BIT BY BIT

2025/02/19 (수)

Session 10 – 데이터베이스

이동인

오프닝

제일 어려웠던 파트는?

조인 알고리즘, 트랜잭션, 인덱싱

Q.inner join이 아닌 left outer join을 써야하는 경우

조인 알고리즘

조인 알고리즘 세 가지? 한국말로 !

중첩 루프 조인

정렬 병합 조인

해시 조인

조인 알고리즘 세 가지? 영어로 !

중첩 루프 조인 (Nexted Loop Join)

정렬 병합 조인

해시 조인

조인 알고리즘 세 가지? 영어로 !

중첩 루프 조인 (Nexted Loop Join)

정렬 병합 조인(Sort Merge Join)

해시 조인(Hash Join)

조인 알고리즘을 이제 하나씩 볼건데,,

정의 설명, 시간복잡도 설명, 실제 작동 예시 설명
하나씩 맡아서 설명해봅시다.

중첩 루프 조인

?

중첩 루프 조인

- 중첩 루프 조인 같은 특정 조인 알고리즘 명시적으로 설정하는 것은 직접적으로 가능하지 않음
- 해당 데이터베이스의 query optimizer가 쿼리를 실행할 때 조인의 종류 중 자동으로 가장 성능이 좋은 알고리즘으로 결정
  - MySQL, PostgreSQL, Oracle, SQL server 모두 해당
- 중첩 루프 조인이란?
  - 한 테이블의 각 행에 대해 다른 테이블의 모든 행을 순차적으로 검사하다 조인하는 방법 ⇒ N*M

정렬 병합 조인

?

정렬 병합 조인

- 정렬병합조인(Sort Merge Join)
  - 두 테이블을 조인할 필드를 기준으로 각각 정렬한 후, 정렬된 순서에 따라 행을 병합하여 조인하는 방법 ⇒ NlogN + MlogM
- 이미 정렬되어있을 때 더 효율적임
- 등가조인에서보단 비등가조인에서 유리
  - 등가조인에서는 해시조인이 유리 O(N+M)
  - 비등가조인에서는 정렬병합조인이 유리

해시 조인

정의 설명 / 시간복잡도 설명 / 특징(장단점) 설명

해시 조인

- 해시조인(Hash Join)
  - 한 테이블(보통은 더 작은 테이블)의 행을 사용하여 해시 테이블을 만든 다음, 다른 테이블을 순회하면서 해시 테이블을 사용하여 조인하는 방법 ⇒ O(N+M)
- 하나의 테이블이 서버 메모리에 온전히 들어간다면 중첩 루프 조인보다 더 효율적
  - 다만,
    - 메모리가 부족할 경우 성능이 저하됨
    - 동등(등가) 조인에서만 사용할 수 있다.
- 방법
  - 빌드 단계(Build Phase)
    - 바이트가 더 작은 테이블로부터 해시 테이블 생성
    - 각 행의 조인 키에 해시 함수를 적용하여 해시 테이블에 저장
  - 프로브 단계 (Probe Phase)
    - 다른 테이블의 각 행에 대한 조인키를 해싱하여 앞서 만든 해시값과 비교해서 해시 테이블에서 매칭되는 행을 찾으면 해당 행들을 조인 결과 테이블에 추가

트랜잭션

용어의 이해

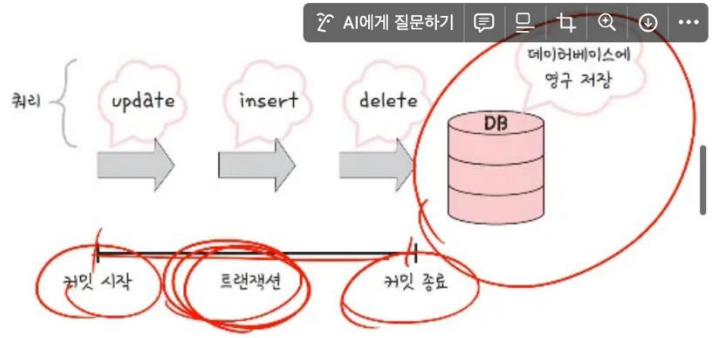
트랜잭션, 커밋, 롤백, 트랜잭션 전파에 대해 설명해주세요.

트랜잭션, 커밋, 롤백, 트랜잭션 전파

- 트랜잭션 전파
  - 트랜잭션을 수행할 때 커넥션 단위로 수행하기 때문에 커넥션 객체를 넘겨서 수행해야함 (싱글톤패턴?)
  - 트랜잭션 전파
    - 커넥션 객체를 넘기지 않고 여러 트랜잭션 관련 메서드의 호출을 하나의 트랜잭션에 묶이도록 하는 것

- 트랜잭션
  - 데이터베이스에서 하나의 논리적 기능을 수행하기 위한 작업의 단위
    - 즉, 여러 개의 쿼리들을 하나로 묶는 단위

- 커밋(commit)
  - 여러 쿼리가 성공적으로 처리되었다고 확정하는 명령어
  - 커밋이 수행되었다 = 하나의 트랜잭션이 성공적으로 수행되었다.



롤백(roll back)
  - 트랜잭션으로 처리한 하나의 묶음 과정을 일어나기 전으로 돌리는 일(취소)

데이터 무결성 네 가지

이름	설명
개체 무결성	기본키로 선택된 필드는 빈 값을 허용하지 않습니다.
참조 무결성	서로 참조 관계에 있는 두 테이블의 데이터는 항상 일관된 값을 유지해야 합니다.
고유 무결성	특정 속성에 대해 고유한 값을 가지도록 조건이 주어진 경우 그 속성 값은 모두 고유한 값을 가집니다.
NULL 무결성	특정 속성 값에 NULL이 올 수 없다는 조건이 주어진 경우 그 속성 값은 NULL이 될 수 없다는 제약 조건입니다.

데이터 무결성 설명해보기

트랜잭션의 특징 네 가지 설명해보기

트랜잭션의 특징 네 가지 설명해보기

- 원자성(atomicity) : all or nothing
  - 모두 수행됐거나, 수행되지 않았거나
    - 트랜잭션 커밋 - 모두 수행됨,
      - 몇개만 수행되는 거 없다.
      - 데이터베이스 사용자는 트랜잭션 과정을 볼 수도 참여할 수도 없고 과정이 모두 끝난 상황만 볼 수 있음
    - 트랜잭션 롤백 - 수행되지 않음
- 일관성(consistency)
  - 허용된 방식으로만 데이터를 변경해야하는 것
  - 데이터 마다의 조건과 규칙을 지켜야함
    - ex. 0원에서 5000원 차감 못하는 것 등
- 지속성(durability)
  - 성공적으로 수행된 트랜잭션은 영원히 반영되어야함
    - 데이터베이스에 시스템 장애가 발생해도 원래 상태로 복구하는 회복 기능이 있어야함을 의미

      ⇒ 데이트베이스 : 체크섬, 저널링, 롤백 등의 기능을 제공
      - 체크섬
        - 중복 검사의 한 형태, 오류 정정을 통해 송신된 자료의 무결성을 보호하는 단순 방법
      - 저널링
        - 파일 시스템 또는 데이터베이스 시스템에 변경 사항을 반영(commit)하기 전에 로깅하는 것

그리고 격리성 !

격리성에 대해 더 자세히 알아보기

격리 수준에 따라 발생할 수 있는 현상 설명

팬텀리드 (phantom reads) :

반복 가능하지 않은 조회(Nonrepeatable Reads) :

•더티 리드(Dirty Reads) :

격리성에 대해 더 자세히 알아보기

격리 수준에 따라 발생할 수 있는 현상 설명

팬텀리드 (phantom reads) : 한 트랜잭션 내에서 동일한 쿼리를 2번 이상 보냈을 때 해당 조회 결과가 다름

반복 가능하지 않은 조회(Nonrepeatable Reads) :

- 더티 리드(Dirty Reads) :

격리성에 대해 더 자세히 알아보기

격리 수준에 따라 발생할 수 있는 현상 설명

팬텀리드 (phantom reads) : 한 트랜잭션 내에서 동일한 쿼리를 2번 이상 보냈을 때 해당 조회 결과가 다름

반복 가능하지 않은 조회(Nonrepeatable Reads) : 한 트랜잭션 내의 같은 행에 두 번 이상 조회가 발생했는데 그 값이 다른 것

- 더티 리드(Dirty Reads) :

격리성에 대해 더 자세히 알아보기

격리 수준에 따라 발생할 수 있는 현상 설명

팬텀리드 (phantom reads) : 한 트랜잭션 내에서 동일한 쿼리를 2번 이상 보냈을 때 해당 조회 결과가 다름

반복 가능하지 않은 조회(Nonrepeatable Reads) : 한 트랜잭션 내의 같은 행에 두 번 이상 조회가 발생했는데 그 값이 다른 것

•더티 리드(Dirty Reads) :한 트랜잭션이 다른 트랜잭션의 아직 커밋되지 않은 데이터를 읽는 현상

인덱싱

인덱싱이란?

인덱싱의 장점

B-Tree
정의, 특징

인덱싱에 비용이 든다

🔷 1. 인덱스의 주요 비용 요인

✅ 1) 인덱스는 추가적인 저장 공간을 사용함

- 인덱스는 테이블과는 별도로 저장되는 **B-Tree 구조의 데이터**
- 즉, **인덱스를 추가할수록 디스크 사용량이 증가!**
- 예를 들어, 테이블이 1GB인데 인덱스를 여러 개 만들면 **추가로 수백 MB~GB의 저장 공간이 필요할 수 있음.**

✅ 2) 인덱스 업데이트(INSERT, UPDATE, DELETE)가 느려짐 (O(log N))

- 인덱스가 존재하면 **새로운 데이터가 삽입될 때마다 B-Tree를 수정해야 함**
- 즉, 데이터를 삽입·삭제할 때마다 **B-Tree 노드를 조정하는 비용이 발생(O(log N))**
- 만약 테이블에 **인덱스가 5개라면?** → 데이터를 추가할 때마다 **5개의 B-Tree를 수정해야 함(즉, 쓰기(Write) 성능이 저하됨!** 🚨)

✅ 3) 인덱스는 테이블과 동기화해야 함

- 데이터가 변경될 때마다 **인덱스도 같이 변경되어야 함**
- 예를 들어, `UPDATE users SET age = 30 WHERE id = 5;` 같은 쿼리를 실행하면:
  1. 테이블에서 `id=5` 값을 변경해야 하고,
  2. 만약 `age` 컬럼에 인덱스가 있다면, 해당 인덱스도 같이 수정해야 함.
- 즉, **데이터를 수정할 때 추가적인 연산이 필요함 → 성능 저하**

✅ 4) 다중 인덱스 사용 시 쿼리 최적화가 복잡해짐

- 인덱스를 여러 개 만들면 쿼리 실행 계획(Query Plan)이 복잡해지고, **오히려 성능이 떨어질 수도 있음**
- 예를 들어, `users` 테이블에 `name`과 `age`에 각각 인덱스를 만들었는데,이런 쿼리를 실행하면 **DB 엔진이 어떤 인덱스를 사용할지 선택해야 함 → 최적화 비용 발생.**

```sql
sql
복사편집
SELECT * FROM users WHERE name = 'Alice' AND age > 30;
```

🔷 2. 인덱스가 비용이 큰 작업

작업	비용
SELECT (조회)	✅ 빠름 (O(log N) + O(1))
INSERT (삽입)	❌ 느려짐 (B-Tree 수정 → O(log N))
DELETE (삭제)	❌ 느려짐 (B-Tree에서 키 삭제 → O(log N))
UPDATE (수정)	❌ 느려짐 (인덱스도 같이 수정해야 함)

✅ **즉, 인덱스는 "조회"는 빠르게 하지만, "삽입/삭제/수정"을 느리게 만듦!** 🚀

인덱싱 최적화 기법 세 가지

대수확장성이 좋은데, 왜 인덱싱 최적화를 해야할까?

감사합니다