신입 개발자 기술면접 질문 : DB

Q&A 데이터베이스의 특징에 대해 설명해주세요.
데이터베이스는 조직화된 데이터의 집합이며, 다음과 같은 특징을 가집니다:

- 구조화된 데이터 저장: 데이터베이스는 데이터를 테이블, 필드, 레코드 등으로 구조화하여 저장합니다. 이를 통해 데이터 검색, 수정, 관리가 용이해집니다.
- 데이터 무결성: 데이터베이스는 무결성 규칙을 통해 데이터의 정확성과 일관성을 유지합니다. 예를 들어, 기본 키와 외래 키 제약 조건을 사용하여 데이터 관계를 정의하고 유지합니다.
- 데이터 보안: 사용자 권한 부여 및 액세스 제어를 통해 데이터 보안을 유지합니다. 민감한 데이터에 대한 액세스는 특정 사용자나 그룹에게만 허용될 수 있습니다.
- 동시성 제어: 여러 사용자가 동시에 데이터베이스에 액세스할 때 발생할 수 있는 충돌을 관리합니다. 이를 위해 트랜잭션 관리와 록(lock) 기법을 사용합니다.
- 백업 및 복구: 데이터베이스는 데이터 손실이 발생할 경우를 대비하여 백업 및 복구 기능을 제공합니다. 이를 통해 시스템 장애나 데이터 손상 후에도 데이터를 복구할 수 있습니다.
- 확장성과 성능: 데이터베이스는 대량의 데이터를 효율적으로 처리할 수 있도록 설계되었습니다. 인덱싱, 쿼리 최적화 등의 기능을 통해 대용량 데이터에서도 높은 성능을 제공합니다.
- 다양한 쿼리 언어 지원: SQL(Structured Query Language)과 같은 쿼리 언어를 통해 사용자는 복잡한 데이터 조작과 질의를 수행할 수 있습니다.

이러한 특징들은 데이터베이스를 효과적인 데이터 관리 도구로 만들어 다양한 분야에서 활용됩니다.

Q&A 데이터베이스 언어(DDL, DML, DCL)에 대해 설명해주세요.
데이터베이스 언어는 데이터베이스 관리 시스템(DBMS)에서 데이터와 데이터 구조를 정의, 조작, 제어하는 데 사용되는 언어입니다. 주로 다음 세 가지 유형으로 분류됩니다:

데이터 정의 언어(DDL, Data Definition Language):
1) DDL은 데이터베이스 스키마와 구조를 정의하는 데 사용됩니다.
주요 명령어에는 CREATE (새로운 테이블, 뷰, 인덱스 등의 데이터베이스 객체를 생성), ALTER (기존 데이터베이스 객체의 구조를 변경), DROP (데이터베이스 객체를 삭제) 등이 있습니다.
DDL 명령은 데이터베이스의 메타데이터를 변경하지만, 실제 데이터에는 직접적인 영향을 주지 않습니다.
데이터 조작 언어(DML, Data Manipulation Language):

2) DML은 데이터를 실제로 조작하고 처리하는 데 사용됩니다.
주요 명령어에는 SELECT (데이터 조회), INSERT (새로운 데이터 삽입), UPDATE (기존 데이터 수정), DELETE (데이터 삭제) 등이 있습니다.

DML을 통해 사용자는 데이터베이스 내의 데이터를 조회하고, 변경할 수 있습니다.
데이터 제어 언어(DCL, Data Control Language):

3) DCL은 데이터베이스에서 데이터 접근 및 권한을 제어하는 데 사용됩니다.
주요 명령어에는 GRANT (사용자나 사용자 그룹에 데이터 접근 권한 부여), REVOKE (기존에 부여된 권한을 제거) 등이 있습니다.
DCL을 통해 데이터베이스 관리자는 사용자의 데이터 접근 권한을 관리할 수 있습니다.
이러한 언어들은 데이터베이스 관리 시스템에서 중요한 역할을 하며, 각각의 용도에 맞게 설계되어 데이터베이스의 효율적인 관리를 가능하게 합니다.

Q&A SELECT 쿼리의 수행 순서를 알려주세요.
FROM, ON, JOIN > WHERE, GROUP BY, HAVING > SELECT > DISTINCT > ORDER BY > LIMIT
SELECT 쿼리의 수행 순서는 사용자가 쿼리를 작성하는 순서와는 다릅니다. 데이터베이스 시스템이 쿼리를 처리하는 내부적인 순서는 다음과 같습니다:

1) FROM 절: 데이터베이스에서 해당 테이블이나 뷰를 찾아 메모리에 로드합니다. 이 단계에서는 조인이 있을 경우 조인 작업도 수행됩니다.
2) WHERE 절: FROM 절에서 가져온 데이터에 대해 조건을 적용하여 필터링합니다. 이때, 지정된 조건에 부합하지 않는 레코드는 제외됩니다.
3) GROUP BY 절: 필터링된 결과에 대해 지정된 컬럼을 기준으로 데이터를 그룹화합니다. 집계 함수(예: SUM, AVG, COUNT 등)와 함께 사용될 때, 이 단계는 각 그룹에 대한 집계 값을 계산하는 데 필요합니다.
4) HAVING 절: GROUP BY 절에 의해 생성된 각 그룹에 대해 조건을 적용합니다. HAVING 절은 그룹화된 결과에 대한 필터 역할을 하며, 집계 함수와 함께 사용됩니다.
SELECT 절: FROM, WHERE, GROUP BY, HAVING 절을 거쳐 얻어진 결과에서 특정 컬럼이나 표현식의 데이터를 선택합니다. 이 단계에서는 실제로 출력될 컬럼이나 값을 결정합니다.
5) ORDER BY 절: 최종적으로 선택된 데이터를 지정된 컬럼을 기준으로 정렬합니다. 이때, ASC (오름차순) 또는 DESC (내림차순)를 지정할 수 있습니다.
6) LIMIT 절 (해당되는 경우): 쿼리 결과의 상위 N개의 레코드만 반환하도록 제한합니다. 주로 큰 데이터셋에서 일부 데이터만 살펴보고 싶을 때 사용됩니다.

이러한 순서는 데이터베이스의 쿼리 최적화와 처리 효율성을 위해 내부적으로 결정되며, 사용자는 이러한 순서를 이해함으로써 보다 효과적인 쿼리를 작성할 수 있습니다.

Q&A 트리거(Trigger)에 대해 간략하게 설명해보세요.
트리거는 특정 테이블에 대한 이벤트에 반응해 INSERT, DELETE, UPDATE 같은 DML 문이 수행되었을 때, 데이터베이스에서 자동으로 동작하도록 작성된 프로그램입니다. 사용자가 직접 호출하는 것이 아닌, 데이터베이스에서 자동적으로 호출한다는 것이 가장 큰 특징입니다.
트리거(Trigger)는 데이터베이스 관리 시스템(DBMS)에서 특정 이벤트가 발생했을 때 자동으로 실행되는 프로시저나 SQL 코드입니다. 이러한 이벤트는 데이터베이스 테이블에 대한 INSERT, UPDATE, DELETE와 같은 DML(Data Manipulation Language) 작업일 수 있습니다. 트리거의 주요 목적은 데이터 무결성을 유지하고,

자동화된 데이터 관리 및 비즈니스 룰의 강제 실행을 위한 것입니다.

트리거의 특징은 다음과 같습니다:
- 자동 실행: 트리거는 정의된 이벤트가 발생하면 자동으로 실행됩니다.
- 이벤트 기반: 트리거는 특정 테이블에 대한 INSERT, UPDATE, DELETE 작업과 같은 특정 이벤트에 연결됩니다.
- 사용자 정의 로직 실행: 트리거를 통해 사용자 정의 로직을 데이터베이스 수준에서 실행할 수 있으며, 이를 통해 데이터 무결성을 보장하고 복잡한 비즈니스 규칙을 구현할 수 있습니다.
- 보안 강화: 데이터 변경에 대한 로깅, 감사 또는 검증 작업을 자동화하여 데이터 보안 및 무결성을 강화할 수 있습니다.
트리거 사용 시 주의해야 할 점은 복잡한 로직이나 너무 많은 처리를 트리거 내부에 넣게 되면 시스템 성능에 부정적인 영향을 줄 수 있다는 것입니다. 따라서 트리거는 신중하게 사용하고 필요한 곳에 적절하게 배치하는 것이 중요합니다.

Q&A Index에 대해 설명하고, 장단점은 무엇이라고 생각하시나요?
인덱스(Index)는 데이터베이스에서 데이터 검색 속도를 향상시키기 위해 사용되는 데이터 구조입니다. 테이블의 하나 이상의 열에 대해 생성할 수 있으며, 이 열의 값에 빠르게 액세스할 수 있도록 키-값 쌍으로 데이터를 저장합니다.

*인덱스의 장점
- 검색 속도 향상: 인덱스를 사용하면 테이블을 전체 스캔하는 대신 인덱스를 통해 데이터를 빠르게 찾을 수 있습니다. 이는 검색 쿼리의 실행 시간을 크게 줄여줍니다.
- 정렬된 데이터 액세스: 인덱스는 데이터를 정렬된 상태로 유지하므로, 정렬된 데이터에 대한 쿼리가 훨씬 빠르게 처리됩니다.
- 데이터 무결성: 유니크 인덱스를 사용하여 열의 고유성을 보장할 수 있어 데이터 무결성을 유지하는 데 도움이 됩니다.
*인덱스의 단점
- 공간 사용: 인덱스는 추가적인 디스크 공간을 사용합니다. 인덱스가 많을수록 더 많은 저장 공간이 필요합니다.
- 유지 관리 비용: 데이터가 삽입, 삭제, 수정될 때마다 인덱스도 업데이트되어야 합니다. 이로 인해 DML(Data Manipulation Language) 작업의 성능이 저하될 수 있습니다.
- 성능 저하 가능성: 잘못 설계된 인덱스는 오히려 성능을 저하시킬 수 있습니다. 예를 들어, 데이터 변경이 많은 테이블에 과도한 인덱스는 성능을 떨어뜨릴 수 있습니다.
인덱스는 데이터 검색 성능을 크게 향상시킬 수 있지만, 설계와 사용 방법에 따라 다른 작업에 영향을 줄 수 있습니다. 따라서 데이터 접근 패턴을 분석하고 실제 애플리케이션의 요구 사항에 맞게 인덱스를 적절히 계획하고 구현해야 합니다.

간단한 예제를 보겠습니다. 예를 들어, 사용자 데이터를 저장하는 데이터베이스 테이블이 있다고 가정해 보겠습니다. 이 테이블에는 id, name, email 열이 있습니다. 여기서 email 열에 인덱스를 생성하는 경우를 살펴보겠습니다.
테이블 구조는 다음과 같습니다:

CREATE TABLE users (id INT PRIMARY KEY, name VARCHAR(100), email VARCHAR(100));
이제 email 열에 대한 인덱스를 생성해 보겠습니다:

CREATE INDEX idx_email ON users(email);
이 인덱스 idx_email은 email 열에 대한 검색을 최적화합니다. 사용자의 이메일 주소로 사용자 정보를 조회하는 쿼리가 자주 실행되는 경우, 이 인덱스는 해당 쿼리의 실행 시간을 단축시킵니다.

예를 들어, 다음과 같은 쿼리가 있다고 가정해 봅시다:
SELECT * FROM users WHERE email = 'example@example.com';
email에 인덱스가 있으면 데이터베이스는 전체 users 테이블을 스캔하는 대신 idx_email 인덱스를 사용하여 훨씬 빠르게 해당 이메일 주소를 가진 사용자를 찾을 수 있습니다.

이 예제에서 볼 수 있듯이, 인덱스는 적절히 사용될 때 데이터 검색 속도를 크게 향상시키지만, 인덱스가 존재하는 열에 데이터 변경이 빈번하게 발생한다면, 인덱스의 유지 관리로 인해 추가적인 오버헤드가 발생할 수 있습니다.

Q&A 정규화에 대해 설명해주세요. 제3정규화까지에 대한 간단한 예제도 부탁합니다.
하나의 릴레이션에 하나의 의미만 존재하도록 릴레이션을 분해하는 과정이며, 데이터의 일관성, 최소한의 데이터 중복, 최대한의 데이터 유연성을 위한 방법입니다.
제1 정규형 : 테이블의 컬럼이 원자 값(Atomic Value; 하나의 값)을 갖도록 분해합니다.
제2 정규형: 제1 정규형을 만족하고, 기본키가 아닌 속성이 기본키에 완전 함수 종속이도록 분해합니다.
※ 여기서 완전 함수 종속이란 기본키의 부분집합이 다른 값을 결정하지 않는 것을 의미
제3 정규형 : 제2 정규형을 만족하고, 이행적 함수 종속을 없애도록 분해합니다.
※ 여기서 이행적 종속이란 A → B, B → C가 성립할 때 A → C가 성립되는 것을 의미
BCNF 정규형 : 제3 정규형을 만족하고, 함수 종속성 X → Y가 성립할 때 모든 결정자 X가 후보키가 되도록 분해합니다.

정규화(Normalization)란? 정규화는 데이터베이스 설계 과정에서 데이터 중복을 줄이고, 데이터 무결성을 유지하기 위해 테이블의 구조를 시스템적으로 조직하는 과정입니다. 정규화를 통해 테이블을 여러 개의 관련된 테이블로 분리하여 데이터베이스의 설계를 개선할 수 있습니다. 정규화 과정은 주로 여러 단계의 '정규 형태'(Normal Form)로 나뉘며, 각 단계는 데이터의 논리적 구조를 점점 더 개선합니다.

1차 정규형(1NF) : 1차 정규형은 모든 컬럼이 원자값(더 이상 분해할 수 없는 값)을 가지도록 테이블을 구조화하는 것입니다. 각 컬럼에는 반복되는 그룹이 없어야 합니다.
예를 들어, Students 테이블이 다음과 같은 데이터를 가지고 있다고 합시다.

Q&A 정규화에는 어떤 장단점이 있는지 설명해주세요.
정규화는 데이터베이스 설계 과정에서 중요한 역할을 하며, 여러 장단점을 가지고 있습니다.
***정규화의 장점**
- 데이터 중복 감소: 정규화는 데이터 중복을 최소화하여 저장 공간을 효율적으로 사용할 수 있게 합니다.

- 데이터 무결성 향상: 데이터 중복이 줄어들면, 데이터의 일관성과 정확성이 향상되어 데이터 무결성이 유지됩니다.
- 업데이트 이상(Anomalies) 방지: 정규화를 통해 삽입, 수정, 삭제 이상을 줄여 데이터베이스의 안정성을 높일 수 있습니다.
- 쿼리 성능 개선: 데이터 중복이 줄어들고 구조가 명확해지면, 쿼리 실행이 더 빨라지고 효율적일 수 있습니다.

***정규화의 단점**

- 조인 연산 증가: 정규화를 통해 테이블이 분리되면, 데이터를 검색하기 위해 여러 테이블을 조인해야 할 경우가 많아집니다. 이는 쿼리의 복잡성을 증가시키고, 때로는 성능 저하를 초래할 수 있습니다.
- 구현 복잡성: 테이블이 많아지고 관계가 복잡해지면, 데이터베이스의 설계와 유지 보수가 더 복잡해질 수 있습니다.
- 성능 저하: 많은 조인이 필요한 복잡한 쿼리는 시스템의 성능을 저하시킬 수 있습니다, 특히 대용량 데이터를 처리할 때 더욱 그렇습니다.
- 응답 시간 지연: 더 많은 테이블 조인과 복잡한 쿼리로 인해 때로는 응답 시간이 지연될 수 있습니다.

정규화는 데이터베이스 설계에서 중요한 단계이지만, 실제 환경에서는 성능과 유지 보수의 용이성을 고려하여 적절한 수준의 정규화를 결정해야 합니다. 때로는 조금의 데이터 중복을 허용하면서 성능을 최적화하는 비정규화(denormalization) 과정이 필요할 수도 있습니다.

Q&A 역정규화를 하는 이유에 대해 아는 대로 설명해주세요.

역정규화(Denormalization)는 데이터베이스 설계 과정에서 의도적으로 데이터 중복을 허용하거나 테이블을 통합하여 정규화된 구조를 단순화하는 과정입니다. 역정규화의 주된 목적은 성능 최적화와 쿼리 처리의 효율성 향상입니다. 역정규화를 하는 주요 이유는 다음과 같습니다:

- 성능 향상: 역정규화를 통해 데이터베이스의 쿼리 성능을 향상시킬 수 있습니다. 데이터가 한 테이블에 중복되어 저장되면, 여러 테이블을 조인하는 복잡한 쿼리를 실행할 필요가 줄어들기 때문에 데이터 접근 시간이 단축됩니다.
- 쿼리의 단순화: 테이블을 통합함으로써 쿼리를 단순화시킬 수 있습니다. 이는 개발자가 더 이해하기 쉽고 관리하기 쉬운 쿼리를 작성할 수 있도록 도와줍니다.
- 조인 연산 감소: 정규화된 데이터베이스에서는 종종 많은 조인 연산이 필요한데, 이는 처리 시간을 증가시킬 수 있습니다. 역정규화는 이러한 조인 연산의 수를 줄여 데이터 검색을 더 빠르게 수행할 수 있게 합니다.
- 트랜잭션 처리 성능 개선: 역정규화를 통해 트랜잭션 처리가 더 빨라질 수 있습니다. 데이터가 한 곳에 중복 저장되면, 업데이트가 더 빠르고 쉽게 이루어질 수 있습니다.
- 읽기 성능 최적화: 읽기 작업이 많은 애플리케이션의 경우, 역정규화는 데이터를 더 빠르게 검색할 수 있게 하여 읽기 성능을 향상시킵니다.

역정규화는 특히 대규모 데이터를 처리하는 데이터 웨어하우스와 같은 시스템에서 유용합니다. 그러나 역정규화는 데이터 중복을 증가시켜 저장 공간을 더 많이 사용하게 하고, 데이터 무결성 관리를 복잡하게 만들 수 있으므로, 성능 이점과 관리 복잡성 사이에서 신중한 균형을 찾아야 합니다.

Q&A 이상 현상의 종류에 대해 설명해주세요.

이상 현상(Anomaly)은 데이터베이스에서 데이터 중복으로 인해 발생하는 구조적 문제입니다. 정규화 과정을

제대로 수행하지 않을 경우, 다음과 같은 세 가지 주요 이상 현상이 발생할 수 있습니다:

1. 삽입 이상 (Insertion Anomaly)
삽입 이상은 새로운 데이터를 추가할 때, 원하지 않는 정보까지 입력해야 하는 문제입니다. 관련 정보 없이 특정 정보만 추가하고자 할 때, 그것이 불가능하여 불필요한 데이터를 입력하게 되는 경우입니다.
예를 들어, 학생과 강의 정보를 하나의 테이블에 저장하는 경우, 특정 강의가 아직 개설되지 않았다면, 그 강의를 수강하는 학생에 대한 정보도 입력할 수 없는 상황이 발생할 수 있습니다.

2. 수정 이상 (Update Anomaly)
수정 이상은 데이터 중복으로 인해 발생하는 문제로, 한 부분의 데이터만 변경할 경우 일관성이 깨져서 정보가 모순되는 현상입니다. 하나의 데이터를 여러 곳에서 수정해야 할 때, 모든 위치에서 일관되게 수정하지 않으면 데이터 불일치가 발생할 수 있습니다.
예를 들어, 동일한 주소 정보가 여러 레코드에 중복되어 있는 경우, 주소 변경 시 모든 레코드를 일관되게 갱신하지 않으면 일부 레코드에는 구 주소가 남아있게 됩니다.

3. 삭제 이상 (Deletion Anomaly)
삭제 이상은 특정 정보를 삭제할 때, 관련된 정보까지 함께 삭제되어 버리는 문제입니다. 중요한 정보가 다른 정보와 함께 불필요하게 삭제되어, 데이터 손실을 초래할 수 있습니다.
예를 들어, 학생과 수강 과목이 한 테이블에 저장되어 있을 때, 한 학생이 탈퇴하면 해당 학생이 수강하는 과목의 정보도 함께 사라지는 상황이 발생할 수 있습니다.

이상 현상은 데이터 중복과 부적절한 테이블 설계에서 기인하므로, 정규화를 통해 테이블 구조를 개선하여 이를 방지할 수 있습니다.

SQL Injection이 무엇인지 설명해보세요.
SQL Injection(이하 SQLi)은 보안 취약점을 이용하여 공격자가 악의적인 SQL 코드를 데이터베이스 시스템에 주입(inject)하여 실행하게 하는 공격 기법입니다. 이 공격은 웹 애플리케이션의 입력 필드나 URL 파라미터를 통해 수행될 수 있으며, 데이터베이스의 보안을 우회하여 민감한 정보를 탈취하거나 데이터베이스를 조작하는 데 사용됩니다.

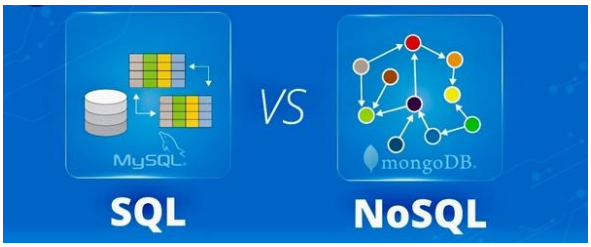
SQLi의 주요 특징 및 위험성
- 데이터 노출: 공격자는 SQLi를 통해 사용자의 개인 정보, 비밀번호, 금융 정보 등 민감한 데이터에 접근할 수 있습니다.
- 데이터 조작: 데이터베이스의 데이터를 무단으로 삽입, 수정, 삭제하는 등의 조작을 할 수 있습니다.
- 인증 우회: 공격자는 SQLi를 사용하여 인증 메커니즘을 우회하고 시스템에 무단으로 접근할 수 있습니다.
- 시스템 공격: 가장 심각한 경우, SQLi를 통해 데이터베이스 서버를 제어하고, 시스템 명령을 실행하여 서버를 완전히 장악할 수도 있습니다.

예제 : 웹 애플리케이션의 로그인 폼에서, 사용자 이름과 비밀번호를 입력받아 다음과 같은 SQL 쿼리를 실행한다고 가정해 봅시다.
SELECT * FROM users WHERE username = '입력된_사용자_이름' AND password = '입력된_비밀번호';

공격자가 사용자 이름 입력 필드에 ' OR '1'='1와 같은 조건을 입력하면, 생성되는 SQL 쿼리는 다음과 같습니다.
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '입력된_비밀번호';
여기서 '1'='1'은 항상 참이 되므로, 이 조건을 만족하는 레코드가 데이터베이스에 있다면 공격자는 인증 과정을 우회할 수 있습니다.

방어 방법
- 입력값 검증: 사용자로부터 받은 입력값에 대해 엄격한 검증을 수행합니다.
- 파라미터화된 쿼리 사용: SQL 쿼리에서 사용자 입력을 파라미터로 처리하여 SQLi를 방지합니다.
- 에러 메시지 관리: 데이터베이스 오류 메시지가 사용자에게 직접 노출되지 않도록 관리합니다.
- 최소 권한 원칙: 데이터베이스 계정에 최소한의 권한만 부여하여 공격 범위를 제한합니다.
SQLi는 매우 심각한 보안 위협이므로, 웹 애플리케이션을 설계하고 개발할 때부터 이러한 공격을 방지하기 위한 조치를 취해야 합니다.

Q&A RDBMS와 NoSQL의 차이에 대해 설명해주세요.



RDBMS(Relational Database Management System, 관계형 데이터베이스 관리 시스템)와 NoSQL(Not Only SQL)은 데이터를 저장, 관리하는 데 사용되는 두 가지 주요한 데이터베이스 유형입니다. 각각의 특징과 차이점은 다음과 같습니다:

RDBMS
- 구조화된 데이터: RDBMS는 엄격한 스키마에 따라 구조화된 데이터를 테이블 형태로 저장합니다. 각 테이블은 열(속성)과 행(레코드)으로 구성되어 있습니다.
- SQL 사용: 데이터를 조회하고 조작하기 위해 SQL(Structured Query Language)을 사용합니다.
- 데이터 무결성과 일관성: RDBMS는 데이터 무결성을 유지하기 위해 엄격한 규칙(예: 외래 키 제약조건)을 적용합니다.
- 트랜잭션 지원: 완전한 ACID(Atomicity, Consistency, Isolation, Durability) 속성을 지원하여, 트랜잭션 처리에 있어 안정성과 신뢰성을 제공합니다.
- 관계형 조인: 테이블 간의 관계를 기반으로 데이터를 조인하여 복잡한 쿼리를 실행할 수 있습니다.

NoSQL
- 비구조화된 데이터: NoSQL 데이터베이스는 스키마가 없거나 유연한 스키마를 가지며, 비구조화된 데이터를 저장하기에 적합합니다.
- 다양한 데이터 모델: 키-값 쌍, 문서, 그래프, 열 기반 등 다양한 데이터 모델을 지원하여, 애플리케이션의 요구사항에 맞게 선택할 수 있습니다.

- 확장성: 수평적 확장성이 뛰어나며, 데이터를 여러 서버에 분산하여 저장하고 처리할 수 있어 대규모 데이터 처리에 유리합니다.
- 유연성: 스키마가 없거나 유연하기 때문에, 데이터 구조가 변경될 경우 쉽게 적응할 수 있습니다.
- 일관성 모델: 일부 NoSQL 시스템은 분산 환경에서 데이터 일관성을 위해 BASE(Basically Available, Soft state, Eventually consistent) 모델을 따릅니다.

차이점

- 데이터 구조와 스키마: RDBMS는 엄격한 스키마 구조를 따르는 반면, NoSQL은 유연한 데이터 구조를 가집니다.
- 확장성: RDBMS는 주로 수직 확장을, NoSQL은 수평 확장을 지원합니다.
- 쿼리 및 데이터 처리: RDBMS는 복잡한 쿼리와 관계형 조인을 지원하는 반면, NoSQL은 대규모 데이터셋에 대한 빠른 접근과 단순한 쿼리에 초점을 맞춥니다.
- 일관성과 트랜잭션: RDBMS는 ACID 속성을 강조하며, NoSQL은 고가용성과 확장성을 위해 BASE 일관성 모델을 채택할 수 있습니다.

각 데이터베이스 유형은 특정 사용 사례와 요구 사항에 따라 장단점을 가지므로, 애플리케이션의 목적과 성능 요구사항을 고려하여 적절한 유형을 선택해야 합니다.

Q&A RDBMS와 NoSQL은 각각 어느 경우에 적합한가요?

RDBMS와 NoSQL은 각기 다른 특성과 장점을 가지고 있어, 사용 사례에 따라 적합한 선택이 달라집니다.

1) RDBMS가 적합한 경우

- 정교한 쿼리와 조인이 필요한 경우: 복잡한 쿼리, 다양한 조인, 서브쿼리 등을 사용하여 상세한 데이터 분석이 필요할 때 RDBMS가 유리합니다.
- 데이터 무결성과 일관성이 중요한 경우: 금융, 의료, 정부 등의 분야에서는 데이터의 정확성과 일관성이 매우 중요합니다. RDBMS는 엄격한 데이터 무결성 규칙을 제공합니다.
- 트랜잭션 처리가 중요한 경우: 은행 시스템이나 온라인 거래 시스템처럼 ACID(Atomicity, Consistency, Isolation, Durability) 트랜잭션 속성을 요구하는 경우 RDBMS가 적합합니다.
- 고정된 스키마를 가진 데이터 모델링: 데이터 구조가 명확하고 변경이 적은 애플리케이션에서는 RDBMS가 효과적입니다.

2) NoSQL이 적합한 경우

- 대규모 분산 데이터 처리가 필요한 경우: 소셜 네트워크, 빅 데이터 분석, 실시간 처리가 필요한 경우와 같이 거대한 데이터 세트를 다뤄야 할 때 NoSQL이 유리합니다.
- 유연한 스키마가 필요한 경우: 데이터 모델이 자주 변경되거나, 다양한 형태의 데이터를 저장해야 하는 경우 NoSQL이 더 적합합니다.
- 빠른 속도와 확장성이 중요한 경우: NoSQL 데이터베이스는 수평적 확장이 가능하여, 서버를 추가함으로써 처리 능력을 쉽게 확장할 수 있습니다.
- 비정형 데이터 처리: 문서, 키-값 쌍, 그래프 기반 데이터 등 비정형 데이터를 다루는 애플리케이션에 NoSQL이 적합합니다.

RDBMS는 일관성과 정확성이 중요한 복잡한 비즈니스 로직 처리에 적합하고, NoSQL은 확장성, 유연성, 대량의 데이터 처리 능력을 요구하는 애플리케이션에 적합합니다. 따라서 애플리케이션의 요구 사항과 데이터의 특성을 잘 파악하여, 가장 적합한 데이터베이스 시스템을 선택해야 합니다.

Q&A 트랜잭션에 대해 아는 대로 설명해주세요.

트랜잭션(Transaction)은 데이터베이스 관리 시스템(DBMS)에서 한 번에 수행되어야 하는 일련의 연산 또는 작업 단위를 말합니다. 트랜잭션은 데이터의 일관성과 무결성을 유지하기 위해 매우 중요한 개념입니다. 트랜잭션은 다음 네 가지 주요 속성, 즉 ACID 속성으로 정의됩니다.

ACID 속성

1) 원자성(Atomicity) : 트랜잭션의 연산은 모두 수행되거나 전혀 수행되지 않아야 합니다. 즉, 하나의 트랜잭션 내의 모든 작업은 전부 완료되거나 전부 실패해야 합니다.

오류가 발생하면, 이미 실행된 작업들을 취소(롤백)하여 트랜잭션 실행 이전의 상태로 돌립니다.

2) 일관성(Consistency) : 트랜잭션이 성공적으로 완료되면, 데이터베이스는 한 일관된 상태에서 다른 일관된 상태로 전환되어야 합니다.

트랜잭션 전후로 모든 데이터베이스 제약 조건이 만족되어야 합니다.

3) 독립성(Isolation) : 동시에 실행되는 트랜잭션들은 서로 영향을 주지 않아야 합니다.

하나의 트랜잭션이 완료되기 전까지는 다른 트랜잭션에서 그 결과를 볼 수 없습니다.

4) 지속성(Durability) : 트랜잭션이 성공적으로 완료되면, 그 결과는 영구적으로 데이터베이스에 반영되어야 합니다.

시스템에 장애가 발생해도 완료된 트랜잭션의 결과는 보존되어야 합니다.

* 트랜잭션의 과정

1) 시작(Start) : 트랜잭션이 시작되며, 데이터베이스 시스템은 이를 추적합니다.

2) 실행(Execute) : 트랜잭션에 포함된 모든 명령어(쿼리)가 실행됩니다.

3) 검증(Validation) : 시스템은 트랜잭션이 ACID 속성을 만족하는지 검증합니다.

4) 커밋(Commit)/롤백(Rollback) : 검증 후, 트랜잭션이 성공적으로 완료되면 커밋되어 변경사항이 데이터베이스에 영구적으로 저장됩니다.

오류가 발생하면 롤백되어 트랜잭션 시작 전의 상태로 되돌아갑니다.

트랜잭션은 데이터베이스의 안정성과 신뢰성을 보장하기 위해 필수적이며, 모든 데이터베이스 시스템에서 중요한 기능입니다.

Q&A 데이터베이스에서 옵티마이저(Optimizer)란 무엇인가요? 아는 대로 설명해주세요.

데이터베이스 옵티마이저(Optimizer)는 SQL 쿼리를 가장 효율적으로 실행하기 위한 최적의 실행 경로를 결정하는 데이터베이스 관리 시스템(DBMS)의 구성 요소입니다. 사용자가 작성한 SQL 쿼리를 실제 데이터에 접근하여 결과를 가져오는 여러 가지 방법 중에서, 리소스 사용을 최소화하고 실행 시간을 단축하는 가장 적합한 방법을 선택합니다.

옵티마이저의 주요 기능

- 쿼리 분석: 사용자로부터 입력된 SQL 쿼리를 분석하여, 테이블, 조인, 필터 등 쿼리의 구성 요소를 이해합니다.
- 실행 계획 생성: 가능한 모든 실행 경로를 고려하여 여러 실행 계획을 생성합니다. 이 때, 테이블 스캔, 인

덱스 스캔, 조인 방식 등 다양한 데이터 접근 방법이 고려됩니다.
- 비용 평가: 각 실행 계획에 대해 예상되는 비용을 계산합니다. 비용은 일반적으로 디스크 I/O, CPU 사용량, 메모리 사용량, 네트워크 통신 등 여러 요소를 포함합니다.
- 최적의 실행 계획 선택: 예상 비용이 가장 낮은 실행 계획을 최적의 계획으로 선택합니다.
- 실행 계획 캐싱: 최적의 실행 계획을 캐시에 저장하여, 같은 또는 유사한 쿼리가 다시 실행될 때 재사용할 수 있게 합니다.

옵티마이저의 유형
- 규칙 기반 옵티마이저(Rule-based Optimizer, RBO): 고정된 규칙 집합을 사용하여 실행 계획을 선택합니다. 과거에는 많이 사용되었으나, 현대의 데이터베이스 시스템에서는 비용 기반 옵티마이저가 더 일반적입니다.
- 비용 기반 옵티마이저(Cost-based Optimizer, CBO): 각 실행 계획에 대한 비용을 계산하고, 가장 비용이 낮은 계획을 선택합니다. 데이터베이스의 실제 통계 정보(예: 테이블 크기, 인덱스의 유무, 데이터 분포 등)를 사용하여 보다 정확한 결정을 내립니다.

데이터베이스 옵티마이저는 쿼리의 성능을 최적화하여 시스템의 전반적인 성능과 효율성을 향상시키는 데 중요한 역할을 합니다.

 DB Lock에 대해 설명해주세요.
데이터베이스 락(DB Lock)은 동시에 여러 사용자나 프로세스가 데이터베이스에 접근할 때 데이터의 일관성과 무결성을 유지하기 위해 사용되는 메커니즘입니다. 락은 특정 데이터 항목에 대한 접근을 제한하여, 한 번에 하나의 트랜잭션이 해당 데이터를 읽거나 변경할 수 있도록 합니다.

* 락의 주요 유형
1) 공유 락(Shared Lock) : 공유 락은 데이터를 읽을 때 사용되며, 여러 트랜잭션이 동시에 같은 데이터 항목을 읽을 수 있게 합니다.
하지만, 공유 락이 걸린 데이터에 대해서는 변경 작업을 할 수 없습니다.
2) 배타 락(Exclusive Lock) : 배타 락은 데이터를 변경할 때 사용되며, 해당 데이터에 대한 독점적인 접근 권한을 제공합니다.
배타 락이 걸린 데이터는 해당 락을 소유한 트랜잭션만이 읽거나 쓸 수 있으며, 다른 트랜잭션은 접근할 수 없습니다.

* 락의 작동 원리
- 락 획득(Lock Acquisition) : 트랜잭션이 데이터를 읽거나 쓰기 전에 락을 획득합니다. 락을 획득할 수 없는 경우, 트랜잭션은 락이 해제될 때까지 대기합니다.
- 락 해제(Lock Release) : 트랜잭션이 완료되면, 즉 커밋이나 롤백이 이루어지면 획득했던 락을 해제합니다. 락이 해제되면, 다른 트랜잭션이 해당 데이터에 대한 접근 권한을 얻을 수 있습니다.

* 락의 목적
- 데이터 일관성 유지: 동시에 여러 트랜잭션이 같은 데이터를 수정하는 것을 방지하여 데이터의 일관성을 유지합니다.
- 동시성 제어: 락을 사용하여 데이터베이스 시스템은 여러 트랜잭션의 동시 실행을 효율적으로 관리할 수

있습니다.

* 락과 관련된 문제
 - 데드락(Deadlock): 두 개 이상의 트랜잭션이 서로 다른 락을 획득하려고 할 때, 상호 대기 상태에 빠져서 어떤 트랜잭션도 진행할 수 없게 되는 상황입니다.
 - 락 경합(Lock Contention): 많은 트랜잭션이 동시에 같은 데이터에 접근하려고 할 때 발생하는 경합으로, 시스템의 성능 저하를 초래할 수 있습니다.

데이터베이스 시스템은 이러한 문제를 관리하고 해결하기 위해 락 타임아웃, 락 에스컬레이션, 데드락 감지 및 복구 메커니즘과 같은 기능을 제공합니다.

Q&A Elastic Search의 키워드 검색과 RDBMS의 LIKE 검색의 차이에 대해 설명해주세요.
Elasticsearch의 키워드 검색과 RDBMS의 LIKE 검색은 데이터를 찾는 방식에 있어 몇 가지 주요 차이점이 있습니다.

Elasticsearch의 키워드 검색 : Elasticsearch는 전문 검색 엔진으로, 대규모 데이터셋에서 빠른 검색 성능을 제공합니다. Elasticsearch의 키워드 검색은 다음과 같은 특징을 가집니다:
 - 역 인덱스 사용: Elasticsearch는 데이터를 저장할 때 역 인덱스를 생성합니다. 이 역 인덱스는 각 키워드와 해당 키워드가 포함된 문서의 위치를 매핑하여, 검색 시 매우 빠른 검색 속도를 제공합니다.
 - 분석기(Analyzer) 사용: 입력 텍스트를 토큰화하고, 표준화하는 과정을 거쳐 인덱싱합니다. 이 과정을 통해 다양한 형태의 텍스트(예: 대소문자 구분, 어근 처리)도 효과적으로 검색할 수 있습니다.
 - 풀 텍스트 검색: Elasticsearch는 풀 텍스트 검색을 지원하여, 문서 전체에서 키워드를 찾을 수 있으며, 복잡한 쿼리(예: 불리언 쿼리, 범위 쿼리)도 가능합니다.

RDBMS의 LIKE 검색 : 텍스트 필드에서 패턴 매칭을 수행하는 방식으로, 주로 다음과 같은 특징을 가집니다:
 - 순차적 스캔: LIKE 검색은 특정 패턴과 일치하는 데이터를 찾기 위해 테이블의 레코드를 순차적으로 스캔합니다. 인덱스가 없는 경우, 이 방법은 데이터가 많을수록 비효율적일 수 있습니다.
 - 와일드카드 사용: % (여러 문자에 일치)와 _ (단일 문자에 일치)와 같은 와일드카드를 사용하여 패턴을 정의할 수 있습니다.
 - 정확한 텍스트 매칭: LIKE는 대소문자를 구분하거나 구분하지 않을 수 있으나, 텍스트의 정확한 패턴을 기준으로 검색합니다.

차이점
 - 검색 속도: Elasticsearch는 빠른 검색 속도를 위해 최적화된 반면, RDBMS의 LIKE 검색은 대량의 데이터에서 성능 저하가 발생할 수 있습니다.
 - 기능성: Elasticsearch는 풀 텍스트 검색, 복잡한 쿼리, 텍스트 분석 기능을 제공하는 반면, RDBMS의 LIKE는 기본적인 텍스트 패턴 매칭에 제한됩니다.
 - 확장성: Elasticsearch는 수평적 확장성이 뛰어나 대용량 데이터 처리에 적합합니다. RDBMS는 주로 수직 확장에 의존합니다.

종합적으로, 대규모 데이터셋에서 복잡한 텍스트 검색 요구가 있는 경우 Elasticsearch와 같은 전문 검색 엔

진이 더 적합합니다. 반면, 간단한 텍스트 검색이 필요한 경우에는 RDBMS의 LIKE 검색이 충분할 수 있습니다.

Q&A DB Tuning이 무엇인지 설명해주세요. 튜닝의 3단계에 대해서도 궁금합니다.

DB 튜닝(Database Tuning)은 데이터베이스 시스템의 성능을 최적화하는 과정입니다. 이는 데이터베이스 서버, SQL 쿼리, 스키마 설계 등 다양한 요소를 조정하여, 처리 속도를 향상시키고 자원 사용을 최적화합니다. DB 튜닝은 시스템의 효율성을 높이고 응답 시간을 단축하여, 전반적인 사용자 경험을 개선하는 데 중요한 역할을 합니다.

DB 튜닝의 3단계

1) 서버 및 하드웨어 튜닝

- 하드웨어 자원 최적화: CPU, 메모리, 스토리지 성능과 네트워크 인프라를 평가하고 최적화합니다.
- 서버 구성 조정: 데이터베이스 서버 설정을 조정하여, 시스템 자원을 효율적으로 사용하도록 합니다. 예를 들어, 메모리 할당, 캐시 크기, 연결 수 제한 등을 조정할 수 있습니다.

2) 데이터베이스 스키마 및 객체 튜닝

- 정규화 및 역정규화: 데이터 중복을 줄이고, 쿼리 성능을 고려하여 스키마를 조정합니다.
- 인덱싱 전략: 쿼리 성능을 향상시키기 위해 적절한 인덱스를 생성하고 관리합니다.
- 파티셔닝: 대용량 테이블을 더 작은 단위로 나누어 관리하여, 검색과 유지 보수를 용이하게 합니다.

3)쿼리 튜닝

- SQL 쿼리 최적화: 비효율적인 쿼리를 식별하고, 실행 계획을 분석하여 쿼리를 최적화합니다.
- 쿼리 성능 분석: 쿼리 실행 계획을 사용하여 쿼리의 성능을 분석하고, 필요한 경우 쿼리 구조를 개선하여 성능을 향상시킵니다.
- 캐싱 전략: 자주 사용되는 쿼리 결과를 캐시하여, 반복적인 데이터베이스 액세스를 줄입니다.

DB 튜닝은 전체 시스템의 성능을 극대화하기 위해 지속적으로 수행되어야 하는 과정입니다. 데이터베이스의 사용 패턴, 데이터 양, 사용자 요구사항 등이 시간에 따라 변할 수 있기 때문에, 정기적인 모니터링과 조정을 통해 시스템을 최적의 상태로 유지해야 합니다.

Q&A 테이블 조인이 궁금합니다. inner join과 outer join, full join의 차이를 설명해주세요.

테이블 조인은 두 개 이상의 테이블에서 관련된 데이터를 결합하기 위해 사용되는 SQL 연산입니다. 조인의 주요 유형에는 내부 조인(Inner Join), 외부 조인(Outer Join) 그리고 전체 조인(Full Join)이 있습니다. 각 조인 유형은 결합할 테이블 간의 관계를 다르게 처리합니다.

* Inner Join (내부 조인)

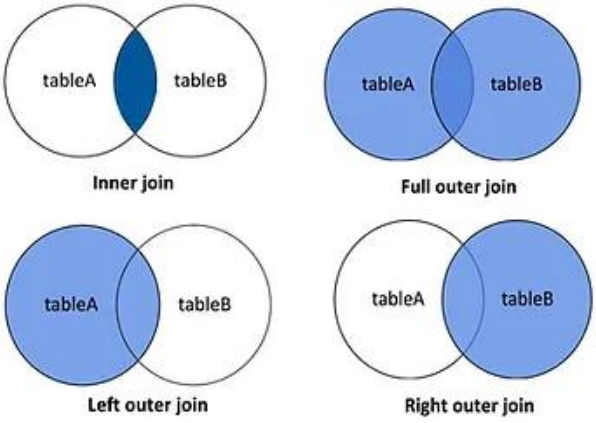
- 정의: 두 테이블의 교집합 부분만을 결과로 반환합니다. 즉, 양쪽 테이블 모두에서 일치하는 행만을 선택합니다.
- 특징: 조인 조건에 맞는 행이 양쪽 테이블에 존재할 경우에만 해당 행을 결과에 포함합니다.
- 예시: 두 테이블 A와 B를 조인할 때, A의 특정 행이 B에도 대응하는 행이 있을 경우, 그 행들만 결과로

나타납니다.

* Outer Join (외부 조인)
외부 조인은 세 가지 하위 유형으로 나뉩니다: 왼쪽 외부 조인(Left Outer Join), 오른쪽 외부 조인(Right Outer Join), 그리고 전체 외부 조인(Full Outer Join).
- Left Outer Join (왼쪽 외부 조인)
  정의: 왼쪽 테이블의 모든 행과 오른쪽 테이블에서 조인 조건에 일치하는 행을 결합합니다.
  특징: 왼쪽 테이블의 행이 조인 조건에 일치하는 오른쪽 테이블의 행이 없더라도, 왼쪽 테이블의 행은 결과에 포함되며, 오른쪽 테이블에서는 NULL 값으로 채워집니다.
- Right Outer Join (오른쪽 외부 조인)
  정의: 오른쪽 테이블의 모든 행과 왼쪽 테이블에서 조인 조건에 일치하는 행을 결합합니다.
  특징: 오른쪽 테이블의 행이 조인 조건에 일치하는 왼쪽 테이블의 행이 없더라도, 오른쪽 테이블의 행은 결과에 포함되며, 왼쪽 테이블에서는 NULL 값으로 채워집니다.

* Full Join (전체 조인)
- 정의: 두 테이블의 합집합을 결과로 반환합니다. 양쪽 테이블에서 조인 조건에 맞는 행이 없는 경우에도, 해당 행들은 결과에 포함되며, 대응하는 행이 없는 테이블 쪽은 NULL 값으로 채워집니다.
- 특징: 두 테이블 중 하나라도 일치하는 행이 있다면 결과에 포함되며, 양쪽 테이블에서 일치하지 않는 행도 모두 결과에 포함됩니다.

각 조인 유형은 데이터를 결합하는 방식에서 차이가 있으며, 사용할 조인 유형을 결정할 때는 반환하려는 데이터의 성격과 요구사항을 고려해야 합니다.



https://www.numpyninja.com/post/understanding-joins-in-postgresql

Q&A JOIN에서 ON과 WHERE의 차이를 설명해주세요.
JOIN 구문에서 ON과 WHERE 절은 조인을 수행할 때 조건을 지정하는 데 사용되지만, 그 목적과 사용 방식에 차이가 있습니다.

* ON 절
- 목적: ON 절은 조인을 수행하는 두 테이블 간의 관계를 명시하는 데 사용됩니다. 주로 두 테이블 사이의 조인 조건을 정의하는 데 사용되며, 어떤 키(열)를 기준으로 두 테이블을 결합할 것인지를 지정합니다.
- 특징: ON 절은 조인이 수행될 때 각 테이블의 행이 결합될 기준을 정의하고, 해당 조건에 맞는 행들만이

조인된 결과로 나타납니다.
- 적용 시점: 조인 과정에서 두 테이블의 관련 행을 찾아 결합할 때 사용됩니다.

* WHERE 절
- 목적: WHERE 절은 조인된 결과에 대한 추가적인 필터링 조건을 제공합니다. 즉, 조인이 완료된 후에 결과 데이터셋에서 특정 조건을 만족하는 행을 선택하는 데 사용됩니다.
- 특징: WHERE 절은 조인된 결과에 대해 조건을 적용하여, 해당 조건에 부합하는 행만을 최종 결과로 반환합니다.
- 적용 시점: 조인이 수행된 후, 최종적으로 반환될 결과셋에서 조건에 맞는 데이터를 필터링할 때 사용됩니다.

예시 : 다음은 employees 테이블과 departments 테이블을 조인할 때 ON과 WHERE의 사용 예시입니다.

```
SELECT employees.name, departments.department_name
FROM employees
JOIN departments ON employees.department_id = departments.id
WHERE employees.salary > 50000;
```

여기서 ON employees.department_id = departments.id: 이 조건은 employees 테이블과 departments 테이블을 어떤 열의 값이 일치하는지에 기반하여 결합합니다. 즉, 두 테이블의 department_id와 id 열을 기준으로 조인을 수행합니다.
WHERE employees.salary > 50000: 이 조건은 조인된 결과에서 직원의 급여가 50000 이상인 행만을 선택합니다.
이처럼 ON은 조인을 어떻게 수행할지를 결정하는 반면, WHERE는 조인된 결과 중에서 어떤 데이터를 최종적으로 선택할지를 결정하는 데 사용됩니다.

Q&A select문에서 HAVING과 WHERE의 차이를 설명해주세요.
SELECT 문에서 HAVING과 WHERE는 모두 데이터를 필터링하는 데 사용되지만, 사용 목적과 적용 시점에 차이가 있습니다.

1) WHERE
목적: WHERE 절은 데이터베이스에서 데이터를 가져오기 전에 특정 조건을 기반으로 행을 필터링하는 데 사용됩니다.
적용 시점: WHERE 절은 GROUP BY 절 이전에 적용되며, 집계 함수를 적용하기 전의 개별 행에 대한 조건을 지정합니다.
기능: WHERE 절은 집계 과정에 포함될 데이터를 선별하여, 결과적으로 집계할 데이터의 양을 줄이거나, 특정 조건을 만족하는 데이터만 집계 대상으로 만듭니다.
2) HAVING
목적: HAVING 절은 GROUP BY를 사용하여 그룹화된 결과에 대해 조건을 적용할 때 사용됩니다. 주로 집계 함수의 결과에 조건을 걸고 싶을 때 활용됩니다.
적용 시점: HAVING 절은 GROUP BY 절 이후에 적용되며, 그룹화된 결과의 집계 값을 기반으로 필터링합니다.

기능: HAVING 절은 이미 그룹화된 데이터에 대한 집계 결과를 기준으로 조건을 적용하므로, 특정 조건을 만족하는 그룹만을 결과로 반환합니다.

예시 : 다음은 employees 테이블에서 각 부서별로 평균 급여가 5000 이상인 부서의 평균 급여를 구하는 예시입니다.

```
SELECT department, AVG(salary) AS avg_salary
FROM employees
WHERE salary > 3000
GROUP BY department
HAVING AVG(salary) > 5000;
```

여기서 WHERE salary > 3000: 이 조건은 employees 테이블에서 급여가 3000 이상인 행만을 선택하여, 이후의 집계 작업(평균 계산)의 대상이 됩니다.
HAVING AVG(salary) > 5000: 이 조건은 그룹화된 결과, 즉 부서별로 계산된 평균 급여가 5000 이상인 그룹만을 결과로 보여줍니다.
이처럼 WHERE과 HAVING은 각각 데이터를 가져오는 단계와 집계한 결과에 대한 필터링 단계에서 조건을 적용하는 데 사용됩니다.

Q&A group by의 문법 및 역할에 대해 설명해주세요.
GROUP BY 문은 SQL에서 데이터를 그룹화하여 집계 함수(예: COUNT, SUM, AVG, MAX, MIN 등)를 사용할 때 필요한 구문입니다. 이 문을 사용하면, 지정된 열의 값이 같은 레코드끼리 그룹을 형성하고, 각 그룹에 대해 집계 연산을 수행할 수 있습니다.

* 문법

```
SELECT column1, column2, AGGREGATE_FUNCTION(column3)
FROM table WHERE condition
GROUP BY column1, column2;
```

SELECT 절에서는 GROUP BY 절에 나열된 열과 집계 함수를 사용한 열만 포함할 수 있습니다.
FROM 절은 집계를 수행할 데이터가 있는 테이블을 지정합니다.
WHERE 절은 그룹화되기 전에 데이터를 필터링하는 데 사용할 수 있습니다. (선택 사항)
GROUP BY 절은 결과 집합을 어떤 열의 값에 따라 그룹화할 것인지를 지정합니다.

* 역할

- 데이터 분류: GROUP BY를 사용하면 특정 열(또는 열들)을 기준으로 데이터를 그룹으로 분류할 수 있습니다.
- 집계 연산: 각 그룹에 대해 집계 함수를 적용하여, 그룹별 합계, 평균, 최대값, 최소값 등을 계산할 수 있습니다.
- 데이터 요약: 큰 데이터 세트에서 중요한 통계 정보를 요약하여, 데이터를 보다 의미 있고 관리하기 쉬운 형태로 제공합니다.
- 보고 및 분석: 비즈니스 인텔리전스, 데이터 분석, 보고서 생성 등에 필요한 데이터를 구성하고 분석하는

데 중요한 역할을 합니다.
예시
SELECT department, COUNT(*) AS employee_count
FROM employees
GROUP BY department;

이 쿼리는 employees 테이블에서 department 열을 기준으로 직원 수를 그룹화하고 계산하여, 각 부서별 직원 수를 나타냅니다. GROUP BY 절은 department 열의 각 고유 값에 대해 집계 함수 COUNT(*)를 적용하여 직원 수를 계산합니다.

Q&A DELETE, TRUNCATE, DROP의 차이를 설명해주세요.
DELETE, TRUNCATE, DROP은 데이터베이스에서 데이터를 제거하는 세 가지 다른 SQL 명령어로, 사용 목적과 영향에 따라 차이가 있습니다.

1) DELETE
- 용도: 테이블에서 특정 행을 제거합니다.
- 특징:
  조건을 사용하여 삭제할 특정 행을 지정할 수 있습니다 (WHERE 절 사용).
  DELETE 연산은 로그를 생성하고, 트랜잭션 내에서 실행되어 롤백이 가능합니다.
  DELETE는 각각의 행을 삭제하면서 무결성 제약 조건을 검사합니다.
  데이터만 삭제되며, 테이블 구조나 인덱스는 유지됩니다.

2) TRUNCATE
- 용도: 테이블의 모든 행을 빠르게 제거합니다.
- 특징:
  TRUNCATE는 WHERE 절을 사용할 수 없으며, 테이블의 모든 행을 삭제합니다.
  테이블을 재사용할 수 있는 상태로 초기화하며, 보통 로그를 생성하지 않아 DELETE보다 빠릅니다 (단, 시스템에 따라 최소한의 로그 생성 가능).
  트랜잭션 내에서 실행되긴 하지만, 대부분의 경우 롤백이 불가능합니다.
  테이블 구조와 인덱스는 유지되지만, auto-increment 필드는 초기화될 수 있습니다.

3) DROP
- 용도: 테이블 또는 데이터베이스 객체 자체를 완전히 제거합니다.
- 특징:
  DROP 명령은 테이블과 그 테이블에 관련된 모든 인덱스, 데이터, 구조를 데이터베이스에서 제거합니다.
  삭제된 테이블은 복구가 불가능하며, 관련 객체도 함께 제거됩니다.
  DROP은 테이블을 데이터베이스에서 완전히 제거하므로, 메모리와 저장 공간을 해제합니다.

이 세 명령어는 사용하는 상황과 필요에 따라 선택해야 합니다. 예를 들어, 단순히 일부 데이터를 삭제하려면 DELETE를, 테이블의 데이터를 빠르게 초기화하고자 하면 TRUNCATE를, 테이블 자체를 완전히 제거하려면 DROP을 사용합니다.

Q&A 데이터베이스 클러스터링과 리플리케이션의 차이에 대해 설명해주세요.

데이터베이스 클러스터링과 리플리케이션은 데이터베이스 관리에서 사용되는 두 가지 중요한 기술로, 데이터의 가용성과 안정성을 높이는 데 목적이 있지만, 각각의 작동 원리와 목적에 차이가 있습니다.

*데이터베이스 클러스터링

- 목적: 고가용성(High Availability, HA)과 부하 분산(Load Balancing)을 제공하기 위함입니다.
- 작동 원리:

클러스터링은 여러 서버 또는 인스턴스가 함께 작동하여 단일 데이터베이스 시스템처럼 보이게 하는 기술입니다.

각 노드(서버)는 같은 데이터에 대한 접근 권한을 가지며, 하나의 노드에 장애가 발생하면 다른 노드가 자동으로 작업을 인계받아 서비스의 중단 없이 운영을 계속할 수 있습니다.

- 특징:

데이터는 클러스터 내의 모든 노드에 걸쳐 실시간으로 동기화됩니다.

클러스터링을 통해 시스템의 장애 허용(Fault Tolerance) 능력을 향상시키고, 요청 처리량을 증가시킬 수 있습니다.

*데이터베이스 리플리케이션

- 목적: 데이터의 복제 및 백업을 통해 데이터의 안정성과 가용성을 보장하기 위함입니다.
- 작동 원리:

리플리케이션은 하나의 데이터베이스 서버(주 서버)로부터 데이터를 다른 서버(복제 서버)로 복사하는 과정입니다.

데이터는 주로 비동기적으로 복제되며, 주 서버에서 발생한 변경사항이 복제 서버로 전송되어 동일한 데이터 상태를 유지합니다.

- 특징:

리플리케이션을 통해 데이터를 지리적으로 분산시킬 수 있으며, 백업 및 재해 복구, 읽기 쿼리의 부하 분산에 효과적입니다.

데이터의 일관성 유지 방법에 따라 동기, 비동기, 반동기 리플리케이션이 있습니다.

* 차이점

- 용도: 클러스터링은 주로 시스템의 가용성과 부하 분산을 목적으로 하며, 리플리케이션은 데이터의 안전성, 백업 및 복구, 읽기 성능 최적화를 목적으로 합니다.
- 데이터 동기화: 클러스터링은 데이터를 노드 간에 실시간으로 동기화하는 반면, 리플리케이션은 주로 비동기적으로 데이터를 복제합니다.
- 시스템 구조: 클러스터링은 여러 노드가 하나의 데이터베이스 시스템을 구성하지만, 리플리케이션은 하나의 주 서버와 하나 또는 여러 개의 복제 서버 간의 데이터 복제 관계를 의미합니다.

데이터베이스 클러스터링과 리플리케이션은 각각의 장단점을 이해하고, 시스템의 요구 사항에 맞게 적절하게 활용해야 합니다.