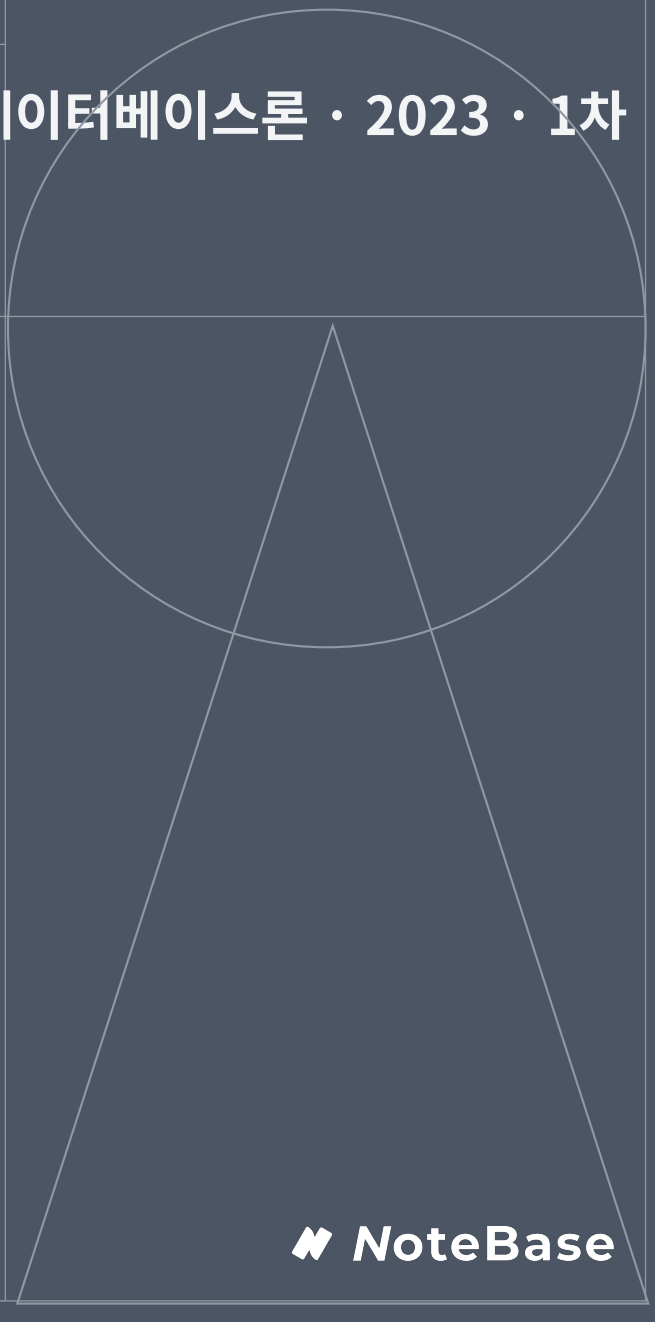
기출OX 해설집

공무원/경찰/공채(순경) · 데이터베이스론 · 2023 · 1차



NoteBase

기출OX 해설집

공무원/경찰/공채(순경) · 데이터베이스론 · 2023 · 1차

공무원/경찰/공채(순경) · 데이터베이스론 · 2023 · 1차

문 1. 논리적 데이터 독립성(logical data independence)과 물리적 데이터 독립성(physical data independence)에 관한 설명으로 가장 적절하지 않은 것은?

① 논리적 데이터 독립성은 개념 스키마가 변경되었을 때 외부 스키마에도 영향을 미치도록 지원한다.

② 물리적 데이터 독립성은 내부 스키마가 변경되어도 개념 스키마에 영향을 미치지 않도록 지원한다.

③ 논리적 데이터 독립성은 외부 스키마와 개념 스키마 사이의 사상(mapping)에 의해 제공된다.

④ 물리적 데이터 독립성은 응용프로그램의 변경 없이 성능 향상을 위한 파일의 접근구조를 수정할 수 있게 한다.

정답 1번

출제 포인트

이 문제는 3단계 스키마 구조에서 논리적 데이터 독립성과 물리적 데이터 독립성이 각각 어느 단계의 변경을 차단하는지 묻는 문제이다.

핵심 개념

논리적 데이터 독립성은 개념 스키마가 변경되어도 외부 스키마나 응용프로그램에 영향을 주지 않도록 하는 성질이며, 외부 스키마와 개념 스키마 사이의 사상으로 제공된다.
물리적 데이터 독립성은 내부 스키마가 변경되어도 개념 스키마나 응용프로그램에 영향을 주지 않도록 하는 성질이며, 개념 스키마와 내부 스키마 사이의 사상으로 제공된다.

NoteBase

선지별 해설

① X
틀린 표현은 '외부 스키마에도 영향을 미치도록'이다. 논리적 데이터 독립성은 개념 스키마가 변경되어도 외부 스키마에 영향을 미치지 않도록 지원한다. 따라서 가장 적절하지 않은 설명이다.

② O
물리적 데이터 독립성은 저장 구조를 나타내는 내부 스키마가 바뀌어도 개념 스키마가 영향을 받지 않게 하는 성질이다.

③ O
논리적 데이터 독립성은 외부 스키마와 개념 스키마 사이의 사상을 조정하여 개념 스키마의 변경이 외부 단계로 전파되지 않게 한다.

④ O
파일의 접근 구조는 내부 스키마에 속한다. 물리적 데이터 독립성이 있으면 응용프로그램을 변경하지 않고도 인덱스 등 파일 접근 구조를 수정하여 성능을 개선할 수 있다.

오답 포인트

①의 '영향을 미치도록'과 '영향을 미치지 않도록'은 정반대이다. 데이터 독립성은 하위 단계의 변경이 상위 단계로 전파되지 않게 하는 성질이라는 점을 놓치면 오답을 고르기 쉽다.

다시 나오면 이렇게 판단

변경된 단계와 보호되는 상위 단계를 확인한다. 개념 스키마 변경으로부터 외부 스키마를 보호하면 논리적 데이터 독립성이고, 내부 스키마 변경으로부터 개념 스키마를 보호하면 물리적 데이터 독립성이다.

암기 포인트

개념 변경 차단은 논리적 독립성, 내부 저장 변경 차단은 물리적 독립성이다.

NoteBase

문 2. 파일시스템(file system)과 비교할 때 DBMS의 장점으로 가장 적절하지 않은 것은?

① 데이터 일관성을 유지한다.

② 데이터 중복성을 반드시 제거한다.

③ 데이터 복구, 보안, 동시성 제어 등을 수행한다.

④ 데이터 독립성을 유지한다.

정답 2번

출제 포인트

이 문제는 파일시스템과 비교한 DBMS의 일반적 장점과 데이터 중복 제어의 의미를 묻는 문제이다.

핵심 개념

DBMS는 여러 사용자가 데이터를 체계적으로 저장·검색·갱신하도록 관리하는 소프트웨어이다. 파일시스템에 비해 데이터 일관성, 복구와 보안, 동시성 제어, 데이터 독립성을 지원하고 불필요한 중복을 줄일 수 있다. 다만 성능이나 가용성 등을 위해 의도적인 중복을 허용할 수 있으므로 모든 중복을 반드시 제거하는 것은 아니다.

선지별 해설

① O — DBMS는 무결성 제약과 중앙화된 관리를 통해 동일한 데이터가 서로 모순되지 않도록 데이터 일관성을 유지할 수 있다.
② X — 틀린 표현은 '중복성을 반드시 제거한다'이다. DBMS는 불필요한 데이터 중복을 제어하거나 최소화하지만, 성능·가용성 등의 목적에 따라 필요한 중복은 허용할 수 있다.
③ O — DBMS는 장애 발생 시의 복구, 접근 권한에 따른 보안, 여러 트랜잭션이 동시에 실행될 때의 동시성 제어 기능을 제공한다.
④ O — DBMS는 데이터의 논리적 구조나 물리적 저장 방식이 바뀌더라도 응용 프로그램에 미치는 영향을 줄이는 데이터 독립성을 지원한다.

오답 포인트

②의 '반드시'가 함정이다. DBMS의 장점은 중복을 무조건 없애는 것이 아니라 불필요한 중복을 통제하여 불일치 가능성을 낮추는 데 있다.

다시 나오면 이렇게 판단

DBMS의 장점을 묻는 선지에서는 일관성·보안·복구·동시성 제어·데이터 독립성을 확인한다. '중복을 완전히 제거한다'처럼 예외를 허용하지 않는 절대적 표현은 중복 제어 또는 최소화로 고쳐 판단한다.

NoteBase

암기 포인트

DBMS는 중복 제거가 아니라 중복 제어이다.

NoteBase

문 3. 시스템 카탈로그(system catalog)라고도 불리는 데이터 사전(data dictionary)의 저장내용으로 가장 적절하지 않은 것은?

① 릴레이션 이름 및 속성

② 무결성 제약조건

③ 릴레이션 스키마에 실제로 저장된 데이터의 집합

④ 스키마 사이의 사상(mapping) 정보

정답 3번

출제 포인트

데이터 사전에 저장되는 메타데이터와 릴레이션에 실제로 저장되는 데이터 인스턴스를 구별하는 문제이다.

핵심 개념

데이터 사전은 데이터베이스의 구조와 관리 정보를 저장하는 시스템 카탈로그이다. 릴레이션 이름·속성, 무결성 제약조건, 스키마 간 사상 등은 메타데이터이므로 데이터 사전에 저장된다. 반면 실제 레코드의 집합은 릴레이션 인스턴스이며 데이터 사전의 저장 내용이 아니다.

선지별 해설

① O — 릴레이션 이름과 속성 정보는 데이터베이스 구조를 설명하는 메타데이터이므로 데이터 사전에 저장된다.
② O — 기본키, 외래키, 도메인 제약 등 무결성 제약조건은 데이터의 유효성을 관리하기 위한 메타데이터이므로 데이터 사전에 저장된다.
③ X — 틀린 표현은 '릴레이션 스키마에 실제로 저장된 데이터의 집합'이다. 이는 데이터 사전의 메타데이터가 아니라 특정 시점의 실제 레코드 집합인 릴레이션 인스턴스이다. 올바르게는 데이터 사전에는 실제 데이터 자체가 아니라 그 데이터를 정의하고 관리하는 스키마 정보가 저장된다고 해야 한다.
④ O — 외부·개념·내부 스키마 사이의 대응 관계인 사상 정보는 데이터 독립성과 변환을 지원하는 메타데이터이므로 데이터 사전에 저장된다.

오답 포인트

③의 '릴레이션 스키마에 실제로 저장된 데이터의 집합'을 스키마 정보로 착각하기 쉽다. 스키마는 구조에 대한 정의이고, 그 구조에 따라 현재 저장된 실제 데이터의 집합은 인스턴스이다.

다시 나오면 이렇게 판단

선지가 데이터의 구조·제약·대응 관계를 설명하면 메타데이터로 판단하고, 실제 행이나 레코드의 값을 가리키면 데이터 인스턴스로 판단한다.

NoteBase

암기 포인트

데이터 사전은 데이터 자체가 아니라 데이터에 관한 데이터인 메타데이터를 저장한다.

NoteBase

문 4. RAID(Redundant Arrays of Independent Disk)에 관한 설명으로 가장 적절하지 않은 것은?

① RAID 0은 여분의 데이터를 갖지 않고, 데이터의 갱신이 중복해서 일어나지 않는다.

② RAID 1은 하나의 드라이브가 고장나면 또 다른 드라이브를 통해 복구할 수 없다.

③ RAID 3은 에러 검출·수정을 위해 별도의 패리티(parity) 드라이브를 사용한다.

④ RAID 5는 별도의 패리티 드라이브 대신 모든 드라이브에 패리티 정보를 나누어 기록하는 방식이다.

정답 2번

출제 포인트

RAID 0, RAID 1, RAID 3, RAID 5가 데이터를 분산하거나 중복 저장하고 패리티를 배치하는 방식을 구별하는 문제이다.

핵심 개념

RAID 0: 데이터를 여러 드라이브에 나누어 기록하는 스트라이핑 방식이며 중복 데이터나 패리티가 없어 장애 복구 기능이 없다.
RAID 1: 같은 데이터를 둘 이상의 드라이브에 복제하는 미러링 방식이며 한 드라이브가 고장 나도 다른 드라이브의 복제본으로 복구할 수 있다.
RAID 3: 데이터를 분산 기록하면서 별도의 전용 패리티 드라이브를 사용하는 방식이다.
RAID 5: 데이터와 패리티를 블록 단위로 여러 드라이브에 분산하여 기록하는 방식이다.

선지별 해설

① O — RAID 0은 스트라이핑만 사용하므로 여분의 데이터가 없고, 미러링처럼 같은 갱신을 중복 수행하지 않는다.
② X — 틀린 표현은 '또 다른 드라이브를 통해 복구할 수 없다'이다. RAID 1은 동일한 데이터를 복제해 두므로 한 드라이브가 고장 나면 정상인 다른 드라이브의 데이터로 복구할 수 있다.
③ O — RAID 3은 오류 검출과 복구에 사용할 패리티 정보를 별도의 전용 패리티 드라이브에 저장한다.
④ O — RAID 5는 전용 패리티 드라이브를 두지 않고 패리티 블록을 구성 드라이브들에 분산하여 기록한다.

오답 포인트

RAID 1의 '중복 저장'을 비효율로만 보고 장애 복구가 불가능하다고 판단하기 쉽다. 그러나 그 복제본 자체가 복구 수단이다. 또한 RAID 3의 전용 패리티와 RAID 5의 분산 패리티를 서로 바꾸어 기억하기 쉽다.

다시 나오면 이렇게 판단

각 RAID 수준을 '데이터 복제 여부'와 '패리티의 위치'로 구별한다. RAID 0은 중복 없음, RAID 1은 미러링, RAID 3은 전용 패리티 드라이브, RAID 5는 분산 패리티에 해당한다.

NoteBase

암기 포인트

0은 분산만, 1은 복제, 3은 전용 패리티, 5는 분산 패리티이다.

문 5. B-트리, B^*-트리, B^+-트리에 관한 설명으로 가장 적절하지 않은 것은?

① B-트리에서 리프(leaf) 노드들은 모두 순차적으로 포인터로 연결되어 있다.

② B^*-트리는 각 노드 분열의 빈도를 줄이기 위해 사용된다.

③ B^+-트리에서 한 노드 안에 있는 키 값들은 오름차순이다.

④ B^+-트리에서 모든 리프는 같은 레벨(level)이다.

정답 1번

출제 포인트

이 문제는 B-트리, B^*-트리, B^+-트리의 노드 구성과 리프 연결 방식의 차이를 묻는다.

핵심 개념

B-트리는 모든 노드에 키와 레코드 포인터를 둘 수 있는 균형 탐색 트리이다. 리프 노드의 순차 연결은 B-트리의 필수 조건이 아니다.
B^*-트리는 노드가 가득 차면 먼저 형제 노드와 키를 재분배하고, 필요한 경우 여러 노드를 분할하여 공간 활용률을 높이고 분할 빈도를 줄인 변형이다.
B^+-트리는 내부 노드를 탐색용 키와 자식 포인터로 사용하고 실제 레코드 접근 정보는 리프에 두며, 리프들을 순차적으로 연결하여 범위 검색에 유리한 균형 트리이다.

선지별 해설

① X
틀린 부분은 'B-트리에서 리프 노드들은 모두 순차적으로 포인터로 연결되어 있다'이다. 리프 노드의 순차 연결은 일반적인 B-트리의 필수 특성이 아니며, 올바르게는 B^+-트리의 리프 노드들이 순차 포인터로 연결된다.

② O
B^*-트리는 노드가 가득 찼을 때 곧바로 분할하지 않고 인접 형제 노드와 먼저 재분배하므로 노드 분할의 빈도를 줄일 수 있다.

③ O
B^+-트리의 각 노드 안에서는 탐색이 가능하도록 키 값이 오름차순으로 정렬된다.

④ O
B^+-트리는 높이 균형을 유지하므로 모든 리프 노드가 같은 레벨에 있다.

NoteBase

오답 포인트

①은 B^+-트리의 대표 특징인 '리프 노드의 순차 연결'을 B-트리의 특징으로 바꾸어 놓은 선지이다. 세 트리 모두 균형 탐색 트리라는 공통점 때문에 리프 구조까지 같다고 판단하기 쉽다.

다시 나오면 이렇게 판단

'리프의 순차 연결'은 B^+-트리, '형제와의 재분배를 통한 분할 감소'는 B^*-트리의 핵심 특징으로 구분한다. 모든 리프가 같은 레벨이라는 성질은 높이 균형을 유지하는 B-트리 계열의 공통 성질로 판단한다.

암기 포인트

B^+: 리프 연결, B^*: 재분배로 분할 감소, B-트리 계열: 모든 리프의 높이 동일이다.

NoteBase

문 6. 데이터 정의어에 관한 설명으로 가장 적절하지 않은 것은?

① CREATE는 테이블과 뷰를 생성하는 명령어이다.

② DROP은 테이블과 뷰를 삭제하는 명령어이다.

③ TRUNCATE는 테이블을 삭제하는 명령어이다.

④ RENAME은 테이블 이름을 변경하는 명령어이다.

정답 3번

출제 포인트

이 문제는 데이터 정의어가 데이터베이스 객체의 구조를 생성·삭제·변경하는 방식과, 테이블의 행만 제거하는 방식을 구분하는 문제이다.

핵심 개념

CREATE는 테이블·뷰 등의 데이터베이스 객체를 생성하고, DROP은 객체 자체를 삭제한다. TRUNCATE는 테이블의 모든 행을 제거하지만 테이블 구조는 남긴다. RENAME은 객체의 이름을 변경한다.

선지별 해설

① O — CREATE는 테이블이나 뷰와 같은 데이터베이스 객체를 생성하는 데이터 정의어이다.
② O — DROP은 지정한 테이블이나 뷰 객체 자체를 데이터베이스에서 삭제하는 데이터 정의어이다.
③ X — 틀린 표현은 '테이블을 삭제'한다는 부분이다. TRUNCATE는 테이블 자체를 삭제하는 것이 아니라 테이블의 모든 행을 제거하고 테이블 구조는 유지한다. 테이블 자체를 삭제하는 명령어는 DROP이다.
④ O — RENAME은 테이블 등 데이터베이스 객체의 이름을 변경하는 데이터 정의어이다.

오답 포인트

③은 TRUNCATE 실행 후 데이터가 없어지는 모습을 '테이블 삭제'로 표현한 것이 함정이다. 행이 모두 제거되는 것과 테이블 객체 및 구조가 삭제되는 것은 다르다.

다시 나오면 이렇게 판단

명령 실행 후 객체의 구조가 남는지를 기준으로 판단한다. 구조까지 없어지면 DROP이고, 구조를 남긴 채 모든 행만 제거하면 TRUNCATE이다.

암기 포인트

CREATE는 생성, DROP은 객체 삭제, TRUNCATE는 전체 행 제거, RENAME은 이름 변경이다.

NoteBase

문 7. 관계 대수의 연산자 중 교환법칙이 성립하지 않은 것은?

① 합집합

② 교집합

③ 자연조인

④ 차집합

정답 4번

출제 포인트

이 문제는 두 피연산 관계의 순서를 바꾸어도 결과가 같은 교환법칙이 관계대수 연산별로 성립하는지를 묻는다.

핵심 개념

교환법칙은 두 관계의 순서를 바꾸어도 연산 결과가 같은 성질, 즉 R S = S R가 성립하는 성질이다. 합집합, 교집합, 자연조인은 교환법칙이 성립하지만 차집합은 어느 관계에서 어느 관계를 빼는지에 따라 결과가 달라진다.

선지별 해설

① X — 합집합은 $R \cup S = S \cup R$이므로 교환법칙이 성립한다. 틀린 판단은 '합집합에서 교환법칙이 성립하지 않는다'는 부분이며, 올바르게는 합집합에서 교환법칙이 성립한다.
② X — 교집합은 $R \cap S = S \cap R$이므로 교환법칙이 성립한다. 틀린 판단은 '교집합에서 교환법칙이 성립하지 않는다'는 부분이며, 올바르게는 교집합에서 교환법칙이 성립한다.
③ X — 자연조인은 공통 속성값이 같은 튜플들을 결합하는 연산이며, 관계를 바꾸어 놓아도 같은 튜플 집합을 얻으므로 $R \bowtie S = S \bowtie R$이다. 틀린 판단은 '자연조인에서 교환법칙이 성립하지 않는다'는 부분이며, 올바르게는 자연조인에서 교환법칙이 성립한다.
④ O — 차집합 $R - S$는 R에는 있으나 S에는 없는 튜플의 집합이므로 일반적으로 $R - S \neq S - R$이다. 따라서 차집합은 교환법칙이 성립하지 않는다.

오답 포인트

자연조인이 두 관계를 결합하므로 순서에 민감하다고 오해하기 쉽다. 그러나 관계대수에서 관계는 튜플의 집합이므로 자연조인의 결과는 피연산 관계의 순서와 무관하다. 반면 차집합은 '앞의 관계에서 뒤의 관계를 제거한다'는 방향성이 있다.

다시 나오면 이렇게 판단

연산 기호의 좌우를 바꾸어 본다. 합집합·교집합·자연조인은 결과가 같고, 차집합은 기준이 되는 앞쪽 관계가 바뀌므로 결과가 일반적으로 달라진다.

NoteBase

암기 포인트

합·교·자연조인은 교환 가능, 차집합은 방향이 있어 교환 불가이다.

NoteBase

문 8. 단일 릴레이션 특징에 관한 설명으로 가장 적절한 것은?

① 동일 이름의 속성들을 가질 수 있다.

② 튜플의 순서가 다르면 다른 릴레이션이다.

③ 속성의 순서가 다르면 릴레이션 스키마도 다르다.

④ 각 속성의 값은 도메인에 정의된 값만을 가지며 그 값은 모두 단일값(원자값)이어야 한다.

정답 4번

출제 포인트

이 문제는 단일 릴레이션에서 속성 이름과 순서, 튜플의 순서, 속성값이 따라야 하는 원칙을 묻는 문제이다.

핵심 개념

릴레이션은 동일한 구조를 따르는 튜플들의 집합이다. 한 릴레이션 안에서 속성 이름은 서로 달라야 하고, 속성과 튜플의 순서는 본질적 의미가 없다. 각 속성값은 해당 속성의 도메인에 속하는 하나의 원자값이어야 한다.

선지별 해설

① X
틀린 표현은 '동일 이름의 속성들을 가질 수 있다'이다. 한 릴레이션을 구성하는 속성들은 서로 구별되는 이름을 가져야 하므로, 올바르게는 '동일 이름의 속성들을 가질 수 없다'이다.

② X
틀린 표현은 '튜플의 순서가 다르면 다른 릴레이션이다'이다. 릴레이션은 튜플의 집합이므로 튜플의 나열 순서는 릴레이션의 의미에 영향을 주지 않는다. 올바르게는 '같은 튜플들로 구성되었다면 튜플의 순서가 달라도 같은 릴레이션이다'이다.

③ X
틀린 표현은 '속성의 순서가 다르면 릴레이션 스키마도 다르다'이다. 릴레이션 스키마에서 속성의 배열 순서는 본질적 의미가 없으므로, 올바르게는 '같은 속성들로 구성되었다면 속성의 순서가 달라도 같은 릴레이션 스키마이다'이다.

④ O
각 속성값은 그 속성에 정의된 도메인의 값이어야 하며, 관계형 모델에서는 하나의 속성 위치에 더 분해되는 값들의 집합이 아니라 하나의 원자값이 들어간다.

NoteBase

오답 포인트

②와 ③은 표 형태로 출력할 때 행과 열이 일정한 순서로 보인다는 점을 이용한 함정이다. 화면에 표시되는 순서와 릴레이션의 논리적 성질을 구분해야 한다.

다시 나오면 이렇게 판단

단일 릴레이션의 성질은 '속성 이름은 유일, 행과 열의 순서는 무의미, 속성값은 도메인에 속하는 원자값'이라는 기준으로 판단한다.

암기 포인트

이름은 서로 다르게, 순서는 의미 없게, 값은 도메인의 원자값으로 기억한다.

NoteBase

문 9. 다음은 무결성의 종류에 관한 설명이다. ㉠~㉢에서 설명하는 무결성의 종류가 바르게 짝지어진 것은?

㉠ 기본키를 구성하는 어떠한 속성값도 NULL 값이거나 중복값을 가질 수 없다.
㉡ 컬럼 데이터 타입, 길이, 유효 값이 일관되게 유지되어야 한다.
㉢ 참조하는 릴레이션의 외래키와 참조되는 릴레이션의 기본키의 도메인은 서로 같아야 한다.

① ㉠: 개체 무결성, ㉡: 참조 무결성, ㉢: 도메인 무결성
② ㉠: 참조 무결성, ㉡: 도메인 무결성, ㉢: 개체 무결성
③ ㉠: 개체 무결성, ㉡: 도메인 무결성, ㉢: 참조 무결성
④ ㉠: 도메인 무결성, ㉡: 참조 무결성, ㉢: 개체 무결성

정답 3번

출제 포인트

기본키, 속성의 허용 범위, 외래키와 참조 대상의 관계를 각각 개체 무결성, 도메인 무결성, 참조 무결성으로 구별하는 문제이다.

핵심 개념

개체 무결성: 각 행을 유일하게 식별하는 기본키의 속성값은 NULL이 될 수 없고 중복될 수도 없다는 원칙이다.
도메인 무결성: 한 속성에 저장되는 값이 정해진 데이터 타입, 길이, 형식, 범위 등 허용된 도메인을 따라야 한다는 원칙이다.
참조 무결성: 외래키가 참조 관계의 일관성을 유지해야 한다는 원칙으로, 외래키와 참조되는 키의 도메인이 호환되어야 하고 외래키 값은 원칙적으로 참조되는 키에 존재하는 값이어야 한다.
따라서 ㉠은 개체 무결성, ㉡은 도메인 무결성, ㉢은 참조 무결성이다.

NoteBase

선지별 해설

① X
㉠을 개체 무결성으로 본 것은 맞지만, ㉡과 ㉢을 서로 바꾸어 연결하였다. 틀린 표현은 '㉡: 참조 무결성, ㉢: 도메인 무결성'이며, 올바르게는 '㉡: 도메인 무결성, ㉢: 참조 무결성'이다.

② X
㉠~㉢을 모두 잘못 연결하였다. 기본키의 NULL·중복 금지는 개체 무결성, 컬럼의 데이터 타입·길이·유효 값 유지는 도메인 무결성, 외래키와 참조되는 기본키의 관계 유지는 참조 무결성이므로 올바른 연결은 '㉠: 개체 무결성, ㉡: 도메인 무결성, ㉢: 참조 무결성'이다.

③ O
㉠의 기본키 제약은 개체 무결성, ㉡의 컬럼 허용 값 제약은 도메인 무결성, ㉢의 외래키와 참조되는 기본키 사이의 제약은 참조 무결성에 해당하므로 세 연결이 모두 맞다.

④ X
㉠~㉢을 모두 잘못 연결하였다. '㉠: 도메인 무결성, ㉡: 참조 무결성, ㉢: 개체 무결성'이 아니라 '㉠: 개체 무결성, ㉡: 도메인 무결성, ㉢: 참조 무결성'이 올바른 연결이다.

오답 포인트

㉡과 ㉢에는 모두 '도메인'과 관련된 표현이 보여 혼동하기 쉽다. 그러나 ㉡은 한 컬럼 자체에 허용되는 값의 범위를 다루므로 도메인 무결성이고, ㉢은 외래키가 다른 릴레이션의 기본키를 참조하는 관계를 다루므로 참조 무결성이다.

다시 나오면 이렇게 판단

기본키의 NULL·중복 금지는 개체 무결성, 속성값의 타입·길이·범위 제한은 도메인 무결성, 외래키와 참조되는 키 사이의 일관성은 참조 무결성으로 판별한다.

암기 포인트

기본키는 개체, 컬럼 값의 범위는 도메인, 외래키의 참조 관계는 참조 무결성이다.

NoteBase

문 10. 외래키(foreign key)의 특징에 관한 설명으로 가장 적절하지 않은 것은?

① 외래키에는 NULL 값과 중복값 등이 허용된다.

② 한 릴레이션에 외래키가 여러 개 존재할 수 있다.

③ 외래키도 기본키(primary key)의 구성 요소가 될 수 있다.

④ 외래키 사용 시 참조하는 릴레이션과 참조되는 릴레이션은 다르다.

정답 4번

출제 포인트

외래키의 값·개수·키 구성 가능성과 자기참조 관계를 구분하는 문제이다.

핵심 개념

외래키는 한 릴레이션의 속성 또는 속성 집합이 다른 릴레이션이나 자기 릴레이션의 후보키를 참조하도록 하는 키이다. 외래키 값은 참조 대상 키의 값과 일치하거나, 허용된 경우 NULL일 수 있다. 기본키와 달리 외래키 자체에는 일반적으로 중복값도 허용된다.

선지별 해설

① O — 외래키에는 일반적으로 중복값이 허용되며, 해당 속성에 NOT NULL 제약이 없다면 NULL도 허용된다.
② O — 한 릴레이션이 여러 릴레이션을 참조하거나 같은 릴레이션을 여러 방식으로 참조할 수 있으므로 외래키가 여러 개 존재할 수 있다.
③ O — 한 속성이 외래키이면서 동시에 기본키의 전부 또는 일부가 될 수 있다. 예를 들어 연결 릴레이션에서는 두 외래키가 복합 기본키를 구성할 수 있다.
④ X — 틀린 표현은 '참조하는 릴레이션과 참조되는 릴레이션은 다르다'이다. 외래키는 다른 릴레이션뿐 아니라 같은 릴레이션의 후보키도 참조할 수 있으므로, 올바르게는 두 릴레이션이 서로 다를 수도 있고 같을 수도 있다.

오답 포인트

④의 '다르다'를 외래키의 필수 조건으로 오해하기 쉽다. 조직도의 상사 번호처럼 한 테이블의 행이 같은 테이블의 다른 행을 참조하는 자기참조 외래키도 가능하다.

다시 나오면 이렇게 판단

외래키 선지는 참조 무결성만 확인하지 않고 NULL·중복 허용 여부, 여러 외래키의 존재 가능성, 기본키와의 중첩 가능성, 자기참조 가능성을 각각 구분해 판단한다.

암기 포인트

외래키는 중복·NULL·복수 지정·기본키와의 중첩·자기참조가 모두 가능하다.

NoteBase

문 11. 관계 대수의 일반 집합 연산자의 결과 릴레이션의 cardinality로 가장 적절하지 않은 것은?

① 일반 집합 연산자: 합집합(R ∪ S), cardinality: $|R \cup S| \leq |R| + |S|$

② 일반 집합 연산자: 교집합(R ∩ S), cardinality: $|R \cap S| \leq \text{MIN}\{|R|, |S|\}$

③ 일반 집합 연산자: 차집합(R - S), cardinality: $|R - S| \leq |S|$

④ 일반 집합 연산자: 카티션 프로덕트(R × S), cardinality: $|R \times S| = |R| \times |S|$

정답 3번

출제 포인트

관계 대수의 합집합·교집합·차집합·카티션 프로덕트에서 결과 릴레이션의 카디널리티가 어떤 입력 릴레이션의 크기로 제한되는지를 묻는 문제이다.

핵심 개념

카디널리티는 릴레이션에 포함된 튜플의 수이다. 합집합은 두 릴레이션의 튜플을 합친 집합, 교집합은 양쪽에 공통인 튜플의 집합, 차집합 $R - S$는 R에는 있고 S에는 없는 튜플의 집합이다. 따라서 $R - S$는 항상 R의 부분집합이다. 카티션 프로덕트는 R의 각 튜플과 S의 각 튜플을 모두 한 번씩 결합한다.

선지별 해설

① O — 합집합에서는 중복 튜플이 제거되므로 $|R \cup S| = |R| + |S| - |R \cap S|$이고, 따라서 $|R \cup S| \leq |R| + |S|$이다.
② O — 교집합은 R과 S 모두의 부분집합이므로 $|R \cap S| \leq \min\{|R|, |S|\}$이다.
③ X — 틀린 표현은 $|R - S| \leq |S|$이다. $R - S$는 R의 부분집합이므로 올바른 상한은 $|R - S| \leq |R|$이다. R과 S가 서로소이면 $|R - S| = |R|$이므로 결과가 $|S|$보다 클 수도 있다.
④ O — R의 각 튜플마다 S의 모든 튜플이 결합되므로 $|R \times S| = |R| \times |S|$이다.

오답 포인트

③에서 차집합 기호 오른쪽의 S를 결과 크기의 기준으로 착각하기 쉽다. $R - S$의 결과로 남을 수 있는 튜플은 처음부터 R에 있던 튜플뿐이므로 상한의 기준은 S가 아니라 R이다.

다시 나오면 이렇게 판단

집합 연산의 결과가 어느 입력의 부분집합인지 먼저 확인한다. 교집합은 양쪽 모두의 부분집합이고, $R - S$는 왼쪽 피연산자 R의 부분집합이며, 카티션 프로덕트는 가능한 모든 튜플 쌍의 수를 곱한다.

암기 포인트

차집합의 크기 상한은 왼쪽 릴레이션이다: $|R - S| \leq |R|$.

NoteBase

문 12. 트리거(trigger)에 관한 설명으로 가장 적절하지 않은 것은?

① 트리거는 데이터의 변경문이 실행될 때 자동으로 실행되는 프로시저이다.

② 트리거는 사건(삽입, 삭제, 갱신) 전(BEFORE)에만 활성화될 수 있다.

③ 트리거는 SELECT에 대해서는 이벤트가 발생하지 않는다.

④ 트리거를 사용하면 하나의 릴레이션이 변경될 때 다른 릴레이션도 변경될 수 있다.

정답 2번

출제 포인트

트리거의 실행 시점과 대상 사건을 구분하는 문제이다.

핵심 개념

트리거는 테이블의 삽입·삭제·갱신과 같은 지정된 사건이 발생하면 DBMS가 자동으로 실행하는 절차이다.
트리거의 실행 시점은 사건 전(BEFORE)으로만 한정되지 않고 사건 후(AFTER)로도 정할 수 있다.

선지별 해설

① O — 트리거는 삽입·삭제·갱신 등 데이터 변경 사건이 발생할 때 DBMS가 자동으로 실행하는 프로시저이다.
② X — 틀린 표현은 '전(BEFORE)에만'이다. 트리거는 사건 전(BEFORE)뿐 아니라 사건 후(AFTER)에도 활성화되도록 정의할 수 있다.
③ O — SELECT는 데이터를 조회할 뿐 삽입·삭제·갱신과 같은 데이터 변경 사건을 일으키지 않으므로, 일반적인 데이터 변경 트리거의 대상 사건이 아니다.
④ O — 한 릴레이션의 변경으로 실행된 트리거가 다른 릴레이션을 삽입·삭제·갱신하도록 정의될 수 있으므로 다른 릴레이션도 변경될 수 있다.

오답 포인트

②는 BEFORE라는 실제 실행 시점을 제시한 뒤 '에만'을 붙여 범위를 잘못 제한한다. 트리거가 사건 발생 전에만 작동한다고 오해하면 AFTER 트리거를 놓치게 된다.

다시 나오면 이렇게 판단

트리거 선지는 사건 종류와 실행 시점을 나누어 판단한다. 사건 종류는 주로 INSERT·DELETE·UPDATE이고, 실행 시점은 BEFORE와 AFTER가 모두 가능하므로 'BEFORE에만'과 같은 배타적 표현을 먼저 확인한다.

암기 포인트

트리거는 변경 사건에 자동 실행되며, 시점은 BEFORE만이 아니라 AFTER도 가능하다.

NoteBase

문 13. Jung이 생성한 다음 부서 테이블에 따라 각 사용자가 갖게 되는 권한에 관한 설명으로 가장 적절하지 않은 것은?

SQL 문	
Jung의 권한부여	GRANT SELECT, INSERT, DELETE ON 부서 TO Son, Han WITH GRANT OPTION;
Son의 권한부여	GRANT SELECT, DELETE ON 부서 TO Lee;

① Jung은 부서 테이블에 UPDATE 할 수 있는 권한이 있다.

② Son은 부서 테이블에 INSERT 할 수 있는 권한이 있다.

③ Han은 부서 테이블에 SELECT 할 수 있는 권한이 있다.

④ Lee는 부서 테이블에 INSERT 할 수 있는 권한이 있다.

정답 4번

출제 포인트

GRANT 문으로 부여된 권한의 종류와 WITH GRANT OPTION에 따른 권한 재부여 범위를 판단하는 문제이다.

핵심 개념

GRANT는 사용자에게 지정한 객체 권한을 부여하는 명령이다. WITH GRANT OPTION을 함께 사용하면 권한을 받은 사용자가 그 권한을 다른 사용자에게 다시 부여할 수 있다. Jung은 Son과 Han에게 SELECT, INSERT, DELETE 권한을 재부여 권한과 함께 주었고, Son은 Lee에게 그중 SELECT와 DELETE만 부여하였다.

선지별 해설

① O — Jung은 부서 테이블의 생성자이므로 테이블에 대한 UPDATE 권한을 갖는다.
② O — Jung이 Son에게 SELECT, INSERT, DELETE 권한을 부여했으므로 Son은 INSERT를 실행할 수 있다.
③ O — Jung이 Han에게 SELECT, INSERT, DELETE 권한을 부여했으므로 Han은 SELECT를 실행할 수 있다.
④ X — 틀린 표현은 'Lee는 INSERT 할 수 있다'이다. Son이 Lee에게 부여한 권한은 SELECT와 DELETE뿐이므로, 올바르게는 'Lee는 부서 테이블에 SELECT와 DELETE를 할 수 있다'이다.

NoteBase

오답 포인트

첫 번째 GRANT 문의 INSERT 권한이 Lee에게도 이어진다고 오해하기 쉽다. WITH GRANT OPTION은 재부여를 허용할 뿐 원래 권한 전체를 자동으로 전달하지 않으며, Lee가 얻는 권한은 두 번째 GRANT 문에 명시된 SELECT와 DELETE뿐이다.

다시 나오면 이렇게 판단

사용자별 권한을 GRANT 문의 수신자와 권한 목록에 따라 단계별로 추적한다. 재부여가 있더라도 새 GRANT 문에 실제로 적힌 권한만 다음 사용자에게 넘어간다.

암기 포인트

WITH GRANT OPTION은 ‘받은 권한을 다시 줄 수 있음’이고, 실제 재부여 범위는 새 GRANT 문에 명시된 권한만이다.

NoteBase

문 14. 다음 SQL 문은 테이블 DEPT와 EMP의 정의문 일부인데, 테이블 정보의 테이블 EMP에서 잘못된 행(row)을 모두 고른 것은? (단, SQL 문에서 제시된 내용만 고려할 것)

SQL 문

```
CREATE TABLE DEPT (
    ...
    PRIMARY KEY(DID));
CREATE TABLE EMP (
    ...
    PRIMARY KEY(ENO),
    FOREIGN KEY(DID) REFERENCES DEPT(DID),
    CHECK(YEAR > 1970 AND YEAR < 2000));
```

테이블 정보

DEPT

DID	DNAME
A	인사부
B	총무부
C	영업부

EMP

	ENO	ENAME	YEAR	DID
㉠	1	이순신	1960	A
㉡	2	박문수	1980	NULL
㉢	3	홍길동	1977	D
㉣	4	NULL	1995	B
㉤	NULL	손오공	1999	B

① ㉠㉡㉣

② ㉠㉢㉤

③ ㉢㉣㉤

④ ㉡㉢㉣

정답 2번

출제 포인트

이 문제는 기본키, 외래키, CHECK 제약조건을 EMP의 각 행에 적용하여 무결성 제약을 위반한 행을 찾는 문제이다.

사례 정리

DEPT의 기본키 DID에는 A, B, C가 존재한다. EMP에서는 ENO가 기본키이고, DID가 DEPT(DID)를 참조하는 외래키이며, YEAR는 1970보다 크고 2000보다 작아야 한다. SQL 문에 ENAME과 DID의 NOT NULL 조건은 제시되지 않았다.

NoteBase

핵심 개념

기본키는 중복될 수 없고 NULL도 허용하지 않는다. 외래키는 참조 대상 기본키에 존재하는 값이어야 하지만, 해당 열에 NOT NULL 조건이 없다면 NULL은 허용된다. CHECK 조건은 해당 열의 값이 제시된 논리식을 만족하도록 제한한다.

적용 기준

㉠은 YEAR가 1960이므로 CHECK 조건을 위반한다. ㉡은 YEAR가 1980이고 DID의 NULL도 허용되므로 잘못된 행이 아니다. ㉢은 DID가 D인데 DEPT의 DID에 D가 없어 외래키를 위반한다. ㉣은 ENAME이 NULL이지만 ENAME에 NOT NULL 조건이 제시되지 않아 허용된다. ㉤은 기본키 ENO가 NULL이므로 기본키 제약을 위반한다. 따라서 잘못된 행은 ㉠, ㉢, ㉤이다.

선지별 해설

① X — ㉣은 ENAME의 NULL을 금지하는 조건이 없어 올바른 행이고, 외래키 위반 행인 ㉢과 기본키 위반 행인 ㉤이 빠져 있다. 틀린 조합 ㉠㉡㉣ → 올바르게는 ㉠㉢㉤이다.
② O — ㉠은 CHECK, ㉢은 외래키, ㉤은 기본키 제약을 각각 위반하므로 잘못된 행을 모두 고른 조합이다.
③ X — ㉣은 허용되는 행인데 포함되었고 CHECK를 위반한 ㉠이 빠져 있다. 틀린 조합 ㉢㉣㉤ → 올바르게는 ㉠㉢㉤이다.
④ X — ㉡과 ㉣은 모두 허용되는 행이고, 실제 위반 행인 ㉠과 ㉤이 빠져 있다. 틀린 조합 ㉡㉢㉣ → 올바르게는 ㉠㉢㉤이다.

오답 포인트

외래키의 NULL을 무조건 위반으로 보거나, 제시되지 않은 ENAME의 NOT NULL 조건을 임의로 가정하면 ㉡ 또는 ㉣을 잘못 고르게 된다. 문제의 단서대로 CREATE TABLE 문에 실제로 제시된 제약조건만 적용해야 한다.

다시 나오면 이렇게 판단

각 행을 기본키의 NULL·중복 여부, 외래키의 참조값 존재 여부, CHECK 식 충족 여부 순서로 검사한다. NULL 허용 여부는 해당 열에 NOT NULL 또는 기본키 조건이 실제로 선언되었는지 확인하여 판단한다.

암기 포인트

기본키는 NULL 불가, 외래키는 참조값 또는 NULL, CHECK는 조건식 충족이다.

NoteBase

문 15. 다음 그림과 같은 로그(log) 기록을 가지는 4개의 트랜잭션(T_1~T_4)을 복구하기 위한 갱신방법에 따른 회복기법으로 올바르게 짝지어진 것은?



① 즉시 갱신: UNDO : T_2, 지연 갱신: REDO : T_2
② 즉시 갱신: REDO : T_3, 지연 갱신: REDO : T_3
③ 즉시 갱신: UNDO : T_3, 지연 갱신: REDO : T_2
④ 즉시 갱신: REDO : T_2, 지연 갱신: REDO : T_3

정답 2번

출제 포인트

이 문제는 검사점과 시스템 고장 사이에 완료된 트랜잭션을 찾고, 즉시 갱신과 지연 갱신에서 필요한 REDO·UNDO 대상을 구분하는 문제이다.

핵심 개념

즉시 갱신은 트랜잭션이 완료되기 전에도 변경값이 데이터베이스에 반영될 수 있으므로, 고장 시 미완료 트랜잭션은 UNDO하고 검사점 이후 완료된 트랜잭션은 REDO한다.
지연 갱신은 트랜잭션이 완료되기 전에는 변경값을 데이터베이스에 반영하지 않으므로 UNDO가 필요 없고, 검사점 이후 완료된 트랜잭션만 REDO한다.
그림에서 T_2는 검사점 전에 완료되고, T_3는 검사점 후 고장 전에 완료되며, T_1과 T_4는 고장 시점에 미완료 상태이다.

NoteBase

선지별 해설

① X

즉시 갱신에서 T_2는 검사점 전에 이미 완료되었으므로 UNDO 대상이 아니다. 지연 갱신에서도 검사점 후 완료된 T_3가 REDO 대상이므로 'REDO: T_2'가 틀렸다. 올바르게는 즉시 갱신과 지연 갱신 모두 'REDO: T_3'이다.

② O

T_3는 검사점 이후에 완료되고 시스템 고장 전에 커밋되었다. 따라서 즉시 갱신과 지연 갱신 모두 T_3의 갱신 결과를 보장하기 위해 REDO한다.

③ X

T_3는 고장 전에 완료된 트랜잭션이므로 즉시 갱신에서 UNDO할 대상이 아니다. 또한 지연 갱신의 REDO 대상은 검사점 전에 완료된 T_2가 아니라 검사점 후 완료된 T_3이다. 올바르게는 즉시 갱신과 지연 갱신 모두 'REDO: T_3'이다.

④ X

즉시 갱신의 'REDO: T_2'가 틀렸다. T_2는 검사점 전에 완료되어 검사점 이후 복구 대상에서 제외되고, T_3가 검사점 후 완료되었으므로 REDO 대상이 된다. 지연 갱신의 'REDO: T_3'은 옳다. 올바른 짝은 '즉시 갱신: REDO: T_3, 지연 갱신: REDO: T_3'이다.

오답 포인트

검사점 전에 완료된 T_2와 검사점 후 고장 전에 완료된 T_3를 혼동하는 것이 함정이다. REDO 여부는 단순히 완료된 트랜잭션인지가 아니라 검사점 이후에 완료되었는지를 기준으로 판단한다.

다시 나오면 이렇게 판단

먼저 검사점 후 고장 전까지 완료된 트랜잭션을 찾아 REDO 대상으로 잡는다. 그다음 고장 시 미완료 트랜잭션은 즉시 갱신에서만 UNDO 대상으로 판단한다.

암기 포인트

검사점 후 완료는 REDO, 고장 시 미완료는 즉시 갱신에서 UNDO이다.

NoteBase

문 16. 트랜잭션의 동시성에 관한 설명으로 가장 적절하지 않은 것은?

① 2단계 락킹(two phase locking)을 사용하면 데드락(deadlock) 현상은 발생하지 않는다.

② 데이터에 공유락(shared lock)이 걸려 있다면 다른 트랜잭션의 공유락 요청을 허용해도 된다.

③ 두 개 이상의 트랜잭션이 각각 자신의 데이터에 대하여 락을 획득하고 상대방 데이터에 대하여 락을 요청하면 데드락이 발생한다.

④ 락(lock)을 사용하면 갱신손실(lost update) 문제를 해결할 수 있다.

정답 1번

출제 포인트

이 문제는 2단계 락킹의 직렬 가능성 보장과 데드락 방지 여부, 공유락의 양립성, 락을 이용한 갱신손실 방지를 구분하는 문제이다.

핵심 개념

2단계 락킹은 트랜잭션이 락을 획득하는 확장 단계와 락을 해제하는 축소 단계를 구분하여 충돌 직렬 가능성을 보장하는 규약이다. 그러나 여러 트랜잭션이 서로 필요한 락을 보유한 채 기다리는 순환 대기는 별개의 문제이므로 데드락이 생길 수 있다. 공유락은 읽기를 위한 락으로 다른 공유락과 양립하지만, 배타락과는 양립하지 않는다. 배타락은 갱신 시 다른 트랜잭션의 접근을 제한하여 갱신손실을 막는 데 사용된다.

선지별 해설

① X — 틀린 표현은 '2단계 락킹을 사용하면 데드락 현상은 발생하지 않는다'이다. 2단계 락킹은 충돌 직렬 가능성을 보장하지만 데드락까지 방지하지는 않는다. 올바르게는 '2단계 락킹을 사용해도 트랜잭션 사이에 순환 대기가 형성되면 데드락이 발생할 수 있다'이다.
② O — 공유락은 데이터를 읽기 위한 락이며 공유락끼리는 양립하므로, 한 트랜잭션이 공유락을 보유하고 있어도 다른 트랜잭션의 공유락 요청을 허용할 수 있다.
③ O — 두 트랜잭션이 서로 상대방이 보유한 데이터에 대해 양립하지 않는 락을 요청하고 기다리면 순환 대기가 형성되어 데드락이 발생한다.
④ O — 갱신 대상에 배타락을 적절히 설정하면 다른 트랜잭션의 동시 갱신을 막을 수 있으므로, 한 트랜잭션의 갱신 결과가 다른 트랜잭션의 갱신으로 사라지는 갱신손실을 방지할 수 있다.

오답 포인트

①은 2단계 락킹이 직렬 가능성을 보장한다는 사실을 데드락 방지까지 보장하는 것으로 바꾼 선지이다. ③에서는 단순히 여러 락이 존재한다는 점이 아니라, 각 트랜잭션이 상대방의 락 해제를 서로 기다리는 순환 대기가 핵심이다.

다시 나오면 이렇게 판단

락킹 규약의 효과를 판단할 때는 '일관된 실행 순서 보장'과 '대기 문제 방지'를 분리한다. 2단계 락킹은 충돌 직렬 가능성을 보장하지만 데드락은 허용할 수 있으며, 공유락끼리는 양립하고 갱신에는 배타락이 필요하다.

암기 포인트

2단계 락킹은 직렬 가능성을 보장하지만 데드락을 보장하지 않는다. 공유락과 공유락은 양립하고, 갱신은 배타락으로 보호한다.

NoteBase

문 17. SQL 뷰(view)의 특성에 관한 설명으로 가장 적절하지 않은 것은?

① 뷰가 정의된 기본 테이블(base table)이 제거되면 뷰도 자동적으로 제거된다.

② 시스템 카탈로그에 저장된다.

③ 뷰는 하나 이상의 기본 테이블에서만 유도되는 가상적 테이블이다.

④ INSERT, UPDATE, DELETE 등의 DML 작업은 경우에 따라 수행되지 않는다.

정답 3번

출제 포인트

SQL 뷰의 정의, 저장 방식, 기본 테이블과의 종속성, 갱신 가능성을 구분하는 문제이다.

핵심 개념

뷰는 하나 이상의 기본 테이블이나 다른 뷰에서 유도되는 가상 테이블이다. 일반적으로 실제 행 자체가 아니라 뷰를 정의한 질의문이 시스템 카탈로그에 저장된다.

선지별 해설

① O — 뷰는 기본 테이블에 의존하므로 그 기본 테이블이 제거되면 해당 뷰도 함께 제거된다.
② O — 뷰의 정의 정보는 시스템 카탈로그에 저장된다. 일반 뷰는 조회 결과의 행을 별도로 저장하는 테이블이 아니다.
③ X — 틀린 표현은 '기본 테이블에서만'이다. 뷰는 하나 이상의 기본 테이블뿐 아니라 다른 뷰에서도 유도될 수 있는 가상 테이블이다.
④ O — 모든 뷰가 갱신 가능한 것은 아니다. 조인, 집계, GROUP BY 등으로 정의된 뷰는 원본 행을 하나로 특정하기 어려워 INSERT, UPDATE, DELETE가 제한될 수 있다.

오답 포인트

③은 '가상적 테이블'이라는 뒷부분은 맞지만, 유도 대상을 '기본 테이블에서만'으로 한정한 것이 함정이다. 뷰를 바탕으로 다시 뷰를 정의할 수도 있다.

다시 나오면 이렇게 판단

뷰 관련 선지는 정의가 저장되는지, 실제 데이터가 저장되는지, 다른 뷰에서도 유도될 수 있는지, 원본 행과 일대일 대응되어 갱신 가능한지를 차례로 확인한다.

암기 포인트

뷰는 질의 정의를 저장한 가상 테이블이며, 기본 테이블뿐 아니라 다른 뷰에서도 만들 수 있고 항상 갱신 가능한 것은 아니다.

NoteBase

문 18. 릴레이션 R(A, B, C, D)는 다음과 같은 함수적 종속성(functional dependency)이 성립한다. ㉠과 ㉡의 물음에 옳은 것을 모두 고른 것은?

릴레이션 R의 함수적 종속성

$A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D$

물음

㉠ 릴레이션 R은 몇 정규형인가?
㉡ 릴레이션을 다음과 같이 분해했을 때 무손실 분해를 만족하는가?
R1(A, B, C), R2(C, D)

① ㉠: 제3정규형, ㉡: 무손실 분해를 만족하지 않음

② ㉠: 제2정규형, ㉡: 무손실 분해를 만족함

③ ㉠: 제2정규형, ㉡: 무손실 분해를 만족하지 않음

④ ㉠: 제3정규형, ㉡: 무손실 분해를 만족함

정답 2번

출제 포인트

함수적 종속성으로 후보키와 정규형을 판정하고, 이항 분해의 공통 속성이 한쪽 릴레이션을 함수적으로 결정하는지로 무손실 분해 여부를 판단하는 문제이다.

핵심 개념

후보키와 정규형: $A^+ = \{A, B, C, D\}$이므로 A가 후보키이다. 후보키가 단일 속성이므로 후보키의 일부에만 종속하는 부분 함수 종속은 생길 수 없어 R은 제2정규형을 만족한다. 그러나 B→C와 C→D에서 B와 C는 슈퍼키가 아니고 C와 D는 비주요 속성이므로 제3정규형은 만족하지 않는다.
무손실 분해: R1(A, B, C)과 R2(C, D)의 공통 속성은 C이다. C→D가 성립하므로 C→R2(C, D)가 성립하고, 두 릴레이션을 자연 조인하면 원래 R을 복원할 수 있다. 따라서 이 분해는 무손실 분해이다.

NoteBase

선지별 해설

① X — ‘제3정규형’과 ‘무손실 분해를 만족하지 않음’이 모두 틀린 표현이다. 올바르게는 R은 제2정규형이고, 분해는 무손실 분해를 만족한다.
② O — R은 제2정규형이며, 공통 속성 C가 R2(C, D)를 함수적으로 결정하므로 분해도 무손실이다.
③ X — ‘무손실 분해를 만족하지 않음’이 틀린 표현이다. 올바르게는 C→D에 의해 C→R2가 성립하므로 무손실 분해를 만족한다.
④ X — ‘제3정규형’이 틀린 표현이다. 올바르게는 B→C와 C→D가 제3정규형 조건을 위반하므로 R은 제2정규형이다. ‘무손실 분해를 만족함’은 옳다.

오답 포인트

A→B→C→D의 연쇄만 보고 모든 속성이 A에 종속하므로 제3정규형이라고 판단하기 쉽다. 제3정규형에서는 B→C와 C→D처럼 결정자가 슈퍼키가 아닌 종속성도 따로 검사해야 한다. 또한 분해 후 각 릴레이션의 속성만 보는 것이 아니라, 교집합 $R1 \cap R2 = \{C\}$가 어느 한쪽 릴레이션을 함수적으로 결정하는지를 확인해야 한다.

다시 나오면 이렇게 판단

먼저 속성 폐쇄로 후보키를 찾고, 단일 속성 키이면 부분 함수 종속이 없으므로 제2정규형 여부를 빠르게 확정한다. 다음으로 각 함수적 종속성에서 결정자가 슈퍼키인지와 종속 속성이 주요 속성인지를 검사해 제3정규형을 판정한다. 이항 분해는 공통 속성이 분해된 릴레이션 중 하나의 전체 속성을 함수적으로 결정하면 무손실이다.

암기 포인트

단일 후보키이면 부분 종속은 없고, 이항 분해에서는 ‘교집합이 한쪽 전체를 결정하면 무손실’이다.

NoteBase

문 19. 다음 데이터베이스 정보에서 “직원이 5명 이상인 부서의 부서명과 그 부서 소속 직원 중 급여가 40,000,000원 이상인 직원의 수”를 검색하는 SQL 질의문 일부인데, SQL 문에서 ㉠과 ㉡에 들어갈 문장 표현으로 가장 적절한 것은?

데이터베이스 정보	
테이블	직원(사번(PK), 이름, 근무부서번호(FK), 급여) 부서(부서번호(PK), 부서명, 사무실)
설명	근무부서번호 : 직원이 근무하는 부서의 부서번호

SQL 질의문

```
SELECT B.부서명, COUNT(*)
      FROM 직원 as A, 부서 as B
      WHERE _____㉠_____
      AND A.급여 >= 40000000
      AND A.근무부서번호 IN (SELECT C.근무부서번호
                             FROM 직원 as C
                             GROUP BY C.근무부서번호
                             HAVING COUNT(*) > 4)
      _____㉡_____;
```

① ㉠: B.부서번호=A.근무부서번호, ㉡: GROUP BY B.부서번호

② ㉠: B.부서번호=A.근무부서번호, ㉡: ORDER BY B.부서번호

③ ㉠: B.부서번호=C.근무부서번호, ㉡: GROUP BY B.부서번호

④ ㉠: B.부서번호=C.근무부서번호, ㉡: ORDER BY B.부서번호

정답 1번

출제 포인트

외부 질의에서 직원과 부서를 연결하는 조인 조건과, 부서별 조건 충족 직원 수를 계산하는 그룹화 절을 구별하는 문제이다.

핵심 개념

조인 조건: 직원 A의 근무부서번호(FK)와 부서 B의 부서번호(PK)를 같게 하여 각 직원을 소속 부서와 연결한다. 따라서 ㉠은 B.부서번호=A.근무부서번호이다.

GROUP BY: 같은 부서의 행을 하나의 그룹으로 묶어 COUNT(*)를 부서별로 계산한다. 따라서 ㉡은 GROUP BY B.부서번호이다.

ORDER BY: 조회 결과의 표시 순서만 정하며, COUNT(*)의 집계 단위를 만들지 않는다.

NoteBase

선지별 해설

① O — B.부서번호=A.근무부서번호로 직원 A와 부서 B를 올바르게 조인하고, GROUP BY B.부서번호로 급여 조건을 충족한 직원을 부서별로 집계한다. 내부 질의는 GROUP BY C.근무부서번호와 HAVING COUNT(*) > 4를 통해 전체 직원이 5명 이상인 부서만 먼저 골라낸다.
② X — 틀린 표현은 ㉡의 ORDER BY B.부서번호이다. ORDER BY는 결과를 정렬할 뿐 부서별 COUNT(*)를 만들지 못한다. 올바르게는 GROUP BY B.부서번호이다.
③ X — 틀린 표현은 ㉠의 B.부서번호=C.근무부서번호이다. C는 괄호 안의 하위 질의에서만 유효한 별칭이므로 외부 WHERE 절에서 참조할 수 없다. 올바르게는 외부 질의의 직원 별칭 A를 사용한 B.부서번호=A.근무부서번호이다.
④ X — ㉠에서 하위 질의 전용 별칭 C를 외부에서 참조하고, ㉡에서 그룹화 대신 정렬을 사용한 두 부분이 모두 틀렸다. 올바르게는 ㉠을 B.부서번호=A.근무부서번호로, ㉡을 GROUP BY B.부서번호로 고친다.

오답 포인트

안쪽 질의의 직원 별칭 C와 바깥 질의의 직원 별칭 A를 혼동하기 쉽다. 또한 '부서번호 순서로 나열한다'는 ORDER BY와 '부서번호별로 COUNT(*)를 계산한다'는 GROUP BY를 바꾸어 생각하면 ②나 ④를 고르게 된다.

다시 나오면 이렇게 판단

먼저 각 별칭의 유효 범위를 확인한다. 외부 질의의 직원과 부서는 FK=PK인 A.근무부서번호=B.부서번호로 연결한다. SELECT에 일반 열과 COUNT(*)가 함께 있고 부서별 개수를 요구하면 정렬이 아니라 GROUP BY가 필요하다고 판단한다.

암기 포인트

A는 외부 직원, C는 하위 질의 직원이다. FK=PK로 조인하고, 부서별 COUNT는 GROUP BY로 만든다.

NoteBase

문 20. 다음 데이터베이스는 학생, 과목, 수강정보 테이블과 참조무결성 제약조건을 갖고 있다. 정의한 참조무결성 제약조건의 영향을 받지 않는 연산들을 모두 고른 것은?

데이터베이스 정보	
테이블	학생(학번(PK), 성명, 학과) 과목(과목번호(PK), 과목명) 수강정보(수강일련번호(PK), 학번(FK), 과목번호(FK), 수강학기)
설명	수강정보의 학번(FK) : 학생의 학번(PK) 참조
연산	
㉠ 학생 테이블에 새로운 행을 삽입하는 삽입연산 ㉡ 학생 테이블에 기존 행의 기본키값을 변경하는 변경연산 ㉢ 수강정보 테이블에서 새로운 행을 삽입하는 삽입연산 ㉣ 수강정보 테이블에서 기존 행을 삭제하는 삭제연산	

① ㉠㉢

② ㉡㉢

③ ㉠㉣

④ ㉡㉣

정답 3번

출제 포인트

참조무결성 제약조건에서 참조되는 부모 키와 이를 참조하는 외래 키에 대한 삽입·변경·삭제 연산이 제약조건의 검사를 받는지 판단하는 문제이다.

사례 정리

학생 테이블은 학번을 기본 키로 가지는 부모 테이블이다. 수강정보 테이블은 학번을 외래 키로 가지는 자식 테이블이며, 이 값은 학생 테이블에 존재하는 학번을 참조해야 한다. 따라서 부모 행의 삽입과 자식 행의 삭제는 참조무결성을 깨뜨리지 않지만, 참조 중인 부모 키의 변경과 자식 행의 삽입은 제약조건의 영향을 받는다.

NoteBase

핵심 개념

참조무결성은 자식 테이블의 외래 키 값이 부모 테이블의 기본 키 값과 일치하거나, 허용된 경우에는 NULL이어야 한다는 제약조건이다. ㄱ은 부모 행 삽입이므로 영향을 받지 않고, ㄴ은 참조되는 부모 키 변경이므로 영향을 받는다. ㄷ은 자식 행 삽입이므로 외래 키 검사를 받아 영향을 받고, ㄹ은 자식 행 삭제이므로 영향을 받지 않는다.

적용 기준

부모 테이블에서는 새 행의 삽입보다 참조 중인 키의 변경·삭제를 주의하고, 자식 테이블에서는 행의 삭제보다 외래 키가 생기거나 바뀌는 삽입·변경을 주의한다. 이 문제에서 제약조건의 영향을 받지 않는 연산은 ㄱ과 ㄹ이다.

선지별 해설

① X — ㄱ은 영향을 받지 않지만, ㄷ은 수강정보에 새 외래 키 값이 생기므로 그 학번이 학생 테이블에 존재하는지 검사받는다. ‘ㄱㄷ’이 아니라 올바른 조합은 ‘ㄱㄹ’이다.
② X — ㄴ은 참조 중인 학생의 학번을 바꾸면 기존 수강정보의 외래 키가 부모 키를 참조하지 못할 수 있고, ㄷ도 새 외래 키의 유효성을 검사받는다. ‘ㄴㄷ’이 아니라 올바른 조합은 ‘ㄱㄹ’이다.
③ O — ㄱ의 부모 행 삽입은 기존 외래 키 참조를 깨뜨리지 않고, ㄹ의 자식 행 삭제는 외래 키 참조 자체를 제거하므로 둘 다 참조무결성 제약조건의 영향을 받지 않는다.
④ X — ㄹ은 영향을 받지 않지만, ㄴ은 참조되는 부모 키를 변경하는 연산이므로 제약조건의 영향을 받는다. ‘ㄴㄹ’이 아니라 올바른 조합은 ‘ㄱㄹ’이다.

오답 포인트

‘기본 키 연산은 항상 안전하다’거나 ‘외래 키가 있는 테이블의 모든 연산이 제약조건의 영향을 받는다’고 판단하면 틀리기 쉽다. 핵심은 테이블의 종류가 아니라 연산 뒤에 자식의 외래 키가 존재하지 않는 부모 키를 가리키게 되는지 여부이다.

다시 나오면 이렇게 판단

먼저 부모 테이블과 자식 테이블을 구분한 다음, 연산 후 고아 외래 키가 생기는지를 확인한다. 부모 행 삽입과 자식 행 삭제는 일반적으로 안전하고, 자식 행 삽입·외래 키 변경과 참조 중인 부모 키 변경·삭제는 제약조건의 검사를 받는다.

암기 포인트

부모는 넣어도 되고, 자식은 지워도 된다. 자식의 참조를 만들거나 부모의 참조 대상을 없애는 연산은 검사받는다.

NoteBase