<보기 1>
함수적 종속성: $AB \rightarrow C, C \rightarrow A, C \rightarrow D$

<보기 2>
분해: R1(A, B, C), R2(C, D)

① (가): 2정규형, (나): 무손실 분해
② (가): 2정규형, (나): 손실 분해
③ (가): 3정규형, (나): 무손실 분해
④ (가): 3정규형, (나): 손실 분해

정답 1번

출제 포인트

이 문제는 기본키가 밑줄 친 AB일 때 2정규형과 3정규형의 구분 기준, 그리고 분해의 무손실 조건을 함께 묻는 문제이다.

핵심 개념

2정규형: 기본키의 일부에만 종속되는 비주요 속성이 없는 정규형이다 / 결정 차이축: 부분 함수 종속 여부 / 키워드: 부분 종속 제거
3정규형: 2정규형이면서 비키 속성을 거치는 이행 함수 종속이 없는 정규형이다 / 결정 차이축: 이행 함수 종속 여부 / 키워드: 이행 종속 제거
무손실 분해: 분해한 릴레이션들을 자연조인했을 때 원래 릴레이션이 정확히 복원되는 분해이다 / 결정 차이축: 공통 속성이 어느 한 분해 릴레이션의 키인지 / 키워드: 공통 속성 키

NoteBase

선지별 해설

① O: AB가 기본키이고 AB의 진부분집합인 A, B만으로는 비주요 속성 C나 D가 결정되지 않으므로 부분 함수 종속은 없다. 따라서 R은 2정규형이다. 다만 C→D가 있어 비키 속성 C를 거쳐 D가 이행 종속되므로 3정규형은 아니다. 또 분해 후 공통 속성은 C이고, C→D이므로 C는 R2(C,D)의 키가 된다. 공통 속성이 한쪽 분해 릴레이션의 키이면 무손실 분해이므로 ①이 옳다.
② X: 틀린 부분은 '(나) 손실 분해'이다. 실제로는 C가 R2(C,D)의 키이므로 자연조인 시 원래 릴레이션이 복원되는 무손실 분해이다.
③ X: 틀린 부분은 '(가) 3정규형'이다. 실제로는 C→D 때문에 비키 속성 사이의 이행 함수 종속이 남아 있으므로 3정규형이 아니다. 올바르게는 2정규형이며, (나)는 무손실 분해이다.
④ X: 틀린 부분은 '(가) 3정규형'과 '(나) 손실 분해'이다. 실제로는 R이 3정규형이 아니라 2정규형이고, 분해도 손실 분해가 아니라 무손실 분해이다.

오답 포인트

C→A가 보여서 바로 2정규형을 부정하거나, C→D를 보고 손실 분해라고 오해하기 쉽다. 이 문항은 기본키의 일부가 비주요 속성을 직접 결정하는지와 공통 속성 C가 분해 릴레이션의 키인지라는 서로 다른 판정 기준을 섞어 놓은 함정이다.

다시 나오면 이렇게 판단

정규형은 먼저 밑줄 친 기본키를 확인한 뒤 부분 함수 종속이 있으면 2정규형 탈락, 이행 함수 종속이 있으면 3정규형 탈락으로 판단한다. 분해는 공통 속성 집합이 한쪽 분해 릴레이션의 키가 되면 무손실이라고 판단한다.

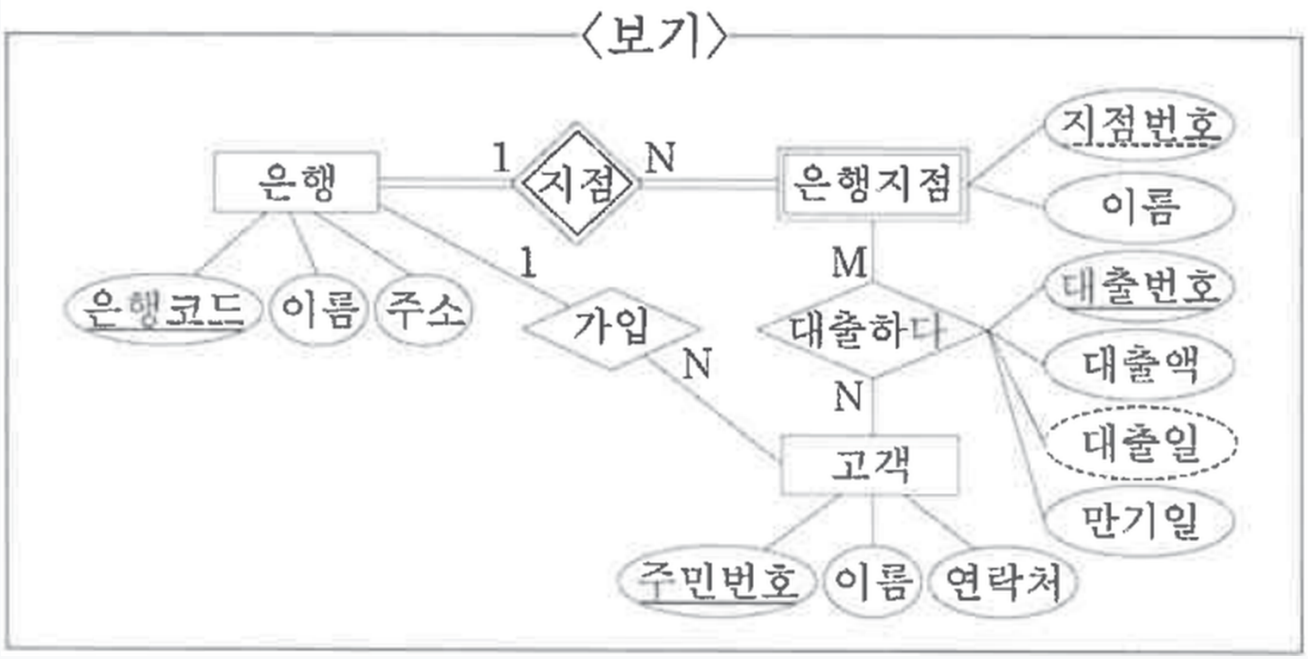
암기 포인트

2정규형은 부분 종속 제거, 3정규형은 이행 종속 제거, 무손실 분해는 공통 속성이 한쪽 릴레이션의 키이면 성립이다.

NoteBase

문 4. <보기>의 대출과 관련하여 작성된 E-R 다이어그램에 대한 설명으로 가장 옳지 않은 것은?

<보기>



① 모든 '고객'은 적어도 하나의 '은행'에 가입되어야 한다.

② E-R 다이어그램에 표현된 관계들의 차수는 모두 2이다.

③ '은행지점' 엔티티를 릴레이션으로 변환하면 속성은 3개 이상이 된다.

④ 모든 '대출'은 1년만 가능하다고 설계되었다면, '대출일'은 '만기일'로부터 유도된 속성이 될 수 있다.

정답 1번

출제 포인트

이 문제는 E-R 다이어그램의 표기에서 카디널리티, 참여 제약, 약한 엔티티, 유도 속성을 정확히 읽어내는지를 묻는 문제이다.

NoteBase

선지별 해설

① X
가입 관계는 '은행' 1 : '고객' N의 이진 관계일 뿐이다. 고객이 가입 관계에 전체참여한다는 이중선이 없으므로 모든 고객이 반드시 어떤 은행에 가입해야 한다고 볼 수는 없다.
틀린 표현은 '모든 고객은 적어도 하나의 은행에 가입되어야 한다'이다. 올바르게는 '고객은 최대 한 은행과 가입 관계를 가질 수 있으나, 반드시 참여해야 한다는 제약은 도식에 나타나지 않는다'이다.

② O
'지점', '가입', '대출하다'는 각각 두 엔티티 집합만 연결한다. 따라서 다이어그램에 표현된 관계들의 차수는 모두 2이다.

③ O
'은행지점'은 약한 엔티티이므로 릴레이션으로 변환할 때 소유자 엔티티 '은행'의 기본키 '은행코드'가 함께 포함된다. 여기에 '지점번호', '이름'이 더해지므로 속성 수는 3개 이상이 된다.

④ O
점선 타원은 유도 속성을 뜻한다. 모든 대출 기간이 1년으로 고정되면 만기일이 주어졌을 때 대출일은 1년을 거슬러 계산할 수 있으므로 '대출일'을 '만기일'로부터 유도되는 속성으로 둘 수 있다.

오답 포인트

①의 '적어도 하나'는 카디널리티가 아니라 참여 제약 문구이다. 1:N만 보고 필수 참여로 오해하면 틀린다. E-R 다이어그램에서 이중선은 전체참여, 1:N은 최대 연결 수를 뜻한다.

다시 나오면 이렇게 판단

E-R 다이어그램 문제는 먼저 표기 의미를 분리해 읽는다. 이중선은 전체참여, 1:N은 최대 개수, 이중 사각형은 약한 엔티티, 점선 타원은 유도 속성으로 판단하면 선지의 옳고 그름이 정리된다.

암기 포인트

참여 제약과 카디널리티는 다르다. 이중선은 반드시 참여, 1:N은 몇 개까지 연결되는지만 뜻한다.

NoteBase

문 5. 테이블 T1(A, B, C)에 대하여 사용자들의 접근 권한을 설정하기 위해 <보기>와 같은 SQL 명령문을 수행하였을 때, 실행할 수 있는 SQL문은? (단, A, B, C는 정수형이다.)

〈보기〉

```
Allen: GRANT SELECT, INSERT ON T1
       TO Brown, Clark WITH GRANT OPTION;
Brown: GRANT SELECT, INSERT ON T1 TO Davis;
Allen: GRANT DELETE ON T1 TO Clark;
Allen: REVOKE INSERT ON T1 FROM Brown
       CASCADE;
Allen: REVOKE GRANT OPTION FOR SELECT
       ON T1 FROM Clark;
```

① Clark: GRANT DELETE ON T1 TO Brown;

② Brown: INSERT INTO T1 VALUES (10, 20, 30);

③ Davis: INSERT INTO T1 VALUES (10, 20, 30);

④ Clark: SELECT * FROM T1 WHERE C < 10;

정답 4번

출제 포인트

WITH GRANT OPTION, REVOKE ... CASCADE, REVOKE GRANT OPTION FOR ...의 효과를 구분하여 현재 각 사용자가 실제로 실행할 수 있는 권한을 판단하는 문제이다.

사례 정리

Allen은 Brown과 Clark에게 SELECT, INSERT를 WITH GRANT OPTION으로 부여하였다.
Brown은 그 권한으로 Davis에게 SELECT, INSERT를 부여하였다.
Allen은 Clark에게 DELETE를 부여하였지만 DELETE에 대한 GRANT OPTION은 주지 않았다.
이후 Brown의 INSERT는 CASCADE로 회수되었고, Clark의 SELECT는 권한 자체가 아니라 GRANT OPTION만 회수되었다.

NoteBase

기출OX 해설집 · 공무원/서울시/7급 · 데이터베이스론 · 2022 · 10월

핵심 개념

WITH GRANT OPTION은 자신이 받은 권한을 다른 사용자에게 다시 부여할 수 있게 하는 권한이다.
REVOKE ... CASCADE는 회수된 권한을 바탕으로 연쇄 부여된 동일 권한도 함께 회수한다.
REVOKE GRANT OPTION FOR SELECT는 SELECT 권한 자체를 없애는 것이 아니라 SELECT를 다시 부여할 수 있는 권한만 없앤다.

적용 기준

각 사용자에 대해 권한 자체와 GRANT OPTION을 분리해서 본다. 또한 REVOKE가 권한 자체를 없애는지, 재부여 권한만 없애는지, CASCADE로 하위 부여까지 끊는지를 순서대로 확인해야 한다.

선지별 해설

① X. Clark은 DELETE 권한은 있지만 DELETE에 대한 GRANT OPTION은 받은 적이 없다. 따라서 DELETE를 Brown에게 다시 부여할 수 없다. 틀린 표현은 'GRANT DELETE ON T1 TO Brown'이고, 올바르게는 'Clark은 DELETE는 수행할 수 있지만 Brown에게 DELETE 권한을 부여할 수는 없다'이다.
② X. Brown의 INSERT 권한은 'REVOKE INSERT ON T1 FROM Brown CASCADE'로 회수되었다. 따라서 Brown은 INSERT문을 실행할 수 없다. 틀린 표현은 'Brown: INSERT INTO T1 VALUES (10, 20, 30);'이고, 올바르게는 'Brown은 더 이상 INSERT를 수행할 수 없다'이다.
③ X. Davis의 INSERT 권한은 Brown이 부여한 것이므로 Brown의 INSERT가 CASCADE로 회수될 때 함께 사라진다. 따라서 Davis도 INSERT문을 실행할 수 없다. 틀린 표현은 'Davis: INSERT INTO T1 VALUES (10, 20, 30);'이고, 올바르게는 'Davis는 INSERT를 수행할 수 없다'이다.
④ O. Allen이 Clark에게서 회수한 것은 'GRANT OPTION FOR SELECT'뿐이므로 Clark의 SELECT 권한 자체는 남아 있다. 따라서 'SELECT * FROM T1 WHERE C < 10;'은 실행할 수 있다.

오답 포인트

④를 틀리는 경우는 'GRANT OPTION 회수'를 'SELECT 권한 회수'와 같은 뜻으로 오해한 경우가 많다. ①도 혼동하기 쉬운데, DELETE 권한을 가진 것과 DELETE를 남에게 다시 줄 수 있는 것은 전혀 다른 권한이다.

다시 나오면 이렇게 판단

권한 문제는 사용자별로 SELECT, INSERT, DELETE와 각 GRANT OPTION의 보유 여부를 따로 적어 본다. 그다음 REVOKE가 나오면 권한 자체 회수인지, GRANT OPTION만 회수인지, CASCADE로 하위 권한까지 끊는지 차례대로 반영하면 된다.

암기 포인트

GRANT OPTION은 남에게 다시 줄 수 있는 권한이고, CASCADE REVOKE는 그 권한으로 뿌린 하위 권한까지 함께 회수한다.

NoteBase

문 6. 로킹(locking)이란 동일한 데이터 항목에 대한 여러 트랜잭션의 임의적인 병행 접근을 방지하는 방법을 말한다. 로킹에 대한 설명으로 가장 옳지 않은 것은?

① 로킹 단위는 레코드, 파일, 데이터베이스 등이 될 수 있다.

② 로킹 단위가 작아질수록 로크 수가 적어 로킹 오버헤드가 감소한다.

③ 교착상태(deadlock)란 두 개 이상의 트랜잭션들이 실행되지 못한 채 다른 트랜잭션이 로크한 항목을 무한정 기다리는 것을 말하며, 교착 상태가 발생하면 일부 트랜잭션을 롤백(rollback)해서 해결할 수 있다.

④ 하나의 스케줄에 있는 모든 트랜잭션들이 2단계 로킹 규약(protocol)을 준수한다면 그 스케줄은 직렬 가능성을 보장한다.

정답 2번

출제 포인트

로킹 단위의 크기에 따른 병행성·오버헤드의 관계와 교착상태, 2단계 로킹 규약의 기본 성질을 묻는 문제이다.

핵심 개념

로킹 단위는 한 번에 잠그는 데이터 범위이다. 단위가 클수록 로크 수와 관리 오버헤드는 줄지만 병행성은 낮아진다.
교착상태는 둘 이상의 트랜잭션이 서로가 점유한 로크의 해제를 기다리며 무한 대기에 빠진 상태이다.
2단계 로킹은 로크를 획득만 하는 확장 단계와 해제만 하는 축소 단계로 나누는 규약이며, 이를 지키면 충돌 직렬 가능성이 보장된다.

선지별 해설

① O: 로킹 단위는 레코드, 블록, 파일, 테이블, 데이터베이스 등 여러 수준으로 설정할 수 있다. 잠금 범위를 어디까지 잡느냐가 로킹 단위이다.
② X: 틀린 부분은 '로킹 단위가 작아질수록 로크 수가 적어 로킹 오버헤드가 감소한다'는 표현이다. 실제로는 로킹 단위가 작아질수록 잠가야 할 항목 수가 많아져 로크 수와 관리 오버헤드는 증가하는 경향이 있다. 대신 동시에 접근할 수 있는 범위가 넓어져 병행성은 높아진다.
③ O: 교착상태는 각 트랜잭션이 상대방이 보유한 로크를 기다려 더 진행하지 못하는 상태이다. 이때 한 트랜잭션을 강제로 종료하거나 롤백하여 점유 로크를 풀면 교착상태를 해소할 수 있다.
④ O: 모든 트랜잭션이 2단계 로킹 규약을 지키면 그 스케줄은 충돌 직렬 가능성을 가진다. 시험에서는 이를 보통 직렬 가능성 보장으로 정리한다.

오답 포인트

세분화된 로킹은 병행성을 높이므로 전체 효율이 좋아 보이지만, 관리해야 할 로크 수는 오히려 증가한다. 병행성 증가와 오버헤드 감소를 같은 방향으로 착각하면 ②를 놓치기 쉽다.

NoteBase

다시 나오면 이렇게 판단

로킹 문제는 먼저 로킹 단위의 크기와 효과를 연결해서 본다. 큰 단위는 로크 수 감소·오버헤드 감소·병행성 감소이고, 작은 단위는 로크 수 증가·오버헤드 증가·병행성 증가이다. 그다음 2단계 로킹은 직렬 가능성 보장, 교착상태는 롤백 등으로 해소 가능이라고 이어서 판단하면 된다.

암기 포인트

큰 로크는 적게 잠그고 덜 관리하지만 덜 동시에 처리하고, 작은 로크는 많이 잠그고 더 관리하지만 더 동시에 처리한다.

NoteBase

문 7. <보기>는 고객 데이터를 관리하는 프로그램의 일부분을 보여주고 있다. <보기>와 같이 파일에 데이터를 저장하는 방식으로 고객의 데이터를 활용하는 프로그램을 작성한 경우에 대한 설명으로 가장 옳지 않은 것은?

<보기>

```
typedef struct {
  int id; // 고객 번호
  char name[20];  // 고객 이름
  char phone[15];  // 고객 휴대폰 번호
  int point;        // 고객이 보유한 포인트
} Client; // 고객 데이터는 data.txt에 저장되어 있음

int main() {
    Client client[100];
    FILE *fp;
    // 고객 데이터 파일을 열고 데이터 읽어 들이기
    fp = fopen("data.txt", "rw");
    ...
}
```

① 동일한 정보가 여러 파일에 저장되어 있을 경우, 데이터 중복성으로 값이 일관되지 못할 수 있다.

② 고객으로부터 수집하는 데이터의 종류가 변경될 경우, 프로그램 코드를 재컴파일해야 한다.

③ 동일한 고객 데이터 파일을 둘 이상의 프로그램에서 접근할 경우, 오류가 발생할 수 있다.

④ DBMS를 사용하더라도, 프로그램 내에 데이터 정의를 저장해야 한다.

정답 4번

출제 포인트

파일 처리 방식의 한계와 DBMS의 프로그램-데이터 독립성 차이를 구분하는 문제이다.

핵심 개념

파일 처리 방식은 각 응용프로그램이 자신이 사용할 데이터 구조를 직접 파일 단위로 관리하는 방식이다. DBMS 방식은 데이터 정의를 스키마와 데이터 사전에 통합 관리하여 응용프로그램과 데이터를 분리하는 방식이다. 결정 차이축은 데이터 정의를 프로그램 내부에 두는지, DBMS가 독립적으로 관리하는지이다.

NoteBase

선지별 해설

① O. 동일한 고객 정보가 여러 파일에 중복 저장되면 한 파일만 수정되고 다른 파일은 갱신되지 않아 데이터 불일치가 발생할 수 있다. 파일 처리 방식의 대표적 문제인 중복성과 불일치 설명으로 맞다. ② O. 고객 데이터 항목이 바뀌면 `struct Client` 같은 레코드 구조와 입출력 처리 코드를 함께 수정해야 하므로 프로그램 변경과 재컴파일이 필요해질 수 있다. 이는 프로그램-데이터 종속성을 설명한 것으로 맞다. ③ O. 같은 데이터 파일을 둘 이상의 프로그램이 동시에 접근하면 잠금과 동시성 통제가 약해 갱신 충돌이나 접근 오류가 생길 수 있다. 파일 처리 방식의 공유 제어 한계를 설명한 것으로 맞다. ④ X. 틀린 부분은 'DBMS를 사용하더라도, 프로그램 내에 데이터 정의를 저장해야 한다'는 표현이다. 실제로는 DBMS가 스키마와 데이터 사전을 통해 데이터 정의를 독립적으로 관리하므로 응용프로그램마다 데이터 정의를 별도로 저장할 필요가 없다.

오답 포인트

②와 ④가 모두 프로그램과 데이터의 관계를 말하므로 혼동하기 쉽다. 파일 처리 방식은 데이터 구조가 바뀌면 프로그램이 직접 영향을 받지만, DBMS는 그 데이터 정의를 시스템 차원에서 관리한다는 점이 핵심 차이이다.

다시 나오면 이렇게 판단

선지에 파일 처리 방식의 단점이 나오면 중복성, 불일치, 프로그램-데이터 종속성, 동시 접근 제어 부족을 먼저 확인한다. 반대로 DBMS의 특징이 나오면 데이터 정의의 독립 관리와 데이터 독립성을 떠올리면 된다.

암기 포인트

파일 처리 방식은 프로그램이 데이터 구조를 안고 가고, DBMS는 데이터 정의를 시스템이 관리한다.

NoteBase

문 8. 데이터베이스에서 해싱(hashing)은 키 값에 직접 산술적인 연산을 적용하여 계산된 레코드의 저장 주소에 따라 레코드를 삽입하거나 접근하는 방식이다. 해싱기법들 중 확장성 해싱(extendible hashing)에 대한 설명으로 가장 옳은 것은?

① 각 버킷(bucket)의 지역 깊이는 전역 깊이보다 크거나 같다.

② 지역 깊이와 전역 깊이가 같은 버킷에서 오버플로가 발생하면 디렉터리 배열 내의 엔트리 수는 두 배가 된다.

③ 확장성 해싱은 앞으로의 파일 크기 증가를 대비해서 공간을 미리 할당해둔다.

④ 레코드 검색은 한 번의 디스크 블록 접근을 필요로 한다.

정답 2번

출제 포인트

이 문제는 확장성 해싱에서 전역 깊이(global depth), 지역 깊이(local depth), 디렉터리 확장 조건을 정확히 구별하는지를 묻는다.

핵심 개념

전역 깊이는 디렉터리가 참조에 사용하는 해시 비트 수이다.
지역 깊이는 특정 버킷을 구분하는 데 실제로 쓰는 해시 비트 수이다.
확장성 해싱은 버킷이 넘칠 때 지역 깊이와 전역 깊이의 관계를 보고 버킷만 분할할지, 디렉터리까지 두 배로 늘릴지를 결정하는 동적 해싱이다.

선지별 해설

1번 X: 틀린 부분은 "지역 깊이는 전역 깊이보다 크거나 같다"이다. 지역 깊이는 전역 깊이를 넘을 수 없으므로 올바르게는 "각 버킷의 지역 깊이는 전역 깊이보다 작거나 같다"이다.
2번 O: 지역 깊이와 전역 깊이가 같은 버킷에서 오버플로가 발생하면 현재 디렉터리 비트 수만으로는 더 이상 분할을 표현할 수 없으므로 전역 깊이를 1 증가시키고 디렉터리 엔트리 수를 두 배로 늘린다.
3번 X: 틀린 부분은 "앞으로의 파일 크기 증가를 대비해서 공간을 미리 할당해둔다"이다. 확장성 해싱은 고정 공간을 미리 크게 잡는 방식이 아니라 필요할 때 버킷을 분할하고 디렉터리를 확장하는 동적 해싱이다.
4번 X: 틀린 부분은 "한 번의 디스크 블록 접근을 필요로 한다"이다. 확장성 해싱의 검색은 디렉터리와 버킷을 통해 이루어지므로 항상 한 번의 디스크 블록 접근으로 끝난다고 단정할 수 없다. 올바르게는 "디렉터리가 메모리에 유지되면 보통 버킷 1회 접근으로 검색할 수 있으나, 일반적으로는 디렉터리 접근까지 함께 고려한다"이다.

NoteBase

오답 포인트

지역 깊이와 전역 깊이의 대소관계를 거꾸로 외우거나, 오버플로가 나면 무조건 디렉터리를 두 배로 늘린다고 이해하면 ①과 ②를 함께 헷갈리기 쉽다. ④처럼 평균적인 접근 효율을 절대적인 접근 횟수로 바꾸어 말하는 선지도 자주 함정이 된다.

다시 나오면 이렇게 판단

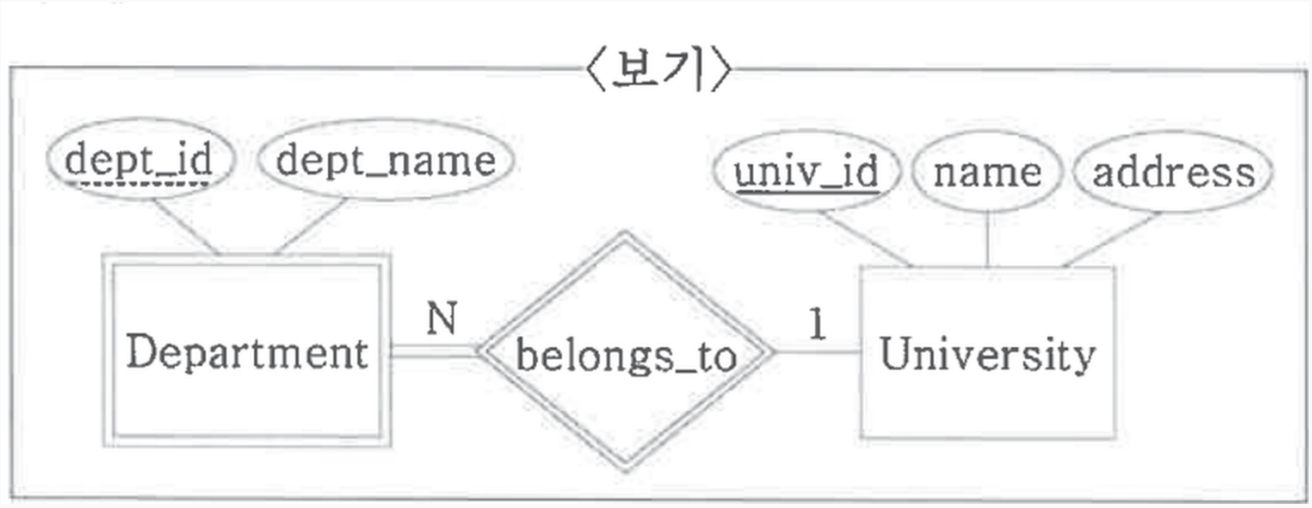
확장성 해싱 선지는 먼저 지역 깊이 ≤ 전역 깊이인지 확인하고, 오버플로 시 지역 깊이=전역 깊이면 디렉터리 확장, 지역 깊이<전역 깊이면 버킷 분할만 수행한다고 판단한다.

암기 포인트

지역 깊이는 버킷 기준, 전역 깊이는 디렉터리 기준이며, 같을 때 넘치면 디렉터리가 두 배가 된다.

NoteBase

문 9. <보기>의 E-R 다이어그램에 대한 설명으로 가장 옳은 것은?



① 관계 belongs_to에서 하나의 Department 개체(entity)는 여러 개의 University 개체에 대응한다.

② Department는 독립적으로 존재할 수 있는 개체들의 타입이다.

③ belongs_to는 식별 관계(identifying relationship)이다.

④ Department의 기본키(primary-key)는 {dept_id}이다.

정답 3번

출제 포인트

약한 개체, 식별 관계, 부분 키, 1:N 카디널리티를 E-R 다이어그램 표기에서 정확히 읽어 내는 문제이다.

선지별 해설

① X: Department 쪽이 N이고 University 쪽이 1이므로 여러 Department가 하나의 University에 속한다. 하나의 Department가 여러 University에 대응한다는 설명이 틀렸다. 틀린 표현은 '하나의 Department 개체는 여러 개의 University 개체에 대응한다'이고, 올바르게 고치면 '여러 Department 개체가 하나의 University 개체에 대응한다'가 된다.

② X: Department는 이중 사각형으로 표시된 약한 개체이므로 독립적으로 존재할 수 있는 강한 개체 타입이 아니다. 틀린 표현은 '독립적으로 존재할 수 있는 개체들의 타입'이고, 올바르게 고치면 'University에 존재를 의존하는 약한 개체 타입'이 된다.

③ O: belongs_to는 이중 마름모로 표시되어 식별 관계이다. Department는 University의 키와 자신의 부분 키를 함께 사용해 식별되므로 belongs_to는 식별 관계가 맞다.

④ X: dept_id는 Department의 부분 키일 뿐 단독 기본키가 아니다. 틀린 표현은 'Department의 기본키는 {dept_id}'이고, 올바르게 고치면 'Department의 기본키는 {univ_id, dept_id}이다'가 된다.

NoteBase

오답 포인트

약한 개체의 이중 사각형과 식별 관계의 이중 마름모를 놓치면 ②와 ④를 강한 개체 기준으로 잘못 판단하기 쉽다. 또한 1:N 카디널리티를 읽을 때 N 쪽의 여러 개체가 1 쪽의 한 개체에 속한다는 방향을 거꾸로 읽으면 ①에 속기 쉽다.

다시 나오면 이렇게 판단

E-R 다이어그램에서는 먼저 이중 사각형인지, 이중 마름모인지, 부분 키 표기가 있는지 확인한다. 그다음 1과 N의 위치를 보고 어느 쪽이 소유자 개체인지, 약한 개체가 어떤 관계로 식별되는지 판단한다.

암기 포인트

약한 개체는 이중 사각형, 식별 관계는 이중 마름모, 부분 키는 단독 기본키가 아니라 소유자 키와 결합되어 식별된다.

NoteBase

문 10. 3단계 데이터베이스 구조에 대한 설명으로 가장 옳은 것은?

① 매핑 개념을 활용하여 각 단계의 스키마 간의 대응 관계를 정의할 수 있다.

② 데이터베이스의 각 단계마다 개념 단계 스키마를 저장한다.

③ 외부 단계의 사용자마다 필요한 데이터가 다를 수 있지만 하나의 스키마만 존재한다.

④ 물리적 저장장치에 데이터베이스를 저장하는 방식이 변경되면 상위 계층의 스키마도 적절하게 수정해야 한다.

정답 1번

출제 포인트

이 문제는 3단계 데이터베이스 구조에서 외부 스키마, 개념 스키마, 내부 스키마와 매핑, 데이터 독립성의 관계를 구별하는지를 묻는다.

핵심 개념

외부 스키마는 사용자나 응용 프로그램별로 필요한 데이터만 보여 주는 사용자 뷰이며 여러 개가 존재할 수 있다.
개념 스키마는 데이터베이스 전체의 논리적 구조를 정의하는 전역 스키마이며 하나만 존재한다.
내부 스키마는 데이터의 물리적 저장 구조와 접근 방법을 정의한다.
매핑은 외부-개념, 개념-내부 스키마 사이의 대응 관계를 정해 서로 변환할 수 있게 한다.

선지별 해설

1번: O. 매핑은 서로 다른 단계의 스키마를 연결하는 대응 관계이다. 외부 스키마와 개념 스키마, 개념 스키마와 내부 스키마 사이의 관계를 매핑으로 정의하므로 옳다.
2번: X. 틀린 표현은 "각 단계마다 개념 단계 스키마를 저장한다"이다. 3단계 구조에는 외부 스키마, 개념 스키마, 내부 스키마가 각각 존재하며 개념 스키마는 전체 데이터베이스에 대해 하나만 존재한다.
3번: X. 틀린 표현은 "하나의 스키마만 존재한다"이다. 외부 단계는 사용자마다 필요한 데이터가 다를 수 있으므로 외부 스키마가 여러 개 존재할 수 있다.
4번: X. 틀린 표현은 "상위 계층의 스키마도 적절하게 수정해야 한다"이다. 물리적 저장 방식이 바뀌어도 내부 스키마만 조정하고 상위의 개념 스키마나 외부 스키마는 그대로 유지할 수 있어야 하는데, 이것이 물리적 데이터 독립성이다.

오답 포인트

외부 스키마와 개념 스키마의 개수를 뒤섞거나, 물리적 저장 방식 변화가 상위 스키마까지 바뀐다고 생각하면 틀리기 쉽다. 특히 3번은 사용자별 데이터 요구가 다르다는 앞부분은 맞지만 뒤의 "하나의 스키마"가 함정이다.

NoteBase

다시 나오면 이렇게 판단

선지를 보면 먼저 어느 단계의 스키마를 말하는지 구분한 뒤, 외부는 여러 개, 개념은 하나, 내부는 물리 저장이라는 기준으로 판단하면 된다. 이어서 저장 방식 변화가 상위 논리 구조에 영향을 주는지 확인하면 데이터 독립성 선지도 바로 걸러진다.

암기 포인트

외부는 여러 뷰, 개념은 전체 논리 하나, 내부는 물리 저장, 매핑은 단계 간 연결이다.

NoteBase

문 11. B-트리는 탐색 성능을 높이기 위해 균형 있게 높이를 유지하는 균형 트리(balanced tree)의 일종이다. <보기>와 같이 나열된 순서로 키가 삽입되어 B-트리가 구성되었을 때, 루트 노드에 존재하는 키 값은? (단, 이 트리의 차수는 3이다.)

<보기>
키 삽입순서: 7, 19, 14, 15, 3, 5, 10, 20, 6

① 7
② 10
③ 14
④ 15

정답 3번

출제 포인트

이 문제는 차수 3인 B-트리에서 키를 삽입할 때 노드 분할이 어떤 순서로 일어나고 최종 루트가 무엇이 되는지를 묻는 문제이다.

올바른 순서

1. 7, 19를 넣으면 루트는 [7, 19]가 된다.
2. 14를 넣으면 [7, 14, 19]가 되어 중간값 14가 올라가고 루트는 [14]가 된다.
3. 15, 3, 5, 10을 차례로 넣으면 루트는 [5, 14], 자식은 [3], [7, 10], [15, 19]가 된다.
4. 20을 넣으면 오른쪽 자식 [15, 19, 20]이 분할되어 19가 올라가고, 루트 [5, 14, 19]도 다시 분할되어 새 루트는 [14]가 된다.
5. 마지막으로 6을 넣으면 왼쪽 내부 노드에서만 분할이 일어나고 루트 [14]는 바뀌지 않는다.

왜 이 순서인가: 차수 3 B-트리는 한 노드에 키를 최대 2개만 둘 수 있으므로 키가 3개가 되는 순간마다 중간값이 부모로 승격된다.

NoteBase

선지별 해설

① X
7은 초기에 루트였지만 14가 삽입되면서 첫 분할이 일어날 때 루트 자리에서 내려간다. 틀린 표현은 최종 루트를 초기 루트와 같게 본 점이다. 올바르게는 7은 최종적으로 왼쪽 내부 노드의 분리 기준 키가 된다.

② X
10은 중간 자식 [7, 10]에 들어가는 값일 뿐 루트로 승격된 적이 없다. 틀린 표현은 삽입된 값이 부모로 자동 승격된다고 본 점이다. 올바르게는 노드가 가득 찼을 때 그 노드의 중간값만 부모로 올라간다.

③ O
20 삽입으로 오른쪽 자식이 [15, 19, 20]이 되어 19가 부모로 올라가고, 그 결과 루트 [5, 14, 19]가 다시 분할되면서 중간값 14가 새 루트가 된다. 마지막 6 삽입 이후에도 루트는 14로 유지된다.

④ X
15는 오른쪽 리프에 남는 값이며 분할 시 중간값이 아니므로 루트가 되지 않는다. 틀린 표현은 오른쪽 자식의 첫 값이 승격된다고 본 점이다. 올바르게는 [15, 19, 20]의 중간값 19가 부모로 올라가고 최종 루트는 14가 된다.

오답 포인트

초기 루트 7이나 마지막에 눈에 띄는 값 15를 그대로 고르기 쉽다. 그러나 B-트리는 삽입값 자체가 아니라 분할 순간의 중간값이 승격된다는 점을 놓치면 루트 변화를 잘못 추적하게 된다.

다시 나오면 이렇게 판단

B-트리 삽입 문제는 매 단계마다 현재 노드의 키 개수를 확인하고, 키가 3개가 되는 순간 중간값이 어디로 올라가는지만 추적하면 된다. 특히 루트가 넘치면 그 중간값이 새 루트가 된다고 판단하면 된다.

암기 포인트

차수 3 B-트리는 키 3개가 되면 분할되고 중간값이 승격되므로 최종 루트는 마지막 분할의 중간값이다.

NoteBase

문 12. 관계 데이터 모델 및 그 제약조건에 대한 설명으로 가장 옳은 것은?

① 한 릴레이션(relation) 내의 투플(tuple)들은 수학적으로 집합이므로 투플들 간의 순서가 중요하지 않지만, 각 투플은 (<애트리뷰트(attribute)>, <값>)의 쌍들의 집합으로 해석되므로 순서가 중요하다.

② 릴레이션 R의 애트리뷰트들의 부분집합 A에 대하여 현재 R의 어떤 임의의 두 투플도 동일한 값을 가지지 않는다면 이 집합 A를 릴레이션 R의 후보키라고 한다.

③ 대학교의 '학생' 릴레이션의 '학년' 애트리뷰트 값이 1~4 사이의 정수 값을 가져야 한다는 조건은 전이 제약조건(transition constraint)이라고 하며, 트리거(trigger)와 주장(assertion)이라는 기법을 이용하여 보장할 수 있다.

④ 릴레이션 R 내의 애트리뷰트 집합 X와 Y 간의 함수 관계는 정규화(normalization) 과정에서 활용될 수 있다.

정답 4번

출제 포인트

이 문제는 관계 데이터 모델에서의 순서 무의미성, 후보키의 조건, 도메인 제약조건, 함수 종속의 활용을 구별할 수 있는지를 묻는다.

핵심 개념

릴레이션은 튜플들의 집합이고, 각 튜플도 애트리뷰트-값 쌍의 집합이므로 둘 다 순서에 의미가 없다.
후보키는 모든 튜플을 유일하게 식별하면서 더 이상 줄일 수 없는 최소 슈퍼키이다.
도메인 제약조건은 애트리뷰트가 가질 수 있는 값의 범위를 제한하는 제약이고, 함수 종속은 정규화의 기준이 된다.

선지별 해설

① X: 튜플 간의 순서가 중요하지 않은 것처럼 튜플 내부의 애트리뷰트 순서도 중요하지 않다. 틀린 표현은 "각 투플은 ... 순서가 중요하다"이다. 올바르게는 "각 투플도 애트리뷰트-값 쌍의 집합으로 해석되므로 순서가 중요하지 않다"이다.
② X: 현재 릴레이션 인스턴스에서 값이 우연히 겹치지 않는다고 해서 후보키가 되지는 않는다. 틀린 표현은 "현재 R의 어떤 임의의 두 투플도 동일한 값을 가지지 않는다면 ... 후보키"이다. 올바르게는 "A가 모든 유효한 릴레이션 상태에서 튜플을 유일하게 식별하면 슈퍼키이고, 그중 최소성을 만족해야 후보키"이다.
③ X: 학년이 1~4의 정수여야 한다는 것은 값의 허용 범위를 제한하는 도메인 제약조건이다. 틀린 표현은 "전이 제약조건(transition constraint)"이다. 올바르게는 "도메인 제약조건(domain constraint)"이며, 전이 제약조건은 상태 변화 과정에 대한 제약을 뜻한다.
④ O: 애트리뷰트 집합 X와 Y 사이의 함수 종속은 릴레이션을 분해하고 이상 현상을 줄이는 정규화 과정의 핵심 기준으로 활용된다.

NoteBase

오답 포인트

②는 "유일성"만 보고 후보키라고 단정하게 만들고, ③은 도메인 제약과 전이 제약을 바꿔 놓은 것이 함정이다. ①도 튜플 내부 순서를 중요하다고 바꿔 놓아 집합 개념을 흐리게 만든다.

다시 나오면 이렇게 판단

관계 모델 문제에서는 먼저 집합인지 순서열인지, 다음으로 키가 유일성과 최소성을 함께 만족하는지, 마지막으로 값의 범위 제약인지 상태 변화 제약인지 구분하면 된다. 정규화가 나오면 함수 종속 여부를 바로 연결하면 된다.

암기 포인트

관계 모델은 순서보다 집합, 키는 유일성+최소성, 학년 1~4는 도메인 제약, 정규화는 함수 종속이다.

NoteBase

문 13. 데이터베이스 스키마가 <보기>와 같을 때, 결과가 다른 질의는?

<보기>
Student(id, name, dept_id)
Dept(id, dept_name, building)

① (_name="Kim"(Student)) _student.dept_id = dept.id (_building="F2"(Dept))
② _building="F2"((_name="Kim"(Student)) _student.dept_id = dept.id Dept)
③ Student _student.dept_id = dept.id (_name="Kim" building="F2"(Dept))
④ _name="Kim" building="F2"(Student _student.dept_id = dept.id Dept)

정답 3번

출제 포인트

이 문제는 관계대수에서 선택 연산의 푸시다운이 가능한 조건과, 선택 조건을 적용할 수 있는 속성 범위를 정확히 구분하는지를 묻는 문제이다.

핵심 개념

선택 푸시다운은 어떤 선택 조건이 한 릴레이션의 속성만 참조하면 조인 전에 그 릴레이션에 먼저 적용해도 결과가 같다는 성질이다.
속성 적용 범위는 선택식이 현재 대상으로 삼는 릴레이션에 실제 존재하는 속성만 사용할 수 있다는 원칙이다.
이 문항에서는 `name="Kim"`은 Student의 속성이고 `building="F2"`는 Dept의 속성이므로, 두 조건은 각각 자기 릴레이션으로 나누어 내릴 수는 있어도 둘 다 Dept에 함께 내릴 수는 없다.

NoteBase

선지별 해설

① O

`name="Kim"`은 Student에, `building="F2"`는 Dept에 각각 미리 적용한 뒤 조인한 형태이다. 두 조건이 각자 한 릴레이션의 속성만 참조하므로 조인 후 전체 결과에 두 조건을 거는 것과 결과가 같다.

② O

먼저 Student에서 `name="Kim"`인 튜플만 남기고 Dept와 조인한 다음, 조인 결과에서 `building="F2"`를 선택한 형태이다. `building` 조건은 Dept 속성만 참조하므로 조인 후에 걸어도 ①, ④와 같은 결과가 나온다.

③ X

틀린 부분은 `σ_name="Kim" ∧ building="F2"(Dept)`이다. Dept의 속성은 `id`, `dept_name`, `building`뿐이므로 `name="Kim"` 조건을 Dept에 적용할 수 없다. 올바르게는 Dept에는 `σ_building="F2"(Dept)`만 적용하고, `name="Kim"`은 `σ_name="Kim"(Student)`로 Student에 적용하거나 조인 후 전체 결과에 적용해야 한다.

④ O

Student와 Dept를 조인한 결과에는 `name`과 `building` 속성이 모두 존재하므로 `σ_name="Kim" ∧ building="F2"`를 조인 뒤에 적용할 수 있다. 이는 ①처럼 조건을 각 릴레이션으로 분배한 표현과 동치이다.

오답 포인트

①, ②, ④는 조건을 거는 위치만 달라 보여도 속성 참조 범위가 맞기 때문에 같은 결과를 낸다. 많이 틀리는 지점은 ③에서 `name="Kim"`도 함께 Dept 쪽으로 내려 보낸 부분을 무심코 지나치는 것이다. 선택 연산은 아무 데나 밀어 넣는 것이 아니라 그 릴레이션이 가진 속성에 대해서만 밀어 넣을 수 있다.

다시 나오면 이렇게 판단

조인과 선택이 섞인 식은 먼저 각 선택 조건이 어느 릴레이션의 속성을 참조하는지 본다. 한 조건이 한 릴레이션의 속성만 쓰면 그 릴레이션 아래로 내릴 수 있고, 현재 릴레이션에 없는 속성이 조건식에 들어 있으면 그 식은 잘못 내려간 것이다.

암기 포인트

선택 푸시다운은 가능하지만, 조건은 자기 속성이 있는 릴레이션까지만 내려간다.

NoteBase

문 14. <보기>의 SQL 질의의 의미로 가장 옳은 것은? (단, /* */는 주석이다.)

〈보기〉

```
SELECT Dnumber, COUNT(*)
FROM DEPARTMENT, EMPLOYEE
WHERE Dnumber = Dno AND Salary >= 40000 AND
        Dno IN ( SELECT     Dno
                 FROM     EMPLOYEE
                 GROUP BY     Dno
                 HAVING     COUNT(*) >= 5 )
GROUP BY Dnumber;
/* 릴레이션 스키마: DEPARTMENT(Dnumber, Dname, Mgr_ssn)
                   EMPLOYEE(ssn, Ename, Address, Salary, Dno) */
```

① 40,000 이상의 급여를 받는 직원들의 부서번호와 부서별 사원 수를 검색하라.

② 40,000 이상의 급여를 받는 직원들이 소속되거나 소속 직원이 5명 이상인 부서의 부서번호와 그 부서에서 40,000 이상의 급여를 받는 사원 수를 검색하라.

③ 5명 이상의 사원이 근무하고 40,000 이상의 급여를 받는 직원들이 소속되어 있는 부서의 부서번호와 그 부서에서 40,000 이상의 급여를 받는 사원 수를 검색하라.

④ 5명 이상의 사원이 근무하는 부서의 부서번호와 부서별 사원 수를 검색하라.

정답 3번

출제 포인트

이 문제는 SQL의 WHERE 조건, 서브쿼리의 HAVING 조건, GROUP BY 이후 COUNT(*)가 각각 무엇을 거르는지 정확히 해석할 수 있는지를 묻는다.

핵심 개념

메인 질의의 Salary >= 40000은 급여가 40000 이상인 직원만 남기는 조건이다.
서브쿼리의 HAVING COUNT(*) >= 5는 부서 전체 직원 수가 5명 이상인 부서 번호만 고르는 조건이다.
최종 COUNT(*)는 위 두 조건을 모두 만족하고 남은 행 수이므로, 결과의 사원 수는 부서 전체 인원이 아니라 해당 부서의 40000 이상 급여자 수이다.

NoteBase

선지별 해설

1번: X. 급여 40000 이상 직원의 부서번호를 찾는 점은 맞지만, 모든 부서를 대상으로 하지 않는다. Dno IN (서브쿼리) 조건 때문에 전체 직원 수가 5명 이상인 부서만 결과에 포함된다. 또한 COUNT(*)는 부서별 전체 사원 수가 아니라 40000 이상 급여를 받는 사원 수이다. 틀린 표현은 '부서별 사원 수'이고, 올바르게는 '직원 수가 5명 이상인 부서에서 40000 이상 급여를 받는 사원 수'이다.
2번: X. 이 질의는 '40,000 이상 급여를 받는 직원들이 소속되거나 소속 직원이 5명 이상인 부서'를 찾는 것이 아니다. 메인 질의는 Salary >= 40000이고 서브쿼리 조건도 함께 만족해야 하므로 OR가 아니라 AND 관계이다. 틀린 표현은 '소속되거나'이고, 올바르게는 '5명 이상이 근무하고 그중 40000 이상 급여자가 있는 부서'이다.
3번: O. 서브쿼리는 직원 수가 5명 이상인 부서를 고르고, 메인 질의는 그 부서들 중 급여가 40000 이상인 직원만 남긴 뒤 부서번호별로 COUNT(*)를 계산한다. 따라서 결과는 5명 이상이 근무하는 부서의 부서번호와 그 부서에서 40000 이상 급여를 받는 사원 수이다.
4번: X. 서브쿼리만 보면 5명 이상의 사원이 근무하는 부서를 찾는 것처럼 보이지만, 최종 SELECT의 COUNT(*)는 메인 질의에서 Salary >= 40000으로 걸러진 행만 센다. 따라서 결과는 부서별 전체 사원 수가 아니다. 틀린 표현은 '부서별 사원 수'이고, 올바르게는 '그 부서에서 40000 이상 급여를 받는 사원 수'이다.

오답 포인트

COUNT(*)를 보면 곧바로 부서 전체 인원을 세는 것으로 착각하기 쉽다. 그러나 COUNT(*)는 WHERE 절을 통과한 행만 세므로, 이 문제에서는 급여 40000 이상 직원만 센다. 또 서브쿼리 조건을 OR처럼 읽으면 2번에 끌리기 쉽다.

다시 나오면 이렇게 판단

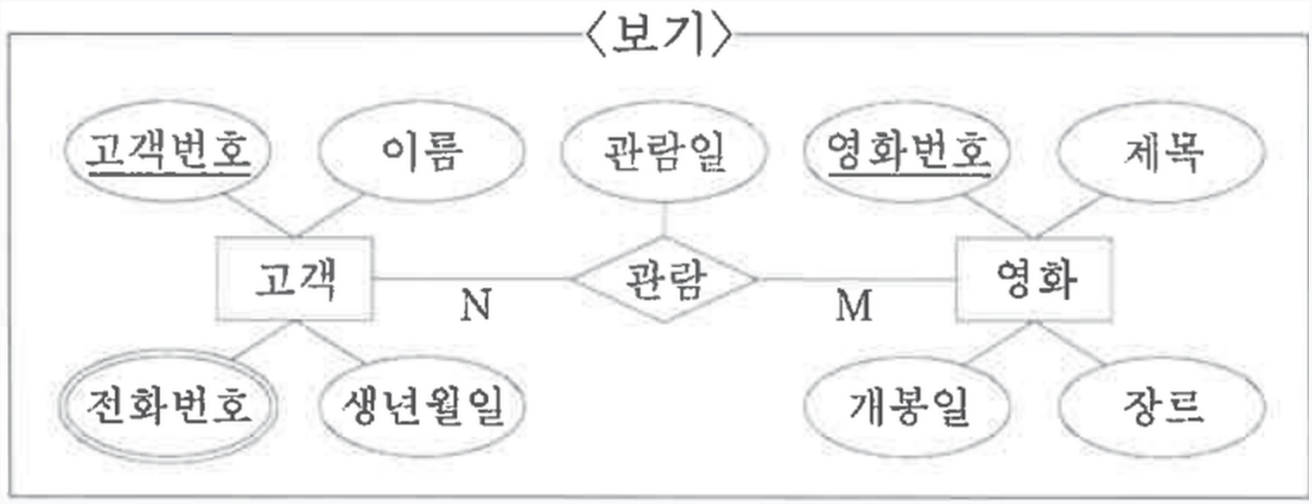
SQL 의미 해석 문제에서는 먼저 메인 질의의 WHERE 절이 남기는 행을 확인하고, 다음으로 서브쿼리가 어떤 집합을 제한하는지 본다. 마지막으로 GROUP BY 뒤의 COUNT가 전체 행을 세는지, 이미 필터링된 행을 세는지 판단하면 된다.

암기 포인트

HAVING은 그룹을 고르고, WHERE는 행을 거르며, COUNT(*)는 최종적으로 남은 행만 센다.

NoteBase

문 15. <보기>의 E-R 다이어그램을 관계형 스키마로 변환한 결과로 가장 옳은 것은?



① 고객(고객번호, 이름, 생년월일, 관람일), 고객전화번호(고객번호, 전화번호), 영화(영화번호, 제목, 장르, 개봉일)

② 고객(고객번호, 이름, 생년월일), 고객전화번호(고객번호, 전화번호), 영화(영화번호, 제목, 장르, 개봉일), 관람(고객번호, 영화번호, 관람일)

③ 고객(고객번호, 이름, 생년월일, 전화번호), 영화(영화번호, 제목, 장르, 개봉일), 관람(고객번호, 영화번호, 관람일)

④ 고객(고객번호, 이름, 생년월일), 영화(영화번호, 제목, 장르, 개봉일), 관람(고객번호, 관람일)

정답 2번

출제 포인트

이 문제는 E-R 다이어그램에서 기본키, 다중값 속성, N:M 관계와 관계 속성을 읽고 이를 관계형 스키마로 정확히 변환하는지를 묻는 문제이다.

NoteBase

선지별 해설

① X: 고객(고객번호, 이름, 생년월일, 관람일)은 틀리다. 관람일은 고객 자체의 속성이 아니라 고객과 영화 사이 관계 관람의 속성이다. 틀린 표현은 "고객(..., 관람일)"이고, 올바르게는 관람(고객번호, 영화번호, 관람일)처럼 별도 관계로 분리되어야 한다.
② O: 고객(고객번호, 이름, 생년월일), 고객전화번호(고객번호, 전화번호), 영화(영화번호, 제목, 장르, 개봉일), 관람(고객번호, 영화번호, 관람일)은 변환 규칙에 맞다. 다중값 속성 전화번호를 별도 릴레이션으로 분리했고, N:M 관계 관람도 두 개체의 기본키를 포함한 별도 릴레이션으로 만들었으며 관계 속성 관람일도 함께 저장했다.
③ X: 고객(고객번호, 이름, 생년월일, 전화번호)는 틀리다. 전화번호는 이중 타원으로 표시된 다중값 속성이므로 고객 테이블 안에 단일 속성처럼 넣지 않는다. 틀린 표현은 "고객(..., 전화번호)"이고, 올바르게는 고객전화번호(고객번호, 전화번호)로 분리해야 한다.
④ X: 관람(고객번호, 관람일)은 틀리다. N:M 관계를 관계형 스키마로 바꾸면 참여 개체의 기본키 둘 다 필요하므로 영화번호가 빠질 수 없다. 또한 전화번호가 다중값 속성인데 고객전화번호 릴레이션도 빠져 있다. 틀린 표현은 "관람(고객번호, 관람일)"이고, 올바르게는 관람(고객번호, 영화번호, 관람일)과 고객전화번호(고객번호, 전화번호)가 추가되어야 한다.

오답 포인트

관계의 속성인 관람일을 개체 고객의 속성처럼 넣거나, 이중 타원인 전화번호를 일반 속성으로 처리하면 오답이 된다. ④처럼 N:M 관계 테이블에서 한쪽 기본키를 빼먹는 실수도 자주 나온다.

다시 나오면 이렇게 판단

E-R 다이어그램을 관계형 스키마로 바꿀 때는 먼저 밑줄 속성으로 기본키를 찾고, 이중 타원은 별도 릴레이션으로 분리하며, N:M 관계는 참여 개체의 기본키를 모두 포함한 별도 릴레이션으로 만든 뒤 관계 속성을 그 릴레이션에 붙이면 된다.

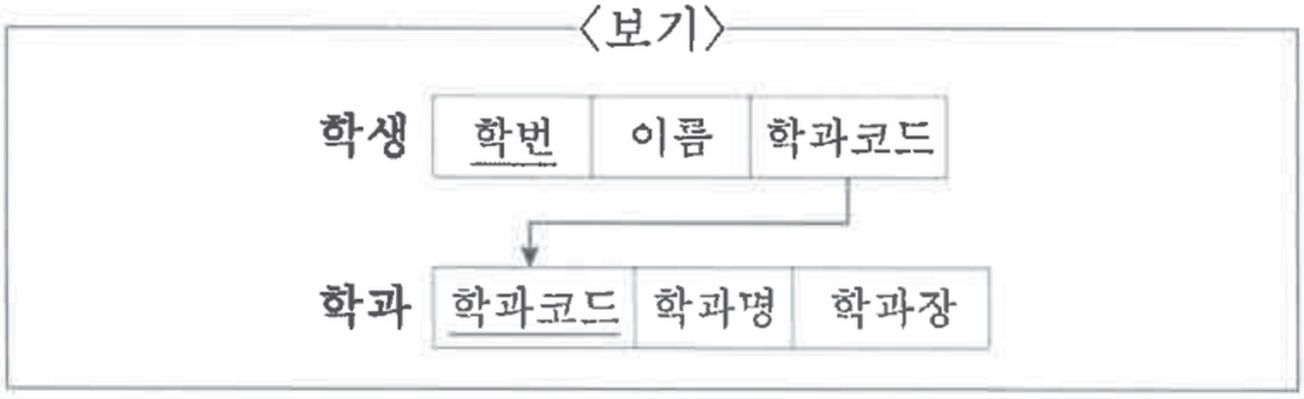
암기 포인트

다중값 속성은 따로 빼고, N:M 관계는 양쪽 키를 모두 가져간다.

NoteBase

문 16. <보기>와 같은 분산 데이터베이스를 단편화하여, 사이트 1에는 학생 테이블, 사이트 2에는 학과 테이블을 저장한다. 제3의 사이트에서 "select 학생.이름, 학과.학과명 from 학생, 학과 where 학생.학과코드 = 학과.학과코드"를 수행하려고 할 때, 사이트 간에 전송되는 데이터의 양을 최소로 한다면, 전송되는 데이터 양[바이트]의 범위는? (단, 학과가 없는 학생은 없으며, 학생과 학과 테이블의 모든 속성들의 크기는 10바이트이고, 학생은 10,000명, 학과는 60개이다.)

<보기>



① 0 이상 1,000 미만

② 1,000 이상 10,000 미만

③ 10,000 이상 100,000 미만

④ 100,000 이상

정답 4번

출제 포인트

이 문제는 분산 질의에서 필요한 속성만 투영한 뒤 조인을 어디서 수행해야 사이트 간 전송량이 최소가 되는지를 계산하는 문제이다.

조건 정리

학생 테이블 속성은 학번, 이름, 학과코드 3개이므로 한 튜플은 30바이트이다.
학과 테이블 속성은 학과코드, 학과명, 학과장 3개이므로 한 튜플은 30바이트이다.
질의 결과에 실제로 필요한 속성은 학생의 이름, 학생의 학과코드, 학과의 학과코드, 학과의 학과명이다.
학생은 10,000명이고 학과는 60개이며, 학과가 없는 학생은 없으므로 결과 행 수도 10,000개이다.

NoteBase

사용할 공식

전송량은 각 사이트에서 다른 사이트로 보내는 필요한 속성들의 총 바이트 수로 계산한다.
필요 없는 속성은 먼저 투영해서 제거하는 것이 최소 전송량의 기본 원리이다.
이 문제에서는 사이트 1에서 이름과 학과코드만 보내면 20바이트 × 10,000 = 200,000바이트이고, 사이트 2에서 학과코드와 학과명만 보내면 20바이트 × 60 = 1,200바이트이다.

계산 과정

1. 학생 테이블에서 최종 결과와 조인에 필요한 속성만 남기면 이름, 학과코드이므로 1명당 20바이트이다.
2. 학과 테이블에서 필요한 속성만 남기면 학과코드, 학과명이므로 1개 학과당 20바이트이다.
3. 사이트 3에서 조인하도록 보내면 학생 쪽 200,000바이트와 학과 쪽 1,200바이트가 전송되어 총 201,200바이트이다.
4. 또는 학과코드, 학과명을 사이트 1로 보내 조인한 뒤 결과 이름, 학과명 10,000건을 사이트 3으로 보내도 1,200 + 200,000 = 201,200바이트이다.
5. 따라서 최소 전송량은 201,200바이트이며 범위는 100,000 이상이다.

선지별 해설

① X: 최소 전송량은 201,200바이트이므로 0 이상 1,000 미만이 될 수 없다. 틀린 표현은 "1,000 미만"이고, 올바르게는 100,000 이상이다.
② X: 최소 전송량은 이미 학생 이름과 학과명을 담은 최종 결과만으로도 200,000바이트이므로 1,000 이상 10,000 미만은 불가능하다. 틀린 표현은 "10,000 미만"이고, 올바르게는 100,000 이상이다.
③ X: 10,000 이상 100,000 미만도 최종 결과 크기 200,000바이트보다 작아서 성립하지 않는다. 틀린 표현은 "100,000 미만"이고, 올바르게는 201,200바이트이므로 100,000 이상이다.
④ O: 필요한 속성만 투영해도 학생 쪽 200,000바이트와 학과 쪽 1,200바이트가 필요하므로 최소 전송량은 201,200바이트이다. 따라서 100,000 이상이 맞다.

오답 포인트

학생과 학과 원본 테이블 전체를 그대로 옮기는 경우만 떠올리면 30바이트 단위 계산에 머물러도 답은 나오지만, 핵심 함정은 필요한 속성만 먼저 투영할 수 있다는 점이다. 또 학과가 없는 학생이 없다는 조건 때문에 결과 행 수가 학생 수 10,000명과 같다는 점을 놓치면 결과 전송량 200,000바이트를 과소평가하게 된다.

다시 나오면 이렇게 판단

분산 질의 전송량 문제는 먼저 최종 결과에 꼭 필요한 속성과 조인 속성만 남긴 뒤 각 테이블의 전송 크기를 계산한다. 그다음 조인을 어느 사이트에서 하더라도 최종 결과 자체의 크기가 하한선이라는 점을 기준으로 최소 전송량을 비교하면 된다.

NoteBase

암기 포인트

분산 조인 최소 전송량은 전체 테이블이 아니라 필요한 속성만 투영한 크기와 최종 결과 크기를 먼저 계산하는 것이 핵심이다.

NoteBase

문 17. <보기>의 부서와 사원 테이블에 대해, 결과 테이블의 카디날리티(cardinality)가 가장 큰 관계 대수식은?

〈보기〉

부서

이름	달성률
생산	60
기획	70
개발	80
영업	90

사원

이름	소속부서	봉급
박찬형	생산	350
홍길동	생산	250
김철수	기획	200
김영희	기획	400
황이수	개발	500
정순희	영업	300

① _달성률>=70(부서) _부서.이름=사원.소속부서(사원)

② _사원.봉급<300(부서×사원)

③ _봉급>=200 and 봉급<500(사원)

④ _달성률<80(부서) _부서.이름=사원.소속부서(사원)

정답 2번

출제 포인트

이 문제는 선택, 조인, 카티션 곱 연산 뒤에 남는 튜플 수를 계산하여 각 관계대수식의 카디널리티를 비교하는 문제이다.

조건 정리

주어진 값: 부서 릴레이션은 4개 튜플이고 사원 릴레이션은 6개 튜플이다.
구해야 할 값: 각 관계대수식의 결과 튜플 수이다.
주의할 조건: 조인은 조건에 맞는 튜플만 결합하고, 카티션 곱은 먼저 모든 조합을 만든 뒤 선택 조건을 적용한다.

NoteBase

사용할 공식

카디널리티는 연산 결과에 포함되는 튜플 개수이다.
선택 σ는 조건을 만족하는 튜플만 남기고, 조인 $\bowtie$는 조인 속성값이 일치하는 튜플만 결합하며, 카티션 곱 $\times$는 두 릴레이션의 모든 조합을 만든다.
이 원리를 쓰는 이유: 각 선지는 결국 조건을 통과한 튜플이 몇 개인지를 세면 비교할 수 있기 때문이다.

계산 과정

1. ①은 달성률이 70 이상인 부서가 기획, 개발, 영업 3개이고, 조인 결과는 기획 2명, 개발 1명, 영업 1명으로 총 4개 튜플이다.
2. ②는 부서×사원으로 먼저 $4 \times 6 = 24$개 조합을 만든다. 여기서 봉급이 300 미만인 사원은 홍길동, 김철수 2명이므로 각 부서와 2명씩 짝지어져 총 $4 \times 2 = 8$개 튜플이 남는다.
3. ③은 봉급이 200 이상 500 미만인 사원이 박찬형, 홍길동, 김철수, 김영희, 정순희로 5명이라서 카디널리티는 5이다.
4. ④는 달성률이 80 미만인 부서가 생산, 기획 2개이고, 조인 결과는 생산 2명과 기획 2명으로 총 4개 튜플이다.
5. 따라서 가장 큰 카디널리티는 8인 ②이다.

선지별 해설

① X
달성률>=70인 부서는 기획, 개발, 영업이다. 이를 사원과 조인하면 김철수, 김영희, 황이수, 정순희만 남아 카디널리티는 4이다.
틀린 판단은 ①이 가장 큰 결과라는 점이다. 올바르게는 ①의 카디널리티는 4이다.

② O
부서×사원은 24개 튜플을 만든다. 여기서 사원.봉급<300 조건을 적용하면 홍길동과 김철수에 해당하는 조합만 남으므로 각 부서마다 2개씩, 총 8개 튜플이 된다.
따라서 네 식 중 카디널리티가 가장 크므로 옳다.

③ X
봉급>=200 and 봉급<500 조건을 만족하는 사원은 5명이다. 황이수의 봉급 500은 500 미만 조건에 맞지 않아 제외된다.
틀린 판단은 ③이 가장 큰 결과라는 점이다. 올바르게는 ③의 카디널리티는 5이며 ②의 8보다 작다.

④ X
달성률<80인 부서는 생산과 기획이다. 이를 사원과 조인하면 박찬형, 홍길동, 김철수, 김영희가 남아 카디널리티는 4이다.
틀린 판단은 ④가 가장 큰 결과라는 점이다. 올바르게는 ④의 카디널리티는 4이다.

NoteBase

오답 포인트

②를 단순히 봉급<300인 사원 2명으로만 세면 카티션 곱의 중복 효과를 놓치게 된다. 이 문항의 함정은 조인과 카티션 곱을 같은 방식으로 세는 실수와, ③에서 봉급 500을 범위에 잘못 포함하는 실수이다.

다시 나오면 이렇게 판단

카디널리티 비교 문제는 관계대수식을 왼쪽부터 따라가며 각 단계 뒤에 몇 개의 튜플이 남는지 직접 센다.
조인이면 매칭되는 튜플 수를 세고, 카티션 곱이면 먼저 전체 조합 수를 만든 뒤 선택 조건으로 걸러 낸다.

암기 포인트

선택은 조건만 남기고, 조인은 값이 맞는 튜플만 남기며, 카티션 곱은 먼저 모든 조합을 만든다.

NoteBase

문 18. <보기>의 두 트랜잭션 T1, T2의 수행에 대한 설명으로 가장 옳지 않은 것은? (단, write(a, b)는 b를 a에 대입한다는 의미이며, 데이터 X, Y, Z의 초깃값은 각각 10, 30, 100이다.)

〈보기〉

트랜잭션 T1	트랜잭션 T2
read(X) A ← X + 10 write(Y, A)	read(Y) read(Z) B ← Y + Z write(X, B)

① T1과 T2가 직렬적으로(serially) 수행되고 완료되었다면 X의 값은 120이거나 130이 된다.

② 교착상태 방지(deadlock prevention) 프로토콜의 wait-die 규칙은 T1이 먼저 시작됐다면 나중에 시작된 T2를 기다리는 것이 허용된다.

③ 엄격한 2단계 로킹(strict 2PL) 기법에 의해 T1의 read(X) 연산이 먼저 수행되면 T2의 write(X, B) 연산은 T1이 완료되거나 철회된 이후에만 가능하다.

④ 데이터 X, Y, Z가 각각 다른 사이트에 분산 저장된 경우, 3단계 완료 프로토콜(3PC)을 사용하기도 한다.

정답 3번

출제 포인트

이 문제는 직렬 스케줄의 결과, wait-die 규칙, strict 2PL의 락 유지 범위, 분산 완료 프로토콜의 용도를 구분하는 문제이다.

핵심 개념

직렬 수행은 한 트랜잭션이 모두 끝난 뒤 다른 트랜잭션이 수행되는 스케줄이다.
wait-die는 먼저 시작한 트랜잭션은 나중에 시작한 트랜잭션이 가진 락을 기다릴 수 있고, 나중에 시작한 트랜잭션은 먼저 시작한 트랜잭션의 락을 요구하면 철회되는 규칙이다.
strict 2PL은 배타 락을 트랜잭션 완료 시점까지 유지하는 기법이며, 공유 락까지 모두 완료 시점까지 유지하는 것은 rigorous 2PL에 가깝다.
3PC는 분산 트랜잭션의 완료를 조정하는 프로토콜이다.

NoteBase

선지별 해설

① O
T1 다음에 T2가 직렬 수행되면 T1이 먼저 $Y = 10 + 10 = 20$으로 바꾸고, 이어 T2가 $X = Y + Z = 20 + 100 = 120$으로 쓴다. 반대로 T2 다음에 T1이 수행되면 T2가 먼저 $X = 30 + 100 = 130$으로 쓰고, 이후 T1은 그 값을 읽어 $Y = 140$으로 바꿀 뿐 X는 130으로 남는다. 따라서 X는 120 또는 130이 된다.

② O
wait-die에서는 타임스탬프가 작은 오래된 트랜잭션은 젊은 트랜잭션이 점유한 락을 기다릴 수 있다. 따라서 T1이 먼저 시작한 오래된 트랜잭션이라면 나중에 시작한 T2를 기다리는 것이 허용된다.

③ X
틀린 표현은 "T1이 완료되거나 철회된 이후에만 가능하다"이다. strict 2PL에서 완료 시점까지 반드시 유지되는 것은 배타 락이며, T1의 read(X)는 공유 락을 사용한다. 따라서 T1이 필요한 락을 모두 확보한 뒤 X에 대한 공유 락을 먼저 해제하면 T2의 write(X, B)는 T1의 종료 전에도 가능하다. 올바르게는 "rigorous 2PL이라면 T1의 모든 락이 종료 시점까지 유지되므로 그 이후에만 가능하다" 또는 "strict 2PL에서는 T1의 공유 락 해제 이후라면 종료 전에도 가능하다"가 된다.

④ O
데이터가 여러 사이트에 분산되어 있고 하나의 트랜잭션 완료를 여러 사이트가 함께 확정해야 하는 경우 분산 완료 프로토콜을 사용한다. 3PC는 그러한 분산 완료 프로토콜의 한 예이므로 진술이 가능하다.

오답 포인트

③은 strict 2PL을 모든 락을 종료 시점까지 유지하는 기법으로 착각하게 만드는 선지이다. strict 2PL에서 끝까지 유지되는 것은 배타 락이고, 공유 락까지 모두 끝까지 유지하는 것은 rigorous 2PL이라는 점을 구분해야 한다.

다시 나오면 이렇게 판단

동시성 제어 선지는 먼저 어떤 락이 필요한지와 그 락을 언제까지 유지하는지를 본다. serial은 실행 순서별 최종값을 직접 계산하고, wait-die는 오래된 트랜잭션은 기다리고 젊은 트랜잭션은 철회된다는 규칙으로 판단하며, strict 2PL은 배타 락만 종료 시점까지 유지한다는 기준으로 판별한다.

암기 포인트

wait-die는 늙은 트랜잭션은 기다리고 젊은 트랜잭션은 죽으며, strict 2PL은 X-lock만 끝까지 잡는다.

NoteBase

문 19. 데이터베이스 시스템이 다운(down)된 시점에 시스템에 <보기>와 같은 데이터베이스 로그가 남아있다고 한다. 데이터베이스 복구를 위해 수행되어야 할 작업에 대한 설명으로 가장 옳지 않은 것은? (단, 〈T_i, X, V_1, V_2〉는 트랜잭션 T_i가 데이터 항목 X의 값을 V_1에서 V_2로 변경했음을 나타내며, 〈checkpoint L〉은 체크포인트가 수행된 시점에 L에 포함된 트랜잭션들은 활동 중(active)이었음을 나타낸다.)

<보기>

〈T_1, start〉
〈T_1, A, 1, 2〉
〈T_2, start〉
〈T_2, B, 3, 4〉
〈T_1, commit〉
〈checkpoint {T_2}〉
〈T_3, start〉
〈T_3, C, 5, 6〉
〈T_3, commit〉
〈T_2, B, 4, 5〉

① A의 값은 수정할 필요 없다.

② B의 값은 최종적으로 4로 복구되어야 한다.

③ T_2는 undo되어야 한다.

④ T_3는 redo되어야 한다.

정답 2번

출제 포인트

이 문제는 체크포인트가 있는 로그에서 커밋 여부에 따라 redo 대상과 undo 대상을 구분하고, 미완료 트랜잭션의 갱신 결과를 어디까지 되돌리는지를 묻는 문제이다.

사례 정리

핵심 사실 1: T1은 A를 1에서 2로 바꾼 뒤 커밋하고, 그 후 체크포인트가 기록되었다.
핵심 사실 2: 체크포인트 시점의 활성 트랜잭션 목록은 {T2}이므로 T2는 그때도 끝나지 않은 상태였다.
핵심 사실 3: T3는 체크포인트 이후 시작하여 C를 5에서 6으로 바꾼 뒤 커밋하였다.
핵심 사실 4: T2는 B를 3에서 4로, 다시 4에서 5로 바꿨지만 커밋 기록 없이 장애가 발생하였다.

NoteBase

핵심 개념

undo는 장애 시점까지 커밋되지 않은 트랜잭션의 변경을 이전 값으로 되돌리는 작업이다.
redo는 커밋은 되었지만 디스크에 반영이 보장되지 않는 변경을 다시 적용하는 작업이다.
체크포인트 이후 복구에서는 체크포인트 시점에 활성 상태였던 미커밋 트랜잭션은 undo 대상으로 보고, 체크포인트 이후 커밋된 트랜잭션은 redo 대상으로 본다.

적용 기준

이 사례에서는 체크포인트 이전에 이미 커밋이 끝난 T1은 복구 대상에서 제외한다. 체크포인트 목록에 남아 있고 끝내 커밋하지 못한 T2는 모든 갱신을 취소한다. 체크포인트 이후 커밋한 T3는 결과를 보장하기 위해 다시 적용한다.

선지별 해설

① O
T1은 A를 1에서 2로 변경한 뒤 체크포인트 이전에 커밋되었다. 따라서 A는 복구 시 다시 수정할 필요가 없다.

② X
B는 T2의 미커밋 갱신 결과이므로 최종적으로 4가 아니라 트랜잭션 시작 전 값인 3으로 돌아가야 한다. 틀린 표현은 "4로 복구"이고, 올바르게는 "3으로 복구"이다.

③ O
T2는 체크포인트 시점에도 활성 상태였고 장애 시점까지 커밋 기록이 없다. 따라서 T2의 갱신은 모두 undo되어야 한다.

④ O
T3는 체크포인트 이후에 시작하여 C를 5에서 6으로 바꾼 뒤 커밋하였다. 장애로 반영이 누락되었을 가능성에 대비해 T3는 redo 대상이다.

오답 포인트

②를 고르지 못하는 경우는 T2의 첫 번째 갱신 로그 <T2, B, 3, 4>만 보고 B의 최종 복구값을 4로 착각하는 경우가 많다. 그러나 미커밋 트랜잭션은 마지막 값만 취소하는 것이 아니라 그 트랜잭션의 전체 갱신을 모두 취소하므로 B는 3으로 돌아간다.

다시 나오면 이렇게 판단

로그 복구 문제는 먼저 각 트랜잭션을 커밋 완료, 체크포인트 시 활성, 장애 시 미커밋으로 나눈다. 그다음 커밋 완료는 redo 후보인지 확인하고, 미커밋은 해당 트랜잭션이 남긴 모든 갱신을 시작 전 값까지 undo한다고 판단한다.

NoteBase

암기 포인트

미커밋은 전부 undo, 체크포인트 후 커밋은 redo, 체크포인트 전 완료는 보통 건드리지 않는다.

NoteBase

문 20. 릴레이션 R(a, b, c, d, e)에서 {a, b}가 후보키(candidate-key)일 때, 이에 대한 설명으로 가장 옳지 않은 것은?

① {a}는 후보키(candidate-key)가 될 수 없다.

② {a, b}는 기본키(primary-key)가 될 수 있다.

③ {a, b, c}는 슈퍼키(superkey)이다.

④ {a, b, c, d, e}는 슈퍼키(superkey)가 아니다.

정답 4번

출제 포인트

이 문제는 후보키, 기본키, 슈퍼키의 정의와 포함관계를 묻는 문제이다.

핵심 개념

후보키는 튜플을 유일하게 식별하는 최소 속성 집합이다.
기본키는 여러 후보키 중 대표로 선택한 후보키이다.
슈퍼키는 튜플을 유일하게 식별할 수 있는 속성 집합으로, 후보키를 포함하는 상위 집합도 모두 슈퍼키이다.

선지별 해설

① O
{a, b}가 후보키이면 최소성이 성립하므로 a만으로는 모든 튜플을 유일하게 식별할 수 없다. 따라서 {a}는 후보키가 될 수 없다.

② O
기본키는 후보키 중 하나를 선택한 키이다. {a, b}가 후보키이므로 기본키로 채택될 수 있다.

③ O
후보키는 곧 슈퍼키이므로, 그 상위 집합인 {a, b, c}도 슈퍼키이다.

④ X
틀린 부분은 "슈퍼키가 아니다"이다. 후보키 {a, b}를 포함하는 {a, b, c, d, e}는 전체 속성 집합이라도 튜플을 유일하게 식별할 수 있으므로 슈퍼키이다.
올바르게는 "{a, b, c, d, e}는 슈퍼키이다."이다.

NoteBase

오답 포인트

후보키의 최소성과 슈퍼키의 포괄성을 뒤섞으면 ④를 맞는 말처럼 보기 쉽다. 최소성은 후보키를 판단하는 조건이고, 슈퍼키는 후보키를 포함하기만 해도 성립한다.

다시 나오면 이렇게 판단

어떤 집합이 후보키로 주어지면 먼저 그 집합의 진부분집합은 후보키가 될 수 없고, 그 집합을 포함하는 상위 집합은 모두 슈퍼키가 된다는 두 규칙을 적용하면 된다.

암기 포인트

후보키는 최소 슈퍼키이고, 슈퍼키는 후보키를 포함한 모든 상위 집합이다.

NoteBase