5~7장 정리

≔ 태그

질문

- nosql에 저장하기 좋은 데이터 형식과 rdb에 저장하기 좋은 형태는 어떤 것이 있을까요
- MVCC에 대해서

질문
5장
    관계형 데이터베이스의 계층
6장
    View
    DBMS 내의 NULL
7장
    트랜잭션
    ACID
    복수 커넥션에서 읽고 쓰기(MVCC)
    잠금 타임아웃과 교착 상태
    해서는 안되는 트랜잭션 처리

5장

커넥션(Connection)

- 로그인을 통해 프롬프트가 표시된 것➡️ 사용자와 MySQL이 접속된 것➡️ Connection

- 커넥션이 유지되는 한 사용자는 데이터베이스와 무언가를 주고받을 수 있음
- 로그인시 나타나는 메세지 중 아래의 문구는 커넥션이 성립되었음을 표시

```
Your MySQL connection id is 8
```

  - `connection id` : MySQL과 사용자간의 커넥션에 붙인 번호
- MySQL은 동시에 여러 커넥션을 유지하는 것이 가능
  - 즉, 동시에 복수의 사용자와 연결 가능
  - 어느 사용자와의 커넥션인지 번호로 관리

세션(Session)

- 커넥션의 시작과 종료 사이에 DBMS와 다양한 교환을 진행하는데, 그 교환의 시작과 종료까지의 단위가 세션(Session)
- 기본적으로 커넥션과 섹션은 1:1 대응되어 커넥션 성립 시 암묵적으로 세션이 시작되고, 세션을 끊으면 커넥션도 끊어지는 경우가 많으므로 세션과 커넥션을 잘 구분하지 않음

커넥션 상태 확인

```
mysql> show status like 'Threads_connected';
```

- `Thread_connected`의 값 = 사용자의 수

로그오프

- 커넥션 끊기

관계형 데이터베이스의 계층

- **인스턴스 > 데이터베이스 > 스키마 > 오브젝트(테이블, 인덱스, 저장 프로시저 등)**
- 트리 구조

스키마(Schema)

- 스키마를 통해 테이블을 용도별로 나누거나 권한 관리 수행 가능
- 스키마 아래에 여러 테이블 존재 가능

오브젝트(Object)

- 최하위 계층
- 테이블(Table), 인덱스(Index), 저장 프로시저(Stored Procedure) 등

DBMS별 계층 구조

- 4계층 구조
  - PostgreSQL, SQL Server, SB2
- 3계층 구조 : 스키마와 데이터베이스의 한 계층을 생략
  - MySQL: 데이터베이스와 스키마를 동일한 것으로 간주 (동의어)
  - Oracle: 4계층 구조로 되어있긴 하지만 인스턴스 아래에 데이터베이스를 한 개만 만들 수 있어 실질적으로 3계층 구조
- ANSI 표준으로 정해진 것은 4계층 구조
- 개발사의 의향에 따라 바뀔 수 있음

6장

View

View → CREATE VIEW 뷰 명 AS SELECT 문;

테이블 대신에 뷰를 사용하는 이점은 다음과 같음

1. 복잡한 SELECT 문을 일일이 매번 기술할 필요 X
2. 필요한 열과 행만 사용자에게 보여줄 수 있다. 갱신 시에도 뷰 정의에 따른 갱신으로 한정 가능
3. 기억 장치의 용량 사용 없는, 즉 데이터 저장 없이 실현 가능. 뷰를 제거해도 참조하는 테이블은 영향 받지 않음

뷰로의 입력, 갱신의 제한

- 어떤 행 대응하는지, 어떤 값을 넣으면 좋을지 모르는 경우에는 갱신 불가
- group by로 집약한 수치, distinct로 얻은 값 갱신
  → 테이블의 어느 행의 수치를 갱신하는 것이 좋은지 판단 불가
- 2가지 이상의 테이블 조합 및 작성한 뷰 갱신 시, 어느 테이블 갱신하면 좋을지 모름

- 뷰에서 원래 테이블의 일부 열만 선택되었다면 데이터를 삽입하려고 해도 선택된 열 이외에 열에 기본값도 없고 NULL도 허용되지 않는 상황에서는 해당 열에 넣을 수 있는 값이 없어서 실질적으로 뷰로의 삽입이 불가

DBMS 내의 NULL

- null은 불명, 적용 불가를 나타냄
- DBMS 내에서는 NULL의 사용을 그다지 권장 X
- 2가지 이유
  1. 인간의 직감에 반하는 3개의 논리 값(true, false, null로 인한 unknown)
     - IS NULL 이 아닌 =NULL 로 하면 unknown 발생
  2. 사칙연산 또는 SQL 함수의 인수에 null이 포함되면 `null의 전파` 발생
     - null은 비교 연산이 불가
- 일부 DBMS에서는 NULL을 포함한 비교 연산 구현됨
  - MySQL : <=>
    - A <=> B : 어느 쪽에 NULL이 포함되어 있어도 올바르게 비교 가능
  - SQL 표준
    - IS NOT DISTINCT FROM이 정의
    - null이 섞여 비교 수행

7장

트랜잭션

- 트랜잭션이란 복수 쿼리를 한 단위로 묶는 것을 말한다.
  - DB에서 갱신 (`INSERT` , `DELETE` , `UPDATE`)는 단일 쿼리만으로 구성된 것이 아니고 복수 쿼리를 연속적으로 수행하는 경우가 대부분이다.
  - 갱신 전의 데이터로 `SELECT` 를 사용할 때 이를 포함해 복수 쿼리를 일관된 형태의 한덩어리도 다뤄야 한다.
- **더 이상 나눌 수 없는 단위 작업**

    - 작업을 쪼개서 작은 단위로 만들 수 없다는 것은 트랜잭션의 핵심 속성인 원자성을 의미한다.
- MySQL에서는 2가지 존재
    - MyISAM : 트랜잭션 사용 불가 단순 구조
        - 간단하고 빠른 읽기 작업을 지원
    - InnoDB : 일반 DBMS와 같은 트랜잭션 구조 사용

ACID

원자성(Atomicity)이란?

- 원자 = 더 이상 쪼개질 수 없는 물질
- 원자성이란 데이터의 변경을 수반하는 **일련의 데이터 조작이 전부 성공할지 전부 실패할지를 보증**하는 구조이다.
- **시스템에서 한 트랜잭션의 연산들이 모두 완료되거나, 반대로 전혀 실행되지 않는 성질.**
- COMMIT, ROLLBACK 사용

Consistency 일관성

- **데이터 조작 전후에 그 상태를 유지하는 것을 보증하는 것을 의미.**
- 트랜잭션 수행 전/후에 데이터 모델의 모든 제약 조건(기본 키, 외래 키, 도메인, 도메인 제약조건 등)을 만족하는 것을 통해 보장한다.
- **한 쪽의 테이블에서만 데이터 변경사항이 이루어져서는 안된다.**
- **트리거 (Trigger)**
    - 트랜잭션의 일관성을 보장하기 위한 방법으로 어떤 이벤트와 조건이 발생했을 때 트리거를 통해 보장한다.
    - 트리거란 방아쇠란 뜻이며, DB 시스템이 자동적으로 수행할 동작을 명시할 때 사용된다. (행위의 시작을 알림)

Isolation 고립성

- **데이터 조작을 복수의 사용자가 동시에 실행해도 '각각의 처리가 모순없이 실행되는 것을 보증한다.' ⇒ 복수의 트랜잭션이 순서대로 실행되는 경우와 동일한 경과를 얻을 수 있는 상태**

- **하나의 트랜잭션 수행시 다른 트랜잭션의 작업이 끼어들지 못하도록 보장하는 것.**
  - 트랜잭션 끼리는 서로를 간섭할 수 없다.
- 트랜잭션이 실행하는 도중에 변경한 데이터는 이 트랜잭션이 완료될 때까지 다른 트랜잭션이 참조하지 못하도록 하는 특성.
  - 동기화
- DB에서는 트랜잭션의 고립성을 보장하기 위해 `잠금(Lock)`을 걸어서 후속 처리를 `블록(Block)`하는 방법이 있다.
  - **잠금 단위에는 테이블 전체, 블록, 행**이 있다.
  - **MySQL은 보통 행 단위의 잠금 기능을 사용한다.**
- 올바른 처리
  - `(1) 현재 빈 싱글룸의 수를 확인한다`를 처리할 때는 `select for update`를 실행하면 `select`한 행에 잠금이 걸리게 된다.
  - 이렇게 하면 후속 처리는 해당 잠금이 해체될 때 (`commit or rollback`)까지 대기하게 되며 올바른 처리를 계속할 수 있게 된다.

- 격리 수준
  - Serializable이 격리 수준이 제일 높고, 역순으로 낮아진다.
  - Serializable > Repeatable Read > Read Committed > Read Uncommitted

- 격리 수준의 완화에 따라 일어나는 현상
  - 더티 읽기 (Dirty Read), 애매한 읽기 (Fuzzy/NonRepeatable Read), 팬텀 읽기 (Phantom Read)

Durability 지속성

- 트랜잭션을 완료(COMMIT)하고 완료 통지를 사용자가 받는 시점에 그 트랜잭션이 영구적이 되어 그 결과를 잃지 않는 것.
    - 컴퓨터가 종료되거나 시스템 장애가 발생해도 계속 저장되는 성질 (RAM이 아닌 SSD에 저장된 상태)
- 지속성 실현
    - MySQL을 포함해 대부분 DBMS는 트랜잭션 조작을 하드 디스크에 "로그"로 기록하고 시스템에 이상이 발생하면 그 로그를 사용해 이상 발생 전의 상태까지 복원하는 것으로 지속성을 실현하고 있다. 이를 커밋 로그 관리라고 부른다

복수 커넥션에서 읽고 쓰기(MVCC)

> 복수의 커넥션(트랜잭션)이 어떻게 동시성을 제공하는지 알아보자

- MySQL은 DBMS의 주류가 된 **MVCC (Multi Versioning Concurrency Control)**라는 기술을 사용한다.
- MVCC?
    - MVCC는 데이터베이스의 동시성을 제어하기 위한 방법 중 하나.
    - 사용자는 데이터에 접근할 때 Snapshot을 읽어오며, 다른 사용자가 변경사항을 완료(commit)할 때까지 변경사항을 볼 수 없다.
    - MVCC는 일반적인 RDBMS보다 빠르게 작동하지만 사용하지 않는 데이터가 쌓이기 때문에 데이터 정리 시스템이 필요하며, 데이터 버전 충돌이 발생하면 애플리케이션에서 이를 해결해야 한다.
        - 데이터를 업데이트하면 이전 데이터를 덮어쓰는 대신, 새로운 버전의 데이터를 UNDO 영역에 생성하고 변경된 내용을 이전 버전의 데이터와 비교해서 기록. 이렇게 하면 여러 버전의 데이터가 하나의 데이터에 존재하게 되며, 사용자는 마지막 버전의 데이터를 읽게 됨.
- MVCC는 일반적인 RDBMS보다 빠르게 작동. 데이터를 읽을 때 다른 사람이 그 데이터를 삭제하거나 수정하더라도 영향을 받지 않는다. 그러나 사용하지 않는 데이터가 계속 쌓이기 때문에 데이터 ㅐ를 정리하는 시스템이 필요.
- MVCC 모델은 하나의 데이터에 대한 여러 버전을 허용하기 때문에 데이터 버전이 충돌. 이러한 문제를 애플리케이션에서 해결해야 한다.
- 또한, 추가 작업으로 인해 UNDO 블록 I/O, CR Copy 생성, CR 블록 캐싱 등의 오버헤드가 발생. 이러한 구조의 MVCC는 문장 수준과 트랜잭션 수준에서 일관성을 유지.

- MVCC에 따른 MySQL의 특성

  MySQL의 InnoDB에서는 Undo Log를 활용해 MVCC 기능을 구현.

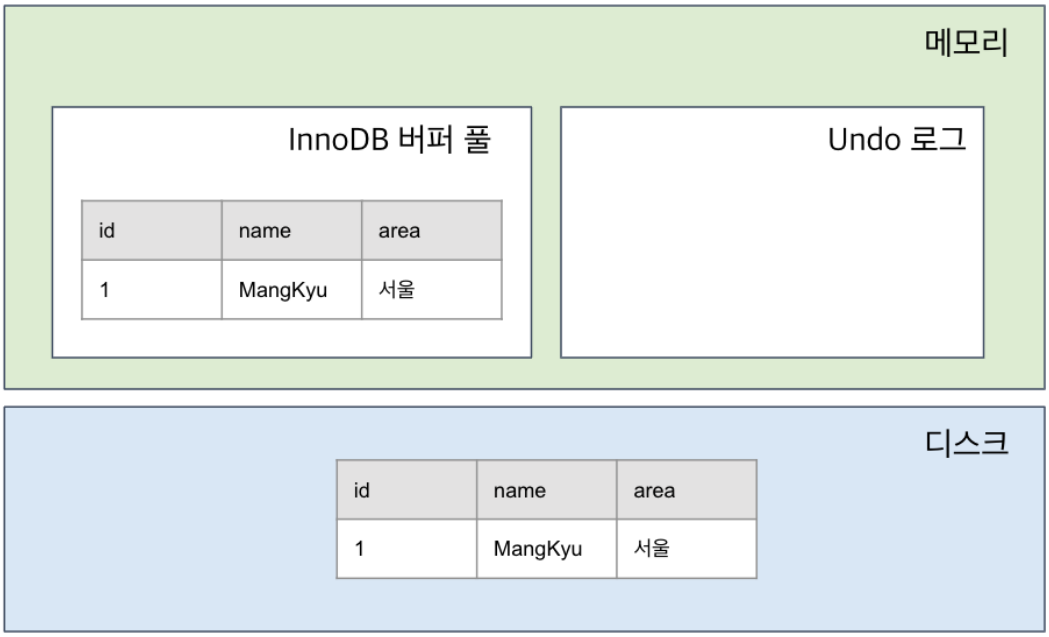
  예를 들어 아래와 같은 CREATE 쿼리문이 실행되었다고 하자.

```
CREATE TABLE member (
    id INT NOT NULL,
    name VARCHAR(20) NOT NULL,
    area VARCHAR(100) NOT NULL,
    PRIMARY KEY(m_id),
    INDEX idx_area(area)
)

INSERT INTO member(id, name, area) VALUES (1, "MangKyu", "서울");
```

  그러면 데이터는 다음과 같은 상태로 저장이 된다.

  메모리와 디스크에 모드 해당 데이터가 동일하게 저장되는 것.



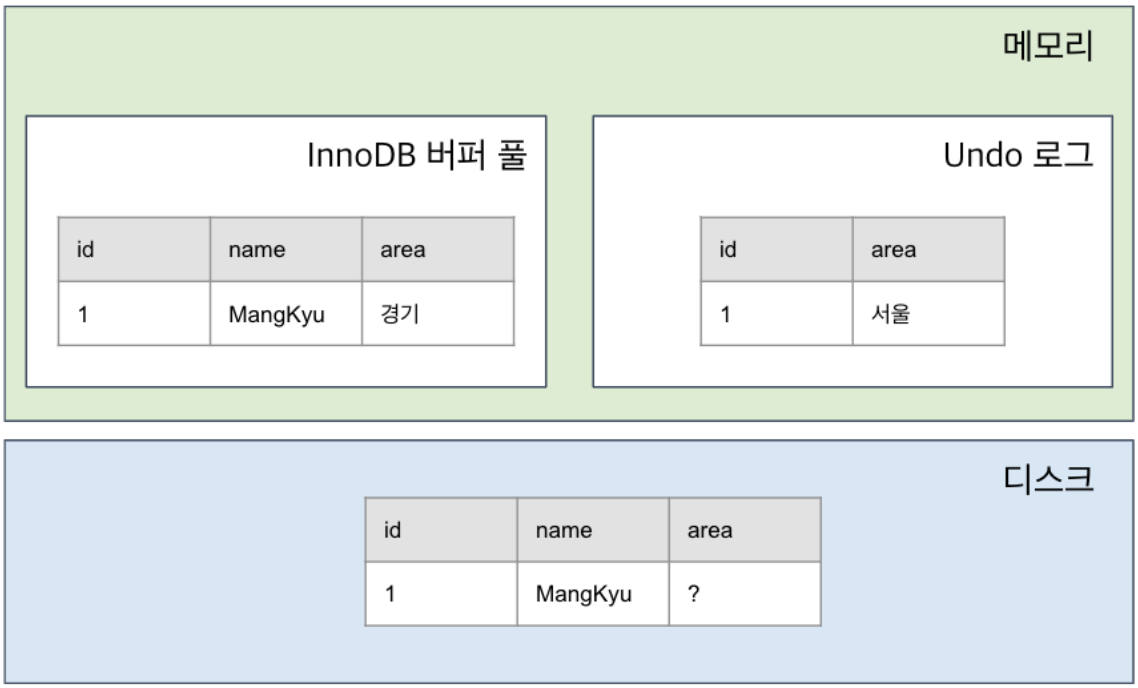
  그리고 다음과 같은 UPDATE 문을 실행시켰다고 하자.

```
UPDATE member SET area = "경기" WHERE id = 1;
```

  UPDATE 문이 실행된 결과를 그림으로 표현하면 다음과 같다.

  먼저 COMMIT 실행 여부와 무관하게 InnoDB 버퍼 풀은 새로운 값으로 갱신. 그리고 Undo 로그에는 변경 전의 값들만 복사.

그리고 InnoDB 버퍼 풀의 내용은 백그라운드 쓰레드를 통해 디스크에 기록되는데, 디스크에도 반영되었는지 여부는 시점에 따라 다를 수 있어서 ?로 표시



💡 Q. COMMIT이나 ROLLBACK이 호출되지 않은 상태에서 다른 사용자가 아래와 같은 쿼리로 데이터를 조회하면 어떤 결과가 반환될까?

```
SELECT * FROM member WHERE id = 1;
```

A. 그 결과는 트랜잭션의 격리 수준(Isolation Level)에 따라 다르다.

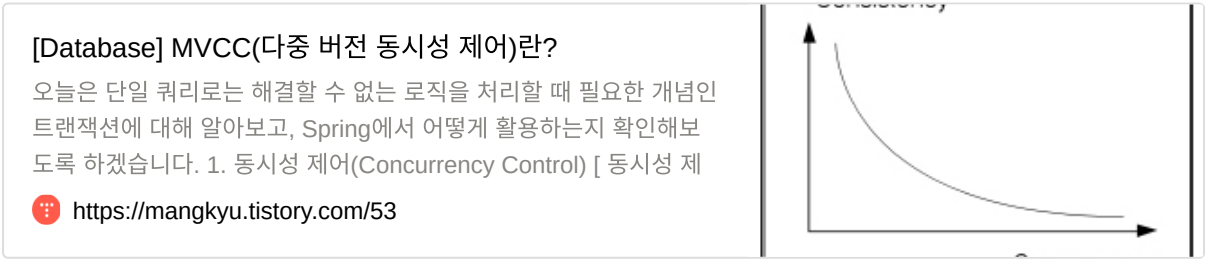
만약 커밋되지 않은 내용도 조회하도록 해주는 READ_UNCOMMITTED라면 버퍼 풀의 데이터를 읽어서 반환하며, 이는 커밋 여부와 무관하게 변경된 데이터를 읽어 반환하는 것이다.

만약 READ_COMMITED 이나 그 이상의 격리 수준(REPEATABLE_READ, SERIALIZABLE)이라면 **변경되기 이전의 Undo 로그 영역의 데이터를 반환하게 된다**. 이것이 가능한 이유는 하나의 데이터에 대해 여러 버전을 관리하는 MVCC 덕분이다.

여기서 Undo Log 영역의 데이터는 커밋 혹은 롤백을 호출하여 InnoDB 버퍼풀도 이전의 데이터로 복구되고, 더 이상 Undo 영역을 필요로 하는 트랜잭션이 더는 없을 때 비로

소 삭제된다.

- 트랜잭션 격리 수준별 외관
  - MVCC의 특성 중 격리수준과 연관(`읽기 내용은 격리 수준에 따라 내용이 바뀌는 경우가 있다.`)
    - 반복 읽기 (REPEATABLE_READ)
    - 커밋된 읽기 (READ_UNCOMMITTED)



잠금 타임아웃과 교착 상태

- `잠금 타임아웃`이란?
  - 갱신과 갱신이 부딪치는 경우에는 나중에 온 갱신이 잠금 대기 상태가 된다.
    잠금을 건 쪽이 언제 잠금을 풀지 알 수 없어서 잠금 대기 상태가 된다.
  - 이때 얼마나 기다릴 지를 설정하는데 이 시간을 지나면 **잠금 타임아웃이 발생한다.**
    - MySQL > innodb_lock_wait_timeout
  - 타임아웃이 발생했을 때 롤백하는 방법은 두 가지이다.
    - 트랜잭션 전체를 롤백하는 경우
    - 쿼리만 롤백하는 경우
- `교착 상태`란?
  - 트랜잭션 A가 테이블 a의 잠금을 얻고, 트랜잭션 B가 테이블 b의 잠금을 얻었다고 해보자.
  - 이 잠금을 유지한 채 서로 잠금을 건 자원에 잠금이 필요한 처리를 실행하면 아무리 기다려도 상황이 바뀌지 않는다.
  - 이를 교착 상태라 한다.

- 교착상태의 빈도를 낮추는 대책

  잠금 타임아웃은 일정 시간 기다리면 상황이 개선될 가능성이 있지만, **교착 상태는 상황이 개선될 가능성이 없습니다.** 이 때문에 일반적인 **DBMS에서는 교착 상태를 독자적으로 검출해 교착 상태를 보고**합니다.

  MySQL도 교착 생태가 일어나면 이를 즉시 인식해 시스템에 영향이 작은 쪽의 트랜잭션을 트랜잭션 개시 시점까지 롤백합니다.

  교착 상태는 일반적인 데이터베이스에서 **발생할 가능성이 있으므로 모든 것을 업앨 수는 없습니다.** 따라서 **애플리케이션 쪽에서는 항상 트랜잭션이 교착 상태를 일으켜 롤백되는 경우 트랜잭션을 재실행 할 수 있는 구조로 만들어야 합니다.**

- DBMS 전반적인 대책
  1. **트랜잭션을 자주 커밋한다.** 이에 따라 **트랜잭션은 더 작은 단위가 되어 교착 상태의 가능성을 낮춘다.**
  2. 정해진 순서로 테이블 순서에 엑세스하게 한다.
  3. 필요 없는 경우에는 읽기 잠금 획득 (SELECT ~ FOR UPDATE 등)의 사용을 피한다.
  4. 쿼리에 의한 잠금 범위를 더 좁히거나 잠금 정도를 더 작은 것으로 한다.(Isolation Level을 낮춘다.)
  5. 한 테이블의 복수 행을 복수의 연결에서 순서 변경 없이 갱신하면 교착 상태가 발생한다. 동시에 많은 연결에서 갱신 때문에 교착 상태가 자주 발생한다면 테이블 단위의 잠금을 회득해 갱신을 직렬화하면 동시선은 떨어지지만 교착 상태는 피할 수 있어서 전체 처리로 보면 좋은 예도 있다.
- MySQL(InnoDB)의 대책
  - 테이블에 적절한 인덱스를 추가해 쿼리가 이를 이용하게 한다.
  - 인덱스가 사용하지 않은 경우, 필요한 행의 잠금이 아닌 스캔한 행 전체에 대해 잠금이 걸리게 된다.

해서는 안되는 트랜잭션 처리

1. 오토 커밋

MySQL에서는 새로운 연결은 모두 기본값으로 오토커밋이 실행된다. 애플리케이션의 잠금을 실행하는 데는 커밋의 부하가 너무 높다. 일정 수 이상의 갱신을 수행하는 처리나 트랜잭션의 기능등은 적절한 단위와 트랜잭션 처리 수준에서 트랜잭션을 이용하지 않도록 한다.

2. 긴 트랜잭션
    - **긴 트랜잭션은 데이터베이스 트랜잭션의 동시성이나 자원의 유효성을 저하한다.** 갱신을 포함한 트랜잭션은 같은 테이블과 행을 갱신하려는 다른 트랜잭션을 블록하고 이것이 장시간 이어지면 블록된 트랜잭션을 타임아웃시킨다.
    - 추가적인 주의사항 → 최대한 트랜잭션을 작게 수행
        - 대량의 처리를 한 개의 트랜잭션이 수행
            - 대량의 갱신 처리 → 대량의 UNDO 로그 트랜잭션 종료까지 유지 → OS 파일시스템 내에 남음
        - 아무 것도 하지 않는 트랜잭션
        - 트랜잭션 중 대화 처리 삽입
            - 사용자의 처리를 끝없이 기다리게 됨, 불가피한 경우 → 상한 삽입
        - 처리 능력 이상의 트랜잭션 수
            - 동시 실행 커넥션 수 상한 설정을 부하 실험을 통해 측정