↑ 목록으로 ← 이전 단원

2단원. **SQL** 실습

> 대상: 전공자 · 정보처리기사 취득자 | 목적: 인프라 직무(네트워크·클라우드·시스템·보안) 기술 면접 대비 중요도: ★★★ 최빈출 · ★★ 자주 출제 · ★ 기본 개념

Q1. INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN의 차이를 설명하세요. ★★★

답안 **INNER JOIN**은 두 테이블에서 조인 조건을 만족하는 행만 결과에 포함시킵니다. **LEFT JOIN**은 왼쪽 테이블의 모든 행을 포함하고, 오른쪽 테이블에 매칭되는 값이 없으면 해당 컬럼을 NULL로 채웁니다. **RIGHT JOIN**은 반대로 오른쪽 테이블의 모든 행을 기준으로 하며, **FULL OUTER JOIN**은 양쪽 테이블의 모든 행을 포함하고 매칭되지 않는 부분은 NULL로 채웁니다. 예를 들어 고객 테이블과 주문 테이블을 LEFT JOIN하면 주문 이력이 없는 고객도 결과에 포함되지만, INNER JOIN을 사용하면 주문한 적 있는 고객만 조회됩니다. MySQL은 FULL OUTER JOIN을 직접 지원하지 않아 UNION으로 LEFT JOIN과 RIGHT JOIN 결과를 합쳐 구현하기도 합니다.

관련 개념 카티전 곱, ON절/WHERE절, NULL 처리, 셀프 조인, 조인 순서와 옵티마이저

Q2. 서브쿼리(**Subquery**)란 무엇이며 종류를 설명하세요. ★★★

답안 서브쿼리는 하나의 SQL 문 안에 포함된 또 다른 SELECT 문으로, 메인 쿼리의 조건절이나 SELECT절, FROM절 등에 사용될 수 있습니다. 스칼라 서브쿼리는 단일 값을 반환해 비교 연산에 사용되고, 다중행 서브쿼리는 여러 값을 반환해 IN, ANY,

ALL 연산자와 함께 사용됩니다. 상관 서브쿼리(**Correlated Subquery**)는 외부 쿼리의 각 행마다 참조되어 재실행되므로 일반 서브쿼리보다 성능이 떨어질 수 있어 실무에서는 JOIN으로 대체 가능한지 검토합니다. FROM절에 사용되는 서브쿼리는 인라인 뷰라고 부르며, 결과를 임시 테이블처럼 활용할 수 있습니다.

관련 개념 인라인 뷰, 상관 서브쿼리, EXISTS, ANY/ALL, 서브쿼리 대 조인 성능

> **Q3. GROUP BY**와 **HAVING**의 역할과 **WHERE**와의 차이를 설명하세요. ★★★

답안 **GROUP BY**는 지정한 컬럼 값을 기준으로 행들을 그룹화하여 COUNT, SUM, AVG 등의 집계 함수를 적용할 수 있게 해줍니다. **HAVING**은 그룹화된 결과에 대해 조건을 걸어 필터링하는 절로, 집계 함수의 결과를 조건으로 사용할 수 있다는 점이 특징입니다. 반면 **WHERE**는 그룹화가 일어나기 전 개별 행 단위로 필터링을 수행하며 집계 함수를 조건에 사용할 수 없습니다. 예를 들어 "부서별 평균 급여가 300만 원 이상인 부서"를 조회하려면 WHERE가 아닌 HAVING AVG(salary) >= 3000000 형태로 작성해야 하며, 실행 순서상 WHERE가 먼저 적용되고 그 다음 GROUP BY, 마지막으로 HAVING이 적용됩니다.

관련 개념 집계 함수, SQL 실행 순서, ROLLUP/CUBE, 그룹화 카디널리티, DISTINCT

> **Q4.** 트랜잭션 격리 수준(**Isolation Level**) 네 가지를 설명하고 각각 어떤 문제를 방지하는지 설명하세요. ★★★

답안 격리 수준은 낮은 순서대로 **READ UNCOMMITTED**, **READ COMMITTED**, **REPEATABLE READ**, **SERIALIZABLE**이 있습니다. READ UNCOMMITTED는 커밋되지 않은 데이터도 읽을 수 있어 더티 리드(**Dirty Read**)가 발생할 수 있습니다. READ COMMITTED는 커밋된 데이터만 읽도록 하여 더티 리드는 방지하지만, 같은 트랜잭션 내에서 동일 쿼리를 두 번 실행했을 때 결과가 달라지는 反복 불가능한 읽기(**Non-repeatable Read**)가 발생할 수 있습니다.

REPEATABLE READ는 트랜잭션이 시작된 시점의 스냅샷을 유지해 반복 읽기 문제를 방지하지만, 새로운 행이 추가되어 나타나는 팬텀 리드(**Phantom Read**)는 완전히 막지 못할 수 있습니다(MySQL InnoDB는 갭 락으로 대부분 방지). SERIALIZABLE은 트랜잭션을 순차적으로 실행한 것과 동일한 결과를 보장하는 가장 엄격한 수준이지만 동시성이 크게 저하됩니다.

관련 개념 더티 리드, Non-repeatable Read, 팬텀 리드, MVCC, 갭 락

> **Q5.** 락(**Lock**)의 종류(공유 락, 배타 락)와 데드락(**Deadlock**)에 대해 설명하세요. ★★★

답안 공유 락(**Shared Lock, S-Lock**)은 데이터를 읽을 때 걸리는 락으로 다른 트랜잭션도 동시에 공유 락을 획득해 읽을 수 있지만 쓰기는 불가능합니다. 배타 락 **(Exclusive Lock, X-Lock)**은 데이터를 수정할 때 걸리는 락으로, 해당 락이 걸린 자원에는 다른 트랜잭션이 공유 락이든 배타 락이든 획득할 수 없습니다. 데드락은 두 개 이상의 트랜잭션이 서로 상대방이 가진 락을 기다리며 무한 대기하는 상태로, 예를 들어 트랜잭션 A가 자원 1을 잠그고 자원 2를 기다리는 동시에 트랜잭션 B가 자원 2를 잠그고 자원 1을 기다리면 발생합니다. DBMS는 대기 그래프(Wait-for Graph)를 통해 데드락을 탐지하면 둘 중 하나의 트랜잭션을 강제로 롤백시켜 해결하며, 애플리케이션 차원에서는 자원 접근 순서를 일관되게 유지해 예방할 수 있습니다.

관련 개념 2단계 락킹(2PL), 데드락 탐지/예방, 타임아웃, 낙관적/비관적 락, 락 에스컬레이션

> **Q6.** 인덱스가 오히려 성능을 저하시킬 수 있는 경우를 설명하세요. ★★

답안 인덱스는 조회 성능을 높이지만 만능은 아닙니다. 카디널리티가 낮은 컬럼(예: 성별처럼 값 종류가 적은 컬럼)에 인덱스를 걸면 옵티마이저가 인덱스를 타지 않고 풀 스캔을 선택할 수 있어 효과가 미미합니다. 또한 인덱스 컬럼에 함수나 연산을 적

용하면(예: WHERE YEAR(date) = 2024) 인덱스를 활용하지 못하는 경우가 많습니다. 데이터의 INSERT/UPDATE/DELETE가 빈번한 테이블에 인덱스가 많으면 매 쓰기 연산마다 인덱스 구조도 갱신해야 하므로 오히려 쓰기 성능이 크게 저하될 수 있습니다. 따라서 인덱스는 조회 패턴을 분석해 실제로 자주 사용되는 컬럼에만 선택적으로 생성하는 것이 중요합니다.

관련 개념 카디널리티, 함수 기반 인덱스, 옵티마이저, 실행 계획(EXPLAIN), 인덱스 유지 비용

> **Q7.** 실행 계획(**Execution Plan**)을 확인하는 이유와 주요 확인 항목을 설명하세요. ★★

답안 실행 계획은 DBMS의 옵티마이저가 쿼리를 어떤 방식으로 처리할지(테이블 접근 방법, 조인 순서, 사용 인덱스 등) 보여주는 정보로, MySQL에서는 `EXPLAIN`, Oracle에서는 `EXPLAIN PLAN`으로 확인합니다. 이를 통해 쿼리가 인덱스를 제대로 사용하는지, 풀 테이블 스캔이 발생하는지, 예상 처리 행 수(rows)가 얼마나 되는지를 사전에 파악할 수 있습니다. 주요 확인 항목으로는 접근 방식(type: const, ref, range, ALL 등), 사용된 키(key), 예상 스캔 행 수(rows), 추가 정보(Extra: Using filesort, Using temporary 등)가 있습니다. 특히 "Using filesort"나 "Using temporary"가 나타나면 정렬이나 임시 테이블 생성으로 성능 저하가 발생할 수 있어 인덱스나 쿼리 구조 개선을 검토해야 합니다.

관련 개념 옵티마이저, 접근 방식(type), Using filesort, 통계 정보, 쿼리 튜닝

> **Q8. UNION**과 **UNION ALL**의 차이를 설명하세요. ★★

답안 **UNION**은 두 개 이상의 SELECT 결과를 합칠 때 중복된 행을 제거하여 반환하는 반면, **UNION ALL**은 중복 제거 없이 모든 결과를 그대로 합칩니다. 중복 제거 과정에는 정렬이나 해시 연산이 필요하기 때문에 UNION은 UNION ALL보다 일반적으로 성능이 느립니다. 따라서 두 결과 집합에 중복이 없다는 것이 보장되거나 중복 제거가 필요 없는 경우에는 UNION ALL을 사용하는 것이 성능상 유리합니다. 또한

UNION으로 결합하려는 각 SELECT 문은 컬럼의 개수와 데이터 타입이 서로 호환되어야 합니다.

관련 개념 중복 제거, 정렬/해시 연산, 컬럼 호환성, 성능 최적화, 집합 연산자 (INTERSECT, MINUS)

> **Q9. DDL, DML, DCL, TCL**을 예시와 함께 구분하여 설명하세요. ★★

답안 **DDL(Data Definition Language)**은 테이블이나 스키마 구조를 정의하는 명령어로 CREATE, ALTER, DROP, TRUNCATE가 있으며 실행 시 자동 커밋되는 경우가 많습니다. **DML(Data Manipulation Language)**은 데이터를 조작하는 명령어로 SELECT, INSERT, UPDATE, DELETE가 해당하며, 트랜잭션 제어의 대상이 됩니다. **DCL(Data Control Language)**은 권한을 부여하거나 회수하는 명령어로 GRANT와 REVOKE가 있습니다. **TCL(Transaction Control Language)**은 트랜잭션을 제어하는 명령어로 COMMIT, ROLLBACK, SAVEPOINT가 있으며, 이를 통해 여러 DML 작업을 하나의 논리적 단위로 묶어 처리할 수 있습니다.

관련 개념 트랜잭션 커밋/롤백, 권한 관리, 스키마 변경, SAVEPOINT, 암시적 커밋

> **Q10. WHERE**절과 **JOIN** 조건에 인덱스를 효과적으로 활용하기 위한 쿼리 작성 팁을 설명하세요. ★★

답안 인덱스를 효과적으로 활용하려면 우선 인덱스가 걸린 컬럼을 가공하지 않고 그대로 조건절에 사용해야 합니다. 예를 들어 `WHERE SUBSTR(name, 1, 3) = 'ABC'`처럼 함수를 적용하면 인덱스를 타지 못하므로 `WHERE name LIKE 'ABC%'`처럼 앞부분 일치 검색으로 바꾸는 것이 좋습니다. 복합 인덱스를 사용할 때는 인덱스에 정의된 컬럼 순서와 조건절의 순서가 일치해야 효율적으로 활용되며, 이를 선행 컬럼 원칙이라고 합니다. 또한 JOIN을 사용할 때는 조인 조건에 사용되는 컬럼에도 인덱스를 생성해두면 조인 연산 속도가 크게 향상되며, 데이터 타입이 다

른 컬럼끼리 비교하면 암묵적 형변환이 발생해 인덱스를 타지 못할 수 있으므로 주의해야 합니다.

관련 개념 복합 인덱스, 선행 컬럼, 암묵적 형변환, LIKE 검색 패턴, 커버링 인덱스

> **Q11.** 정렬 함수인 **ROW_NUMBER(), RANK(), DENSE_RANK()**의 차이를 설명하세요. ★

답안 세 함수 모두 윈도우 함수로 OVER절과 함께 정렬 순위를 매기지만 동점 처리 방식이 다릅니다. **ROW_NUMBER()**는 값이 같더라도 무조건 고유한 순번을 1, 2, 3, 4처럼 순차적으로 부여합니다. **RANK()**는 동점인 행에 같은 순위를 부여하되 다음 순위는 동점 개수만큼 건너뛰어(예: 1, 2, 2, 4) 부여합니다. **DENSE_RANK()**는 동점에 같은 순위를 부여하지만 다음 순위를 건너뛰지 않고 연속적으로(예: 1, 2, 2, 3) 부여합니다. 예를 들어 부서별 급여 순위를 매길 때 동점자를 어떻게 처리할지에 따라 적절한 함수를 선택해야 합니다.

관련 개념 윈도우 함수, PARTITION BY, OVER절, 동점 처리, TOP-N 쿼리

> **Q12.** 옵티마이저(**Optimizer**)의 역할과 통계 정보가 쿼리 성능에 미치는 영향을 설명하세요. ★

답안 옵티마이저는 SQL 문을 실행하기에 가장 효율적인 실행 계획을 자동으로 선택해주는 DBMS의 핵심 구성 요소로, 대부분의 현대 DBMS는 비용 기반 옵티마이저**(CBO)**를 사용합니다. CBO는 테이블의 행 수, 데이터 분포, 인덱스 유무 등의 통계 정보를 바탕으로 여러 실행 경로의 예상 비용(디스크 I/O, CPU 사용량)을 계산해 가장 저렴한 계획을 선택합니다. 만약 통계 정보가 오래되어 실제 데이터 분포와 차이가 크면 옵티마이저가 비효율적인 실행 계획(예: 인덱스가 있음에도 풀 스캔 선택)을 세울 수 있으므로, 대량의 데이터 변경 후에는 통계 정보를 갱신(ANALYZE TABLE 등)해주는 것이 중요합니다. 이는 실무에서 갑자기 쿼리가 느려지는 원인 중 하나로 자주 언급됩니다.

관련 개념 비용 기반 옵티마이저(CBO), 통계 정보 갱신, 실행 계획 캐시, 히스토그램, 인덱스 선택도