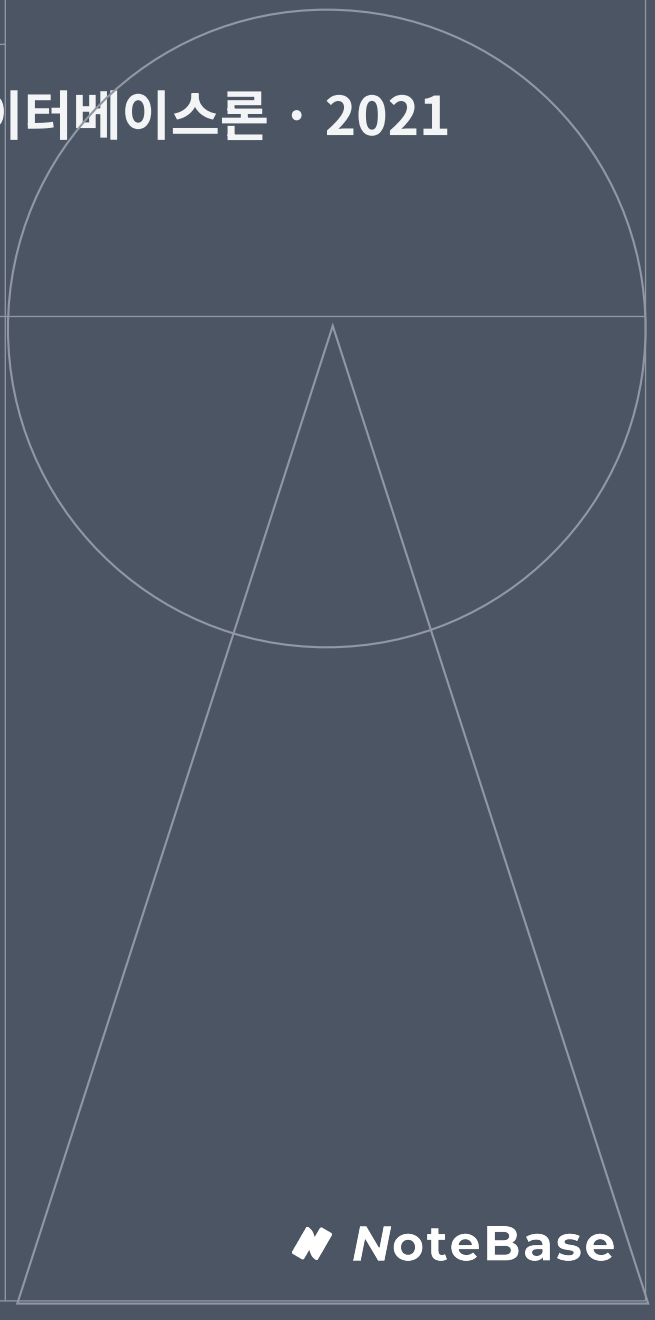
기출OX 해설집

공무원/경찰/경력채용 · 데이터베이스론 · 2021



NoteBase

기출OX 해설집

공무원/경찰/경력채용 · 데이터베이스론 · 2021

공무원/경찰/경력채용 · 데이터베이스론 · 2021

문 1. DBMS를 사용하여 얻을 수 있는 장점으로 가장 적절하지 않은 것은?

① 데이터베이스 사용자들은 기술적인 지식의 보유 정도에 따라 다양한 그룹으로 나누어지므로 DBMS는 이들 각각에 적합한 인터페이스를 제공한다.

② 제어된 중복성(controlled redundancy) 기능을 통해 데이터 중복을 완전히 제거함으로써 변경 작업의 비효율성, 메모리 낭비, 정보의 불일치 문제 등을 해결할 수 있다.

③ DBMS의 보안(security) 및 권한(authorization) 서브시스템을 통해 권한이 없는 사용자의 데이터베이스 접근을 통제할 수 있다.

④ DBMS는 데이터베이스에 대한 질의와 갱신을 효율적으로 수행하는 기능을 제공하기 위해 질의 처리와 최적화 모듈을 제공한다.

정답 2번

간단한 해설

제어된 중복성은 **중복의 완전한 제거가 아니라 필요한 수준으로 통제**하는 것이다. 불필요한 중복을 줄이면서 성능상 필요한 중복은 일관성이 유지되도록 관리한다.

선지별 해설

① O: DBMS는 사용자 역할과 기술 수준에 맞는 접근 방식을 제공한다. 응용 프로그램용 인터페이스, SQL 기반 질의 도구, 일반 사용자를 위한 화면·양식 등으로 데이터 이용을 지원할 수 있다.

② X: 제어된 중복성은 **불필요한 중복을 줄이고 필요한 중복은 관리**한다는 뜻이지, 모든 중복을 완전히 제거한다는 뜻이 아니다. 성능을 위해 일부 데이터를 의도적으로 중복 저장할 수도 있으며, 이때 갱신 과정에서 서로 다른 값이 생기지 않도록 일관성을 유지해야 한다.

③ O: DBMS는 인증과 권한 검사를 통해 사용자가 허용된 데이터와 연산에만 접근하도록 통제한다. 예를 들어 특정 사용자에게 조회 권한만 주고 수정 권한은 주지 않을 수 있다.

④ O: 질의 처리·최적화 모듈은 요청을 해석하고 실행 방법을 비교하여 효율적인 계획을 선택한다. 인덱스 사용 여부나 조인 방법·순서 등을 고려하여 조회와 데이터 변경 작업의 처리 비용을 줄이는 데 기여한다.

NoteBase

문 2. 데이터 독립성(data independence)을 지원하는 데이터베이스 시스템에서 특정 테이블에 대한 검색성능을 향상시키기 위해 인덱스를 추가하였다. 그 결과로 인해 변경될 가능성이 있는 것으로 가장 적절한 것은?

① 응용프로그램 자료 구조

② 외부 스키마(external schema)

③ 개념 스키마(conceptual schema)

④ 내부 스키마(internal schema)

정답 4번

간단한 해설

인덱스는 데이터의 논리적 의미가 아니라 검색을 위한 저장·접근 방법이다. 따라서 추가로 바뀌는 것은 **물리적 접근 경로를 명세하는 내부 스키마**이다.

핵심 풀이

인덱스 추가는 데이터의 의미나 테이블의 논리적 구성을 바꾸는 것이 아니라, 필요한 레코드를 더 빠르게 찾을 **물리적 접근 경로를 추가**하는 작업이다. 저장 구조와 접근 방법을 명세하는 내부 스키마가 변경 대상이므로 ④에 해당한다.

물리적 데이터 독립성은 **내부 스키마가 바뀌어도 개념 스키마를 유지**할 수 있게 한다. 따라서 인덱스를 추가했다는 이유만으로 전체 논리 구조인 개념 스키마나 사용자별 관점인 외부 스키마, 이를 이용하는 응용프로그램 자료 구조를 변경할 필요는 없다. 내부와 개념 수준의 대응은 DBMS가 처리한다.

NoteBase

문 3. DBMS 구성 요소 중 예비컴파일러(precompiler)에 대한 설명으로 적절한 것을 모두 고른 것은?

㉠ 응용프로그램에 포함된 DML 명령문을 추출한다.
㉡ DML 명령문을 번역하여 목적코드를 생성한다.
㉢ 질의문을 파싱, 분석, 컴파일하여 코드 생성 후 런타임 데이터베이스 처리기를 호출하여 실행한다.
㉣ 추출된 DML을 DML 컴파일러에 전달한다.

① ㉢㉣
② ㉠㉣
③ ㉡㉢
④ ㉠㉡

정답 2번

간단한 해설

예비컴파일러는 응용 프로그램에 포함된 **DML을 추출하여 DML 컴파일러로 전달**한다. DML의 목적코드 생성이나 질의의 실행 전체를 맡는 구성 요소와는 구별된다.

보기별 해설

㉠ O: 예비컴파일러는 호스트 언어로 작성한 응용 프로그램에서 **삽입된 DML 명령문을 찾아 분리**한다. 일반 프로그램 코드와 데이터베이스 조작 명령이 서로 다른 처리 경로로 전달되도록 하는 역할이다.

㉡ X: 추출된 DML을 번역하여 데이터베이스 접근을 위한 목적코드를 만드는 것은 이 구성 모형에서 **DML 컴파일러**의 역할이다. 예비컴파일러의 DML 분리·전달 역할과 구별된다.

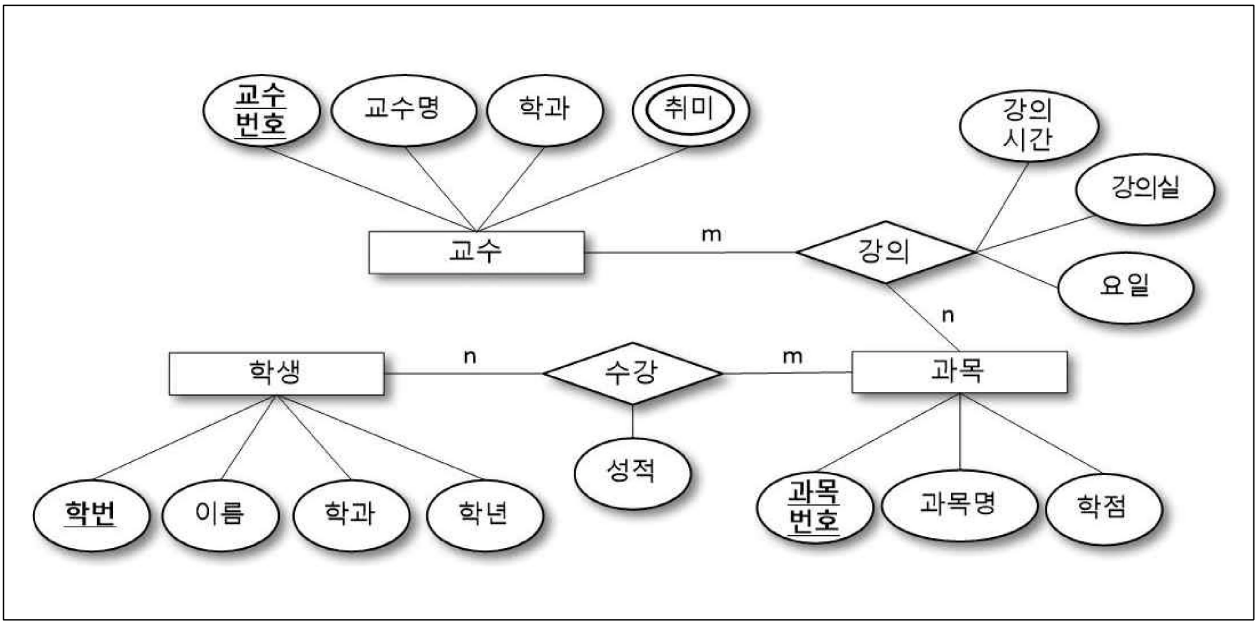
㉢ X: 질의를 파싱·분석하고 컴파일하여 실행 코드나 계획을 만든 뒤 런타임 처리기로 실행하는 일은 **질의 처리 경로**에 해당한다. 예비컴파일러는 응용 프로그램의 내장 DML을 분리하는 구성 요소이지, 이 질의 처리·실행 전체를 담당하지 않는다.

㉣ O: 예비컴파일러는 추출한 DML을 **DML 컴파일러에 전달**하여 번역되도록 한다. DML을 분리한 나머지 호스트 언어 코드는 해당 언어의 컴파일러가 처리하는 구조이다.

따라서 예비컴파일러의 역할에 해당하는 것은 ㉠·㉣이며, 이 조합은 ②이다.

NoteBase

문 4. 다음 ER 다이어그램을 관계 스키마로 변환하였을 때 결과에 포함되는 릴레이션으로 가장 적절하지 않은 것은? (단, 밑줄 친 애트리뷰트는 기본키이다)



① 과목(과목번호, 과목명, 학점)

② 학생(학번, 이름, 학과, 학년)

③ 강의(과목번호, 학번, 요일, 강의시간, 강의실)

④ 교수취미(교수번호, 취미)

정답 3번

간단한 해설

강의 관계의 참여 개체는 **교수와 과목**이며 다대다 관계이다. 학번이 아니라 교수번호를 포함하고, 교수번호와 과목번호의 조합을 기본키로 삼아야 한다.

NoteBase

선지별 해설

① O: 과목 개체에는 과목번호, 과목명, 학점이 있고 과목번호가 식별자이다. 따라서 과목(과목번호, 과목명, 학점)을 만들고 **과목번호를 기본키**로 지정하는 것은 적절하다.

② O: 학생 개체에는 학번, 이름, 학과, 학년이 있고 학번이 식별자이다. 따라서 학생(학번, 이름, 학과, 학년)을 만들고 **학번을 기본키**로 지정한다.

③ X: 강의는 학생과 과목이 아니라 **교수와 과목 사이의 다대다 관계**이다. 따라서 강의(교수번호, 과목번호, 요일, 강의시간, 강의실)로 변환하며, 그림의 관계 제약에 따른 기본키는 **(교수번호, 과목번호)**이다. 선지는 교수번호 대신 학번을 넣었고 과목번호만 기본키로 삼았으므로 모두 부적절하다. 학생과 과목의 관계는 별도의 수강 관계이다.

④ O: 취미는 이중 타원으로 표시된 교수의 다중값 속성이다. 이를 교수 릴레이션의 단일 속성에 여러 값으로 저장하지 않고 교수취미(교수번호, 취미)로 분리한다. 한 교수의 여러 취미를 구별하도록 **(교수번호, 취미)를 복합 기본키**로 삼는다.

따라서 그림을 관계 스키마로 변환한 결과로 가장 적절하지 않은 것은 ③이다.

NoteBase

문 5. 다음과 같은 함수적 종속성을 가진 릴레이션 R을 제3정규형으로 변환하였을 때 결과에 포함되는 릴레이션으로 가장 적절하지 않은 것은? (단, 제3정규화 후 릴레이션의 이름은 임의로 정한 것임)

R(a, b, c, d, e), 기본키 : (a, d)

R.a → R.b
R.a → R.c
R.(a,d) → R.e
R.b → R.c

① R1(a, b)

② R2(b, c)

③ R3(a, d, e)

④ R4(a, b, c)

정답 4번

간단한 해설

a가 b를, b가 c를 결정하므로 a, b, c를 함께 두면 **비키 속성 사이의 이행적 종속**이 남는다. 이를 (a, b)와 (b, c)로 분리하고, 복합키에 종속되는 e는 (a, d, e)에 둔다.

선지별 해설

① O: 원래 기본키는 (a, d)이지만 b는 a만으로 결정된다. 이 부분 함수 종속을 분리하면 R1(a, b)를 얻으며, 이 릴레이션에서는 a가 키이다. 주어진 종속성 아래에서 제3정규형을 만족한다.

② O: b가 c를 결정하므로 R2(b, c)로 분리한다. 이 릴레이션에서는 b가 키이며, a를 거쳐 c가 결정되는 이행적 종속을 별도 릴레이션으로 옮기는 역할을 한다.

③ O: e는 복합키 (a, d)에 종속된다. a에만 종속되는 b와 c를 분리한 뒤 R3(a, d, e)를 남기면 (a, d)가 키이고, 제시된 종속성에서 e의 부분·이행 종속이 없으므로 제3정규형에 적합하다.

④ X: R4(a, b, c)에서는 a가 키이지만 **키가 아닌 b가 비주요 속성 c를 결정**한다. 따라서 a → b → c의 이행적 종속이 남아 제3정규형을 만족하지 않는다. R1(a, b)와 R2(b, c)로 나누어야 한다.

NoteBase

문 6. <보기 1>은 어느 상품주문 데이터베이스의 관계 스키마를 보인 것이며, <보기 2>는 제약조건들을 나열한 것이다. <보기 1>의 데이터베이스에 대한 연산으로 인해 발생할 수 있는 제약조건 위반 사항을 표시한 것으로 가장 적절하지 않은 것은? (단, 밑줄 친 애트리뷰트는 기본키이다)

<보기 1>
고객(고객ID, 고객명, 주소, 전화번호)
상품(상품코드, 상품명, 가격)
주문(고객ID, 상품코드, 주문수량)
<보기 2>
㉠ 개체 무결성 제약조건(entity integrity constraint)
㉡ 참조 무결성 제약조건(referential integrity constraint)
㉢ 키 제약조건(key constraint)

① 새로운 고객 레코드를 고객 테이블에 삽입한다 - ㉠㉢
② 새로운 주문 레코드를 주문 테이블에 삽입한다 - ㉠㉡㉢
③ 특정 상품 레코드를 상품 테이블에서 삭제한다 - ㉡
④ 특정 주문 레코드를 주문 테이블에서 삭제한다 - ㉡

정답 4번

간단한 해설

주문은 고객과 상품을 참조하는 자식 테이블이다. **주문 삭제는 참조를 제거할 뿐 참조 대상을 없애지 않으므로**, 제시된 관계의 참조무결성을 위반하지 않는다.

NoteBase

선지별 해설

고객의 기본키는 고객ID, 상품의 기본키는 상품코드이며, 주문의 기본키는 고객ID와 상품코드의 조합이다. 상품주문 관계에서 주문의 고객ID와 상품코드는 각각 고객과 상품을 참조한다. ㉠은 기본키의 NULL 금지, ㉡은 외래키 참조 대상의 존재, ㉢은 키 값의 유일성을 요구한다.

① O: 고객을 삽입할 때 고객ID가 NULL이면 ㉠을, 기존 고객ID와 중복되면 ㉢을 위반할 수 있다. 고객은 이 관계에서 참조되는 쪽이므로 새 고객을 추가하는 것만으로 주문의 참조무결성이 깨지지는 않는다.

② O: 주문을 삽입할 때 기본키 구성 속성 중 하나라도 NULL이면 ㉠을 위반한다. 고객ID나 상품코드가 해당 부모 테이블에 없으면 ㉡을, **(고객ID, 상품코드)의 조합이 기존 주문과 중복**되면 ㉢을 위반할 수 있다. 따라서 세 제약 모두 검토 대상이다.

③ O: 삭제하려는 상품을 기존 주문이 참조하면 그 주문의 참조 대상이 사라져 ㉡ 위반 문제가 생길 수 있다. 실제 DBMS는 삭제를 제한하거나 정의된 연쇄 동작 등을 수행하여 무결성을 유지한다. 상품 삭제 자체가 남은 상품의 키를 NULL이나 중복 값으로 만드는 것은 아니다.

④ X: 주문은 고객과 상품을 참조하는 자식 테이블이다. **주문 행을 삭제하면 참조가 없어질 뿐, 부모 레코드가 사라지는 것은 아니다**. 제시된 관계에서 주문 삭제는 참조 대상 없는 외래키를 만들지 않으므로 ㉡ 위반 가능 연산으로 표시한 것은 부적절하다.

NoteBase

문 7. 관계 데이터 모델의 연산에 대한 설명으로 가장 적절하지 않은 것은?

① 관계해석은 원하는 정보와 그 정보를 어떻게 유도하는가를 기술하는 절차적 언어이다.

② 관계해석과 관계대수는 관계 데이터베이스를 처리하는 기능과 능력 면에서 동등하다.

③ 관계해석은 수학의 Predicate Calculus에 기반을 두고 있다.

④ 관계해석에는 튜플 관계해석과 도메인 관계해석이 있다.

정답 1번

간단한 해설

관계해석은 결과를 얻는 절차가 아니라 결과가 만족할 조건을 표현한다. **관계해석은 비절차적, 관계대수는 절차적**이라는 구분이 핵심이다.

선지별 해설

① X: 관계해석은 원하는 결과가 만족해야 할 조건을 기술하는 **비절차적 언어**이다. 데이터를 어떤 연산 순서로 얻을지까지 표현하는 절차적 성격은 관계대수에 해당하므로 두 언어의 특징을 바꾸어 설명했다.

② O: 기본 관계대수와 **안전한 식으로 제한한 관계해석의 표현 능력**은 동등하다. 같은 질의 결과를 서로 다른 형식으로 표현할 수 있다는 의미이며, 실행 속도나 구현 방식까지 같다는 뜻은 아니다.

③ O: 관계해석은 술어논리(Predicate Calculus)에 바탕을 두어 결과에 포함될 데이터의 조건을 나타낸다. 비교 조건과 논리 연결자, 존재·전칭 한정자 등을 이용하는 **논리적 조건 명세**가 핵심이다.

④ O: 튜플 관계해석은 변수가 튜플 전체를 나타내고, 도메인 관계해석은 변수가 개별 속성값을 나타낸다. 두 방식은 **변수가 대상으로 삼는 단위**가 다르지만 모두 조건으로 결과를 기술한다.

NoteBase

문 8. 결과 테이블 T를 구하기 위해 테이블 R과 테이블 S에 적용한 연산으로 가장 적절한 것은?

R

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

S

A	D	E
a1	d1	e1
a1	d2	e2
a2	d3	e3
a3	d1	e4
a3	d4	e5
a3	d2	e6
a4	d2	e7

T

A	B	C	D	E
a1	b1	c1	d1	e1
a1	b1	c1	d2	e2
a2	b2	c2	d3	e3
a3	b3	c3	d1	e4
a3	b3	c3	d4	e5
a3	b3	c3	d2	e6
a4	b4	c4	d2	e7

① 외부 조인(outer join)

② 자연 조인(natural join)

③ 세미 조인(semi join)

④ 셀프 조인(self join)

정답 2번

간단한 해설

두 테이블의 **공통 속성 A가 같은 행을 결합하고 A를 한 번만 표시**한 결과이다. a1·a2·a3·a4의 대응 행이 모두 결합되어 7행이 되는 자연 조인으로 설명할 수 있다.

NoteBase

핵심 풀이

공통 속성 확인: R의 속성은 A·B·C이고 S의 속성은 A·D·E이다. 두 테이블에 공통인 **A의 값이 같은 행을 결합**하고, 공통 속성 A를 결과에 한 번만 남기면 T의 A·B·C·D·E 구조가 된다.

행별 대응: R의 a1 행은 S의 a1 두 행과 결합해 2행이 되고, a2는 1행, a3는 3행, a4는 1행이 된다. a5는 S에 대응하는 값이 없어 제외된다. 따라서 **2+1+3+1=7행**으로 그림의 T와 일치한다.

다른 조인과의 구별: 세미 조인은 한쪽 테이블의 속성만 반환하므로 양쪽의 B·C와 D·E를 모두 가진 T를 설명하지 못한다. 셀프 조인은 같은 테이블을 자신과 연결하는 방식인데 여기서는 서로 다른 R과 S를 결합한다. 외부 조인은 지정한 쪽의 미매칭 행을 보존하는 데 초점이 있다. R을 보존하면 a5도 남지만 T에는 없다. 단, S를 보존하는 오른쪽 외부 조인은 이 자료에서 우연히 같은 결과가 될 수 있으므로 모든 외부 조인이 반드시 다른 결과라고 단정하지는 않는다.

이 문제에서 공통 속성의 동등 조건과 공통 열의 중복 제거를 가장 직접적으로 나타내는 연산은 **자연 조인**이며 ②가 가장 적절하다.

NoteBase

문 9. (가)의 SQL 문은 어느 회사 데이터베이스의 직원과 부서 테이블 정의 중 일부를 보인 것이며, (나)는 부서 테이블의 부서장ID에 대한 제약조건 요구사항을 설명한 것이다. (가)의 SQL 문에서 ㉠에 들어갈 제약조건 표현으로 가장 적절한 것은?

(가)
CREATE TABLE 직원 (
.....
PRIMARY KEY(직원ID));

CREATE TABLE 부서 (
.....
PRIMARY KEY(부서번호),
FOREIGN KEY(부서장ID) REFERENCES 직원(직원ID)
____㉠____);

(나) 부서 테이블의 제약조건
- 부서의 부서장인 직원이 퇴사하여 해당 직원 레코드를 삭제하게 되면 후임 부서장이 임명될 때까지 해당 부서 레코드의 부서장ID는 NULL 값을 가져야 한다.
- 부서장인 직원의 직원ID가 변경되면 해당 부서 레코드의 부서장ID 값도 같이 변경되어야 한다.

① ON DELETE SET NULL ON UPDATE CASCADE

② ON DELETE CASCADE ON UPDATE SET NULL

③ ON DELETE CASCADE ON UPDATE SET DEFAULT

④ ON DELETE SET NULL ON UPDATE SET DEFAULT

정답 1번

간단한 해설

직원이 삭제되면 부서장ID만 NULL로 바꾸므로 **ON DELETE SET NULL**을 사용한다. 직원ID 변경을 부서장ID에 전파하려면 **ON UPDATE CASCADE**를 사용한다.

NoteBase

구성 요소별 해설

삭제 조건 → **ON DELETE SET NULL**: 참조 대상인 직원 레코드가 삭제될 때, 이를 참조하던 부서 레코드는 유지하고 부서장ID만 NULL로 바꾼다. 부서장ID는 NULL을 허용해야 한다. ON DELETE CASCADE라면 부서 레코드까지 삭제하므로 요구사항에 맞지 않는다.

변경 조건 → **ON UPDATE CASCADE**: 참조 대상 직원의 직원ID가 변경되면, 그 직원을 참조하는 부서장ID에도 새로운 직원ID를 전파한다. SET NULL은 참조값을 NULL로, SET DEFAULT는 미리 정한 기본값으로 바꾸므로 변경된 직원ID를 그대로 따라가게 하는 요구와 다르다.

따라서 ㉠에는 ON DELETE SET NULL ON UPDATE CASCADE가 들어가며, 해당하는 것은 ①이다.

NoteBase

문 10. 보기의 데이터베이스에서 “자신이 속한 부서의 부서장보다 더 많은 급여를 받는 직원의 이름과 급여를 검색하라”라는 질의를 SQL로 표현한 것으로 가장 적절한 것은?

<보기>
직원(사번, 이름, 전화번호, 근무부서, 급여)
부서(부서코드, 부서명, 부서장, 사무실)

- 근무부서: 직원이 근무하는 부서의 부서코드 값을 가짐
- 부서장: 부서장인 직원의 사번을 값으로 가짐

① SELECT E.이름, E.급여
FROM 직원 E, 직원 M, 부서 D
WHERE E.근무부서=D.부서코드 AND D.부서장=M.사번
AND E.급여 > M.급여;

② SELECT E.이름, E.급여
FROM 직원 E, 직원 M
WHERE E.근무부서 IN (SELECT D.부서코드 FROM 부서 D
WHERE D.부서장=M.사번)
AND E.급여 > M.급여;

③ SELECT E.이름, E.급여
FROM 직원 E, 직원 M
WHERE EXISTS (SELECT * FROM 부서 D
WHERE M.근무부서=D.부서코드
AND D.부서장=E.사번)
AND E.급여 > M.급여;

④ SELECT E.이름, E.급여
FROM 직원 E
WHERE E.근무부서 IN (SELECT D.부서코드 FROM 부서 D
WHERE D.부서장.급여 < E.급여);

정답 1번

출제 포인트

직원과 그 직원이 속한 부서의 부서장을 연결한 뒤, 직원의 급여가 부서장의 급여보다 큰지를 판단하는 SQL 질의이다.

NoteBase

핵심 개념

자기 조인은 같은 테이블을 서로 다른 별칭으로 사용하여 행 사이의 관계를 연결하는 방법이다. 직원 E는 검색할 직원, 직원 M은 비교할 부서장을 나타낸다. 부서 D의 부서장에는 급여가 아니라 직원의 사번이 저장되어 있으므로, D.부서장과 M.사번을 연결해야 부서장의 급여를 얻을 수 있다. IN은 값이 하위 질의 결과에 포함되는지, EXISTS는 하위 질의 결과 행이 존재하는지를 판단한다.

선지별 해설

이 문제의 O/X는 가장 적절한 선지로 선택되는지를 표시한다. X가 언제나 SQL 자체의 오류를 뜻하는 것은 아니다.

① O: E.근무부서=D.부서코드로 직원의 소속 부서를 찾고, D.부서장=M.사번으로 그 부서의 부서장을 찾는다. E.급여 > M.급여는 직원이 자신의 부서장보다 급여를 많이 받는 조건이다. SELECT E.이름, E.급여도 요구한 출력과 일치한다. 필요한 두 연결 조건과 급여 비교를 직접 제시한다.

② X: 선택되지 않은 선지이지만, 주어진 구조에서는 ①과 같은 조건을 구현하는 유효한 질의이다. 하위 질의는 M이 부서장인 부서의 코드를 반환하고, E.근무부서가 그 코드에 포함되는지 확인하므로 M은 E의 소속 부서장이 된다. 이어 E.급여 > M.급여를 검사한다. ①은 이 연결 관계를 두 조인 조건으로 직접 드러내고, ②는 상관 하위 질의와 IN으로 표현한다. ②의 IN이나 상관 참조를 문법 오류로 교정해서는 안 된다.

③ X: M.근무부서=D.부서코드와 D.부서장=E.사번은 M의 소속 부서장이 E라는 관계를 만든다. 따라서 부하 직원 M보다 급여가 많은 부서장 E를 검색하는 방향으로 관계가 뒤바뀐다. M.근무부서=D.부서코드는 E.근무부서=D.부서코드로, D.부서장=E.사번은 D.부서장=M.사번으로 바꾸어야 한다.

④ X: D.부서장.급여는 부서장 속성에 저장된 사번에서 급여를 바로 꺼낼 수 있다고 가정한다. 그러나 제시된 관계형 구조에서 부서장은 사번 값이며, 직원 행이나 급여 속성을 포함하는 객체가 아니다. 직원 M을 추가하고 D.부서장=M.사번으로 연결한 뒤, E.급여 > M.급여로 비교해야 한다.

오답 포인트

③은 급여 비교 부호가 맞아 보여도 직원과 부서장의 연결 방향이 반대이다. ④는 부서장이라는 속성 이름을 직원 객체처럼 해석하게 만든다. 또한 ②는 IN을 사용했다는 이유만으로 틀린 질의가 되는 것은 아니며, 이 구조에서는 ①과 의미가 같다.

다시 나오면 이렇게 판단

검색 대상 별칭과 비교 대상 별칭을 구분하고, 직원의 근무부서에서 부서코드로, 부서의 부서장에서 직원의 사번으로 연결되는지 확인한다. 그다음 검색 대상의 급여가 부서장의 급여보다 큰지, 출력 열이 검색 대상의 이름과 급여인지 확인한다. 하위 질의도 같은 연결 관계를 구현하는지 따져 판단한다.

암기 포인트

직원의 근무부서 → 부서코드, 부서의 부서장 → 직원의 사번, 급여 비교는 직원 E > 부서장 M이다.

NoteBase

문 11. SQL 트리거(trigger)에 대한 설명으로 가장 적절하지 않은 것은?

① 트리거는 데이터베이스의 일관성을 유지하는 데 유용하지만, 과도하게 사용하면 복잡한 상호 의존성을 초래할 수 있다.

② 트리거는 어떤 이벤트가 일어날 때 조건이 참이면 트리거에 정의된 명령문이 수행되고, 그렇지 않으면 아무 동작도 하지 않는다.

③ 테이블 수준의 트리거와 열 수준의 트리거로 구분한다.

④ 트리거는 데이터의 변경(삽입, 삭제, 수정) 작업이 일어날 때 작업을 수행한다.

정답 3번

간단한 해설

트리거의 실행 단위는 **문장 수준과 행 수준**으로 구별한다. 특정 열의 갱신을 조건으로 삼는 것과, 문장당 또는 행마다 실행되는지를 나누는 것은 서로 다른 기준이다.

선지별 해설

① O: 트리거는 데이터 변경을 검사하거나 관련 데이터를 함께 갱신하여 일관성 유지에 도움을 준다. 다만 트리거의 작업이 다른 트리거를 연쇄적으로 실행시킬 수 있어, 과도한 사용은 실행 흐름과 상호 의존성을 복잡하게 만든다.

② O: 트리거는 사건·조건·동작의 구조로 이해할 수 있다. 지정한 사건이 발생하면 조건을 검사하고, 조건이 참일 때 정의된 동작을 수행한다. 조건이 거짓이면 그 트리거의 동작은 수행하지 않으며, 이것이 원래 SQL 문장 자체도 반드시 취소된다는 뜻은 아니다.

③ X: 실행 단위를 기준으로 하는 기본 구분은 **문장 수준 트리거와 행 수준 트리거**이다. 문장 수준은 SQL 문장당 한 번, 행 수준은 영향을 받는 각 행에 대해 실행한다. 특정 열의 갱신을 발동 조건으로 지정할 수 있다는 사실과 이러한 실행 단위 분류는 별개이다.

④ O: 데이터 변경 트리거는 INSERT, DELETE, UPDATE 같은 사건에 연결되어 자동 실행된다. 해당 작업의 전·후 또는 대신 실행하도록 정의할 수 있으며, 사용자가 트리거를 일반 프로시저처럼 직접 호출하는 방식과 구별된다.

NoteBase

문 12. B-트리에 대한 설명으로 가장 적절하지 않은 것은?

① 모든 단말 노드는 항상 같은 레벨에 위치한다.

② 인덱스로 사용되면 데이터 블록(또는 데이터 레코드)에 대한 포인터는 내부 노드와 단말 노드들에 저장한다.

③ 한 노드에서 킷값을 삭제한 후 그 노드의 반 이상이 비게 되면 그 노드는 인접한 노드와 합쳐져서 하나의 노드가 될 수도 있다.

④ 삽입 연산 중 한 노드에서 오버플로가 발생하면 그 노드를 두 노드로 분할하고 중간 킷값은 분할된 노드와 부모 노드에 모두 저장한다.

정답 4번

간단한 해설

B-트리의 분할에서는 **중간 키가 부모로 이동**하므로 분할된 자식 노드에 같은 키를 중복 저장하지 않는다. B+트리의 단말 분할에서 경계 키를 복사하는 경우와 구별해야 한다.

선지별 해설

① O: B-트리는 균형 잡힌 다진 탐색 트리로, **모든 단말 노드가 같은 깊이**에 있다. 삽입·삭제 시 분할·재분배·병합을 통해 이 성질과 노드 점유 조건을 유지한다.

② O: 전형적인 B-트리 인덱스에서는 키에 대응하는 **데이터 레코드 포인터가 내부 노드에도 존재**할 수 있으며 단말 노드에만 한정되지 않는다. 실제 데이터 항목을 단말에 모으는 B+트리의 기본 구조와 구별해야 한다.

③ O: 삭제로 최소 점유 조건을 충족하지 못하면 형제 노드에서 키를 재분배하거나 **인접 형제 노드와 병합**할 수 있다. 병합 시 부모의 구분 키를 함께 조정하며, 루트에는 별도의 최소 점유 조건이 적용된다.

④ X: B-트리에서 노드를 분할하면 **중간 키를 부모로 올려 이동**시키고, 그보다 작은 키와 큰 키를 각각 두 자식 노드에 나눈다. 따라서 중간 키를 부모와 분할된 자식에 모두 보관한다는 설명은 틀리다. B+트리의 단말 분할에서 경계 키를 부모에 복사하는 방식과 혼동하지 않아야 한다.

NoteBase

문 13. 기존 테이블에 새로운 칼럼(column)을 추가할 때 사용하는 명령문으로 가장 적절한 것은?

① INSERT

② ALTER

③ UPDATE

④ DROP

정답 2번

간단한 해설

새 열을 추가하면 테이블의 구조가 바뀐다. **구조 변경 명령인 ALTER TABLE**에 열 추가 절을 사용한다.

핵심 풀이

열 추가는 기존 행의 값을 바꾸는 작업이 아니라 **테이블의 정의·구조를 변경**하는 작업이다. 따라서 ALTER TABLE에 ADD COLUMN 절을 사용한다. 예를 들어 ALTER TABLE 학생 ADD COLUMN 이메일 VARCHAR(100);은 학생 테이블에 이메일 열을 추가하며, 제시된 명령 중 ALTER가 해당한다.

INSERT는 **새 행을 삽입**하고 UPDATE는 기존 행의 열 값을 수정한다. DROP은 테이블 같은 객체를 삭제할 때 쓰는 명령으로, 열 추가와 목적이 다르다. '새로운 데이터 추가'와 '새로운 열 정의 추가'를 구별하면 혼동을 피할 수 있다.

NoteBase

문 14. RAID(Redundant Arrays of Inexpensive Disks)에 대한 설명으로 가장 적절하지 않은 것은?

① 레벨 0는 장애 복구 능력이 없다.

② 레벨 1은 미러링(mirroring) 방식이다.

③ 레벨 2는 Reed-Solomon 에러 정정 코드를 사용한다.

④ 레벨 5는 모든 디스크에 패리티 정보를 나누어 저장한다.

정답 3번

간단한 해설

RAID 2는 비트 수준 스트라이핑에 **해밍 코드 기반 오류 정정**을 사용하는 방식이다. Reed-Solomon 코드로 설명한 부분이 부적절하다.

선지별 해설

① O: RAID 0은 데이터를 여러 디스크에 나누어 저장하는 스트라이핑을 사용하지만, 미러 사본이나 패리티 같은 **복구용 중복 정보를 두지 않는다**. 따라서 RAID 구성 자체로 디스크 고장에 따른 손실을 재구성할 수 없다. 별도 백업에서 복구할 가능성까지 부정하는 말은 아니다.

② O: RAID 1은 같은 데이터를 두 디스크 등에 중복 저장하는 **미러링** 방식이다. 한 복제본을 읽을 수 없을 때 정상 복제본을 사용할 수 있도록 한다.

③ X: RAID 2는 비트 수준으로 데이터를 분산하고 **해밍 코드(Hamming code)** 기반 오류 정정 정보를 저장하는 방식이다. Reed-Solomon 코드를 RAID 2의 오류 정정 코드로 제시한 부분을 해밍 코드로 바로잡아야 한다.

④ O: RAID 5는 블록 수준 스트라이핑과 **분산 패리티**를 사용한다. 패리티를 전용 디스크 하나에 몰아넣지 않고 스트라이프마다 여러 디스크에 분산 배치하여, 한 디스크 고장 시 남은 데이터와 패리티로 재구성할 수 있다.

NoteBase

문 15. 병행제어(concurrency control)를 위해 사용하는 로킹(locking) 기법에 대한 설명으로 가장 적절한 것은?

① 2단계 로킹기법에서 만약 로크 전환(lock conversion)을 허용한다면 로크 상승(공유 로크 상태에서 배타 로크로 변경)은 반드시 확장단계(growing phase) 동안에 수행되어야 한다.

② 특정 데이터 항목에 대해 한 트랜잭션이 배타 로크(exclusive lock)를 보유하고 있으면 다른 트랜잭션에서는 공유 로크(shared lock)만 보유할 수 있다.

③ 특정 스케줄에 참여하고 있는 모든 트랜잭션이 기본적 2단계 로킹기법(basic 2-Phase Locking)을 준수한다면 교착상태(deadlock)가 발생하지 않는다.

④ 트랜잭션이 특정 데이터 항목에 대하여 읽기 연산을 수행하기 위해서는 반드시 배타 로크를 보유하여야 한다.

정답 1번

간단한 해설

공유 로크를 배타 로크로 바꾸는 상승은 권한을 강화하므로 **2단계 로킹의 확장 단계**에서만 허용된다. 기본 2단계 로킹 자체가 교착상태까지 막는 것은 아니다.

선지별 해설

① O: 2단계 로킹의 확장 단계에서는 로크를 획득하거나 강하게 만들 수 있고, 축소 단계에서는 해제하거나 약하게 만들 수 있다. 공유 로크에서 배타 로크로의 **상승은 권한을 강화하는 동작**이므로 확장 단계에서 수행해야 한다.

② X: 기본적인 공유·배타 로크 호환 규칙에서 한 트랜잭션의 배타 로크는 다른 트랜잭션의 **공유 로크와 배타 로크 모두**와 충돌한다. 공유 로크끼리는 함께 보유할 수 있지만, 배타 로크 보유 중 다른 트랜잭션에 공유 로크를 허용하지 않는다.

③ X: 기본 2단계 로킹은 충돌 직렬 가능성을 보장하지만 **교착상태를 방지하지는 않는다**. 예를 들어 T1이 A를 잠근 채 B를 기다리고 T2가 B를 잠근 채 A를 기다리면, 둘 다 아직 확장 단계이면서도 교착상태가 생길 수 있다.

④ X: 공유·배타 로킹 모델에서 읽기에는 **공유 로크로 충분**하다. 배타 로크를 보유한 경우에도 읽을 수 있지만, 읽기 연산을 위해 반드시 배타 로크를 가져야 하는 것은 아니다.

따라서 로크 상승을 확장 단계로 제한한 ①이 가장 적절하다.

NoteBase

문 16. 로그를 이용한 데이터베이스 회복(recovery) 기법과 DBMS 캐시(cache) 관리 기법에 대한 설명으로 가장 적절하지 않은 것은?

① WAL(Write-Ahead Logging) 기법은 디스크에 데이터를 기록하기 전에 로그 레코드가 먼저 디스크에 기록되는 것을 보장한다.

② steal 방식의 캐시 관리 규칙에서는 트랜잭션이 완료(commit)되기 전까지는 그 트랜잭션이 갱신한 캐시 버퍼 페이지가 디스크에 기록될 수 없다.

③ force 방식의 캐시 관리 규칙에서는 트랜잭션이 갱신한 모든 페이지가 트랜잭션이 완료(commit)되기 전에 즉시 디스크에 반영된다.

④ no-steal 규칙을 사용하면 회복 처리 중 UNDO 연산이 필요 없고, force 규칙을 사용하면 회복 처리 중 REDO 연산이 필요 없다.

정답 2번

간단한 해설

steal/no-steal은 미완료 변경의 디스크 반영 허용 여부를, force/no-force는 커밋까지 변경 페이지를 디스크에 강제로 반영하는지를 구분한다. 미완료 변경의 기록을 금지하는 것은 steal이 아니라 no-steal이다.

선지별 해설

① O: WAL은 변경된 데이터 페이지를 디스크에 쓰기 전에 **그 변경에 대응하는 로그를 먼저 영구 저장**하도록 한다. 데이터가 먼저 반영되고 회복에 필요한 로그가 사라지는 상황을 방지하기 위한 순서 규칙이다.

② X: **steal은 미완료 트랜잭션이 변경한 페이지도 디스크에 기록할 수 있는 정책**이다. 이를 금지하는 정책은 no-steal이다. steal에서는 나중에 해당 트랜잭션이 취소되거나 장애로 중단되면, 이미 반영된 변경을 UNDO로 되돌릴 필요가 있다.

③ O: force는 트랜잭션의 커밋이 완료되기까지 해당 트랜잭션이 갱신한 모든 페이지를 디스크에 반영하도록 하는 정책이다. 여기서 즉시 반영한다는 표현은 커밋 시점의 강제 반영을 뜻하며, 개별 갱신 연산이 일어날 때마다 반드시 곧바로 디스크에 써야 한다는 뜻은 아니다.

④ O: 데이터 디스크가 보존되는 일반적인 시스템 장애 회복을 기준으로, no-steal에서는 미완료 변경이 디스크에 남지 않으므로 UNDO가 필요 없다. force에서는 완료된 변경이 이미 디스크에 반영되어 있으므로 REDO가 필요 없다. 미완료 변경의 반영 여부는 UNDO, 완료된 변경의 반영 보장은 REDO 필요성과 연결된다.

NoteBase

문 17. 트랜잭션 T1과 T2가 다음과 같은 스케줄로 실행될 때 발생할 수 있는 문제점으로 가장 적절한 것은?

시간 ↓

T1	T2
read x; x = x + 20; write x;	

	read x;
	x = x * x;
	write x;
	read y;
	y = y * y;
	write y;

read y; y = y + 20; write y;	

① 부정 판독(dirty read)

② 모순성(inconsistency)

③ 연쇄 복귀(cascade rollback)

④ 반복할 수 없는 읽기(unrepeatable read)

정답 2번

간단한 해설

x는 T1→T2, y는 T2→T1 순서로 갱신된다. 이처럼 데이터별 처리 순서가 달라져 하나의 직렬 실행 결과와 일치하지 않을 수 있는 **모순성**이 핵심이다.

NoteBase

핵심 풀이

스케줄 추적: x에는 T1의 20 더하기가 먼저 적용되고 T2의 제곱이 나중에 적용된다. 반면 y에는 T2의 제곱이 먼저 적용되고 T1의 20 더하기가 나중에 적용된다. 즉, **x와 y에서 두 트랜잭션의 처리 순서가 서로 뒤바뀐다.**

결과 비교: 초깃값을 x_0, y_0라 하면 그림의 최종 값은 $x = (x_0 + 20)^2,\ y = y_0^2 + 20$이다. T1→T2의 직렬 실행이라면 y도 $(y_0 + 20)^2$가 되어야 하고, T2→T1의 직렬 실행이라면 x도 $x_0^2 + 20$이 되어야 한다. 따라서 그림은 일반적으로 어느 한 직렬 실행 순서의 결과와도 일치하지 않는 **모순성**을 일으킬 수 있다.

개념 구별: 부정 판독은 다른 트랜잭션이 아직 커밋하지 않은 값을 읽는 문제이다. 그림에는 커밋 시점이 없으므로 그 가능성 자체를 배제할 수는 없지만, 여기서 드러나는 핵심은 두 데이터에 적용되는 순서의 불일치이다. 연쇄 복귀는 한 트랜잭션의 취소가 의존하던 다른 트랜잭션의 취소로 이어지는 현상인데 취소 과정은 제시되지 않았다. 반복할 수 없는 읽기는 같은 데이터를 다시 읽을 때 값이 달라지는 현상인데 그림에는 같은 항목의 반복 읽기가 없다.

따라서 스케줄의 상반된 갱신 순서가 직접 보여 주는 문제는 ② **모순성**이다.

NoteBase

문 18. 데이터베이스에 대한 SQL 주입(SQL Injection) 공격유형으로 가장 적절하지 않은 것은?

① 코버트 채널(covert channels)

② 함수 호출 주입

③ SQL 조작

④ 코드 주입

정답 1번

간단한 해설

SQL 주입 유형은 입력을 실행 구문에 끼워 넣는 방식으로 구별한다. **코버트 채널은 은밀한 정보 전달 통로**에 관한 개념이므로 제시된 주입 방식들과 분류 기준이 다르다.

개념 구분

SQL 주입은 입력값이 단순 데이터로 취급되지 않고 SQL 구문의 일부로 해석되어 원래 질의의 의미나 실행 동작이 바뀌는 공격이다. SQL 조작은 질의의 조건·구조를 바꾸는 방식, 코드 주입은 의도하지 않은 실행 코드를 끼워 넣는 방식, 함수 호출 주입은 데이터베이스 함수 호출을 삽입하는 방식으로 구분할 수 있다. 공통점은 **입력이 실행 구문의 일부가 되는 것**이다.

코버트 채널은 원래 허용된 통신 경로가 아닌 저장 상태나 시간 변화 등을 이용해 정보를 은밀하게 전달하는 통로이다. 이는 **정보 전달 경로에 관한 개념**으로, 이 문제의 SQL 구문 주입 방식 분류와 다르므로 ①이 해당한다. SQL 주입 공격에서 시간차 등을 정보 추출에 활용할 수 있다는 사실과, 코버트 채널 자체를 해당 주입 유형으로 분류하는 것은 구별한다.

NoteBase

문 19. 하둡(Hadoop) 에코시스템에 대한 설명으로 적절한 것을 모두 고른 것은?

㉠ 피그(Pig)는 맵리듀스 위에 SQL 인터페이스를 제공한다.
㉡ 우지(Oozie)는 잡(job)의 작업 흐름을 예약하고 실행하기 위한 서비스이다.
㉢ 스쿱(sqoop)은 관계형 데이터베이스와 HDFS(Hadoop Distributed File System) 사이에 효율적으로 데이터를 이동시키기 위한 라이브러리와 실행 환경이다.
㉣ HBase는 HDFS를 기반으로 하는 컬럼 기반 키-값 저장소이다.

① ㉠㉡㉢
② ㉠㉡㉣
③ ㉠㉢㉣
④ ㉡㉢㉣

정답 4번

간단한 해설

Pig는 SQL이 아니라 **Pig Latin으로 데이터 처리 흐름을 표현**한다. Oozie는 작업 예약·실행, Sqoop은 관계형 데이터베이스와 Hadoop 간 전송, HBase는 컬럼 계열 키-값 저장을 담당한다.

보기별 해설

㉠ X: Pig는 SQL 인터페이스가 아니라 **Pig Latin이라는 데이터 흐름 언어**로 분석 과정을 작성하는 플랫폼이다. Hadoop에서 SQL 계열 질의 인터페이스를 제공하는 대표적인 도구는 Hive이며, Pig와 구별된다.

㉡ O: Oozie는 Hadoop 작업들의 의존 관계와 실행 순서를 워크플로로 관리하고, 정해진 시간이나 데이터 조건에 따라 작업을 예약·실행하는 서비스이다. **작업 흐름의 조정과 스케줄링**이 중심 역할이다.

㉢ O: Sqoop은 관계형 데이터베이스와 Hadoop 사이에서 대량 데이터를 가져오거나 내보내는 도구이다. 데이터베이스의 테이블을 HDFS로 가져오거나 HDFS 데이터를 관계형 데이터베이스로 내보내는 **양방향 데이터 전송**을 지원한다.

㉣ O: HBase는 Hadoop 환경에서 대규모 데이터를 저장하는 분산 데이터베이스로, 일반적인 분산 구성에서 HDFS를 기반으로 한다. 행 키와 컬럼 패밀리·컬럼 등의 식별자로 값에 접근하는 **와이드 컬럼 계열 키-값 저장 구조**이며, 관계형 SQL 테이블이나 단순 파일 저장소와 구별된다.

따라서 적절한 설명은 ㉡·㉢·㉣이며, 해당 조합은 ④이다.

NoteBase

문 20. NoSQL 시스템의 종류로 가장 적절하지 않은 것은?

① 키-값(key-value) NoSQL 시스템

② 문서 기반(document-based) NoSQL 시스템

③ 테이블 기반(table-based) NoSQL 시스템

④ 그래프 기반(graph-based) NoSQL 시스템

정답 3번

간단한 해설

NoSQL의 대표 유형은 **키-값·문서·와이드 컬럼·그래프**이다. “테이블 기반”은 이 대표 분류의 유형명과 맞지 않지만, NoSQL이 테이블을 전혀 사용하지 않는다는 뜻은 아니다.

개념 구분

NoSQL의 대표적인 데이터 모델 분류는 **키-값, 문서, 와이드 컬럼(컬럼 패밀리), 그래프**이다. 키-값 모델은 키로 값을 찾고, 문서 모델은 문서 단위로 데이터를 저장하며, 그래프 모델은 개체와 관계를 중심으로 데이터를 표현한다.

문제는 이러한 대표 유형의 명칭을 묻고 있다. 이 분류에서 “테이블 기반”은 와이드 컬럼과 나란히 별도의 대표 유형으로 통상 열거되는 명칭이 아니다. 따라서 제시된 유형명 중 가장 적절하지 않은 것을 골라야 한다.

다만 이를 **NoSQL에는 테이블이 없다는 뜻으로 이해하면 안 된다**. Cassandra처럼 테이블·행·열이라는 구조와 용어를 사용하는 NoSQL도 있다. 테이블 사용 여부 자체가 관계형 데이터베이스와 NoSQL을 가르는 충분한 기준은 아니며, 이 문항의 판단은 대표 분류 명칭에 한정된다.

이러한 대표 유형 분류를 기준으로 가장 적절하지 않은 것은 ③ 테이블 기반 NoSQL 시스템이다.

NoteBase