INDEX

1. 데이터베이스 인덱스란 무엇이며, 왜 사용하나요?

> 인덱스는 **데이터베이스에서 검색 성능**을 높이기 위해 사용하는 **자료구조**이다. 책의 목차처럼 원하는 데이터를 빠르게 찾을 수 있도록 도와준다.

- 사용하는 이유? 데이터를 찾을 때 Full Table Scan을 하면 시간이 오래 걸리는데, 이때 인덱스를 쓰면 특정 값을 빠르게 찾을 수 있어서 성능이 향상된다.

2. B-Tree 인덱스와 Hash 인덱스의 차이점은 무엇인가요?

B-Tree 인덱스

> B-Tree 인덱스는 가장 일반적으로 사용되는 인덱스 구조로, **데이터를 정렬된 트리 형태**로 저장하고, 검색 시 트리를 따라 내려가면서 값을 찾는 방식이다.

B-Tree 인덱스의 가장 큰 장점은 `범위 검색이 가능` 하다는 점이다. ex) WHERE age > 30, BETWEEN, LIKE 'abc%' 등의 조건에서 유리하다.

또한 정렬이 되어 있기 때문에 `ORDER BY`, `GROUP BY` 와 같은 쿼리에도 효율적으로 사용될 수 있다.

Hash 인덱스

> Hash 인덱스는 데이터를 **해시 함수**를 통해 변환하여 인덱스를 구성한다. **"정확히 일치하는 값" 을 매우 빠르게 찾을 수** 있는 구조이다.

Hash인덱스는 동등 조건에서 뛰어난 성능을 보인다. ex) WHERE id = 12345

하지만, 값의 순서를 알 수 없기 때문에 **범위 검색이 불가능**하다. ex) WHERE id > 1000 Hash 인덱스는 불가능

B-Tree 인덱스 VS. Hash 인덱스

B-Tree 인덱스는 범위 검색, 정렬, 다양한 조건 처리에 유연하여 대부분에 DB에서 기본 인덱스 타입으로 사용된다. 반면, Hash 인덱스는 정확한 값 비교에는 특화되어 있지만, 범위나 정렬 관련 쿼리에는 적합하지 않다.

실제 MySQL(InnoDB)에서는 대부분 B-Tree 인덱스를 기본으로 사용하고 있고, Hash 인덱스는 특정 스토리지 엔진(Memory 엔진 등)에서만 제한적으로 사용된다.

3. 인덱스를 사용할 때의 장점과 단점을 설명해주세요.

인덱스 사용의 장점

1. 조회 성능 향상
2. WHERE, JOIN, ORDER BY, GROUP BY 등의 쿼리 최적화 가능
    - 인덱스는 원하는 조건에 해당하는 레코드만 빠르게 조회할 수 있도록 도와준다.

인덱스 사용의 단점

1. 인덱스를 위한 추가 저장공간 필요
    - 인덱스는 별도의 자료구조(B-Tree 등)를 테이블과 별도로 저장한다.
2. 쓰기 성능 저하 (INSERT, UPDATE, DELETE): 인덱스도 갱신해야하기 때문
3. 너무 많은 인덱스는 오히려 성능 저하를 유발할 수 있음
    - 여러개의 인덱스를 추가했을 때, 쿼리 실행시 어떤 인덱스를 쓸지 판단하는 비용 발생

=> 인덱스는 읽기 성능을 높이지만, 쓰기 성능을 희생하는 구조이다. 따라서 읽기 위주의 시스템(검색 서비스, 리포트 대시보드 등)은 인덱스를 적극적으러 활용하고, 쓰기 위주의 시스템(로그 수집, 트랜잭션 처리 등)은 최소한의 인덱스만 활용한다.

4. 인덱스 스캔 방법에는 어떤 것들이 있나요?

Full Table Scan

> 인덱스 없이 테이블 전체를 훑는 스캔 방법

Index Scan / Index Range Scan

> 인덱스를 따라 범위 내 값을 검색

Index Only Scan

> 필요한 데이터가 모두 인덱스에 있어 테이블 접근 생략

Bitmap Index Scan

> 대량의 조건에 대한 효율적인 검색

Unique Index Scan

> 고유한 인덱스에서 정확히 하나읙 값만 찾을 때

5. 복합 인덱스(Composite Index)란 무엇이며, 어떤 경우에 유용한가요?

> 복합 인덱스란, 두 개 이상의 컬럼을 조합해 만든 인덱스이다.

복합 인덱스가 유용한 경우

- WHERE 조건이 여러 컬럼에 걸쳐 있을 때
- 자주 같이 조회되는 컬럼이 있을 때
- 특정 컬럼 조홥으로 정렬, 그룹화 할 때

6. 인덱스 설계 시 고려해야 할 핵심 요소는 무엇인가요?

- **조회 패턴** -> 어떤 쿼리가 자주 실행되는지?

- **선택도(Selectivity)** -> 컬럼의 중복이 적을 수록 인덱스 효율이 높아짐 (주민번호 vs. 성별)
- **컬럼 순서** -> 복합 인덱스일 경우 자주 사용하는 조건을 앞에 두는 게 좋음
- **쓰기 비용** -> 인덱스가 많을 수록 쓰기 성능 저하
- **커버링 인덱스 고려** -> 인덱스 만으로 쿼리가 해결되도록 설계하면 성능이 향상