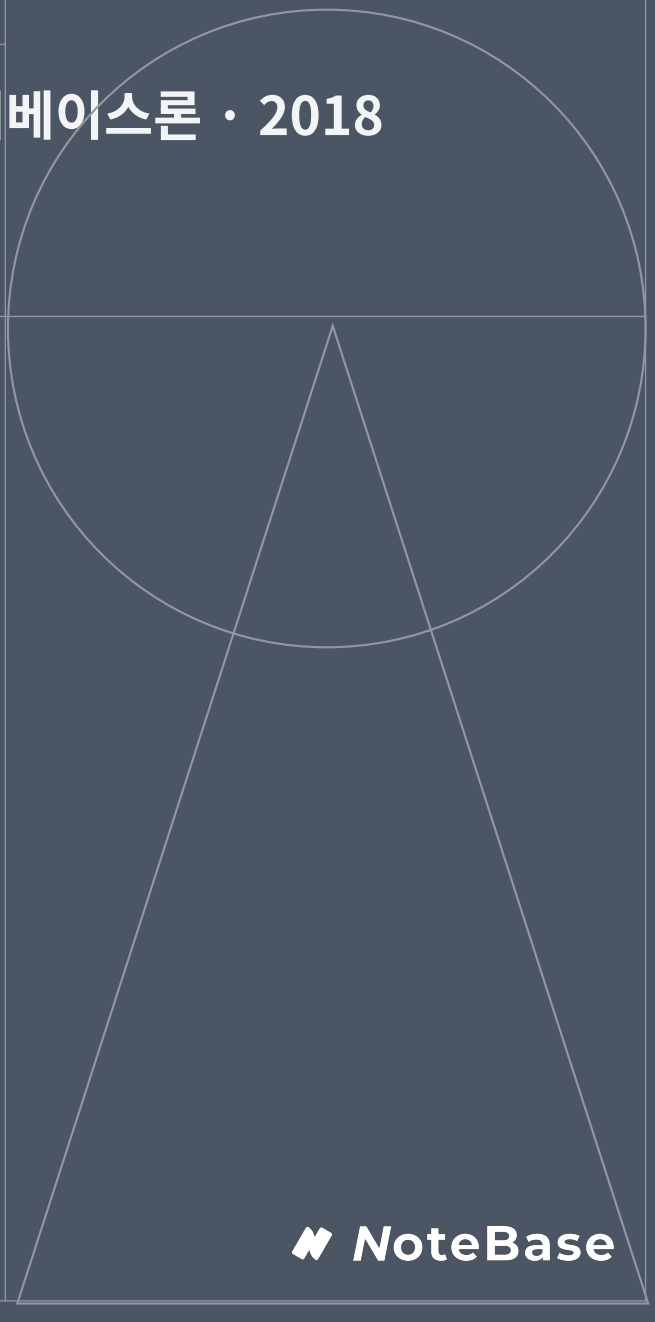
기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2018



NoteBase

기출OX 해설집

공무원/국가직/7급 · 데이터베이스론 · 2018

공무원/국가직/7급 · 데이터베이스론 · 2018

문 1. 파일처리 시스템에서 데이터 중복의 단점에 대한 설명으로 옳지 않은 것은?

① 같은 데이터가 여러 곳에 중복되어 있어 동일 수준의 보안이 가능하다.

② 데이터 저장공간에 대한 추가 비용이 소요된다.

③ 데이터 간의 불일치로 인해 데이터 일관성이 결여된다.

④ 데이터 갱신 시 중복된 모든 데이터를 찾아내어 갱신해야 하므로 갱신 비용이 추가된다.

정답 1번

출제 포인트

이 문제는 파일처리 시스템에서 데이터 중복이 가져오는 대표적 부작용과 보안상의 한계를 구별하는 문제이다.

핵심 개념

데이터 중복은 같은 데이터가 여러 파일이나 여러 위치에 반복 저장되는 상태이다.
데이터 일관성은 같은 데이터를 어디서 보아도 같은 값으로 유지되는 성질이다.
파일처리 시스템에서는 중복된 데이터를 각각 관리하므로 저장공간 낭비, 갱신 비용 증가, 일관성 저하, 보안 통제의 어려움이 생긴다.

선지별 해설

① X: 틀린 부분은 "동일 수준의 보안이 가능하다"이다. 데이터가 여러 곳에 흩어져 있으면 각 파일마다 보안 통제를 따로 적용해야 하므로 오히려 동일한 수준의 보안을 유지하기 어렵다. 올바르게는 "같은 데이터가 여러 곳에 중복되어 있어 일관된 보안 관리가 어렵다"이다.
② O: 중복 저장은 같은 내용을 여러 번 저장하므로 저장공간이 추가로 필요하다. 데이터 중복의 대표적 단점이다.
③ O: 중복된 데이터 중 일부만 수정되면 서로 다른 값이 남을 수 있으므로 데이터 일관성이 깨진다. 이것이 데이터 불일치 문제이다.
④ O: 데이터 갱신 시 중복된 모든 데이터를 함께 찾아 수정해야 하므로 작업량과 비용이 증가한다. 일부만 갱신하면 일관성 문제도 함께 발생한다.

NoteBase

오답 포인트

①은 "여러 곳에 저장된다"는 사실을 보안 강화로 착각하게 만드는 선지이다. 데이터 중복은 백업 개념과 다르며, 파일처리 시스템에서는 중복될수록 보안 정책을 통일하기가 더 어려워진다.

다시 나오면 이렇게 판단

데이터 중복의 효과를 묻는 문제에서는 먼저 저장공간, 갱신 비용, 일관성, 보안 관리의 네 가지 축으로 나누어 본다. 이때 "중복이 많을수록 관리가 쉬워진다"는 방향의 표현은 대부분 반대로 판단하면 된다.

암기 포인트

데이터 중복은 공간 낭비, 갱신 부담, 일관성 저하, 보안 관리 어려움으로 이어진다.

NoteBase

문 2. 다음 데이터베이스 설계의 주요 단계를 순서대로 바르게 나열한 것은?

ㄱ. 정규화 ㄴ. 물리적 설계
ㄷ. 요구사항 수집 및 분석 ㄹ. 개념적 설계
ㅁ. 데이터베이스 튜닝 ㅂ. 논리적 설계

① ㄱ, ㄷ, ㄹ, ㅂ, ㄴ, ㅁ
② ㄱ, ㄷ, ㅂ, ㄹ, ㅁ, ㄴ
③ ㄷ, ㄹ, ㅂ, ㄱ, ㄴ, ㅁ
④ ㄷ, ㅂ, ㄹ, ㄱ, ㅁ, ㄴ

정답 3번

출제 포인트

이 문제는 데이터베이스 설계의 전체 흐름에서 요구사항 수집 및 분석, 개념적 설계, 논리적 설계, 정규화, 물리적 설계, 데이터베이스 튜닝의 선후관계를 묻는 문제이다.

올바른 순서

1. 요구사항 수집 및 분석(ㄷ) - 사용자 요구와 업무 규칙을 정리하는 단계이다.
2. 개념적 설계(ㄹ) - 요구사항을 ER 모델 등으로 표현하여 개체와 관계를 구조화하는 단계이다.
3. 논리적 설계(ㅂ) - 개념 모델을 관계 스키마 등 DBMS 독립적인 논리 구조로 변환하는 단계이다.
4. 정규화(ㄱ) - 논리 스키마의 중복과 이상 현상을 줄이도록 릴레이션을 정제하는 단계이다.
5. 물리적 설계(ㄴ) - 저장 구조, 파일 조직, 인덱스 등 실제 저장 방식을 결정하는 단계이다.
6. 데이터베이스 튜닝(ㅁ) - 구축된 데이터베이스의 성능을 점검하고 접근 경로와 저장 구조를 개선하는 단계이다.

왜 이 순서인가: 개념적 설계는 요구사항 분석 결과를 바탕으로 하고, 논리적 설계와 정규화가 끝나야 물리적 저장 구조를 구체화할 수 있으며, 튜닝은 실제 물리 구조와 성능 결과를 기준으로 수행된다.

NoteBase

선지별 해설

① X
- 틀린 부분: 정규화(ㄱ)가 요구사항 수집 및 분석(ㄷ)보다 앞에 배치되어 있다.
- 이유: 정규화는 논리적 스키마가 나온 뒤 중복과 이상 현상을 줄이는 작업이므로 설계 첫 단계에 올 수 없다.
- 올바르게는: ㄷ, ㄹ, ㅂ, ㄱ, ㄴ, ㅁ이다.

② X
- 틀린 부분: 정규화(ㄱ)가 가장 앞에 있고, 물리적 설계(ㄴ)가 데이터베이스 튜닝(ㅁ)보다 뒤에 있다.
- 이유: 물리적 설계가 먼저 이루어져야 그 저장 구조를 대상으로 성능 조정인 튜닝을 수행할 수 있다.
- 올바르게는: ㄷ, ㄹ, ㅂ, ㄱ, ㄴ, ㅁ이다.

③ O
- 이유: 요구사항 분석 후 개념 모델을 만들고, 이를 논리 모델로 변환한 다음 정규화로 스키마를 다듬고, 그 뒤 물리적 설계와 튜닝으로 이어지는 정상적인 흐름이다.

④ X
- 틀린 부분: 논리적 설계(ㅂ)가 개념적 설계(ㄹ)보다 앞에 있고, 데이터베이스 튜닝(ㅁ)이 물리적 설계(ㄴ)보다 앞에 있다.
- 이유: 개념 모델이 논리 모델의 전제이고, 물리적 저장 구조가 결정되어야 튜닝이 가능하다.
- 올바르게는: ㄷ, ㄹ, ㅂ, ㄱ, ㄴ, ㅁ이다.

오답 포인트

정규화를 독립된 맨 앞 단계로 외우거나, 물리적 설계와 튜닝의 순서를 바꾸면 틀리기 쉽다. 정규화는 논리 설계 결과를 다듬는 작업이고, 튜닝은 물리 구조와 성능 결과를 보고 조정하는 후행 단계이다.

다시 나오면 이렇게 판단

데이터베이스 설계 절차는 요구사항 파악, 개념 모델 작성, 논리 스키마 변환, 정규화, 물리 저장 구조 결정, 성능 튜닝 순서로 판단한다. 각 단계가 바로 앞 단계의 산출물을 입력으로 받는지 확인하면 선후관계가 정리된다.

암기 포인트

요개논정물튜의 흐름으로 묶이면 순서가 정리된다.

NoteBase

문 3. 함수적 종속성과 정규화에 대한 설명으로 옳지 않은 것은?

① 함수적 종속성 X→Y는 릴레이션 내 임의의 두 튜플에서 속성 X의 값이 같을 경우, 속성 Y의 값도 항상 같음을 의미한다.

② 함수적 종속성은 릴레이션 상태(외연)의 특성이다.

③ 제2정규형에서 릴레이션의 모든 비주요 속성들은 릴레이션의 기본키에 완전하게 함수적으로 종속한다.

④ 정규화는 함수적 종속성과 기본키를 기반으로 주어진 릴레이션 스키마를 분석하는 과정이다.

정답 2번

출제 포인트

이 문제는 함수적 종속성의 성질이 릴레이션 스키마에 대한 제약인지, 현재의 릴레이션 상태에 대한 관찰인지와 제2정규형의 의미를 구별할 수 있는지를 묻는다.

핵심 개념

함수적 종속성은 속성 집합 X의 값이 같으면 속성 Y의 값이 반드시 하나로 정해지는 스키마 수준의 제약이다.
제2정규형은 모든 비주요 속성이 기본키 전체에 완전 함수 종속하고 부분 함수 종속이 없는 상태이다.
정규화는 함수적 종속성과 키를 기준으로 스키마를 분석하고 분해하여 이상 현상을 줄이는 과정이다.

선지별 해설

1번: O. 함수적 종속성 $X \rightarrow Y$는 임의의 두 튜플에서 X 값이 같으면 Y 값도 같아야 함을 뜻하므로 설명이 맞다.
2번: X. 틀린 부분은 "릴레이션 상태(외연)의 특성"이라는 표현이다. 함수적 종속성은 현재 들어 있는 데이터 몇 건만 보고 우연히 성립하는 성질이 아니라, 해당 릴레이션 스키마가 항상 만족해야 하는 제약이다. 올바르게는 "함수적 종속성은 릴레이션 스키마(내포)에서 정의되는 의미적 제약이다."라고 고쳐야 한다.
3번: O. 제2정규형은 모든 비주요 속성이 기본키에 완전 함수 종속해야 한다는 정규형이므로 설명이 맞다.
4번: O. 정규화는 함수적 종속성과 기본키를 바탕으로 스키마를 분석하는 과정이므로 설명이 맞다.

오답 포인트

함수적 종속성을 현재 데이터 상태에서 관찰되는 규칙으로 착각하면 2번에 끌린다. 시험에서는 "스키마의 제약"과 "현재 릴레이션 상태의 모습"을 뒤바꾸는 표현이 자주 함정으로 나온다.

다시 나오면 이렇게 판단

함수적 종속성 관련 선지가 나오면 먼저 그것이 스키마 차원의 제약을 말하는지, 현재 인스턴스의 상태를 말하는지 구분한다. 정규형 선지는 각 정규형이 제거하려는 종속 유형이 무엇인지로 판단한다.

NoteBase

암기 포인트

함수적 종속성은 스키마의 제약, 제2정규형은 부분 함수 종속 제거, 정규화는 종속성과 키를 기준으로 한 스키마 분해이다.

NoteBase

문 4. 위치 기반 서비스(Location-Based Services)를 제공하는 시스템의 요소로 옳지 않은 것은?

① 푸시(push) 기반 서비스
② 최근접 주변 질의
③ TSQL2 언어
④ 윈도우 질의

정답 3번

출제 포인트

위치 기반 서비스 시스템에서 쓰이는 대표 서비스와 공간 질의를 구별하고, 시간 데이터 질의 언어와 혼동하지 않는지를 묻는 문제이다.

핵심 개념

푸시 기반 서비스는 사용자의 위치나 상황에 맞추어 시스템이 먼저 정보를 전달하는 서비스이다.
최근접 주변 질의는 현재 위치를 기준으로 가장 가까운 객체를 찾는 공간 질의이다.
윈도우 질의는 지정한 영역 안에 포함되는 객체를 찾는 공간 질의이다.
TSQL2는 시간 정보를 포함한 데이터를 질의하기 위한 시간 데이터베이스 언어이다.

선지별 해설

1번 O: 푸시 기반 서비스는 사용자의 현재 위치나 이동 상태에 따라 쿠폰, 교통, 경보 같은 정보를 시스템이 먼저 보내는 방식이므로 위치 기반 서비스의 구성 요소로 볼 수 있다.
2번 O: 최근접 주변 질의는 사용자의 위치에서 가장 가까운 시설이나 객체를 찾는 질의이므로 위치 기반 서비스에서 매우 자주 쓰이는 공간 질의이다.
3번 X: 틀린 부분은 "TSQL2 언어"를 위치 기반 서비스의 요소로 본 점이다. TSQL2는 시간 데이터베이스를 위한 질의 언어로서 시간 축의 변화와 이력을 다루는 데 초점이 있다. 올바르게는 위치 기반 서비스의 질의 요소로 최근접 주변 질의나 윈도우 질의 같은 공간 질의가 들어간다.
4번 O: 윈도우 질의는 지도상의 특정 범위 안에 있는 객체를 찾는 질의이므로 위치 기반 서비스 시스템에서 사용하는 전형적인 공간 질의이다.

오답 포인트

공간 데이터와 시간 데이터를 함께 다루는 시스템이라는 이유만으로 위치 기반 서비스와 시간 데이터베이스 언어를 같은 범주로 묶으면 틀린다. 3번은 "위치"가 아니라 "시간"을 처리 대상으로 하는 언어라는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

선지가 서비스인지, 공간 질의인지, 질의 언어인지부터 분류하면 판단이 빨라진다. 위치 기반 서비스 문제에서 주변, 거리, 영역 같은 표현이 나오면 공간 질의를 떠올리고, 시간 이력이나 유효 시간 표현이 나오면 시간 데이터베이스 개념으로 분리한다.

암기 포인트

LBS는 위치와 공간 질의, TSQL2는 시간과 이력 처리이다.

NoteBase

문 5. SQL 질의문은 동치인 관계대수식으로 변환할 수 있다. 다음 SQL 질의문을 관계대수식으로 변환한 것 중 옳은 것은? (단, 제품 테이블 P, 판매 테이블 S, 집계함수 표현을 위한 관계대수 연산자 기호는 $\mathcal{J}$로 한다)

Schema: S(sId, sDate, pId, sAmount)
P(pId, pName)

```
SQL: SELECT SUM(sAmount)
     FROM P, S
     WHERE S.pId = P.pId AND P.pName='P1'
     GROUP BY sDate;
```

① $\mathfrak{I}_{SUM(sAmount)}($
$\pi_{P.pName}(\sigma_{P.pName='P1'} \bowtie_{S.pId=P.pId} P)$
$)$

② ${}_{sDate}\mathfrak{I}_{SUM(sAmount)}($
$\pi_{S.sAmount}(\sigma_{P.pName='P1'}(S \bowtie_{S.pId=P.pId} P))$
$)$

③ $\mathfrak{I}_{SUM(sAmount)}($
$S \bowtie_{S.pId=P.pId}(\sigma_{P.pName='P1'}(P))$
$)$

④ ${}_{sDate}\mathfrak{I}_{SUM(sAmount)}($
$\sigma_{P.pName='P1'}(P) \bowtie_{P.pId=S.pId} S$
$)$

정답 4번

출제 포인트

이 문제는 SQL의 선택 조건, 조인 조건, 그리고 GROUP BY sDate가 관계대수식의 집계 연산자에 정확히 반영되는지를 묻는 문제이다.

핵심 개념

선택은 P.pName='P1' 조건으로 제품 테이블 P에서 필요한 튜플만 남기는 단계이다.
조인은 S.pId=P.pId 조건으로 판매 정보와 제품 정보를 결합하는 단계이다.
그룹 집계는 sDate를 그룹 속성으로 두고 각 날짜별로 SUM(sAmount)를 계산하는 단계이므로, sDate와 sAmount가 집계 직전까지 유지되어야 한다.

NoteBase

선지별 해설

① X
집계 연산자에 그룹 속성 sDate가 없고, P.pName만 투영하여 SUM(sAmount)를 계산할 수 없는 식이 된다.
틀린 표현은 P.pName만 남긴 투영 부분이다.
올바르게는 sDate와 sAmount가 유지된 상태에서 P.pName='P1' 선택과 pId 조인을 거친 뒤 sDate별 SUM(sAmount)를 계산해야 한다.

② X
sDate로 그룹화하는 표기는 있으나, π_{S.sAmount}로 sDate를 제거하므로 GROUP BY sDate를 수행할 수 없다.
틀린 표현은 S.sAmount만 투영한 부분이다.
올바르게는 그룹 기준 속성 sDate를 함께 유지하거나 투영을 생략한 상태에서 sDate별 SUM(sAmount)를 계산해야 한다.

③ X
P에서 P1만 선택한 뒤 S와 조인한 단계는 맞지만, 집계 연산자에 sDate가 없어 날짜별 합계가 아니라 전체 합계만 계산한다.
틀린 표현은 그룹 속성 없이 SUM(sAmount)만 둔 집계 부분이다.
올바르게는 sDate를 그룹 속성으로 명시한 뒤 집계해야 하므로 sDate를 포함한 그룹 집계식이 되어야 한다.

④ O
P에서 pName='P1'인 튜플을 먼저 선택하고, 그 결과를 S와 pId로 조인한 뒤, sDate를 기준으로 그룹화하여 SUM(sAmount)를 계산한다.
선택, 조인, 그룹 집계가 SQL의 WHERE와 GROUP BY에 맞게 모두 반영되어 동치인 관계대수식이 된다.

오답 포인트

이 문제의 함정은 집계 함수 SUM(sAmount)만 보고 식을 고르는 데 있다. GROUP BY sDate가 있으면 집계 연산자에 sDate가 반드시 함께 표시되어야 하고, 집계 전에 sDate가 투영으로 사라지면 바로 오답이 된다.

다시 나오면 이렇게 판단

SQL을 관계대수식으로 바꿀 때는 선택 조건이 어느 테이블 속성에 붙는지, 조인 속성이 무엇인지, GROUP BY 속성이 집계 연산자에 남아 있는지를 순서대로 확인하면 된다. 집계 문제에서는 집계 대상 속성과 그룹 기준 속성이 집계 직전까지 모두 살아 있는지를 보면 된다.

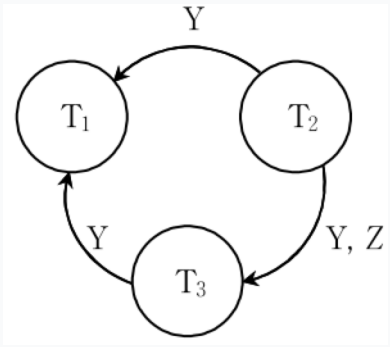
NoteBase

암기 포인트

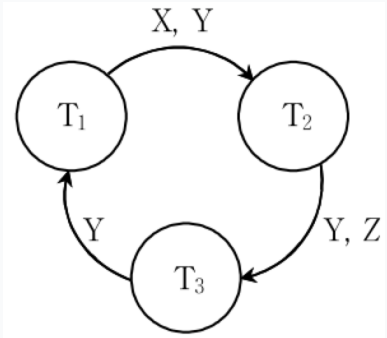
GROUP BY가 있으면 집계식에 그룹 속성이 함께 들어가며, 그룹 속성을 먼저 지우면 집계 결과도 성립하지 않는다.

NoteBase

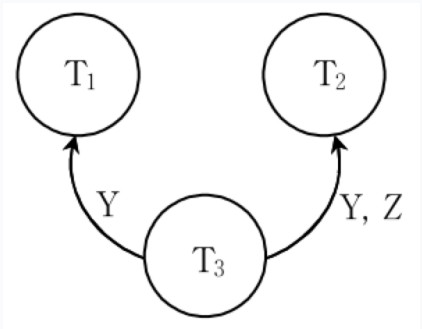
문 6. 다음 트랜잭션의 실행그래프 및 그와 동치인 직렬 가능한 스케줄의 쌍들이다. 이 쌍들 중 옳지 않은 것은?

① 

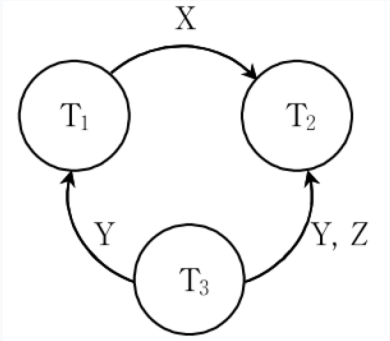
동치 직렬 스케줄: T2→T3→T1

② 

동치 직렬 스케줄: T3→T1→T2

③ 

동치 직렬 스케줄: T3→T2→T1

④ 

동치 직렬 스케줄: T3→T1→T2

정답 2번

NoteBase

출제 포인트

이 문제는 선행그래프의 간선 방향을 읽어 충돌 직렬가능성 여부와 가능한 동치 직렬 스케줄을 판단하는 문제이다.

핵심 개념

선행그래프에 사이클이 없으면 그 그래프의 위상정렬 순서가 동치 직렬 스케줄이 된다. 반대로 사이클이 있으면 충돌 직렬가능하지 않으므로 동치 직렬 스케줄이 존재하지 않는다.

선지별 해설

① O. 간선은 T2→T3, T2→T1, T3→T1이다. 이 순서를 모두 만족하는 위상정렬이 T2→T3→T1이므로 제시된 직렬 스케줄은 맞다.

② X. 간선은 T1→T2, T2→T3, T3→T1이어서 T1→T2→T3→T1의 사이클이 생긴다. 사이클이 있는 선행그래프는 충돌 직렬가능하지 않으므로 동치 직렬 스케줄이 존재하지 않는다. 틀린 표현은 '동치 직렬 스케줄: T3→T1→T2'이고, 올바르게는 '이 그래프는 동치 직렬 스케줄이 없다'이다.

③ O. 간선은 T3→T1, T3→T2이다. T1과 T2 사이에는 선후 제약이 없으므로 T3만 먼저 오면 된다. 따라서 T3→T2→T1은 가능한 동치 직렬 스케줄이다.

④ O. 간선은 T3→T1, T1→T2, T3→T2이다. 이를 만족하는 위상정렬은 T3→T1→T2뿐이므로 제시된 직렬 스케줄은 맞다.

오답 포인트

②처럼 각 간선은 맞게 읽고도 마지막에 아무 직렬 순서나 적어 버리는 실수가 자주 나온다. 선행그래프에서 가장 먼저 볼 것은 개별 간선이 아니라 사이클 존재 여부이다. 사이클이 보이면 특정 직렬 순서를 찾는 단계로 가지 않고 곧바로 '동치 직렬 스케줄이 없다'로 판단된다.

다시 나오면 이렇게 판단

선행그래프 문제는 먼저 사이클 유무를 확인한다. 사이클이 있으면 충돌 직렬가능하지 않다. 사이클이 없으면 간선 방향을 모두 만족하는 위상정렬을 하나 구해 제시된 직렬 스케줄과 비교하면 된다.

암기 포인트

선행그래프는 순환이면 불가, 비순환이면 위상정렬이다.

NoteBase

문 7. 다음 '사원' 테이블을 생성하는 SQL문에서 부서의 속성값을 '인사', '연구', '영업'으로 제한하고자 한다. ㉠, ㉡에 들어갈 내용으로 옳은 것은?

```
CREATE TABLE 사원
(사번        NUMBER      NOT NULL,
 이름        CHAR(10),
 직급        CHAR(10),
 부서        CHAR(10)    ㉠    (부서  ㉡  ('인사', '연구', '영업')));
```

① ㉠: UNIQUE, ㉡: ON

② ㉠: CHECK, ㉡: ON

③ ㉠: UNIQUE, ㉡: IN

④ ㉠: CHECK, ㉡: IN

정답 4번

출제 포인트

이 문제는 CREATE TABLE 문에서 값의 허용 범위를 제한하는 제약조건과 그 안에서 쓰는 비교 구문을 구별할 수 있는지를 묻는다.

핵심 개념

CHECK는 컬럼 값이 특정 조건을 만족하는지만 검사하는 제약조건이다. UNIQUE는 컬럼 값의 중복 여부를 막는 제약조건이다. IN은 값이 제시된 목록에 포함되는지를 검사하는 연산자이고, ON은 JOIN에서 결합 조건을 지정할 때 주로 쓰는 구문이다.

NoteBase

선지별 해설

1번: X. ㉠의 "UNIQUE"가 틀린 표현이다. UNIQUE는 같은 부서명이 여러 행에 반복되는 것을 막는 제약이지, 부서값을 '인사', '연구', '영업'으로만 제한하는 제약이 아니다. ㉠은 CHECK로 고쳐야 한다. ㉡의 "ON"도 틀리다. 허용 목록 안에 포함되는지를 검사할 때는 IN을 써서 CHECK (부서 IN ('인사', '연구', '영업')) 형태가 된다.
2번: X. ㉠의 CHECK는 맞다. 하지만 ㉡의 "ON"이 틀리다. ON은 조인 조건에서 쓰는 구문이므로 여기서는 IN으로 고쳐야 한다.
3번: X. ㉡의 IN은 맞다. 하지만 ㉠의 "UNIQUE"가 틀리다. 이 문제는 값의 중복 금지가 아니라 값의 허용 범위 제한이므로 CHECK로 고쳐야 한다.
4번: O. CHECK는 컬럼 값이 조건을 만족하는지 검사하는 제약조건이고, IN은 값이 제시된 목록에 속하는지 검사한다. 따라서 SQL문은 CHECK (부서 IN ('인사', '연구', '영업'))가 되어야 한다.

오답 포인트

UNIQUE와 CHECK를 모두 '제약조건'으로만 외우면 헷갈린다. UNIQUE는 중복 금지, CHECK는 조건 만족 여부라는 역할 차이를 구분해야 한다. 또 ON을 조건이라는 말만 보고 고르면 틀리기 쉽고, 목록 제한에는 IN이 들어간다는 점을 함께 잡아야 한다.

다시 나오면 이렇게 판단

선지가 '특정 값들만 허용한다'는 의미이면 먼저 CHECK 제약조건을 떠올리고, 괄호 안에서는 목록 포함 여부를 검사하는 IN을 연결한다. 반대로 '중복을 허용하지 않는다'는 의미일 때만 UNIQUE를 고른다.

암기 포인트

허용값 목록 제한은 CHECK + IN, 중복 금지는 UNIQUE이다.

NoteBase

문 8. 2개의 계좌 A1, A2의 잔액이 각각 2,000,000원, 3,000,000원이라고 하자. 이 상태에서 각 계좌의 관리를 수행하는 트랜잭션 T1 및 T2의 실행이 다음과 같을 때, 발생한 현상으로 옳은 것은?

1: T2가 계좌 A2에 2,500,000원을 더해 5,500,000원이 됨
2: T1이 계좌 A1에 1,500,000원을 더해 3,500,000원이 됨
3: T2가 계좌 A1의 잔액(3,500,000원)이 충분하다고 판단하여 A1에서 A2로 2,500,000원 이체를 실행함
4: T1이 알 수 없는 이유로 트랜잭션의 수행이 실패함

① 오손 판독(dirty read)
② 반복할 수 없는 판독(non-repeatable read)
③ 부정확한 요약(incorrect summary)
④ 유령 판독(phantom read)

정답 1번

출제 포인트

이 문제는 트랜잭션이 아직 확정되지 않은 값을 다른 트랜잭션이 읽었을 때 발생하는 동시성 이상 현상을 사례에 적용할 수 있는지를 묻는 문제이다.

사례 정리

초기 잔액은 A1이 2,000,000원, A2가 3,000,000원이다.
T1이 A1에 1,500,000원을 더해 A1을 3,500,000원으로 만든 뒤 아직 확정되지 않은 상태이다.
T2는 그 3,500,000원을 읽고 A1에서 A2로 2,500,000원을 이체한 뒤, 나중에 T1은 실패한다.

핵심 개념

오손 판독은 아직 commit되지 않은 다른 트랜잭션의 변경 값을 읽는 현상이다.
적용 요건은 다른 트랜잭션의 미확정 데이터 읽기와 그 후 원래 트랜잭션의 rollback 또는 실패 가능성이다.

적용 기준

이 사례의 핵심은 T2가 T1이 확정하지 않은 A1=3,500,000원을 근거로 이체를 진행했다는 점이다.
즉 읽은 값이 나중에 무효가 될 수 있는 미확정 값인지가 판단 기준이다.

NoteBase

선지별 해설

1번: O. T2는 T1이 아직 확정하지 않은 A1의 잔액 3,500,000원을 읽고 이를 근거로 이체를 수행했다. 이후 T1이 실패하므로 T2가 읽은 값은 더러워진 값이었고, 이는 오손 판독에 해당한다.
2번: X. 반복할 수 없는 판독은 같은 트랜잭션이 같은 데이터를 두 번 읽을 때 값이 달라지는 현상이다. 이 사례는 동일 데이터의 재읽기 문제가 아니라 미확정 값을 읽은 문제이므로, 틀린 표현은 '반복할 수 없는 판독'이고 올바르게는 'commit 전 데이터를 읽은 오손 판독'이다.
3번: X. 부정확한 요약은 여러 행이나 여러 값의 합계·평균 같은 요약 계산 중 일부 값만 갱신되어 전체 요약이 일관되지 않게 되는 현상이다. 이 사례는 요약 계산이 아니라 미확정 단일 값 읽기가 핵심이므로, 틀린 표현은 '부정확한 요약'이고 올바르게는 '미확정 잔액을 읽은 오손 판독'이다.
4번: X. 유령 판독은 같은 범위 질의를 반복했을 때 삽입·삭제된 행 때문에 결과 집합이 달라지는 현상이다. 이 사례에는 범위 검색이나 행의 추가·삭제가 없으므로, 틀린 표현은 '유령 판독'이고 올바르게는 'rollback될 수 있는 값을 읽은 오손 판독'이다.

오답 포인트

잔액이 바뀌었다는 겉모습만 보고 반복할 수 없는 판독으로 오해하기 쉽다. 그러나 이 문제의 본질은 같은 값을 다시 읽었는지가 아니라 commit되지 않은 값을 읽었는지에 있다.

다시 나오면 이렇게 판단

동시성 이상 문제는 먼저 어떤 트랜잭션이 읽은 값이 확정된 값인지부터 본다. 미확정 값을 읽으면 오손 판독, 같은 값을 다시 읽어 값이 달라지면 반복할 수 없는 판독, 범위 결과 집합이 달라지면 유령 판독으로 판단한다.

암기 포인트

commit 전 값 읽기=오손 판독, 같은 값 재조회 후 변경=반복할 수 없는 판독, 범위 결과 집합 변경=유령 판독이다.

NoteBase

문 9. 로킹(locking)을 이용한 DBMS 동시성 제어 기법의 설명으로 옳은 것은?

① 2단계 로킹 프로토콜(two-phase locking protocol)의 확대 단계에서는 로크를 해제할 수 없지만, 축소 단계에는 로크를 해제한 후 같은 항목에 대하여 다시 로크를 획득할 수 있다.

② 로킹 기법을 사용하여 트랜잭션의 동시성을 제어하면 항상 직렬 가능(serializable)한 스케줄을 보장한다.

③ 다중 단위 로킹(multiple granularity locking) 프로토콜에서 SIX(intention-shared-exclusive) 로크가 걸려있는 노드에 IS(intention-shared) 로크를 추가로 걸 수 있다.

④ 2단계 로킹 프로토콜을 활용하면 불필요한 교착상태(deadlock)를 회피할 수 있어서 트랜잭션의 철회(rollback)를 막을 수 있다.

정답 3번

출제 포인트

이 문제는 2단계 로킹과 다중 단위 로킹의 단계 규칙 및 호환성 규칙을 구별할 수 있는지를 묻는다.

핵심 개념

2단계 로킹은 확대 단계에서는 로크만 획득하고 축소 단계에서는 로크만 해제하는 규칙으로 충돌 직렬 가능성을 확보하는 기법이다.
다중 단위 로킹은 트리 구조 자원에 IS, IX, S, SIX, X 로크를 두어 상위 노드에서 의도를 먼저 표시하는 기법이다. SIX는 해당 노드 자체에는 공유 잠금을, 하위 노드에는 배타 잠금을 걸 의도를 함께 나타내는 로크이므로 IS와는 호환되지만 S, IX, SIX, X와는 함께 둘 수 없다.

선지별 해설

1번: X. 2단계 로킹의 확대 단계에서는 로크를 획득만 할 수 있고 해제할 수 없다. 축소 단계에서는 반대로 로크를 해제만 할 수 있으므로, 한 번 해제한 뒤 같은 항목에 다시 로크를 획득할 수 있다는 부분이 틀리다. 올바르게는 축소 단계에 들어가면 어떤 항목에도 새 로크를 획득할 수 없다.
2번: X. 로킹을 사용한다는 사실만으로 항상 직렬 가능 스케줄이 보장되지는 않는다. 임의의 lock-unlock만으로는 비직렬 가능 스케줄이 나올 수 있고, 직렬 가능성을 보장하는 대표 규칙은 2단계 로킹 프로토콜이다. 올바르게는 적절한 로킹 프로토콜, 특히 2단계 로킹을 따를 때 충돌 직렬 가능성이 보장된다.
3번: O. 다중 단위 로킹의 호환성 행렬에서 SIX와 IS는 서로 호환된다. 따라서 어떤 노드에 SIX 로크가 걸려 있어도 다른 트랜잭션은 그 노드에 IS 로크를 추가로 걸어 하위 노드에 공유 로크를 걸겠다는 의도를 표시할 수 있다.
4번: X. 2단계 로킹은 직렬 가능성을 높이지만 교착상태를 자동으로 회피하지는 않는다. 여러 트랜잭션이 서로 상대가 가진 로크 해제를 기다리면 deadlock이 발생할 수 있으므로 철회가 전혀 일어나지 않는다는 결론도 틀리다. 올바르게는 2단계 로킹은 교착상태를 유발할 수 있어 별도의 예방, 회피, 검출 기법이 필요하다.

NoteBase

오답 포인트

2단계 로킹이 직렬 가능성을 높인다는 사실과 교착상태를 막는다는 주장을 혼동하면 4번으로 흔들리기 쉽다. 또 SIX를 단순한 공유 로크의 확장판으로만 보면 IS와의 호환 여부를 틀리기 쉽다.

다시 나오면 이렇게 판단

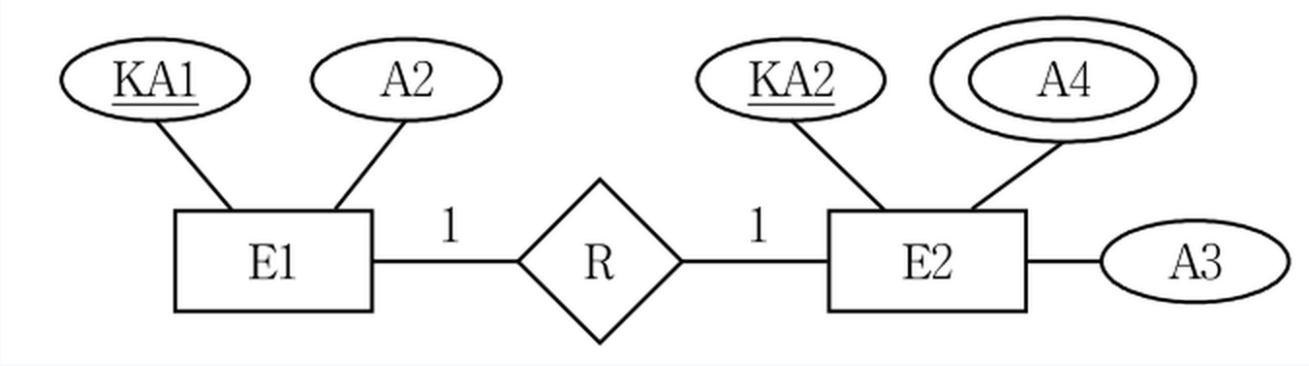
로킹 문제는 먼저 단계 규칙과 호환성 규칙을 분리해서 본다. 2단계 로킹은 확대 단계에서는 획득만, 축소 단계에서는 해제만 가능하다고 판단하고, 다중 단위 로킹은 IS, IX, S, SIX, X 호환성 행렬에서 해당 조합이 함께 가능한지만 확인하면 된다.

암기 포인트

2PL은 획득 후 해제, 해제 후 재획득 불가이다. SIX는 공유+하위 배타 의도 로크이므로 IS와는 호환된다.

NoteBase

문 10. 다음 ERD(Entity-Relationship Diagram)를 관계형 데이터베이스 스키마로 변환한 것으로 옳지 않은 것은?(단, 밑줄은 기본키이다)



① E1(KA1, A2)
E2(KA2, A3, KA1)
E21(KA2, A4)

② E1(KA1, A2, KA2)
E2(KA2, A3)
E21(KA2, A4)

③ E1(KA1, A2)
E2(KA2, A3)
R(KA1, KA2)
E21(KA2, A4)

④ E1(KA1, A2, KA2)
E2(KA2, A3, A4)

정답 4번

출제 포인트

이 문제는 ERD에서 1:1 관계와 다치 속성을 관계형 스키마로 변환하는 규칙을 정확히 구분할 수 있는지를 묻는다.

NoteBase

선지별 해설

1번: O. E1과 E2의 관계는 1:1이므로 E1의 기본키 KA1을 E2에 외래키로 포함시키는 방식이 가능하다. 또한 A4는 이중 타원으로 표시된 다치 속성이므로 E2에 직접 넣지 않고 E21(KA2, A4)처럼 별도 릴레이션으로 분리한 점이 옳다.
2번: O. 1:1 관계는 반대로 E2의 기본키 KA2를 E1에 외래키로 포함시키는 방식으로도 변환할 수 있다. A4를 E21(KA2, A4)로 분리했으므로 다치 속성 처리도 맞다.
3번: O. 1:1 관계를 별도의 릴레이션 R(KA1, KA2)로 두는 방법도 가능하다. 두 엔터티의 기본키를 관계 릴레이션에 포함시키는 방식이며, A4를 별도 릴레이션 E21(KA2, A4)로 분리한 점도 옳다.
4번: X. A4는 E2의 다치 속성이므로 E2(KA2, A3, A4)처럼 같은 릴레이션 안에 두면 한 튜플에 여러 값을 담게 되어 1정규형에 맞지 않는다. 틀린 표현은 E2(KA2, A3, A4)이고, 올바르게는 E2(KA2, A3)와 E21(KA2, A4)로 분리해야 한다. 1:1 관계를 반영하려면 E1(KA1, A2, KA2)처럼 외래키를 한쪽에 포함시키는 방식은 가능하다.

오답 포인트

1:1 관계 처리에만 시선이 가면 다치 속성 A4를 놓치기 쉽다. 이 문제의 핵심 함정은 관계 R의 변환 방식보다 이중 타원으로 표시된 A4를 별도 릴레이션으로 분리해야 한다는 점에 있다.

다시 나오면 이렇게 판단

ERD를 관계형 스키마로 바꿀 때는 먼저 이중 타원 같은 다치 속성이 있는지 확인하고, 있으면 별도 릴레이션으로 분리한다. 그다음 1:1 관계는 한쪽 엔터티에 외래키를 넣거나 별도 관계 릴레이션으로 둘 수 있는지를 판단한다.

암기 포인트

다치 속성은 별도 릴레이션으로 분리되고, 1:1 관계는 한쪽 외래키 편입이나 관계 릴레이션 생성으로 변환된다.

NoteBase

문 11. (가), (나), (다)는 데이터베이스의 보안 위험에서 보호되어야 하는 성질을 설명하고 있다. ㉠, ㉡, ㉢에 들어갈 용어로 옳은 것은?

(가) (㉠)은 정보가 부적절한 변경으로부터 보호되어야 하는 성질을 의미한다.
(나) (㉡)은 적법한 권리가 있는 사용자 또는 프로그램이 객체를 사용할 수 있어야 하는 성질을 의미한다.
(다) (㉢)은 권한을 갖지 않은 노출로부터 데이터를 보호해야 하는 성질을 의미한다.

① ㉠ 무결성, ㉡ 가용성, ㉢ 조직성
② ㉠ 무결성, ㉡ 가용성, ㉢ 기밀성
③ ㉠ 기밀성, ㉡ 무결성, ㉢ 가용성
④ ㉠ 기밀성, ㉡ 무결성, ㉢ 조직성

정답 2번

출제 포인트

이 문제는 데이터베이스 보안의 기본 성질인 무결성, 가용성, 기밀성의 정의를 정확히 구별하는지를 묻는 문제이다.

핵심 개념

무결성은 정보가 부적절하게 변경되지 않도록 보호하는 성질이다.
가용성은 적법한 권한이 있는 사용자나 프로그램이 필요할 때 객체를 사용할 수 있게 하는 성질이다.
기밀성은 권한이 없는 노출이나 접근으로부터 데이터를 보호하는 성질이다.
정리하면 무결성은 변경 방지, 가용성은 사용 가능, 기밀성은 비인가 노출 방지가 핵심 키워드이다.

NoteBase

선지별 해설

① X
㉠을 무결성, ㉡을 가용성으로 본 점은 맞지만 ㉢을 조직성으로 둔 부분이 틀리다. 권한을 갖지 않은 노출로부터 데이터를 보호하는 성질은 조직성이 아니라 기밀성이다.

② O
㉠은 정보의 부적절한 변경을 막는 무결성이고, ㉡은 권한 있는 사용자가 객체를 사용할 수 있어야 하는 가용성이며, ㉢은 비인가 노출로부터 데이터를 보호하는 기밀성이다. 세 개념의 대응이 모두 정확하다.

③ X
㉠을 기밀성으로 둔 부분이 틀리다. 정보의 부적절한 변경으로부터 보호하는 성질은 기밀성이 아니라 무결성이다.
또한 ㉡을 무결성으로 둔 부분도 틀리다. 권한 있는 사용자의 사용 가능성은 무결성이 아니라 가용성이다.
㉢을 가용성으로 둔 부분도 틀리다. 비인가 노출 방지는 가용성이 아니라 기밀성이다.

④ X
㉠을 기밀성으로 둔 부분이 틀리다. 부적절한 변경 방지는 기밀성이 아니라 무결성이다.
㉡을 무결성으로 둔 부분도 틀리다. 적법한 사용 가능성은 무결성이 아니라 가용성이다.
㉢을 조직성으로 둔 부분도 틀리다. 권한 없는 노출 방지는 조직성이 아니라 기밀성이다.

오답 포인트

무결성과 기밀성을 모두 보안 개념으로만 묶어 두면 ③, ④처럼 서로 자리를 바꾼 선지에 흔들리기 쉽다. 이 문항의 함정은 '변경'과 '노출'을 구분하지 못하게 만드는 데 있다. 변경 방지는 무결성이고, 노출 방지는 기밀성이라는 축을 놓치면 오답이 된다.

다시 나오면 이렇게 판단

선지에서 핵심 동사를 먼저 본다. '변경'이 나오면 무결성, '사용할 수 있어야 한다'가 나오면 가용성, '노출로부터 보호'가 나오면 기밀성으로 바로 연결하면 된다.

암기 포인트

무결성은 안 바뀜, 가용성은 쓸 수 있음, 기밀성은 안 새어 나감이다.

NoteBase

문 12. 다음 관계형 데이터베이스 릴레이션의 특성으로 옳은 것만을 모두 고르면?

ㄱ. 한 릴레이션에는 동일한 값을 가지는 튜플이 두 개 이상 존재할 수 없다.
ㄴ. 한 릴레이션에서 한 속성의 값은 모두 같은 도메인에 속해야 한다.
ㄷ. 한 릴레이션에서 튜플의 순서는 중요하지 않다.
ㄹ. 한 릴레이션에서 속성의 순서는 중요하지 않다.
ㅁ. 한 릴레이션에서 속성은 다중값(multiple value)을 포함할 수 있다.

① ㄴ, ㅁ
② ㄷ, ㄹ
③ ㄱ, ㄹ, ㅁ
④ ㄱ, ㄴ, ㄷ, ㄹ

정답 4번

출제 포인트

관계형 데이터베이스에서 릴레이션의 기본 성질인 중복 배제, 도메인 제약, 튜플 순서 무관성, 속성 순서 무관성, 원자값 원칙을 구분하는 문제이다.

핵심 개념

릴레이션은 튜플의 집합이므로 동일한 값을 가지는 튜플이 중복되어 존재할 수 없다. 한 속성의 값들은 모두 같은 도메인에 속해야 하며, 튜플의 순서와 속성의 순서는 관계 모델의 논리적 의미를 바꾸지 않는다. 또한 관계형 모델의 속성값은 원자값이어야 하므로 다중값을 한 속성에 그대로 담을 수 없다.

선지별 해설

① X: ㄴ은 옳지만 ㅁ이 틀리므로 오답이다. 관계형 릴레이션의 속성값은 원자값이어야 하므로 한 속성이 다중값을 포함할 수는 없다. 틀린 표현은 "속성은 다중값을 포함할 수 있다"이고, 올바르게는 "속성은 하나의 원자값만 가진다"이다.
② X: ㄷ과 ㄹ은 옳지만 "ㄷ, ㄹ만" 고른 점이 틀리다. ㄱ의 중복 튜플 불허와 ㄴ의 동일 도메인 조건도 함께 옳다. 올바르게는 "ㄱ, ㄴ, ㄷ, ㄹ"을 골라야 한다.
③ X: ㄱ과 ㄷ은 옳지만 ㅁ이 틀리므로 오답이다. 틀린 표현은 "ㅁ 포함"이고, 올바르게는 "ㅁ을 제외한 ㄱ, ㄴ, ㄷ, ㄹ"이다.
④ O: ㄱ, ㄴ, ㄷ, ㄹ은 모두 관계형 데이터베이스 릴레이션의 기본 성질에 해당한다. ㄱ은 릴레이션이 집합이므로 중복 튜플을 허용하지 않는다는 뜻이고, ㄴ은 각 속성값이 동일한 도메인에 속해야 한다는 뜻이며, ㄷ과 ㄹ은 순서가 의미를 결정하지 않는다는 뜻이다.

NoteBase

오답 포인트

ㅁ의 다중값 허용 여부를 개체-관계 모델의 다중값 속성과 혼동하면 틀리기 쉽다. 관계형 모델의 릴레이션은 제1정규형을 전제로 하므로 속성값의 원자성을 요구한다.

다시 나오면 이렇게 판단

릴레이션의 성질을 묻는 문제는 먼저 집합인지 여부를 떠올려 중복과 순서를 판단하고, 다음으로 각 속성값이 같은 도메인에 속하는지와 한 칸에 원자값만 들어가는지를 확인하면 된다.

암기 포인트

릴레이션은 중복 없음, 순서 무관, 도메인 동일, 값은 원자값이다.

NoteBase

문 13. 다음은 장바구니 트랜잭션의 예이다. 연관규칙(Association Rule)을 발견하기 위해 Apriori 알고리즘을 적용한다고 가정할 때, 다음 설명으로 옳지 않은 것은?

TID	Items
1	A, F
2	A, B, D, E
3	B, C, D
4	A, B, D, F
5	A, C, D, F

① 항목집합 {A}의 지지도(support)는 80%이다.

② 항목집합 {A, B, D}가 빈발하면 항목집합 {A, D}도 빈발한다.

③ 규칙 A→{B, D}의 신뢰도(confidence)가 {A, B}→D의 신뢰도보다 크거나 같다.

④ 규칙 A→B의 지지도는 40%이다.

정답 3번

출제 포인트

이 문제는 장바구니 데이터에서 지지도와 신뢰도를 계산하고, Apriori의 하향 폐쇄 성질을 적용할 수 있는지를 묻는 문제이다.

조건 정리

전체 거래 수는 5건이다.
A가 포함된 거래는 TID 1, 2, 4, 5로 4건이다.
{A, B, D}가 포함된 거래는 TID 2, 4로 2건이다.
{A, B}가 포함된 거래는 TID 2, 4로 2건이다.

NoteBase

계산 과정

지지도는 해당 항목집합이 포함된 거래 수를 전체 거래 수로 나눈 값이다.
신뢰도는 규칙의 전건과 후건이 함께 나온 거래 수를 전건이 나온 거래 수로 나눈 값이다.
따라서 support({A}) = 4/5 = 80%, support({A, B}) = 2/5 = 40%이다.
confidence(A → {B, D}) = support({A, B, D}) / support({A}) = 2/4 = 50%이다.
confidence({A, B} → D) = support({A, B, D}) / support({A, B}) = 2/2 = 100%이다.

선지별 해설

① O
{A}는 5개 거래 중 4개 거래에 포함되므로 지지도는 80%이다.

② O
Apriori는 빈발 항목집합의 모든 부분집합도 빈발해야 한다는 하향 폐쇄 성질을 이용한다. 따라서 {A, B, D}가 빈발하면 부분집합인 {A, D}도 빈발한다.

③ X
A → {B, D}의 신뢰도는 50%이고, {A, B} → D의 신뢰도는 100%이다. 따라서 "크거나 같다"가 틀린 표현이다. 올바르게는 "A → {B, D}의 신뢰도는 {A, B} → D의 신뢰도보다 작다"이다.

④ O
규칙 A → B의 지지도는 규칙이 성립하는 거래, 즉 {A, B}가 함께 나온 거래의 비율이다. {A, B}는 2건이므로 지지도는 2/5 = 40%이다.

오답 포인트

③은 신뢰도의 분모가 달라진다는 점을 놓치면 헷갈리기 쉽다. A → {B, D}는 분모가 support({A})이고, {A, B} → D는 분모가 support({A, B})이므로 같은 분자를 써도 값이 달라진다.

다시 나오면 이렇게 판단

연관규칙 문제는 먼저 전체 거래 수와 각 항목집합의 등장 횟수를 센 뒤, 지지도는 전체 거래 수로 나누고 신뢰도는 전건 등장 횟수로 나눈다. 빈발 여부를 묻는 문장은 Apriori의 부분집합 성질부터 확인하면 된다.

암기 포인트

지지도는 전체 기준, 신뢰도는 전건 기준이며, 빈발집합의 부분집합은 항상 빈발집합이다.

NoteBase

문 14. 다음 EMPLOYEE, PROJECT, WORKSON 테이블로 구성된 데이터베이스에서, “모든 프로젝트에 참가하는 직원의 이름을 검색하라.”를 수행하기 위한 SQL문으로 옳은 것은?

EMPLOYEE(eNo, eName)
PROJECT(pNo, pName)
WORKSON(eNo, pNo)

①
```
SELECT eName
FROM  EMPLOYEE E, WORKSON W, PROJECT P
WHERE E.eNo = W.eNo
      AND W.pNo = P.pNo;
```

②
```
SELECT eName
FROM EMPLOYEE E
WHERE NOT EXISTS (
        SELECT P.pNo
        FROM PROJECT P
        WHERE NOT EXISTS (
                SELECT W.pNo
                FROM WORKSON W
                WHERE P.pNo = W.pNo AND E.eNo = W.eNo
        )
);
```

③
```
SELECT eName
FROM EMPLOYEE E
WHERE EXISTS (
          ( SELECT pNo FROM PROJECT )
                NOT EXCEPT
          ( SELECT pNo
            FROM WORKSON W
            WHERE E.eNo=W.eNo)
);
```

④
```
SELECT eName
FROM EMPLOYEE E
WHERE EXISTS (
          ( SELECT pNo FROM PROJECT)
                EXCEPT
          ( SELECT pNo
            FROM WORKSON W
            WHERE E.eNo = W.eNo )
);
```

정답 2번

출제 포인트

이 문제는 SQL에서 '모든 프로젝트에 참여한다'는 전체 조건을 이중 NOT EXISTS 또는 공집합 검사로 표현할 수 있는지를 묻는다.

NoteBase

핵심 개념

전체 조건 검사는 '빠진 프로젝트가 하나도 없어야 한다'로 바꾸어 표현한다. 단순 조인은 '하나라도 참여한 경우'만 확인하고, 집합 방식은 PROJECT 전체 집합에서 해당 직원의 참여 프로젝트 집합을 뺀 결과가 공집합인지로 판단한다.

선지별 해설

1번 X: EMPLOYEE, WORKSON, PROJECT를 조인하면 어떤 프로젝트든 한 건이라도 참여한 직원이 반환된다. '모든 프로젝트'를 확인하는 조건이 없으므로 전체 조건을 표현하지 못한다. 틀린 표현은 전체 조건이 빠진 단순 조인이라는 점이고, 올바르게는 PROJECT 전체에 대해 빠진 pNo가 없는지를 검사해야 한다.
2번 O: 바깥쪽 NOT EXISTS는 '조건을 만족하지 못하는 프로젝트가 존재하지 않는다'는 뜻이다. 안쪽 NOT EXISTS가 각 프로젝트 P에 대해 해당 직원 E의 WORKSON 행이 없는 경우를 찾으므로, 결국 어떤 프로젝트도 빠지지 않은 직원만 남는다.
3번 X: 틀린 표현은 `EXISTS`와 `NOT EXCEPT` 조합이다. 모든 프로젝트 참여 여부는 차집합 결과가 비어 있는지를 보아야 하므로 `NOT EXISTS ((SELECT pNo FROM PROJECT) EXCEPT (SELECT pNo FROM WORKSON W WHERE E.eNo = W.eNo))`처럼 써야 한다. 제시된 형태는 올바른 공집합 검사 구문이 아니다.
4번 X: `(SELECT pNo FROM PROJECT) EXCEPT (SELECT pNo FROM WORKSON W WHERE E.eNo = W.eNo)`는 그 직원이 참여하지 않은 프로젝트 목록이다. 여기에 `EXISTS`를 쓰면 빠진 프로젝트가 하나라도 있는 직원을 고르게 되므로 의미가 정반대이다. 틀린 표현은 `EXISTS`이고, 올바르게는 차집합이 없다는 뜻의 `NOT EXISTS`여야 한다.

오답 포인트

단순 조인 결과가 나오면 곧바로 정답처럼 보이기 쉽고, `EXISTS`와 `NOT EXISTS`를 한 글자 차이로 뒤집어 읽기 쉽다. 이 문항의 함정은 '하나라도 참여'와 '모두 참여'를 구분하지 못하게 만드는 데 있다.

다시 나오면 이렇게 판단

발문에 '모든', '항상', '전부'가 나오면 먼저 전체 조건을 빠진 항목이 없는지의 문제로 바꾼다. 그다음 PROJECT 전체 집합에서 해당 직원의 참여 집합을 뺀 결과가 공집합인지, 또는 '참여하지 않은 프로젝트가 존재하지 않는지'를 확인한다.

암기 포인트

'모든 A를 만족'은 '만족하지 않는 A가 없다'로 바꾸어 이중 NOT EXISTS나 `NOT EXISTS (A EXCEPT B)`로 처리한다.

NoteBase

문 15. 하루 수천 건의 거래가 발생하는 온라인 쇼핑몰에서 '주문정보' 테이블은 대용량의 데이터를 저장해야 하고 트랜잭션이 자주 발생하기 때문에 주문정보 시스템의 성능에 크게 영향을 준다. 다음 질의가 빈번하게 발생하는 상태에서 성능향상을 위하여 '주문정보' 테이블의 '주문일자' + '순번' 속성으로 구성된 키(key)에 대해 인덱스를 생성하고자 할 때, 성능향상에 적합한 인덱스 구조로 옳은 것만을 모두 고르면?

```
SELECT *
FROM 주문정보
WHERE 주문일자 BETWEEN '03-01-2016' AND '03-31-2016';
```

ㄱ. B^{+}-트리 인덱스
ㄴ. B^{*}-트리 인덱스
ㄷ. 해시 인덱스
ㄹ. 비트맵(bitmap) 인덱스

① ㄱ
② ㄱ, ㄴ
③ ㄱ, ㄴ, ㄷ
④ ㄱ, ㄴ, ㄹ

정답 2번

출제 포인트

범위 검색이 빈번한 대용량 OLTP 테이블에서 복합키 인덱스 구조별 적합성을 구별하는 문제이다.

핵심 개념

B-트리 인덱스는 키 값을 정렬된 상태로 유지해 범위 검색에 적합한 균형 트리 인덱스이다.
B*트리 인덱스는 B-트리 계열로 노드 활용률을 더 높인 구조이며 역시 정렬 기반 범위 검색에 적합하다.
해시 인덱스는 해시 함수로 버킷을 찾으므로 동등 비교에는 빠르지만 키 순서가 없어 범위 검색에 부적합하다.
비트맵 인덱스는 카디널리티가 낮고 갱신이 적은 분석형 환경에 유리하며, 트랜잭션이 잦은 주문정보 테이블에는 부적합하다.

NoteBase

선지별 해설

① X: ①은 ㄱ만 옳다고 본 선택지이다. B-트리 인덱스는 주문일자 범위 조건에 적합하므로 ㄱ은 맞지만, 같은 정렬 기반 트리 구조인 ㄴ B*트리 인덱스도 범위 검색에 적합하므로 ①은 불완전하다. 틀린 표현은 'ㄱ만'에 해당하는 조합이며, 올바르게는 'ㄱ, ㄴ'이다.
② O: ②는 ㄱ, ㄴ을 고른 선택지이다. 주문일자가 복합키의 선두 속성이고 질의가 BETWEEN 범위 조건이므로 정렬 순서를 유지하는 B-트리와 B*트리 인덱스가 모두 적합하다.
③ X: ③은 해시 인덱스인 ㄷ까지 포함한 선택지이다. 해시 인덱스는 '=' 조건의 동등 검색에는 유리하지만 키 순서가 없어 주문일자 범위 검색에는 부적합하다. 틀린 표현은 'ㄷ 포함'이며, 올바르게는 해시 인덱스를 제외한 'ㄱ, ㄴ'이다.
④ X: ④는 비트맵 인덱스인 ㄹ까지 포함한 선택지이다. 비트맵 인덱스는 주로 읽기 위주의 저카디널리티 분석 환경에 적합하고, 거래와 갱신이 자주 발생하는 주문정보 테이블에는 잠금·갱신 비용 측면에서 부적절하다. 틀린 표현은 'ㄹ 포함'이며, 올바르게는 비트맵 인덱스를 제외한 'ㄱ, ㄴ'이다.

오답 포인트

범위 검색이라는 조건을 놓치고 '인덱스면 다 빠르다'고 보면 해시 인덱스를 포함하기 쉽다. 또 대용량이라는 말만 보고 비트맵 인덱스를 떠올리면 OLTP의 빈번한 트랜잭션 조건을 놓치게 된다.

다시 나오면 이렇게 판단

인덱스 문제는 먼저 질의 조건이 동등 검색인지 범위 검색인지 확인한다. 범위 검색이면 정렬을 유지하는 B-트리 계열을 우선 떠올리고, 해시는 제외한다. 이어서 업무가 OLTP처럼 갱신이 잦은지, DW처럼 조회 위주인지 확인해 비트맵 인덱스 포함 여부를 판단한다.

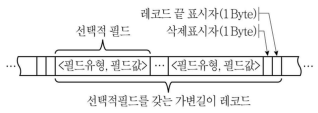
암기 포인트

범위 검색은 B-트리 계열, 동등 검색은 해시, 비트맵은 저카디널리티·조회 위주 환경이다.

NoteBase

문 16. 다음의 '선택적필드'를 갖는 가변길이 레코드 10,000개가 저장된 PLAYER 파일이 있다. PLAYER 파일 내의 레코드는 <필드유형, 필드값>으로 표현되는 '선택적필드'들로 구성되어 있으며, 레코드 끝 부분에 1바이트의 '삭제표시자'와 1바이트의 '레코드 끝 표시자'를 포함한다. PLAYER 파일에 저장되어 있는 레코드들의 상태는 <선택적필드를 갖는 가변길이 레코드 저장 상태> 표에 제시되어 있으며, 레코드를 저장하기 위한 블록 구조 역시 <PLAYER 파일 구성을 위한 블록 구조> 표에 제시되어 있다. 이와 같은 상태에서 PLAYER 파일에 저장되어 있는 레코드들의 평균 크기와 필요한 블록 수를 계산한 결과는?

<PLAYER 파일의 구조>



<선택적필드를 갖는 가변길이 레코드 저장 상태>

필드이름	필드유형 크기	필드값 크기	선택적필드 참여율
TEAM	1바이트	3바이트	90%
SALARY	1바이트	2바이트	80%
INJURY	1바이트	9바이트	10%
UNIVERSITY	1바이트	9바이트	70%

○ 필드이름은 '선택적필드'를 식별하기 위한 것으로 '선택적필드'의 '필드유형' 값으로 코드화되어 저장된다.
○ 필드유형 크기는 '필드유형'을 저장하기 위한 항목으로 파일 내에 있는 모든 '선택적필드'에서 1바이트의 크기를 갖는다.
○ 필드값 크기는 '선택적필드'에서 '필드유형'을 제외한 '필드값'을 저장하기 위한 크기로 '선택적필드'에 따라 적합한 크기를 갖는다.
○ 선택적필드 참여율은 파일 내 전체 레코드들에서 해당 '선택적필드'가 나타나는 비율을 의미한다.

<PLAYER 파일 구성을 위한 블록 구조>

블록조직의 형태	블록 크기	기타
신장(spanned) 조직	2400바이트	파일 내 다음 번 블록을 가리키기 위한 4바이트 크기 포인터 필드 포함

① 레코드들의 평균 크기: 70바이트, 필요한 블록의 수: 293개

② 레코드들의 평균 크기: 71바이트, 필요한 블록의 수: 296개

③ 레코드들의 평균 크기: 70바이트, 필요한 블록의 수: 291개

④ 레코드들의 평균 크기: 71바이트, 필요한 블록의 수: 294개

정답 1번

출제 포인트

이 문제는 선택적 필드의 평균 저장 크기를 기대값으로 구한 뒤, 블록 포인터 4바이트를 제외한 실제 저장 가능 용량으로 필요한 블록 수를 계산하는 문제이다.

NoteBase

조건 정리

주어진 값:
- TEAM: 1 + 39 = 40바이트, 참여율 90%
- SALARY: 1 + 29 = 30바이트, 참여율 80%
- INJURY: 1 + 9 = 10바이트, 참여율 10%
- UNIVERSITY: 1 + 9 = 10바이트, 참여율 70%
- 삭제표시자 1바이트, 레코드 끝 표시자 1바이트
- 레코드 수 10,000개
- 블록 크기 2400바이트, 다음 블록 포인터 4바이트

구해야 할 값:
- 레코드 평균 크기
- 필요한 블록 수

주의할 조건:
- 선택적 필드는 참여율만큼만 평균에 반영한다.
- 블록당 실제 레코드 저장 가능 용량은 2400 − 4 = 2396바이트이다.

사용할 공식

평균 레코드 크기 = 각 선택적 필드의 필드 전체 크기 × 참여율 합 + 관리 바이트

$$(40 \times 0.9) + (30 \times 0.8) + (10 \times 0.1) + (10 \times 0.7) + 2$$

필요 블록 수 = $\leq$ft 전체 레코드 크기블록당 저장 가능 크기

이 공식을 쓰는 이유:
이 문제의 레코드는 선택적 필드가 확률적으로 나타나므로 평균 크기를 기대값으로 계산해야 한다. 또 신장 조직은 블록마다 다음 블록을 가리키는 포인터 4바이트가 필요하므로 2400바이트 전체를 그대로 레코드 저장에 쓸 수 없다.

NoteBase

계산 과정

1. 선택적 필드의 평균 저장 크기를 구한다.
$40 \times 0.9 = 36, \quad 30 \times 0.8 = 24, \quad 10 \times 0.1 = 1, \quad 10 \times 0.7 = 7$
2. 선택적 필드 기대 크기를 합하면 $36 + 24 + 1 + 7 = 68$바이트이다.
3. 삭제표시자 1바이트와 레코드 끝 표시자 1바이트를 더하면 평균 레코드 크기는 $68 + 2 = 70$바이트이다.
4. 전체 레코드 크기는 $70 \times 10{,}000 = 700{,}000$바이트이다.
5. 블록당 실제 저장 가능 크기는 $2400 - 4 = 2396$바이트이므로 필요한 블록 수는
$\left\lceil \frac{700{,}000}{2396} \right\rceil = \lceil 292.153\cdots \rceil = 293$
이다.

선지별 해설

① O
- 평균 레코드 크기 70바이트, 필요한 블록 수 293개가 계산값과 일치한다.
- 선택적 필드 기대 크기 68바이트에 관리 바이트 2바이트를 더해 70바이트가 되고, 블록당 2396바이트를 기준으로 올림하면 293개이다.

② X
- 71바이트는 선택적 필드 기대 크기 68바이트에 관리 바이트 2바이트를 더한 값과 맞지 않는다.
- 필요한 블록 수도 296개가 아니라 293개이다.
- 틀린 표현은 "71바이트, 296개"이고, 올바르게는 "70바이트, 293개"이다.

③ X
- 평균 레코드 크기 70바이트는 맞지만 필요한 블록 수 291개는 부족하다.
- 신장 조직의 블록은 포인터 4바이트를 제외한 2396바이트만 저장에 쓰므로 $700{,}000 \div 2396$을 올림한 293개가 필요하다.
- 틀린 표현은 "291개"이고, 올바르게는 "293개"이다.

④ X
- 평균 레코드 크기부터 71바이트로 잘못 계산되었고, 블록 수도 294개가 아니다.
- 평균 크기는 70바이트이며 이를 기준으로 계산한 블록 수는 293개이다.
- 틀린 표현은 "71바이트, 294개"이고, 올바르게는 "70바이트, 293개"이다.

오답 포인트

선택적 필드를 모두 항상 저장된다고 보고 단순 합산하거나, 블록 크기 2400바이트 전체를 저장 공간으로 쓰는 실수가 잦다. 이 문제는 참여율을 곱한 기대값 계산과 블록 포인터 4바이트 차감이 동시에 들어간다는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

가변길이 레코드 계산형 문제는 각 필드의 전체 크기를 먼저 구한 뒤 참여율을 곱해 평균 레코드 크기를 만든다. 그다음 블록의 관리 영역을 뺀 실제 저장 가능 용량으로 나누고, 블록 수는 항상 올림으로 판단한다.

암기 포인트

선택적 필드 평균 크기는 필드 크기 합이 아니라 필드 크기 × 참여율의 합이며, 블록 수는 관리 바이트를 뺀 뒤 올림한다.

NoteBase

문 17. 릴레이션 스키마 R(A, B, C, D, E, F, G, H, I)에 대하여 다음과 같은 함수적 종속성의 집합 F가 주어졌을 때, 스키마 R의 속성들의 부분집합 {A, G}에 대하여, F 하의 {A, G}의 폐포(closure) {A, G}+로 옳은 것은?

F = {A→B, BH→D, C→EF, BG→IE, AE→C}

① {A, B, C, E, G, H, I}

② {A, C, D, E, G, I, F}

③ {A, C, D, E, G, H, I}

④ {A, B, C, E, G, I, F}

정답 4번

출제 포인트

이 문제는 함수적 종속성 집합으로부터 속성 집합 $\{A, G\}$의 폐포를 단계적으로 구할 수 있는지를 묻는다.

조건 정리

주어진 함수적 종속성은 $A \to B, BH \to D, C \to EF, BG \to IE, AE \to C$이다.
구해야 할 값은 $\{A, G\}^+$이다.
폐포는 현재 얻은 속성들만으로 적용 가능한 함수적 종속성을 반복 적용하여 구한다.

사용할 공식

폐포 계산은 시작 집합에 포함된 속성에서 출발하여, 좌변이 모두 포함되는 함수적 종속성의 우변을 계속 추가하는 방식으로 진행된다.
이 문제는 특정 속성 집합이 어떤 속성들까지 결정하는지를 묻기 때문에 폐포 계산 원리를 그대로 적용한다.

계산 과정

처음에는 $\{A, G\}$에서 시작한다.
$A \to B$를 적용하면 B가 추가되어 $\{A, B, G\}$가 된다.
$BG \to IE$를 적용하면 I, E가 추가되어 $\{A, B, E, G, I\}$가 된다.
$AE \to C$를 적용하면 C가 추가되어 $\{A, B, C, E, G, I\}$가 된다.
$C \to EF$를 적용하면 F가 추가되어 $\{A, B, C, E, F, G, I\}$가 된다.
$BH \to D$는 H가 없으므로 적용할 수 없다.
따라서 $\{A, G\}^+ = \{A, B, C, E, F, G, I\}$이다.

NoteBase

선지별 해설

1번: X. 집합에 H가 포함되어 있는데, 주어진 함수적 종속성으로는 H를 새로 얻을 방법이 없다. 틀린 표현은 H의 포함이며, 올바르게는 $\{A, B, C, E, F, G, I\}$가 되어야 한다.
2번: X. 집합에 D가 포함되어 있는데, D를 얻으려면 $BH \rightarrow D$를 적용해야 한다. 폐포에는 H가 없으므로 D는 포함될 수 없다. 또한 B가 빠져 있는 것도 틀렸으며, 올바르게는 $\{A, B, C, E, F, G, I\}$가 되어야 한다.
3번: X. D와 H를 포함한 점이 모두 틀리다. H를 얻을 종속성이 없고, D도 H 없이는 결정되지 않는다. 올바르게는 D와 H를 제외하고 B, F를 포함한 $\{A, B, C, E, F, G, I\}$가 되어야 한다.
4번: O. $A \rightarrow B$, $BG \rightarrow IE$, $AE \rightarrow C$, $C \rightarrow EF$를 순서대로 적용하면 $\{A, B, C, E, F, G, I\}$가 된다. D와 H는 끝까지 추가되지 않으므로 이 선지가 맞다.

오답 포인트

폐포 문제에서 가장 흔한 실수는 우변에 보이는 속성을 바로 넣거나, 한 번 얻은 속성으로 다시 적용할 수 있는 종속성을 끝까지 추적하지 않는 점이다. 이 문항에서는 $BH \rightarrow D$를 보고 B만 있으면 D가 나온다고 착각하거나, $C \rightarrow EF$에서 F를 끝까지 추가하지 않으면 오답이 된다.

다시 나오면 이렇게 판단

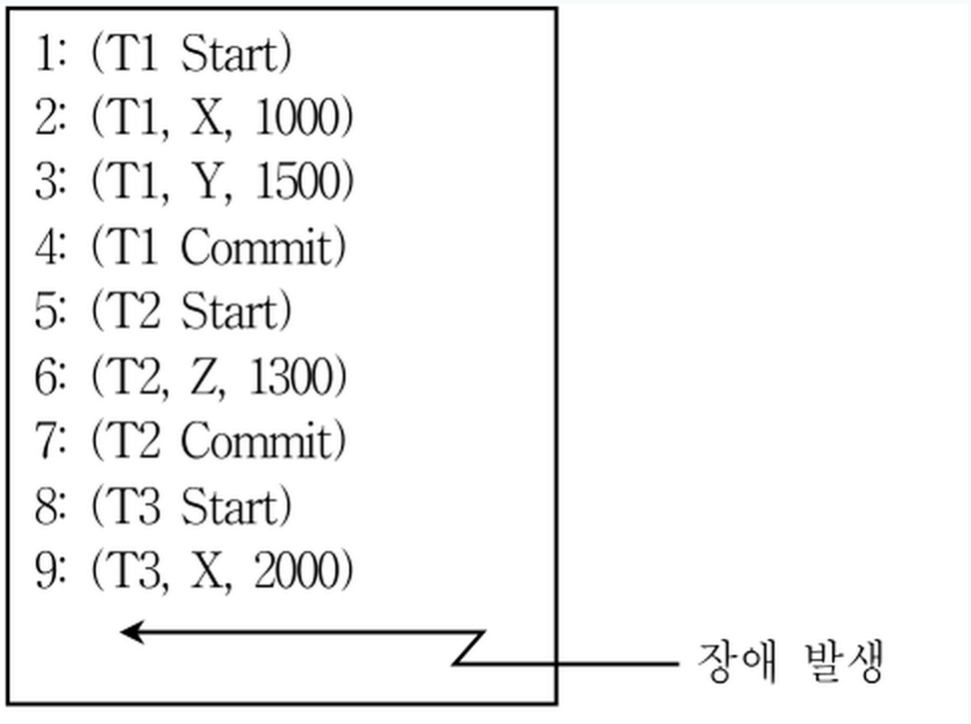
폐포 문제는 시작 집합을 적고, 현재 집합만으로 좌변이 충족되는 종속성만 순서대로 표시한 뒤, 더 이상 새 속성이 추가되지 않을 때 멈추면 된다. 중간에 새로 얻은 속성으로 다시 적용 가능한 종속성이 있는지 끝까지 확인하면 된다.

암기 포인트

폐포는 시작 집합에서 출발해 적용 가능한 함수적 종속성의 우변을 더 이상 늘지 않을 때까지 반복해서 붙인 최종 집합이다.

NoteBase

문 18. 다음은 장애 발생 시 지연 갱신 회복 기법으로 복구하는 데이터베이스 시스템의 로그 기록이다. T3를 수행하던 도중 시스템 장애가 발생했을 때 수행되는 복구 절차로 옳은 것은?



① T1과 T2를 redo 연산을 수행하여 복구하고, T3에 대해서는 undo 연산을 수행한다.

② T1과 T2를 undo 연산을 수행하여 복구하고, T3에 대해서는 redo 연산을 수행한다.

③ T1과 T2를 redo 연산을 수행하여 복구하고, T3에 대해서는 별도의 연산을 수행하지 않는다.

④ T1과 T2는 이미 데이터베이스에 반영되어 있으므로 별도의 연산을 수행하지 않고, T3를 redo 연산을 수행하여 복구한다.

정답 3번

출제 포인트

이 문제는 지연 갱신 회복 기법에서 커밋된 트랜잭션과 미커밋 트랜잭션을 어떻게 처리하는지를 묻는다.

사례 정리

T1은 Start, 갱신 로그, Commit까지 기록되어 있다.
T2도 Start, 갱신 로그, Commit까지 기록되어 있다.
T3는 Start와 갱신 로그만 있고 Commit 기록이 없는 상태에서 장애가 발생하였다.

NoteBase

핵심 개념

지연 갱신은 트랜잭션이 커밋되기 전에는 데이터베이스에 실제 갱신 결과를 반영하지 않는 회복 방식이다.
따라서 회복 시에는 Commit 기록이 있는 트랜잭션만 redo 대상이 되고, Commit 기록이 없는 트랜잭션은 데이터베이스에 반영된 내용이 없으므로 undo 대상이 아니다.

적용 기준

이 사례에서는 로그에 Commit 기록이 있는지 여부가 판단 기준이다.
T1과 T2는 Commit 기록이 있으므로 redo 대상이고, T3는 Commit 기록이 없으므로 별도 회복 연산을 하지 않는다.

선지별 해설

1번 X: T1과 T2를 redo 하는 부분은 맞지만, T3에 undo를 수행한다는 부분이 틀리다. 지연 갱신에서는 미커밋 갱신이 데이터베이스에 반영되지 않았으므로 undo가 필요하지 않다. 올바르게는 T3에 대해서는 별도의 연산을 수행하지 않는다.
2번 X: T1과 T2는 Commit이 끝난 트랜잭션이므로 undo 대상이 아니라 redo 대상이다. 또한 T3는 Commit이 없으므로 redo 대상도 아니다. 올바르게는 T1과 T2는 redo 하고, T3는 별도 연산을 하지 않는다.
3번 O: T1과 T2는 Commit 기록이 있으므로 redo 연산으로 복구하고, T3는 Commit 전에 장애가 발생했으므로 별도의 연산을 수행하지 않는다. 지연 갱신의 회복 원칙에 맞는 설명이다.
4번 X: T1과 T2가 이미 데이터베이스에 반영되었으므로 아무 연산도 하지 않는다는 부분이 틀리다. 지연 갱신에서는 Commit된 트랜잭션이라도 장애 시점에 실제 반영이 끝났다고 볼 수 없으므로 redo가 필요하다. 또한 T3는 미커밋이므로 redo 대상이 아니다. 올바르게는 T1과 T2를 redo 하고, T3는 별도 연산을 하지 않는다.

오답 포인트

즉시 갱신과 지연 갱신을 혼동하면 T3에 undo를 해야 한다고 착각하기 쉽다. 이 문항의 함정은 미커밋 트랜잭션이 보이면 곧바로 undo를 떠올리게 만드는 점에 있다. 지연 갱신에서는 미커밋 내용이 애초에 데이터베이스에 반영되지 않았다는 점이 핵심이다.

다시 나오면 이렇게 판단

복구 문제는 먼저 갱신 방식이 즉시 갱신인지 지연 갱신인지부터 구분한다.
지연 갱신이면 Commit 기록이 있는 것만 redo 하고, Commit 기록이 없는 것은 아무 연산도 하지 않는다고 판단한다.

암기 포인트

지연 갱신은 No-UNDO/REDO 방식이므로 커밋된 트랜잭션만 redo 대상이다.

NoteBase

문 19. 다음은 트랜잭션 격리수준(Isolation Level) 설정을 위한 SQL 표준 형식이다. 이를 이용하여 설정할 수 있는 트랜잭션 격리수준에 대한 설명으로 옳지 않은 것은?

```
SET TRANSACTION {READ ONLY | READ WRITE}
    ISOLATION LEVEL {READ UNCOMMITTED
                   | READ COMMITTED
                   | REPEATABLE READ
                   | SERIALIZABLE }
```

① READ UNCOMMITTED로 설정된 트랜잭션은 완료되지 않은 트랜잭션이 생성한 데이터 값을 읽어가는 것을 허용하는 것이며, READ WRITE와 함께 설정될 수 없다.

② READ COMMITTED로 설정된 트랜잭션은 동일 조건의 데이터 집합을 반복하여 읽을 경우, 먼저 읽었던 데이터 집합에 있던 데이터 값과 새로이 읽은 데이터 집합에 있는 데이터 값이 달라져 있을 수 있다.

③ REPEATABLE READ로 설정된 트랜잭션은 동일 조건의 데이터 집합을 반복하여 읽을 경우, 읽혀지는 집합을 구성하는 로우들의 수가 달라질 수 있다.

④ SERIALIZABLE로 설정된 트랜잭션은 격리성을 완전하게 유지함으로써 트랜잭션의 일관성을 보존하고 동시에 수행 중인 트랜잭션들의 처리율을 향상시킨다.

정답 4번

출제 포인트

이 문제는 SQL 표준의 트랜잭션 격리수준별로 허용되는 이상 현상과 성능 특성을 구분하는 문제이다.

핵심 개념

READ UNCOMMITTED는 아직 커밋되지 않은 변경까지 읽을 수 있는 가장 낮은 격리수준이다.
READ COMMITTED는 커밋된 데이터만 읽지만 같은 행을 다시 읽을 때 값이 달라질 수 있다.
REPEATABLE READ는 이미 읽은 행의 값 변경은 막지만 조건에 맞는 행 수 자체가 달라지는 팬텀 리드는 남을 수 있다.
SERIALIZABLE은 동시 실행 결과가 직렬 실행과 같도록 보장하는 가장 높은 격리수준이다.

NoteBase

선지별 해설

① O. READ UNCOMMITTED는 미완료 트랜잭션이 기록한 값을 읽는 더티 리드를 허용한다. 또한 SQL 표준에서는 READ UNCOMMITTED가 READ WRITE 트랜잭션에는 허용되지 않으므로 진술이 맞다.
② O. READ COMMITTED에서는 같은 조건으로 다시 조회할 때 먼저 읽은 행의 값이 다른 커밋 결과로 바뀔 수 있다. 이는 비반복 읽기 현상에 해당하므로 진술이 맞다.
③ O. REPEATABLE READ는 이미 읽은 기존 행의 값은 반복 조회 시 같게 유지되지만, 같은 조건에 새 행이 추가되거나 기존 행이 제외되어 행 수가 달라질 수 있다. 이는 팬텀 리드에 해당하므로 진술이 맞다.
④ X. SERIALIZABLE은 격리성을 가장 강하게 보장하여 일관성을 높이지만 동시성은 가장 낮아지고 처리율도 대체로 떨어진다. 틀린 표현은 '처리율을 향상시킨다'이다. 올바르게는 '처리율이 향상되는 것이 아니라 감소할 수 있다'이다.

오답 포인트

격리수준이 높을수록 무조건 좋은 성능이 나온다고 오해하면 ④를 고르지 못한다. 이 문항은 이상 현상 감소와 동시성 저하가 함께 간다는 점을 묻는 함정이다.

다시 나오면 이렇게 판단

격리수준 문제는 더티 리드, 비반복 읽기, 팬텀 리드가 어디까지 허용되는지를 먼저 대입하면 된다. 그다음 격리수준이 올라갈수록 일관성은 강해지고 동시성 및 처리율은 대체로 낮아진다고 판단하면 된다.

암기 포인트

격리수준은 READ UNCOMMITTED → READ COMMITTED → REPEATABLE READ → SERIALIZABLE 순으로 강해지고, 허용 이상 현상은 줄며 처리율은 대체로 낮아진다.

NoteBase

문 20. <보기 1>은 관계형 데이터베이스 스키마와 이 스키마에 대하여 “1980년 이후에 출생하고 프로젝트명이 ‘DBMS개발’인 프로젝트를 수행하는 연구원 이름을 검색하라.”를 SQL로 표현한 것이다. <보기 2>는 경험적 최적화 기법을 적용하여 최적화를 수행하기 위해, <보기 1>의 SQL 질의문에서 생성되는 초기 질의트리를 최적화된 질의트리로 변환하는 과정에서 유도되는 관계대수식의 목록이다. 최적화 순서에 맞도록 바르게 나열한 것은? (단, 연구원 테이블 R, 프로젝트 테이블 P, 프로젝트 팀 테이블 T라 한다)

<보기 1>

Schema: R(<u>rNo</u>, rName, bDate, address)

P(<u>pNo</u>, pName, pPeriod)

T(<u>rNo, pNo</u>, pBudget)

SQL 질의문:

```
SELECT rName
FROM R, T, P
WHERE pName = 'DBMS개발'
      AND P.pNo = T.pNo AND R.rNo = T.rNo
      AND bDate > '12-31-1979';
```

<보기 2>

ㄱ. $\pi_{rName}($
$\pi_{rNo}($
$\pi_{pNo}(\sigma_{pName='DBMS개발'}(P))$
$\bowtie_N \pi_{rNo,pNo}(T)$
$) \bowtie_N \pi_{rNo,rName}(\sigma_{bDate>'12-31-1979'}(R))$
$)$

ㄴ. $\pi_{rName}(\sigma_{P.pNo=T.pNo\ AND\ T.rNo=R.rNo\ AND\ pName='DBMS개발'\ AND\ bDate>'12-31-1979'}((R \times T) \times P))$

ㄷ. $\pi_{rName}($
$\sigma_{P.pNo=T.pNo}($
$\sigma_{T.rNo=R.rNo}(\sigma_{bDate>'12-31-1979'}(R) \times T)$
$\times \sigma_{pName='DBMS개발'}(P)$
$)$
$)$

ㄹ. $\pi_{rName}($
$(\sigma_{pName='DBMS개발'}(P) \bowtie_N T)$
$\bowtie_N \sigma_{bDate>'12-31-1979'}(R)$
$)$

ㅁ. $\pi_{rName}($
$\sigma_{R.rNo=T.rNo}($
$\sigma_{P.pNo=T.pNo}(\sigma_{pName='DBMS개발'}(P) \times T)$
$\times \sigma_{bDate>'12-31-1979'}(R)$
$)$
$)$

① ㄱ → ㄴ → ㅁ → ㄷ → ㄹ

NoteBase

② ㄴ → ㄱ → ㄷ → ㄹ → ㅁ

③ ㄴ → ㄷ → ㅁ → ㄹ → ㄱ

④ ㄱ → ㄴ → ㄷ → ㄹ → ㅁ

정답 3번

출제 포인트

이 문제는 SQL로부터 만든 초기 질의트리를 경험적 최적화 규칙에 따라 선택 푸시다운, 곱셈(조인) 순서 정리, 카티전곱+선택의 자연조인 치환, 투영 축소 순으로 바꿀 수 있는지를 묻는다.

올바른 순서

1. ㄴ — FROM절의 세 테이블 카티전곱 $(R \times T) \times P$ 위에 WHERE절의 네 조건(조인 조건 두 개와 선택 조건 두 개)을 하나의 선택 σ로 묶어 올린 초기 질의트리이다. SQL의 FROM절이 카티전곱으로, WHERE절이 단일 선택으로 곧이곧대로 옮겨진 출발점이라 최적화의 기준점이 된다.
2. ㄷ — 통합 선택을 분해하여 *bDate* 조건은 R 바로 위로, *pName* 조건은 P 바로 위로 내려보낸(선택 푸시다운) 형태이다. 다만 카티전곱 ×과 조인 조건 선택 $\sigma_{T.rNo = R.rNo}$, $\sigma_{P.pNo = T.pNo}$은 아직 그대로 남아 있다.
3. ㅁ — 여전히 카티전곱 ×과 조인 조건 선택을 쓰지만, 먼저 선택이 적용된 P와 T를 결합한 $(\sigma_{pName}(P) \times T)$를 만든 뒤 R과 곱하도록 연산(곱셈) 순서를 재배치한 형태이다. 즉 조인을 자연조인으로 바꾸기 전에, 튜플이 줄어든 릴레이션끼리 먼저 묶이도록 곱셈 순서만 조정한 중간 단계이며 아직 조인 연산은 등장하지 않는다.
4. ㄹ — 카티전곱과 그 위에 붙은 조인 조건 선택을 하나의 자연조인 $\bowtie_N$으로 치환한 형태이다. $\sigma_{pName}(P) \bowtie_N T$와 $\sigma_{bDate}(R)$을 자연조인으로 묶어, ㅁ의 '카티전곱+선택' 구조가 비로소 조인 연산으로 정리된다. ㅁ과 ㄹ의 결정적 차이는 $\times + \sigma$(ㅁ)와 $\bowtie_N$(ㄹ)이다.
5. ㄱ — 조인에 필요한 속성만 남기도록 π_{pNo}, $\pi_{rNo, pNo}$, $\pi_{rNo, rName}$ 등의 투영을 각 릴레이션 아래로 내리고(투영 푸시다운) 최종적으로 *rName*만 출력하는 최적화 완료형이다.

왜 이 순서: 먼저 선택 연산으로 튜플 수를 줄여야(ㄷ) 이후 조인 비용이 감소하고, 곱셈 순서를 정리한 뒤(ㅁ) 카티전곱+선택을 자연조인으로 바꾸며(ㄹ), 조인 구조가 확정된 다음에야 조인 속성을 해치지 않으면서 투영을 안전하게 줄일 수 있다(ㄱ).

NoteBase

선지별 해설

1번: X. ㄱ을 맨 앞에 두었는데, ㄱ은 π_{pNo}, $\pi_{rNo,\,pNo}$처럼 필요한 속성만 남기는 투영 푸시다운이 끝난 최적화 완료형이므로 가장 마지막에 와야 한다. 초기 질의트리보다 앞에 올 수 없다. 올바르게는 초기트리 ㄴ으로 시작해 ㄱ으로 끝난다.

2번: X. ㄴ 다음에 바로 ㄱ을 두었는데, 초기 트리에서 곧바로 투영을 줄이면 뒤의 조인에 필요한 *pNo*와 *rNo*를 남겨야 하는지 판단할 수 없다. 선택 조건을 먼저 각 릴레이션 아래로 내린 ㄷ이 먼저 와야 한다. 올바르게는 "ㄴ → ㄷ"이다.

3번: O. ㄴ은 초기 질의트리, ㄷ은 선택 푸시다운, ㅁ은 카티전곱 ×을 유지한 채 선택된 *P*와 *T*를 먼저 묶는 곱셈 순서 재배치, ㄹ은 그 카티전곱+선택을 자연조인 $\bowtie_N$으로 바꾼 변환, ㄱ은 최종 투영 축소이므로 경험적 최적화의 흐름과 일치한다.

4번: X. ㄱ을 맨 앞에 둔 점이 먼저 틀리다. 또한 ㄷ 다음에 곧바로 ㄹ로 건너뛰어 곱셈 순서를 정리하는 ㅁ 단계를 빠뜨렸고, 그 ㅁ을 맨 뒤로 보냈다. ㅁ은 아직 카티전곱 ×과 선택을 쓰는 중간 단계이므로 이미 자연조인 $\bowtie_N$으로 정리된 ㄹ보다 뒤에 올 수 없다. 올바른 순서는 "ㄴ → ㄷ → ㅁ → ㄹ → ㄱ"이다.

오답 포인트

투영 연산 ㄱ을 빨리 적용하면 항상 더 좋다고 오해하기 쉽지만, 조인 키인 *rNo*와 *pNo*가 아직 필요하므로 투영은 마지막 단계에 가까워야 한다. 또한 ㅁ(카티전곱 × + 조인 조건 선택을 유지한 곱셈 순서 재배치)과 ㄹ(그 카티전곱+선택을 자연조인 $\bowtie_N$으로 치환)을 같은 단계로 혼동해 ㅁ을 건너뛰는 실수가 자주 나온다. 두 식은 구조가 다르므로 반드시 ㅁ 다음에 ㄹ이 온다.

다시 나오면 이렇게 판단

트리를 보면 먼저 초기 카티전곱 ×에 통합 선택이 얹힌 형태인지 확인하고, 다음으로 단일 릴레이션 선택이 아래로 내려갔는지 본다. 그다음 곱셈 순서가 정리되었는지, 이어서 카티전곱+선택이 자연조인 $\bowtie_N$으로 바뀌었는지 확인하고, 마지막에 필요한 속성만 남기는 투영 π 축소가 오는지를 보면 순서를 고를 수 있다.

암기 포인트

경험적 질의 최적화는 선택 먼저, 곱셈 순서 정리와 조인 변환 다음, 투영 마지막의 흐름으로 정리된다.

NoteBase