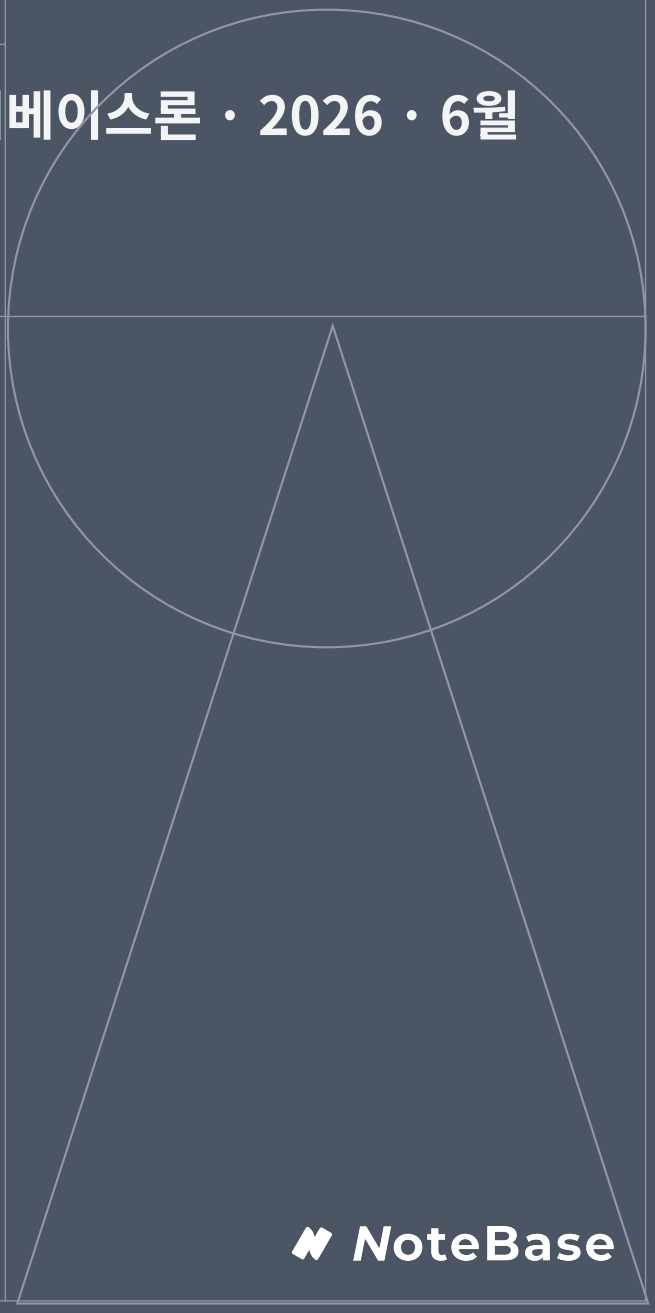
기출OX 해설집

공무원/서울시/9급 · 데이터베이스론 · 2026 · 6월



NoteBase

기출OX 해설집

공무원/서울시/9급 · 데이터베이스론 · 2026 · 6월

공무원/서울시/9급 · 데이터베이스론 · 2026 · 6월

문 1. SQL을 실행할 때 디스크 접근 횟수를 줄여 검색 속도를 높이기 위해 사용되고, 필요한 데이터를 빠르게 검색하기 위해 사용되는 것으로 가장 옳은 것은?

① 인덱스(index)

② 트랜잭션(transaction)

③ 뷰(view)

④ 트리거(trigger)

정답 1번

간단한 해설

인덱스는 필요한 데이터로 빠르게 접근할 수 있는 **검색 경로**를 제공한다. 검색 조건에 맞는 인덱스를 이용하면 테이블 전체를 읽는 것보다 디스크 접근 횟수를 줄일 수 있다.

개념 구분

인덱스(index): 검색에 쓰이는 열의 값을 바탕으로 필요한 데이터를 빠르게 찾도록 만든 자료구조이다. 검색 조건에 맞는 인덱스를 이용하면 **테이블 전체를 순차적으로 읽는 작업을 줄일 수 있다**. 예를 들어 학생 테이블에서 특정 학번을 찾을 때, 학번 인덱스를 이용하면 모든 학생 행을 차례로 확인하는 대신 해당 데이터로 접근할 수 있다.

트랜잭션(transaction): 여러 데이터베이스 연산을 **하나의 논리적 작업 단위**로 묶은 것이다. 작업 전체의 성공이나 취소를 관리하는 단위로, 검색을 위한 접근 경로와 구별된다.

뷰(view): 질의로 정의한 **가상 테이블**이다. 사용자에게 필요한 데이터의 형태를 제공하며, 일반적인 뷰 자체가 검색용 인덱스를 만드는 것은 아니다.

트리거(trigger): 데이터의 삽입·갱신·삭제 등 **지정한 사건이 발생할 때 자동으로 실행**되는 절차이다. 사건에 반응하여 작업을 수행하는 기능으로, 데이터의 검색 경로와 구별된다.

NoteBase

문 2. 트랜잭션(transaction)의 특성 중 지속성(durability)에 대한 설명으로 가장 옳은 것은?

① 트랜잭션 내의 모든 연산은 완전히 수행되거나 전혀 수행되지 않은 상태로 유지되어야 한다.

② 트랜잭션이 수행되는 동안에는 다른 트랜잭션이 해당 데이터에 접근해서는 안 된다.

③ 트랜잭션이 성공적으로 완료된 후에는 시스템 장애가 발생하더라도 그 결과가 영구적으로 반영되어야 한다.

④ 트랜잭션 수행 전후 데이터베이스는 일관성 있는 상태를 유지해야 한다.

정답 3번

간단한 해설

지속성은 **성공적으로 완료된 트랜잭션의 결과**가 이후 시스템 장애에도 영구적으로 보존되는 특성이다.

개념 구분

지속성(durability): 트랜잭션이 커밋되어 성공적으로 완료되면 **시스템 장애가 발생해도 그 결과가 보존**되어야 한다. 예를 들어 계좌 이체가 완료되었다면, 이후 시스템이 중단되었다가 복구되어도 이체 결과가 유지되어야 한다. ③의 '성공적으로 완료된 후'와 '영구적으로 반영'이 이 특성을 나타낸다.

원자성(atomicity): 트랜잭션의 연산은 **전부 반영되거나 전혀 반영되지 않아야** 한다. 계좌 이체에서 출금만 반영되고 입금은 누락되는 것처럼 일부 연산만 처리된 상태를 허용하지 않는다는 뜻이다. ①은 완료된 결과의 보존이 아니라 연산 전체의 처리 여부를 설명한다.

고립성(isolation): 동시에 실행되는 트랜잭션들이 서로의 미완료 작업에 부당하게 간섭하지 않도록 하는 특성이다. ②의 접근 제한은 **수행 중인 트랜잭션 사이의 간섭**에 관한 내용으로, 완료 후 결과를 보존하는 지속성과 구별된다.

일관성(consistency): 트랜잭션 수행 전후에 데이터베이스가 **정해진 무결성 제약조건을 만족하는 상태**를 유지해야 한다. ④는 데이터베이스 상태의 적합성을 설명하며, 장애 이후에도 완료된 결과가 남는지를 설명하는 것은 아니다.

NoteBase

문 3. 데이터 제어어(DCL, data control language)의 특성으로 가장 옳지 않은 것은?

① 무결성 유지

② 데이터베이스 관리자(DBA)가 주로 사용

③ 접근 제어 및 권한 부여

④ 스키마 정의

정답 4번

간단한 해설

④의 '스키마 정의'는 데이터베이스의 구조를 정하는 **데이터 정의어(DDL)**의 기능이다. 데이터 제어어(DCL)는 사용자에게 허용할 작업을 정하는 **접근 제어와 권한 관리**를 담당한다.

선지별 해설

① O: 무결성은 데이터가 정해진 규칙을 만족하여 정확성과 일관성을 유지하는 성질이다. DCL은 **허가되지 않은 데이터 변경을 제한**하는 등 데이터 사용을 통제하여 무결성 유지에 기여한다.

② O: 데이터베이스 관리자(DBA)는 데이터베이스의 보안과 사용자 권한을 관리한다. 따라서 사용자별로 허용할 작업을 정하는 DCL을 **DBA가 주로 사용**한다. 다만 필요한 권한을 가진 다른 사용자도 사용할 수 있으므로 DBA만 사용하는 언어라는 뜻은 아니다.

③ O: DCL은 사용자에게 데이터베이스 객체에 대한 권한을 부여하거나 회수하여 접근을 제어한다. 대표적으로 **GRANT는 권한 부여, REVOKE는 권한 회수**에 사용한다. 예를 들어 조회 권한만 부여하고 수정 권한은 부여하지 않으면 해당 사용자는 데이터를 조회할 수 있지만 수정할 수는 없다.

④ X: 스키마는 테이블과 속성, 제약조건 등 데이터베이스의 구조를 기술한 것이다. 이를 정의하거나 변경하는 것은 DCL이 아니라 **데이터 정의어(DDL)**의 기능이다. 테이블을 생성하는 CREATE TABLE, 구조를 변경하는 ALTER TABLE 등이 이에 해당한다. DCL은 이러한 구조 자체를 정의하기보다 그 객체에 대한 사용자 권한을 관리한다.

NoteBase

문 4. 릴레이션 R의 차수(degree)가 3이고 카디널리티(cardinality)가 5이며, 릴레이션 S의 차수는 5이고 카디널리티가 7이다. 이 두 릴레이션을 카티션 프로덕트(cartesian product)했을 때 생성된 릴레이션의 차수와 카디널리티로 옳은 것은?

① 8, 12

② 8, 35

③ 15, 12

④ 15, 35

정답 2번

간단한 해설

카티션 프로덕트에서는 **차수는 두 차수의 합**, **카디널리티는 두 카디널리티의 곱**이 된다. 따라서 결과 릴레이션은 속성 8개와 튜플 35개를 갖는다.

계산 과정

차수 계산: 차수는 릴레이션의 속성 수이다. 카티션 프로덕트로 만든 각 튜플은 R의 속성 3개와 S의 속성 5개를 모두 포함하므로, 결과 릴레이션의 차수는 3 + 5 = 8이다. 두 릴레이션의 **속성을 모두 이어 붙이므로 더한다**.

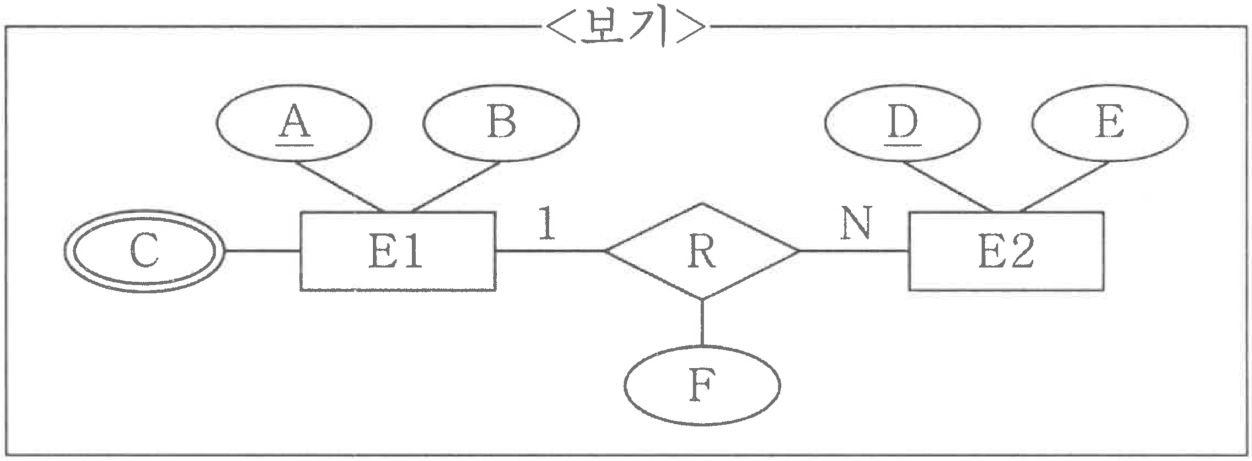
카디널리티 계산: 카디널리티는 릴레이션의 튜플 수이다. 카티션 프로덕트는 **R의 각 튜플을 S의 모든 튜플과 짝짓는다**. R의 튜플 하나마다 S의 튜플 7개와 결합하여 결과 튜플 7개가 생기고, R에는 튜플이 5개 있으므로 전체 튜플 수는 5 × 7 = 35이다.

결과 릴레이션의 차수와 카디널리티는 각각 8과 35이며, 이는 ②의 '8, 35'에 해당한다.

NoteBase

문 5. <보기>의 E-R 다이어그램(Entity-Relationship diagram)을 관계 데이터베이스 스키마로 변환했을 때, 가장 옳은 것은? (단, 밑줄 친 속성은 기본키(primary key)이다.)

<보기>



① E1(A, B, C)
E2(D, E, A, F)

② E1(A, B, C, D, F)
E2(D, E)

③ E1(A, B, D, F)
E2(D, E)
E3(A, C)

④ E1(A, B)
E2(D, E, A, F)
E3(A, C)

정답 4번

간단한 해설

이중 타원으로 표시된 C는 **다중값 속성**이므로 별도 릴레이션으로 분리한다. E1과 E2가 1:N 관계이므로 E1의 기본키 A와 관계 속성 F는 **N측인 E2**에 둔다.

NoteBase

구성 요소별 해설

E1: A는 밑줄 친 기본키이고 B는 일반 속성이다. C는 이중 타원으로 표시된 다중값 속성으로, 한 E1 개체가 여러 C 값을 가질 수 있다. 따라서 C를 별도로 분리하고 E1(A, B)를 구성하며, 기본키는 A로 유지한다.

E2와 관계 R: E1 쪽의 1과 E2 쪽의 N은 하나의 E1 개체에 여러 E2 개체가 연결될 수 있음을 뜻한다. 각 E2 개체에 연결된 E1을 나타내려면 **N측 릴레이션에 1측의 기본키**를 외래키로 둔다. 따라서 E2에 A를 추가하여 E1의 A를 참조하게 한다. 관계 속성 F도 같은 E2 행에 저장하면 해당 E2와 E1 사이의 관계를 표현할 수 있으므로, 별도의 R 릴레이션 없이 E2(D, E, A, F)를 구성한다. E2 개체는 D만으로 식별되므로 기본키는 D이며, A나 F를 기본키에 추가하지 않는다.

E3: 다중값 속성 C를 저장할 릴레이션은 E1의 기본키 A와 C를 묶어 E3(A, C)로 구성한다. A는 E1을 참조하는 외래키이다. 같은 A에 여러 C 값이 대응할 수 있고, 서로 다른 E1 개체가 같은 C 값을 가질 수도 있으므로 **A와 C를 함께 복합 기본키**로 삼는다. 예를 들어 한 개체의 C 값이 두 개이면 같은 A를 가진 두 행에 C 값을 하나씩 저장한다.

따라서 E1(A, B), E2(D, E, A, F), E3(A, C)를 구성하고 기본키를 각각 A, D, (A, C)로 지정한 ④와 일치한다.

NoteBase

문 6. 관계 대수의 연산에 대한 설명으로 가장 옳지 않은 것은?

① 셀렉트(select) 연산은 릴레이션에서 주어진 조건을 만족하는 투플(tuple)을 추출한다.

② 프로젝트(project) 연산은 릴레이션에서 원하는 속성(attribute)만을 추출하며, 결과에서 중복 투플은 제거된다.

③ 자연 조인(natural join)은 두 릴레이션에서 같은 이름을 가진 속성에 대해 동등 조인을 수행하며, 결과에서 조인 속성은 한 번만 나타난다.

④ 디비전(division) 연산 R÷S의 결과는 S에 있는 모든 값에 대응되지 않는 R의 투플들을 반환한다.

정답 4번

간단한 해설

④의 '모든 값에 대응되지 않는'이 잘못되었다. 디비전은 S의 **모든 값과 빠짐없이 대응하는** R의 나머지 속성값을 구하는 연산이다.

선지별 해설

① O: 셀렉트는 조건을 만족하는 **행인 투플을 선택**한다. 예를 들어 학생 릴레이션에서 학년이 2인 투플만 추출하면, 해당 학생들의 행은 남고 각 행의 속성 구성은 그대로 유지된다.

② O: 프로젝트는 지정한 **열인 속성을 선택**한다. 일부 속성을 제외하면 원래 서로 달랐던 투플들이 같은 값을 가질 수 있는데, 관계 대수의 릴레이션은 집합이므로 결과에서 중복 투플을 제거한다. 예를 들어 학생들의 학과만 추출했을 때 같은 학과가 여러 번 나타나더라도 결과에는 한 번만 남는다.

③ O: 자연 조인은 두 릴레이션에서 **이름이 같은 모든 속성의 값이 일치**하는 투플들을 결합한다. 결합된 두 투플의 공통 속성값은 같으므로, 결과에서는 공통 조인 속성을 중복하여 두지 않고 각각 한 번만 나타낸다.

④ X: 디비전은 '대응되지 않는' 투플을 찾는 연산이 아니라, S의 **모든 투플과 대응하는** R의 나머지 속성값을 구하는 연산이다. S의 속성들이 R에 포함되어 있을 때, R에서 S의 속성을 제외한 속성값을 후보로 삼고 각 후보가 S의 모든 투플과 결합된 형태로 R에 존재하는지 확인한다. 예를 들어 R이 학생·과목으로 이루어진 수강 릴레이션이고 S에 과목 A와 B가 있다면, 두 과목을 모두 수강한 학생만 결과에 남는다. A만 수강한 학생은 제외되며, 결과에는 과목을 제외한 학생 속성만 나타난다.

NoteBase

문 7. 릴레이션 $R(A, B, C, D, E, F)$에서 함수적 종속성 (functional dependency)이 <보기>와 같이 주어졌을 때, 릴레이션 R의 후보키(candidate key)로 가장 옳은 것은?

<보기>

$A \rightarrow B,\ C \rightarrow D,\ \{B, C\} \rightarrow E,\ E \rightarrow F$

① {A, C}

② {A, D}

③ {A, B, C}

④ {A, C, E}

정답 1번

간단한 해설

$\{A, C\}$에서 함수적 종속성을 적용하면 B, D, E, F를 차례로 얻어 **모든 속성을 결정**할 수 있다. A나 C 중 어느 하나를 빼면 모든 속성을 결정하지 못하므로 **최소성**도 만족한다.

NoteBase

계산 과정

후보키는 릴레이션의 모든 속성을 결정하는 유일성과, 그 집합에서 **어떤 속성도 더 뺄 수 없는 최소성**을 만족하는 속성 집합이다. 속성 집합 X의 폐포 X^+는 주어진 함수적 종속성으로 X에서 결정할 수 있는 모든 속성의 집합이다. 폐포가 R의 전체 속성 집합이면 X는 슈퍼키이며, 최소성까지 만족하면 후보키이다.

필수 속성 확인: A와 C는 주어진 함수적 종속성의 **오른쪽에 한 번도 나타나지 않는다**. 따라서 다른 속성에서 새로 얻을 수 없으므로, 모든 속성을 결정하려는 집합에는 A와 C가 모두 포함되어야 한다.

속성폐포 계산: $\{A, C\}$에서 시작한다. $A \to B$로 B를, $C \to D$로 D를 추가하면 $\{A, B, C, D\}$가 된다. 이제 **B와 C가 모두 있으므로** $\{B, C\} \to E$를 적용하여 E를 얻고, $E \to F$로 F를 얻는다. 따라서 $\{A, C\}^+ = \{A, B, C, D, E, F\}$이며, 이는 R의 모든 속성을 결정한다는 뜻이다.

최소성 확인: C를 빼면 $\{A\}^+ = \{A, B\}$이고, A를 빼면 $\{C\}^+ = \{C, D\}$이다. 어느 쪽도 전체 속성을 얻지 못하므로 A와 C는 모두 필요하다. 따라서 $\{A, C\}$는 유일성과 최소성을 함께 만족한다.

다른 집합과 구별: ②의 $\{A, D\}^+ = \{A, B, D\}$이므로 C, E, F를 결정하지 못한다. ③의 $\{A, B, C\}$와 ④의 $\{A, C, E\}$는 모든 속성을 결정하지만, 각각 B와 E를 빼도 $\{A, C\}$가 남아 모든 속성을 결정한다. 따라서 두 집합은 **최소성을 만족하지 않는 슈퍼키**이다.

따라서 후보키는 $\{A, C\}$이며, ①의 속성 집합에 해당한다.

NoteBase

문 8. <보기>의 정규화 과정을 순서대로 바르게 나열한 것은?

<보기>
ㄱ. 후보키(candidate key)가 아닌 결정자 제거
ㄴ. 부분 함수 종속성을 제거
ㄷ. 속성을 원자값만 갖도록 분해
ㄹ. 이행적 함수 종속성을 제거

① ㄱ → ㄴ → ㄷ → ㄹ

② ㄱ → ㄷ → ㄴ → ㄹ

③ ㄷ → ㄱ → ㄴ → ㄹ

④ ㄷ → ㄴ → ㄹ → ㄱ

정답 4번

간단한 해설

제3정규형에서도 릴레이션 전체를 식별하지 못하는 결정자에 따른 종속 관계가 남을 수 있으며, 이를 추가로 분해하는 단계가 **BCNF**이다. 따라서 **이행적 함수 종속성 제거**는 후보키가 아닌 결정자를 제거하는 단계보다 앞선다.

NoteBase

올바른 순서

ㄷ → ㄴ → ㄹ → ㄱ

ㄷ. 속성을 원자값만 갖도록 분해: 제1정규형을 만드는 단계이다. 각 칸에 여러 값을 묶어 넣은 반복 집단 등을 정리하여 **각 속성이 하나의 원자값**을 갖도록 한다. 이후의 함수 종속성에 따른 정규화는 이 제1정규형을 전제로 한다.

ㄴ. 부분 함수 종속성을 제거: 제1정규형에서 제2정규형으로 진행하는 단계이다. 어느 후보키에도 속하지 않는 속성이 **복합 후보키의 일부에만 종속**되는 관계를 분리한다. 예를 들어 수강 릴레이션의 후보키가 '학번과 과목번호'인데 학생명은 학번만으로 결정된다면, 학생 정보를 별도 릴레이션으로 분해한다.

ㄹ. 이행적 함수 종속성을 제거: 제2정규형에서 제3정규형으로 진행하는 단계이다. 후보키에 속하지 않는 속성이 다른 비키 속성을 거쳐 후보키에 종속되는 관계를 분리한다. 예를 들어 학번이 후보키인 학생 릴레이션에서 학과번호가 학과명을 결정한다면, 학과명은 학번에 **학과번호를 거쳐 간접적으로 종속**된다. 학과번호와 학과명을 별도 릴레이션으로 분해하면 이러한 종속 관계를 해소할 수 있다.

ㄱ. 후보키가 아닌 결정자 제거: 제3정규형에서 BCNF로 진행하는 단계이다. 결정자는 다른 속성의 값을 결정하는 속성 또는 속성 집합이다. BCNF의 정확한 조건은 자신에게 포함되지 않은 속성을 결정하는 모든 결정자가 **슈퍼키**, 즉 튜플 전체를 식별할 수 있는 속성 집합이어야 한다는 것이다. 보기의 표현은 이 조건을 만족하도록 릴레이션을 분해하는 단계를 가리키며, 제3정규형보다 더 엄격한 조건을 적용한다.

NoteBase

문 9. 관계 데이터 모델에서 외래키(foreign key)에 대한 설명으로 가장 옳지 않은 것은?

① 하나의 릴레이션에는 여러 개의 외래키가 존재할 수 있다.

② 외래키가 되는 속성(attribute)과 그 외래키의 참조 대상이 되는 릴레이션 내 기본키(primary key) 속성의 도메인은 동일해야 한다.

③ 외래키는 다른 릴레이션을 참조하기 때문에 널(null) 값을 가질 수 없다.

④ 외래키는 자신이 속한 릴레이션의 기본키를 참조하는 것도 가능하다.

정답 3번

간단한 해설

외래키는 별도의 NOT NULL 제약 등이 없으면 **널 값을 가질 수 있다**. 널 값을 금지하는 기본키의 개체 무결성과, 널이 아닌 외래키 값이 참조 대상에 존재해야 한다는 참조 무결성을 구별해야 한다.

선지별 해설

① O: 외래키는 릴레이션 사이의 참조 관계를 나타내므로, 한 릴레이션에 **여러 개의 외래키**를 둘 수 있다. 예를 들어 주문 릴레이션의 고객번호는 고객 릴레이션을, 상품번호는 상품 릴레이션을 참조하도록 각각 외래키로 지정할 수 있다.

② O: 도메인은 속성이 가질 수 있는 값의 범위이다. 외래키와 참조 대상 기본키의 대응 속성은 **동일한 도메인**을 가져야 하므로, 같은 종류와 범위의 값으로 참조 관계를 맺는다. 외래키가 여러 속성으로 구성되면 각 대응 속성끼리 이 조건을 만족해야 한다.

③ X: 외래키라는 이유만으로 널 값이 금지되지는 않는다. 참조 무결성은 외래키 값이 널이 아니면 참조 대상의 키 값과 일치하도록 요구한다. 따라서 '널 값을 가질 수 없다'는 **'별도의 NOT NULL 제약 등이 없으면 널 값을 가질 수 있다'**로 고쳐야 한다. 기본키는 개체 무결성에 따라 널 값을 가질 수 없으므로, 외래키가 기본키의 구성 속성이기도 하면 널 값이 허용되지 않는다.

④ O: 외래키는 다른 릴레이션뿐 아니라 **자신이 속한 릴레이션의 기본키**도 참조할 수 있다. 이를 자기 참조라고 한다. 예를 들어 사원 릴레이션에서 사원번호를 기본키로 두고, 상사번호를 같은 릴레이션의 사원번호를 참조하는 외래키로 둘 수 있다.

NoteBase

문 10. 한 번에 로크(lock) 되어야 할 데이터의 크기를 '로크 단위'라고 한다. 이 단위의 크기에 대한 설명으로 가장 옳지 않은 것은?

① 로크 단위가 작아지면 로크를 설정하는 오버헤드가 증가한다.

② 로크 단위가 커지면 트랜잭션(transaction)들의 동시 수행 가능성이 증가한다.

③ 릴레이션(테이블), 블록, 레코드 등이 로크 단위가 될 수 있다.

④ 로크 단위가 커지면 로크 단위가 작을 때보다 설정된 로크의 수가 적다.

정답 2번

간단한 해설

②의 '동시 수행 가능성이 증가한다'는 설명은 반대이다. 로크 단위가 커지면 한 로크가 더 넓은 데이터 범위를 묶어 다른 트랜잭션의 접근을 제한하므로, **동시 수행 가능성은 감소한다**.

선지별 해설

① O: 같은 데이터 범위를 처리할 때 로크 단위가 작으면 더 많은 로크를 설정해야 한다. 이에 따라 로크의 설정·해제와 관리에 드는 **오버헤드가 증가한다**.

② X: 로크 단위가 커지면 다른 트랜잭션이 접근하려는 데이터까지 한꺼번에 잠길 수 있어 **동시 수행 가능성이 감소한다**. 예를 들어 한 트랜잭션이 테이블 전체에 배타적 로크를 설정하면, 다른 트랜잭션은 그 테이블의 서로 다른 레코드를 변경하려 해도 기다려야 한다. 따라서 '증가한다'를 '감소한다'로 고쳐야 한다.

③ O: 로크는 **테이블·블록·레코드 등 여러 크기의 데이터 단위**에 설정할 수 있다. 테이블 단위 로크는 테이블 전체를, 블록 단위 로크는 해당 블록을, 레코드 단위 로크는 개별 레코드를 잠그는 방식이다.

④ O: 같은 데이터 범위를 잠근다면 로크 단위가 클수록 **필요한 로크 수가 줄어든다**. 예를 들어 테이블 전체를 잠글 때 테이블 단위 로크는 하나면 되지만, 레코드 단위 로크는 각 레코드에 따로 설정해야 한다.

NoteBase

문 11. <보기 1>의 테이블 R1과 R2에 대해 <보기 2>의 SQL문을 수행한 결과에서 투플(tuple)의 개수는? (단, 밑줄 친 속성은 기본키(primary key)이고 R1 테이블의 C는 R2 테이블을 참조하기 위한 외래키(foreign key)이다.)

<보기 1>

R1

A	B	C
A1	B1	1
A2	B2	2
A3	B3	3
A4	B4	1
A5	B5	2
A6	B6	3
A7	B7	2

R2

C	D	E
1	고급	E1
2	특급	E2
3	중급	E2
4	초급	E1

<보기 2>

```
SELECT A, C
FROM R1
WHERE C NOT IN (SELECT C
                FROM R2
                WHERE D = '고급' OR D = '중급' );
```

① 2

② 3

③ 4

④ 6

정답 2번

NoteBase

간단한 해설

부질의는 D가 '고급' 또는 '중급'인 행의 C 값인 1과 3을 반환한다. **NOT IN은 이 값들을 제외**하므로 R1에서 C가 2인 행만 남으며, 결과는 3개 투플이다.

계산 과정

1. 부질의 결과 구하기: R2에서 D = '고급' OR D = '중급'은 두 조건 중 **하나라도 만족하는 행**을 고른다. '고급' 행의 C는 1이고 '중급' 행의 C는 3이므로, SELECT C의 결과는 1과 3이다. '특급'과 '초급' 행은 포함되지 않는다.

2. NOT IN 조건 적용하기: 바깥 질의의 C는 R1의 C를 뜻한다. C NOT IN (1, 3)은 **C가 1도 아니고 3도 아닌 행**을 남긴다. R1의 7개 행 중 C가 1인 A1·A4와 C가 3인 A3·A6을 제외하면, C가 2인 A2·A5·A7의 3개 행이 남는다.

3. 최종 투플 수 세기: SELECT A, C는 남은 행에서 A와 C 두 속성의 값을 출력하므로 결과 투플은 (A2, 2), (A5, 2), (A7, 2)이다. C 값이 같더라도 A 값이 서로 달라 별개의 투플이다. 따라서 결과 투플 수는 7 - 4 = 3개이며, ②의 3에 해당한다.

NoteBase

문 12. 데이터베이스 질의 최적화에 대한 설명으로 가장 옳지 않은 것은?

① 주어진 질의를 처리할 수 있는 동등한 관계 대수들을 찾아, 가장 효율적인 질의 실행 계획을 선택하는 과정이다.

② 비용 기반 최적화는 다양한 실행 계획의 비용을 추정하여 가장 낮은 비용의 실행 계획을 선택하는 것이다.

③ 조인(join) 순서는 질의 처리 성능에 영향을 미치지 않으므로 어떤 순서로 조인을 수행해도 최적화에 무관하다.

④ 경험적(heuristic) 최적화의 예는 셀렉트(select) 연산을 가능한 이른 단계에 수행하여 중간 결과의 크기를 줄이는 것이다.

정답 3번

간단한 해설

조인 순서가 달라지면 **중간 결과의 크기와 처리 비용**이 달라질 수 있다. 따라서 조인 순서는 질의 처리 성능에 영향을 미치며, 질의 최적화에서 중요한 고려 대상이다.

선지별 해설

① O: 질의 최적화는 **동일한 결과를 산출하는** 관계 대수식과 실행 방법을 검토하여 효율적인 실행 계획을 선택하는 과정이다. 같은 결과를 얻더라도 연산 순서나 자료 접근 방법에 따라 처리 비용이 달라지므로 실행 계획을 비교한다.

② O: 비용 기반 최적화는 각 실행 계획에 필요한 디스크 입출력, CPU 연산 등의 비용을 추정하고, **검토한 후보 중 추정 비용이 가장 낮은 계획**을 선택한다. 테이블의 튜플 수나 값의 분포 같은 통계 정보가 비용 추정에 활용되며, 추정 비용이 실제 실행 시간과 항상 일치하는 것은 아니다.

③ X: 조인 순서는 **중간 결과의 크기와 후속 연산의 작업량**을 바꾸므로 질의 처리 성능에 영향을 미친다. 예를 들어 세 테이블을 내부 조인할 때, 첫 조인에서 매우 큰 중간 결과를 만드는 계획보다 작은 중간 결과를 만드는 계획이 후속 조인의 처리량을 줄일 수 있다. 최종 결과가 같아도 처리 비용은 달라질 수 있으므로, '어떤 순서로 조인을 수행해도 최적화에 무관하다'는 설명은 조인 순서를 고려하여 효율적인 실행 계획을 선택해야 한다는 내용으로 고쳐야 한다.

④ O: 경험적 최적화는 일반적으로 효율을 높이는 규칙을 적용하여 질의를 변환한다. 셀렉트는 조건을 만족하는 튜플만 남기는 연산이므로, **결과의 동등성을 유지하는 범위에서 일찍 수행**하면 중간 결과를 줄일 수 있다. 예를 들어 한 테이블의 속성만을 검사하는 조건을 내부 조인 전에 적용하면, 조인에 투입되는 튜플 수를 줄일 수 있다.

NoteBase

문 13. 데이터베이스 설계와 관련하여 가장 옳지 않은 것은?

① 요구 사항 분석 단계에서는 사용자의 요구 사항을 수집하고 분석하여, 사용자가 의도하는 데이터베이스의 용도를 파악해야 한다.

② 개념적 설계 단계에서는 요구 사항 분석 단계에서 파악한 사용자의 요구 사항을 지정된 DBMS에 적합한 개념적 구조로 표현한다.

③ 논리적 설계 단계에서는 개념적 설계 단계에서 만들어진 개념적 구조로부터 특정 목표 DBMS가 처리할 수 있는 스키마를 생성한다.

④ 물리적 설계 단계에서는 인덱스 구조나 내부 저장 구조, 접근 경로 등을 결정한다.

정답 2번

간단한 해설

②의 '지정된 DBMS에 적합한'이 잘못되었다. 개념적 설계는 **특정 DBMS와 독립적으로** 데이터와 그 관계를 표현하며, 목표 DBMS가 처리할 수 있는 스키마로 변환하는 것은 논리적 설계이다.

선지별 해설

① O: 요구 사항 분석은 사용자가 어떤 데이터를 필요로 하고, 그 데이터로 어떤 업무를 수행하려는지 파악하는 단계이다. 필요한 데이터와 조회·수정 등의 처리 요구를 수집하여 **데이터베이스의 용도와 요구 사항**을 명확히 한다.

② X: 개념적 설계는 요구 사항을 바탕으로 개체, 속성, 관계 등을 정리하여 E-R 다이어그램과 같은 개념적 구조로 표현하는 단계이다. 이 구조는 **특정 DBMS에 종속되지 않는다**. 따라서 '지정된 DBMS에 적합한 개념적 구조'가 아니라 'DBMS와 독립적인 개념적 구조'로 고쳐야 한다.

③ O: 논리적 설계는 개념적 구조를 **목표 DBMS가 처리할 수 있는 논리적 스키마**로 변환하는 단계이다. 예를 들어 관계형 DBMS를 사용한다면 개체와 관계를 릴레이션으로 변환하고 속성과 키 등을 정한다.

④ O: 물리적 설계는 논리적 스키마의 데이터를 실제 저장장치에 어떻게 저장하고 찾아갈지 정하는 단계이다. 저장 공간과 처리 성능 등을 고려하여 **인덱스, 내부 저장 구조, 접근 경로**를 결정한다.

NoteBase

문 14. <보기>는 데이터베이스 보안 및 권한 제어를 위해 관리자(DBA)와 사용자들이 순차적으로 수행한 SQL 명령의 흐름이다. 모든 명령이 수행된 직후, 최종적인 사용자들의 권한 상태에 대한 설명으로 가장 옳은 것은?

<보기>
1. 관리자가 사용자 A에게 테이블 T에 대한 권한을 부여함
GRANT SELECT ON T TO A WITH GRANT OPTION;
2. 사용자 A가 사용자 B에게 권한을 부여함
GRANT SELECT ON T TO B WITH GRANT OPTION;
3. 사용자 B가 사용자 C에게 권한을 부여함
GRANT SELECT ON T TO C;
4. 관리자가 사용자 A의 권한을 회수함
REVOKE SELECT ON T FROM A CASCADE;

① 사용자 A의 권한만 회수되고, 사용자 B와 C가 부여받은 권한은 그대로 유지된다.

② 사용자 A와 B의 권한만 회수되고, 사용자 C의 권한은 유지된다.

③ 관리자가 직접 권한을 부여한 대상은 사용자 A뿐이므로, 연쇄 회수(CASCADE) 구문은 문법 오류를 발생시킨다.

④ 사용자 A의 권한이 회수되면서, A로부터 파생된 사용자 B와 C의 SELECT 권한도 모두 연쇄적으로 회수된다.

정답 4번

간단한 해설

B와 C의 SELECT 권한은 모두 A에게서 이어진 권한 부여에 의존한다. A의 권한을 **CASCADE로 회수**하면 이 경로에서 파생된 B와 C의 권한도 연쇄적으로 회수된다.

NoteBase

계산 과정

SELECT는 테이블의 데이터를 조회하는 권한이며, WITH GRANT OPTION은 받은 권한을 다른 사용자에게 부여할 수 있게 하는 옵션이다. 보기의 명령으로 형성되는 권한 부여 경로는 관리자 → A → B → C이므로, 각 명령이 이 경로에 미치는 영향을 차례로 추적한다.

1. 관리자 → A: A는 테이블 T의 SELECT 권한과 **다른 사용자에게 부여할 수 있는 권한**을 함께 받는다. 따라서 다음 단계에서 A가 B에게 SELECT 권한을 부여할 수 있다.

2. A → B: B도 SELECT 권한을 WITH GRANT OPTION과 함께 받는다. 이때 A의 권한은 유지되고, B는 A에게 받은 권한을 근거로 C에게 권한을 부여할 수 있다.

3. B → C: C는 SELECT 권한을 받지만 **WITH GRANT OPTION은 받지 않는다**. 이 단계 직후 A와 B는 조회와 권한 부여가 가능하고, C는 조회만 가능하다. C의 권한은 B에게서, B의 권한은 A에게서 파생된다.

4. 관리자의 회수: REVOKE SELECT ON T FROM A CASCADE는 A의 SELECT 권한과 그 부여 권한을 회수한다. **CASCADE는 회수 대상에게 의존하는 권한까지 연쇄적으로 회수**하므로, A에게 받은 B의 권한과 다시 B에게 받은 C의 권한도 회수된다. 관리자가 B나 C에게 직접 권한을 부여하지 않았어도 적용되므로 문법 오류가 발생하는 것은 아니다.

최종적으로 보기의 경로로 부여된 A·B·C의 SELECT 권한은 모두 회수된다. 이는 A로부터 파생된 B와 C의 권한까지 연쇄적으로 회수된다는 ④의 설명과 일치한다.

NoteBase

문 15. 관계 대수식의 질의 최적화(query optimization) 과정에서 실행 시간을 단축하기 위해 동등 규칙(equivalence rule)을 적용하여 연산의 순서를 변경할 수 있다. 다음 중 관계 대수식의 동등 변환으로 가장 옳지 않은 것은?

① _c1 c2(R)≡_c1(_c2(R))

② $\pi_A(\pi_{A,B}(R)) \equiv \pi_A(R)$

③ $\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$ (단, 조건 c는 릴레이션 R의 속성에만 관련된 조건임)

④ $\pi_A(R \cup S) \equiv \pi_A(R) \bowtie \pi_A(S)$

정답 4번

간단한 해설

프로젝션은 합집합에 대해 분배되므로, 각 릴레이션에서 속성 A를 추출한 결과도 **자연 조인이 아니라 합집합**으로 결합해야 한다.

선지별 해설

① O: 셀렉션은 조건을 만족하는 튜플을 고르는 연산이다. 두 조건을 AND로 묶으면 두 조건을 모두 만족해야 하므로, 먼저 조건 c_2로 고른 뒤 조건 c_1으로 다시 골라도 결과가 같다. 이는 **AND 조건의 셀렉션을 연속된 셀렉션으로 분해**하는 규칙이다.

② O: 프로젝션은 지정한 속성만 남기는 연산이다. A와 B를 남긴 결과에서 다시 A만 추출하면 처음부터 A만 추출한 것과 같다. 안쪽 프로젝션이 **최종적으로 필요한 속성 A를 보존**하므로 중간 프로젝션을 생략할 수 있다.

③ O: 조건 c가 **R의 속성에만 관련**되므로, 조인 결과의 튜플이 조건을 만족하는지는 그 튜플을 구성한 R의 튜플만으로 결정된다. 따라서 R과 S를 조인한 뒤 조건 c로 고르는 것과, R에서 조건 c를 만족하는 튜플만 먼저 골라 S와 조인하는 것은 결과가 같다.

④ X: 합집합 가능한 R과 S에서 속성 A를 추출하는 올바른 변환은 $\pi_A(R \cup S) \equiv \pi_A(R) \cup \pi_A(S)$이다. 합집합은 어느 한쪽에 있는 튜플도 포함하지만, 자연 조인은 공통 속성값이 일치하는 튜플끼리 결합하므로 **합집합을 자연 조인으로 바꿀 수 없다**. 예를 들어 A만 가진 두 릴레이션에 각각 값 1과 값 2가 하나씩 있으면, 합집합에는 두 값이 모두 남지만 자연 조인 결과는 빈 릴레이션이다.

NoteBase

문 16. <보기 1>과 <보기 2>를 실행하여 저장된 데이터에 대해 <보기 3>의 SQL문을 실행할 때, 조회되는 투플(tuple)의 수가 가장 적게 나오는 SQL문과 가장 많이 나오는 SQL문을 옳게 짝지은 것은? (단, P1과 P2 테이블의 첫 번째 속성명은 COL1이다.)

<보기 1>

```
INSERT INTO P1 VALUES (1, 4);
INSERT INTO P1 VALUES (2, 5);
INSERT INTO P1 VALUES (3, 6);
INSERT INTO P1 VALUES (4, 7);
```

<보기 2>

```
INSERT INTO P2 VALUES (1, 4);
INSERT INTO P2 VALUES (2, 5);
INSERT INTO P2 VALUES (NULL, 6);
INSERT INTO P2 VALUES (NULL, 7);
```

<보기 3>

```
ㄱ. SELECT * FROM P1, P2
    WHERE P1.COL1 = P2.COL1;
ㄴ. SELECT * FROM P1 LEFT OUTER JOIN P2
    ON P1.COL1 = P2.COL1;
ㄷ. SELECT * FROM P1 RIGHT OUTER JOIN P2
    ON P1.COL1 = P2.COL1;
ㄹ. SELECT * FROM P1 FULL OUTER JOIN P2
    ON P1.COL1 = P2.COL1;
```

① ㄱ, ㄴ

NoteBase

② ㄱ, ㄹ

③ ㄴ, ㄷ

④ ㄷ, ㄹ

정답 2번

간단한 해설

등가 조건으로 연결되는 것은 COL1이 1과 2인 두 쌍뿐이므로 **내부 조인은 2개**의 투플을 반환한다. 완전 외부 조인은 양쪽에서 연결되지 않은 투플도 모두 보존하므로 **6개**를 반환한다.

선지별 해설

P1의 COL1 값은 1, 2, 3, 4이고, P2의 COL1 값은 1, 2, NULL, NULL이다. NULL과의 등호 비교는 참이 아니라 알 수 없음(UNKNOWN)이므로, P1.COL1 = P2.COL1을 만족하는 쌍은 값이 1인 쌍과 2인 쌍뿐이다. 외부 조인은 조건에 맞는 상대가 없는 투플도 보존하며, 이때 상대 테이블의 열을 NULL로 채운다.

ㄱ. 내부 조인: WHERE의 등가 조건을 만족하는 두 쌍만 조회한다. P1의 (1, 4)는 P2의 (1, 4)와, P1의 (2, 5)는 P2의 (2, 5)와 연결되므로 결과는 **2개**이다.

ㄴ. 왼쪽 외부 조인: **왼쪽 테이블 P1의 모든 투플**을 보존한다. 연결되는 2개에 더하여, P1의 (3, 6)과 (4, 7)도 P2 쪽 열을 NULL로 채워 조회하므로 2 + 2 = 4개이다.

ㄷ. 오른쪽 외부 조인: **오른쪽 테이블 P2의 모든 투플**을 보존한다. 연결되는 2개에 더하여, P2의 (NULL, 6)과 (NULL, 7)도 P1 쪽 열을 NULL로 채워 조회하므로 2 + 2 = 4개이다. 두 투플은 COL1이 모두 NULL이어도 각각 별개의 투플로 남는다.

ㄹ. 완전 외부 조인: **양쪽 테이블에서 연결되지 않은 투플을 모두 보존**한다. 연결되는 2개, P1에서만 남는 2개, P2에서만 남는 2개를 합쳐 2 + 2 + 2 = 6개이다. 연결되는 두 쌍은 왼쪽·오른쪽에 각각 있다고 해서 중복으로 세지 않는다.

조회되는 투플 수는 ㄱ 2개, ㄴ 4개, ㄷ 4개, ㄹ 6개이다. 가장 적은 것은 ㄱ, 가장 많은 것은 ㄹ이므로 ②의 짝과 일치한다.

NoteBase

문 17. 전통적인 파일 처리 시스템(file-processing system)의 문제점을 극복하기 위해 도입된 데이터베이스 관리 시스템(DBMS, database management system)의 특징으로 가장 옳지 않은 것은?

① 응용 프로그램과 데이터 간의 상호 의존성을 높여 데이터 처리의 효율성을 극대화한다.

② 다수의 사용자가 동시에 동일한 데이터에 접근하고 갱신할 수 있도록 동시성 제어(concurrency control) 기능을 제공한다.

③ 데이터의 중복을 제어하여 저장 공간의 낭비를 막고 데이터의 일관성을 유지한다.

④ 물리적 데이터 구조가 변경되더라도 응용 프로그램에 영향을 주지 않는 물리적 데이터 독립성을 제공한다.

정답 1번

간단한 해설

①의 '상호 의존성을 높여'는 DBMS의 특징과 반대이다. DBMS는 응용 프로그램이 데이터의 저장 방식에 직접 의존하는 정도를 낮추어 **데이터 독립성**을 제공한다.

선지별 해설

① X: 전통적인 파일 처리 시스템에서는 응용 프로그램이 파일의 구조나 저장 방식을 직접 다루므로, 이들이 바뀌면 프로그램도 수정해야 하는 문제가 있다. DBMS는 데이터의 구조와 접근을 관리하여 **응용 프로그램과 데이터 간의 의존성을 낮춘다**. 따라서 '상호 의존성을 높여'는 '데이터 독립성을 높여'로 고쳐야 한다.

② O: 여러 사용자의 작업이 동시에 수행되면 한 사용자의 갱신 결과가 다른 사용자의 작업에 의해 덮어써지는 등의 문제가 생길 수 있다. DBMS는 **동시성 제어**를 통해 충돌하는 작업의 실행 순서를 조정하여, 여러 사용자가 데이터를 공유하면서도 일관성을 유지하도록 한다. 동시 접근을 지원한다는 것이 충돌하는 모든 갱신을 제한 없이 동시에 실행한다는 뜻은 아니다.

③ O: 같은 데이터를 여러 파일에 각각 저장하면 공간이 낭비되고, 일부만 수정되어 서로 다른 값이 남을 수 있다. DBMS는 데이터를 통합적으로 관리하고 **불필요한 중복을 제어**하여 이러한 낭비와 불일치를 줄인다. 중복 제어는 모든 중복을 반드시 없앤다는 의미는 아니다.

④ O: 물리적 데이터 독립성은 **저장 구조가 바뀌어도 응용 프로그램을 수정하지 않아도 되는 성질**이다. 예를 들어 데이터의 논리적 구조는 유지하면서 저장 위치나 인덱스를 변경하더라도, DBMS가 접근 방식을 처리하므로 응용 프로그램은 기존 방식으로 데이터를 이용할 수 있다.

NoteBase

문 18. <보기 1>의 시험(EXAM) 테이블에는 6건의 시험성적 데이터가 저장되어 있다. <보기 2>의 SQL문 실행 결과에 포함되지 않는 것은?

<보기 1>

EXAM

no	name	course	score
1	Kim	DataBases	20
2	Lee	DataBases	35
3	Kang	Algorithms	25
4	Jeon	Networks	30
5	Seo	Systems	25
6	Shin	Systems	20

<보기 2>

```
SELECT name
FROM EXAM AS E1
WHERE score >= (SELECT AVG(score)
                FROM EXAM AS E2
                WHERE E2.course = E1.course);
```

① Lee

② Seo

③ Shin

④ Jeon

정답 3번

NoteBase

간단한 해설

이 SQL문은 성적이 **같은 과목의 평균 이상**인 행의 이름을 출력한다. Systems의 평균은 22.5점이므로, 20점인 Shin은 조건을 충족하지 못해 결과에서 제외된다.

계산 과정

비교 기준: E1과 E2는 같은 EXAM 테이블에 붙인 별칭이다. 부질의의 E2.course = E1.course는 현재 검사하는 E1 행과 **같은 과목의 행만** 골라 AVG(score)로 평균을 구한다. 자기 자신의 성적도 평균에 포함된다. 바깥 WHERE절은 현재 행의 score가 이 평균 이상인지 검사하므로, 평균과 같은 성적도 결과에 포함된다.

DataBases: Kim은 20점, Lee는 35점이므로 과목 평균은 $\frac{20+35}{2} = 27.5$점이다. Kim은 $20 < 27.5$이므로 제외되고, Lee는 $35 \geq 27.5$이므로 포함된다.

Algorithms: 해당 과목의 행은 Kang의 25점 한 건뿐이므로 평균도 25점이다. Kang은 $25 \geq 25$를 만족하므로 포함된다.

Networks: 해당 과목의 행은 Jeon의 30점 한 건뿐이므로 평균도 30점이다. Jeon은 $30 \geq 30$을 만족하므로 포함된다.

Systems: Seo는 25점, Shin은 20점이므로 과목 평균은 $\frac{25+20}{2} = 22.5$점이다. Seo는 $25 \geq 22.5$이므로 포함되지만, Shin은 $20 < 22.5$이므로 제외된다.

출력되는 이름은 Lee, Kang, Jeon, Seo이다. 제시된 선택지 중 결과에 포함되지 않는 이름은 ③ Shin이다.

NoteBase

문 19. <보기>는 즉시 갱신(immediate update) 기법을 사용하는 데이터베이스의 로그 파일 기록이다. 시스템 장애가 발생한 직후 회복 관리자가 수행해야 할 조치로 가장 옳은 것은? (단, 체크포인트 수행 시 메모리 버퍼의 갱신 내용은 모두 디스크에 안전하게 기록된다고 가정한다.)

<보기>

로그 기록
<T1, start>
<T1, A, 10, 20>
<T2, start>
<T2, B, 15, 30>
<T1, commit>
<checkpoint, active {T2}>
<T3, start>
<T3, C, 40, 50>
<T2, commit>
-- 시스템 장애 발생 --

① T1과 T2에 대해 Redo 연산을 수행하고, T3에 대해서는 아무 작업도 수행하지 않는다.

② T2에 대해 Redo 연산을 수행하고, T3에 대해서는 Undo 연산을 역순으로 수행한다.

③ T2와 T3 모두 장애 발생 시점에 완전하게 반영되지 않았으므로 Undo 연산을 역순으로 수행한다.

④ T1, T2, T3 모든 트랜잭션에 대해 순차적으로 Redo 연산을 수행하여 시스템을 복구한다.

정답 2번

간단한 해설

체크포인트 이후 커밋한 T2는 **Redo 대상**이고, 장애 때까지 커밋하지 않은 T3는 **Undo 대상**이다. 즉시 갱신에서는 커밋 전의 변경도 디스크에 반영될 수 있으므로, T3의 변경은 취소해야 한다.

NoteBase

대응 관계 해설

즉시 갱신은 트랜잭션이 커밋되기 전에도 변경 내용을 디스크에 반영할 수 있는 기법이다. Redo는 로그의 변경 후 값을 다시 적용하여 커밋된 결과를 보장하고, Undo는 변경 전 값으로 되돌려 미완료 트랜잭션의 영향을 제거한다. 이 문제에서는 체크포인트 때 메모리 버퍼의 갱신 내용이 모두 디스크에 기록되지만, 이것이 진행 중인 트랜잭션의 커밋을 뜻하지는 않는다.

T1 — 복구 작업 불필요: 로그에서 T1의 commit은 체크포인트보다 앞에 있다. A를 10에서 20으로 바꾼 결과는 **커밋된 상태로 체크포인트에서 디스크 기록까지 보장**되므로, T1에 대한 Redo나 Undo는 필요하지 않다.

T2 — Redo: 체크포인트의 active {T2}는 그때 T2가 아직 실행 중이었다는 뜻이다. 이후 장애 전에 commit이 기록되었으므로, T2는 **체크포인트 때 실행 중이었다가 이후 커밋한 트랜잭션**으로서 Redo 대상으로 분류한다. 로그의 변경 후 값인 B = 30을 다시 적용하여 커밋 결과를 보장한다. 제시된 B의 변경은 체크포인트 전에 이루어져 이미 디스크에 기록되었지만, 같은 값 30을 다시 기록해도 결과는 달라지지 않는다.

T3 — 역순 Undo: 체크포인트 이후 시작되었고, 장애 전까지 commit 기록이 없다. 즉시 갱신으로 미커밋 변경이 디스크에 반영되었을 수 있으므로 **로그를 역순으로 따라가며 변경 전 값으로 복원**한다. 제시된 T3의 갱신은 C를 40에서 50으로 바꾼 한 건이므로, Undo로 C를 40으로 되돌린다. 여러 번 갱신했다면 나중 변경부터 취소해야 앞선 변경까지 차례로 되돌릴 수 있다.

따라서 T1에는 별도 복구 작업을 하지 않고, T2에는 Redo를, T3에는 역순 Undo를 수행하는 조합이므로 ②와 일치한다.

NoteBase

문 20. 대규모 의사결정 지원을 위한 데이터 웨어하우스(data warehouse)의 다차원 모델링 기법 중, <보기>의 설명에 해당하는 스키마 구조로 가장 옳은 것은?

<보기>
· 중앙에 매출, 수량 등의 트랜잭션 수치 데이터를 담고 있는 하나의 거대한 '사실(fact) 테이블'이 존재한다.
· 사실 테이블을 둘러싼 '차원(dimension) 테이블'들이 존재하는데, 이 차원 테이블들이 제3정규형(3NF) 등으로 정규화되어 있고 사실 테이블을 중심으로 뻗어 나가는 다단계 계층 구조를 이룬다.
· 차원 테이블 내의 데이터 중복을 최소화하여 저장 공간을 절약할 수 있으나, 사용자의 질의 수행 시 여러 테이블을 조인(join)해야 하므로 모델 구조가 복잡해지고 검색 성능이 저하될 수 있다.

① 스노우플레이크 스키마(snowflake schema)
② 스타 스키마(star schema)
③ 팩트 컨스텔레이션(fact constellation)
④ 데이터 큐브(data cube)

정답 1번

간단한 해설

차원 테이블을 **정규화하여 여러 단계의 테이블로 나눈 구조**가 스노우플레이크 스키마이다. 중복 데이터는 줄어들지만, 계층을 따라 여러 테이블을 조인해야 하므로 질의가 복잡해지고 검색 성능이 저하될 수 있다.

핵심 풀이

스노우플레이크 스키마: 사실 테이블은 매출·수량 같은 분석 대상 수치를 담고, 차원 테이블은 상품·지역·시간 같은 분석 기준을 담는다. <보기>처럼 사실 테이블을 중심으로 차원 테이블을 제3정규형(3NF) 등으로 정규화하여 **다단계 계층으로 분리**한 구조가 스노우플레이크 스키마이다. 중앙에 사실 테이블이 있다는 점보다 차원 테이블이 정규화되어 뻗어 나간다는 점이 결정적인 단서이다.

정규화의 효과: 예를 들어 상품마다 상품 분류명을 반복 저장하는 대신, 상품 테이블과 상품 분류 테이블을 분리하고 분류 식별자로 연결할 수 있다. 이때 분류명을 별도 테이블에서 관리하므로 중복을 줄이고 저장 공간을 절약할 수 있다. 다만 매출을 상품 분류별로 조회하려면 사실 테이블에서 상품 테이블을 거쳐 상품 분류 테이블까지 연결해야 하므로 **조인이 늘어나** 질의 구조가 복잡해지고 검색 성능이 저하될 수 있다.

스타 스키마와의 구분: 스타 스키마는 차원 정보를 대체로 **비정규화된 차원 테이블**에 모아 사실 테이블과 직접 연결한다. 따라서 차원의 계층 정보를 여러 테이블로 분리한 스노우플레이크 스키마보다 구조와 조인이 단순하다. <보기>의 '차원 테이블 정규화'와 '다단계 계층 구조'는 스타 스키마와 구별되는 특징이다.

NoteBase