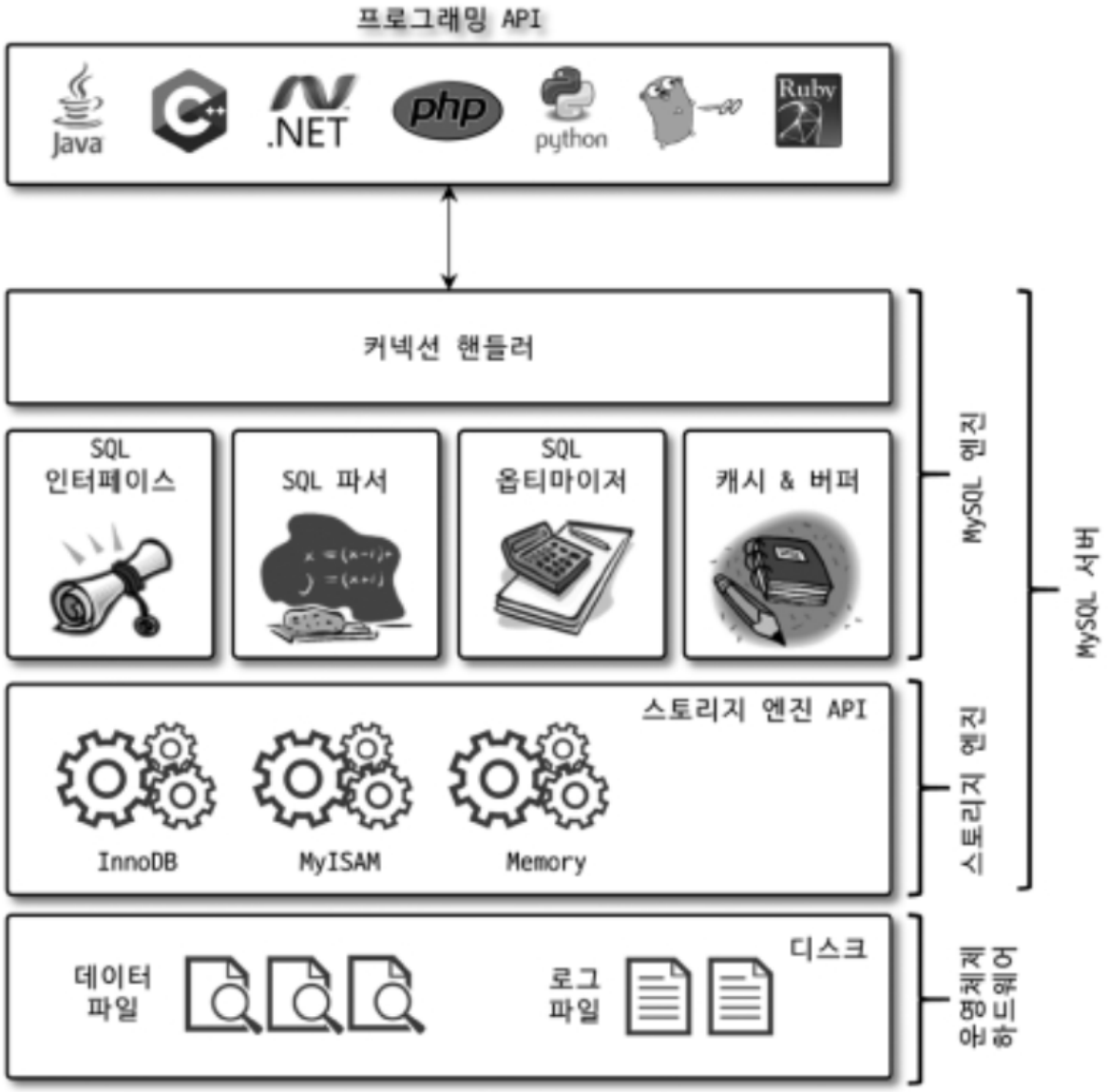
4. 아키텍처

↗ Books	Real MySQL 8.0

MySQL 서버는 MySQL 엔진과 스토리지 엔진으로 구분할 수 있다.

이 때, 핸들러 API를 만족하면 스토리지 엔진을 구현해서 MySQL 서버에 추가해서 사용할 수 있다.

1. MySQL 엔진 아키텍처



일반 상용 RDBMS와 같이 대부분의 프로그래밍 언어로부터 접근 방법을 모두 지원한다.

1. MySQL 엔진과 스토리지 엔진

- **MySQL 엔진**
  클라이언트로부터의 접속 및 쿼리 요청을 처리하는 커넥션 핸들러, SQL 파서 및 전처리기, 쿼리의 최적화된 실행을 위한 옵티마이저가 중심을 이룬다.

MySQL 엔진이 머리 역할을 한다는 것은 옵티마이저가 큰 비중을 차지한다. 실행 계획에 대해 잘 이해해야만 더 최적화된 방법으로 실행 계획을 수립할 수 있도록 유도할 수 있기 때문이다.

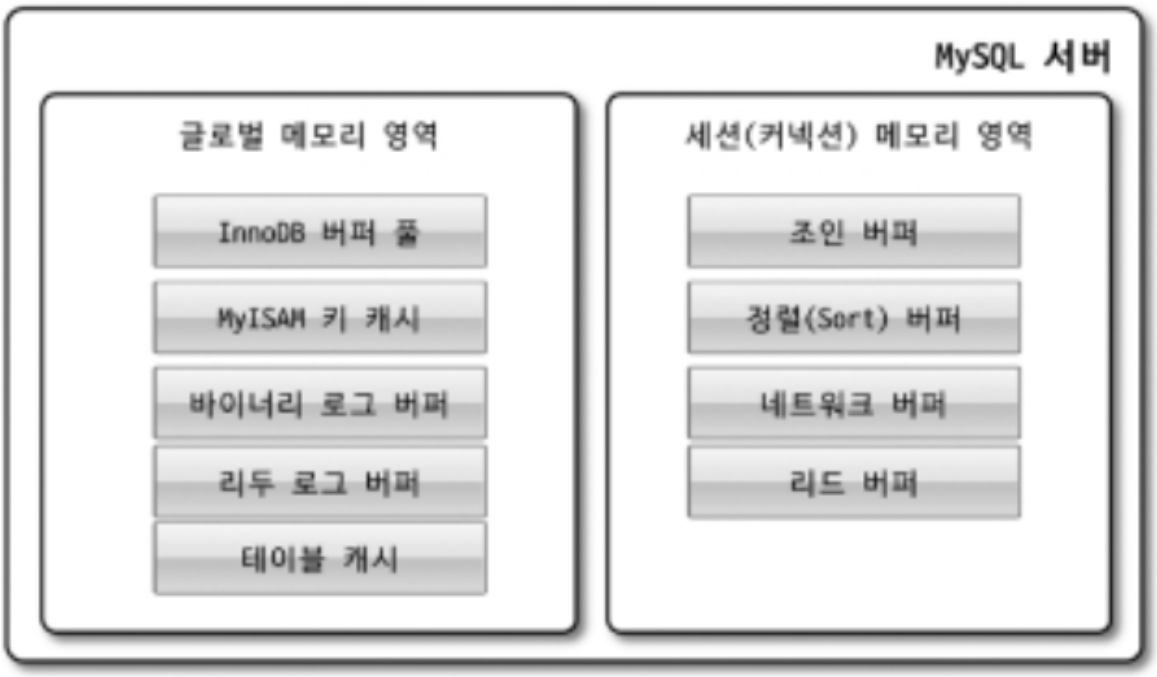
- **스토리지 엔진**
  실제 데이터를 저장하거나 읽어오는 부분을 전담하며 여러 개를 동시에 사용하는 것이 가능하다.
  데이터를 쓰거나 읽어야 할 때 스토리지 엔진에 보내는 요청을 핸들러(Handler) 요청이라 하고, 여기서 사용되는 API를 핸들러 API라고 한다.

2. MySQL 스레딩 구조

MySQL 서버는 스레드 기반으로 작동하며 크게 포그라운드(Foreground) 스레드와 백그라운드(Background) 스레드로 구분할 수 있다.

- **포그라운드 스레드(클라이언트 스레드)**
  최소한 MySQL 서버에 접속된 클라이언트 수만큼 존재하며 주로 요청된 쿼리 문장을 처리한다.
  요청이 끝나고 커넥션을 종료하면 스레드 캐시(Thread cache)로 되돌아가지만 스레드 캐시에 이미 일정 개수 이상의 대기 중인 스레드가 있을 경우 스레드를 종료시켜 스레드 캐시에 일정 개수만 유지하도록 한다.
- **백그라운드 스레드**
  MyISAM는 별로 해당 사항이 없는 부분이지만 InnoDB는 여러 작업이 백그라운드로 처리된다.
  가장 중요한 것은 로그 스레드와 쓰기 스레드다. MySQL 5.5 버전부터 쓰기 스레드와 읽기 스레드 개수를 2개 이상 지정할 수 있게 되었다.
  InnoDB에서도 데이터를 읽는 작업은 주로 클라이언트 스레드에서 처리되기 때문에 읽기 스레드는 많이 설정할 필요가 없지만 쓰기 스레드는 일반적인 내장 디스크를 사용할 경우 2~4 정도, DAS나 SAN과 같은 스토리지를 사용할 경우 디스크를 최적으로 사용할 수 있을 만큼 충분히 설정하는 것이 좋다.

3. 메모리 할당 및 사용 구조



MySQL에서 사용되는 메모리 공간은 크게 글로벌 메모리 영역과 로컬 메모리 영역으로 구분할 수 있다.

- **글로벌 메모리 영역**
  MySQL 서버가 시작될 때 운영체제로부터 할당된다.
  일반적으로 하나의 메모리 공간만 할당된다.
- **로컬 메모리 영역**
  MySQL 서버상에 존재하는 클라이언트 스레드가 쿼리를 처리하는 데 사용하는 메모리 영역이다.
  클라이언트가 MySQL 서버에 접속하면 스레드를 하나씩 할당하며 절대 공유되어 사용되지 않는다는 특징이 있다.

4. 플러그인 스토리지 엔진 모델

MySQL은 검색어 파서, 인증 방식, 여러 스토리지 엔진 등의 많은 것들이 플러그인 형태로 구현되어 있다. 또한 필요에 따라 직접 스토리지 엔진을 개발해서 사용하는 것도 가능하다.

쿼리가 실행되는 과정에서 데이터 읽기/쓰기 작업만 스토리지 엔진에 의해 처리되기 때문에 스토리지 엔진을 개발해서 사용하더라도 DBMS의 전체 기능이 아니라 일부 기능만 수행하게 된다.

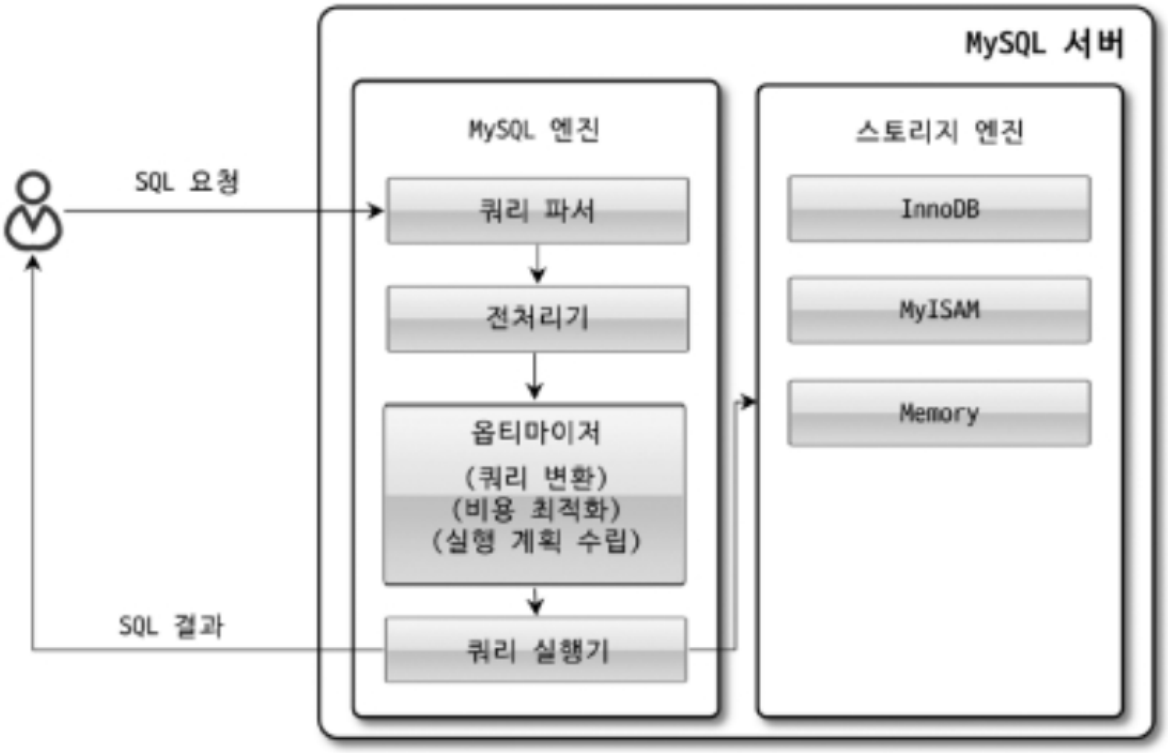
5. 컴포넌트

플러그인 아키텍처의 큰 단점은 아래와 같다.

- 플러그인은 오직 MySQL 서버와 인터페이스할 수 있고, 플러그인끼리는 통신할 수 없음
- 플러그인은 MySQL 서버의 변수나 함수를 직접 호출하기 때문에 안전하지 않음(캡슐화 안 됨)
- 플러그인은 상호 의존 관계를 설정할 수 없어서 초기화가 어려움

이러한 단점 때문에 MySQL 8.0부터는 기존의 플러그인 아키텍처를 대체하기 위한 컴포넌트 아키텍처가 지원된다.

6. 쿼리 실행 구조



- **쿼리 파서**
  사용자 요청으로 들어온 쿼리 문장을 토큰으로 분리해 트리 형태의 구조로 만들어내는 작업을 한다.
  쿼리 문장의 기본적인 문법 오류는 이 과정에서 발견된다.
- **전처리기**
  파서 트리를 기반으로 쿼리 문장에 구조적인 문제점이 있는지 확인한다.

해당 객체의 존재 여부와 객체의 접근 권한 등을 확인한다.

- **옵티마이저**

  사용자의 요청으로 들어온 쿼리 문장을 저렴한 비용으로 가장 빠르게 처리할지를 결정하는 역할을 담당한다.

- **실행 엔진**
  만들어진 계획대로 각 핸들러에게 요청해서 받은 결과를 또 다른 핸들러 요청의 입력으로 연결하는 역할을 수행한다.

- **핸들러(스토리지 엔진)**
  MySQL 실행 엔진의 요청에 따라 데이터를 디스크로 저장하고 읽어오는 역할을 담당한다.

7. 쿼리 캐시

쿼리 캐시는 SQL 실행 결과를 저장하고 동일 쿼리가 실행되었을 때 즉시 결과를 반환해주는 역할을 한다.

매우 빠른 성능을 보이지만 테이블의 데이터가 변경되면 변경된 테이블과 관련된 것들은 모두 삭제해야 하기 때문에 심각한 동시 처리 성능 저하를 유발한다.

결국 MySQL 8.0으로 올라오면서 제거된 기능이다.

8. 스레드 풀

MySQL 서버 엔터프라이즈 에디션에서 제공되는 기능으로 커뮤니티 에디션에서 사용하고 싶다면 동일 버전의 Percona Server에서 라이브러리를 설치해서 사용할 수 있다.

Percona Server는 MySQL 데이터베이스 관리 시스템의 개량된 버전의 오픈소스 소프트웨어다.
MySQL 커뮤니티 버전의 기능을 향상시키고 추가적인 성능 최적화를 제공한다.
성능 향상의 목적이라면 MySQL과 호환성이 좋기 때문에 더 공부해서 사용해보는 것도 나쁘지 않을 것 같다.

스레드 풀은 내부적으로 사용자의 요청을 처리하는 스레드 개수를 줄여서 동시 처리되는 요청이 많다 하더라도 MySQL 서버의 CPU가 제한된 개수의 스레드 처리에만 집중할 수 있게 해서 서버의 자원 소모를 줄이는 것이 목적이다.

하지만 실제 서비스에서 눈에 띄는 성능 향상을 보여준 경우는 드물고 스케줄링 과정에서 CPU 시간을 제대로 확보하지 못하는 경우 쿼리 처리가 더 느려지는 사례도 발생할 수 있다.

9. 트랜잭션 지원 메타데이터

데이터베이스 서버에서 테이블의 구조 정보와 스토어드 프로그램 등의 정보를 데이터 딕셔너리 또는 메타데이터라고 하는데, MySQL 5.7버전까지는 테이블의 구조를 FRM 파일에 저장하고 일부 스토어드 프로그램 또한 파일 기반으로 관리했다.

하지만 이러한 파일 기반의 메타데이터는 생성 및 변경 작업이 트랜잭션을 지원하지 않기 때문에 테이블의 생성 또는 변경 도중에 MySQL 서버가 비정상적으로 종료되면 일관되지 않은 상태로 남는 문제가 있었다. (=데이터베이스나 테이블이 깨졌다.)

MySQL 8.0 버전부터는 데이터 딕셔너리 정보를 모두 InnoDB 테이블에 저장하도록 개선했다.

또한 사용자 인증과 권한 등의 서버가 작동하는 데 기본적으로 필요한 테이블(시스템 테이블)도 모두 InnoDB 스토리지 엔진을 사용하도록 개선했다.

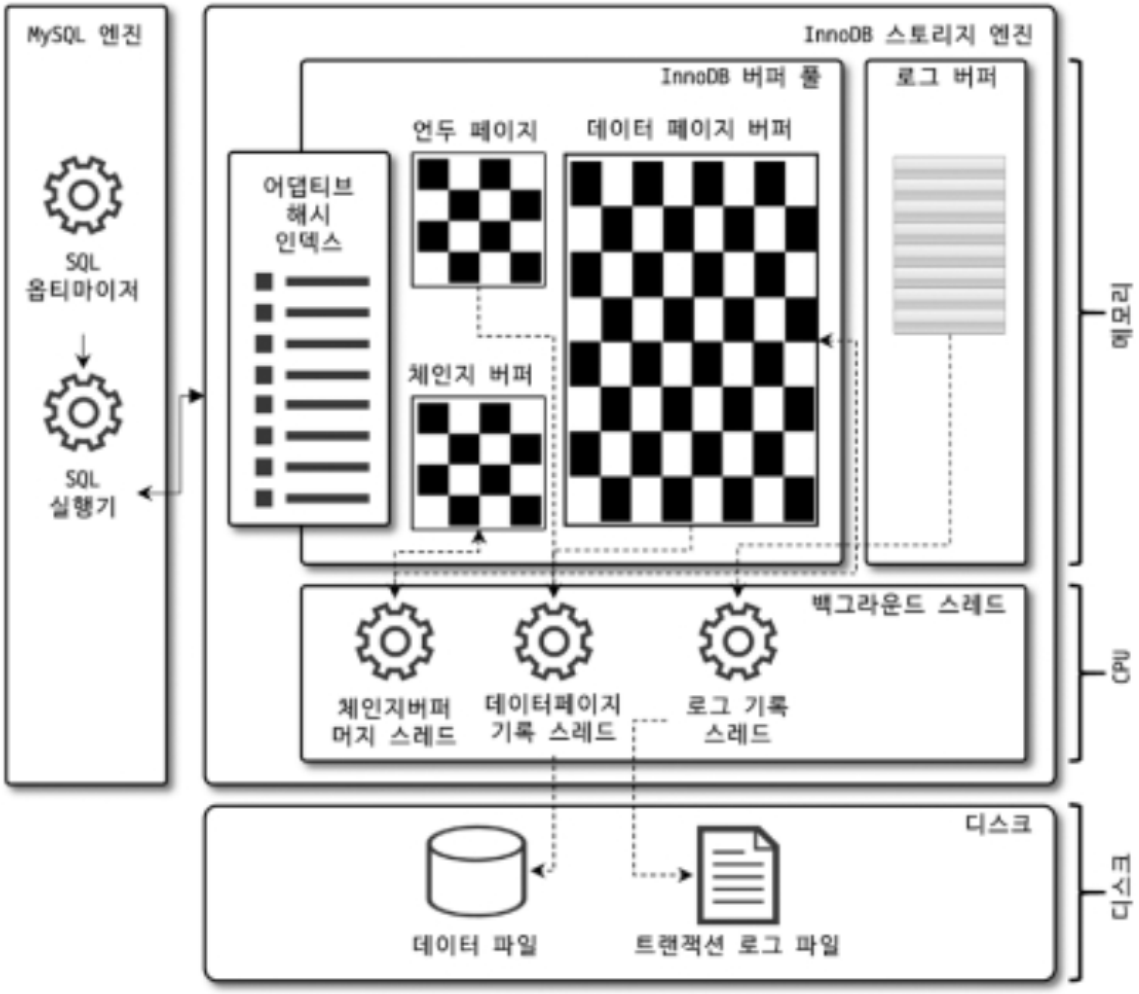
시스템 테이블과 데이터 딕셔너리 정보를 모두 모아서 mysql DB에 저장하고 있다.

mysql DB는 통째로 `mysql.ibd`라는 이름의 테이블스페이스에 저장되기 때문에 각별히 주의해야 한다.

이렇게 개선되면서 이제 스키마 변경 작업 중간에 MySQL 서버가 비정상적으로 종료된다고 하더라도 문제가 생기지 않게 되었다.

---

2. InnoDB 스토리지 엔진 아키텍처



InnoDB는 MySQL에서 사용할 수 있는 스토리지 엔진 중 거의 유일하게 레코드 기반의 잠금을 제공하며, 그 때문에 높은 동시성 처리가 가능하고 안정적이며 성능이 뛰어나다.

1. 프라이머리 키에 의한 클러스터링

InnoDB의 모든 테이블은 기본적으로 프라이머리 키를 기준으로 클러스터링되어 저장된다. 즉, 프라이머리 키 값의 순서대로 디스크에 저장되며 모든 세컨더리 인덱스는 레코드의 주소 대신 프라이머리 키의 값을 논리적인 주소로 사용한다.

프라이머리 키를 이용한 레인지 스캔은 상당히 빨리 처리될 수 있으며 이러한 이유 때문에 쿼리 실행 계획에서 프라이머리 키는 기본적으로 다른 보조 인덱스에 비해 비중이 높게 설정된다.

MyISAM 스토리지 엔진에서는 클러스터링 키를 지원하지 않는다.

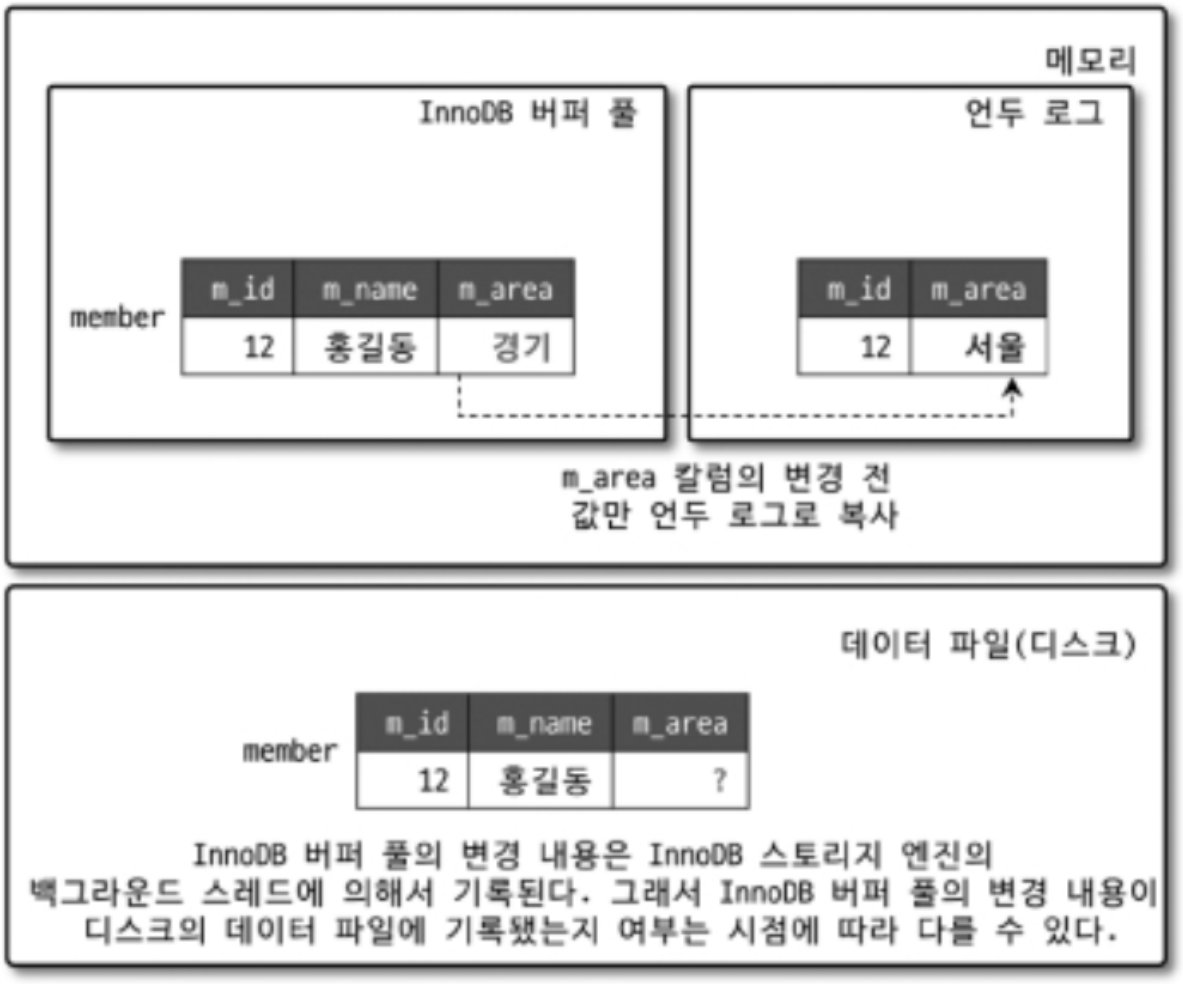
2. 외래 키 지원

사용자가 외래 키를 생성하지 않는 경우 InnoDB 자체적으로 인덱스를 생성하지만, 이렇게 생성된 키는 사용자가 접근할 수 없기 때문에 개발할 때 외래 키의 존재에 주의하는 것이 좋다.

3. MVCC(Multi Version Concurrency Control)

MVCC의 가장 큰 목적은 잠금을 사용하지 않는 일관된 읽기를 제공하는 데 있다.

InnoDB는 언두 로그(Undo log)를 이용해 이 기능을 구현한다.



값이 변경되면 변경되기 이전의 값은 언두 로그에 저장하고 버퍼 풀에는 새로운 값으로 업데이트된다. 커밋을 하고 언두 영역을 필요로 하는 트랜잭션이 더는 없을 때 삭제된다.

아직 커밋이나 롤백이 되지 않은 상태에서 이 데이터를 조회한다면 트랜잭션 격리 수준에 따라 다르게 값이 반환된다.

이처럼 하나의 트랜잭션에서 데이터에 접근하는 경우 데이터의 다중버전 상태 중 보장되는 버전에 맞는 값을 반환하여 처리하는 방법을 MVCC라고 한다.

4. 잠금 없는 일관된 읽기(Non-Locking Consistent Read)

InnoDB 스토리지 엔진은 MVCC 기술을 이용해 잠금을 걸지 않고 읽기 작업을 수행한다.

잠금을 걸지 않기 때문에 다른 트랜잭션의 변경 작업과 관계 없이 읽기 작업이 가능하다.

이렇게 일관된 읽기를 위해 언두 로그를 계속 유지해야 하기 때문에 트랜잭션이 시작됐다면 가능한 한 빨리 롤백이나 커밋을 통해 트랜잭션을 완료하는 것이 좋다.

5. 자동 데드락 감지

InnoDB 스토리지 엔진은 데드락 감지 스레드를 가지고 있어서 주기적으로 교착 상태에 빠진 트랜잭션들을 찾아서 그중 하나를 강제 종료한다.

6. 자동화된 장애 복구

MySQL 서버가 시작될 때 완료되지 못한 트랜잭션이나 디스크에 일부만 기록된 데이터 페이지 등에 대한 일련의 복구 작업이 자동으로 진행된다.

디스크나 서버 하드웨어 이슈로 자동 복구를 못 하는 경우도 발생할 수 있는데 MySQL 서버의 설정 파일에 `innodb_force_recovery` 시스템 변수를 설정해서 서버를 시작해야 한다.

7. InnoDB 버퍼 풀

InnoDB 스토리지 엔진에서 가장 핵심적인 부분으로, 디스크의 데이터 파일이나 인덱스 정보를 메모리에 캐시해 두는 공간이다. 쓰기 작업을 지연시켜 일괄 작업으로 처리할 수 있게 해주는 버퍼 역할도 같이 한다.

- **버퍼 풀의 크기 설정**
  MySQL 5.7 버전부터는 InnoDB 버퍼 풀의 크기를 동적으로 조절할 수 있는데 버퍼 풀의 크기를 줄이는 작업은 서비스 영향도가 매우 크기 때문에 가능한 적절히 작은 값으로 설정하고 상황을 봐 가면서 증가시키는 방법이 최적이다.
- **버퍼 풀의 구조**
  버퍼 풀이라는 메모리 공간을 페이지 크기의 조각으로 쪼개어 InnoDB 스토리지 엔진이

데이터를 필요로 할 때 해당 데이터 페이지를 읽어서 각 조각에 저장한다.
페이지 크기 조각을 관리하기 위해 InnoDB 스토리지 엔진은 크게 LRU 리스트와 플러시 리스트, 프리 리스트라는 3개의 자료 구조를 관리한다.

💡 버퍼 풀에 대해서 제대로 이해가 가지 않아서 정리를 하지 못 했다🫠

8. Double Write Buffer

InnoDB 스토리지 엔진의 리두 로그는 리두 로그 공간의 낭비를 막기 위해 페이지의 변경된 내용만 기록한다. 이로 인해 더티 페이지를 디스크 파일로 플러시할 때 일부만 기록되는 문제가 발생하면 그 페이지의 내용은 복구할 수 없을 수도 있다.

이렇게 페이지가 일부만 기록되는 현상을 파셜 페이지(Partial-page) 또는 톤 페이지(Torn-page)라고 하는데 이러한 문제를 막기 위해 Double-Write 기법을 이용한다.

9. 언두 로그

InnoDB 스토리지 엔진은 변경되기 이전 버전의 데이터를 별도로 백업하는데, 이 데이터를 언두 로그(Undo Log)라고 한다.

언두 로그의 데이터는 크게 두 가지 용도로 사용된다.

1. 트랜잭션의 롤백 대비용
2. 트랜잭션의 격리 수준을 유지하면서 높은 동시성을 제공

MySQL 5.5 이전 버전에서는 언두 로그 공간은 다시 줄어들지 않았지만 이후 버전부터 공간의 문제점이 완전히 해결됐다. 하지만 여전히 서비스 중인 MySQL 서버에서 활성 상태의 트랜잭션이 장시간 유지되는 것은 성능상 좋지 않기 때문에 항상 모니터링하는 것이 좋다.

언두 로그가 저장되는 공간을 언두 테이블스페이스(Undo Tablespace)라고 한다. MySQL 8.0 버전부터는 언두 테이블스페이스를 동적으로 추가하고 삭제할 수 있게 되었다. 또한 언두 테이블스페이스의 불필요한 공간을 잘라내는 방법은 자동과 수동 두 가지가 있는데 MySQL 8.0부터는 두 가지 방법 모두 지원된다.

10. 체인지 버퍼

RDBMS에서 레코드가 삽입 혹은 수정될 때 해당 테이블에 포함된 인덱스를 업데이트하는 작업도 필요한데, 테이블에 인덱스가 많다면 상당히 많은 자원이 소모된다.

InnoDB는 변경해야 할 인덱스 페이지가 버퍼 풀에 있으면 바로 업데이트를 수행하지만 디스크로부터 읽어와서 업데이트해야 한다면 이를 즉시 실행하지 않고 임시 공간에 저장해 두고 바로 사용자에게 결과를 반환하는 형태로 성능을 향상시키게 된다. 이때 사용하는 임시 메모리 공간을 체인지 버퍼(Change Buffer)라고 한다.

11. 리두 로그 및 로그 버퍼

리두 로그는 하드웨어나 소프트웨어 등 여러가지 문제점으로 인해 MySQL 서버가 비정상적으로 종료됐을 때 데이터 파일에 기록되지 못한 데이터를 잃지 않게 해주는 안전장치다.

거의 모든 DBMS에서 데이터 파일은 쓰기보다 읽기 성능을 고려한 자료 구조를 가지고 있기 때문에 변경된 데이터를 기록하는 것은 상대적으로 큰 비용이 필요하다. 이로 인한 성능 저하를 막기 위해 쓰기 비용이 낮은 자료 구조를 가진 리두 로그를 가지고 있다.

MySQL 8.0 버전부터 InnoDB 스토리지 엔진의 리두 로그를 아카이빙할 수 있는 기능이 추가됐다. 데이터 변경이 많아서 리두 로그가 덮어쓰인다고 하더라도 백업이 실패하지 않게 해주는 기능이다.

12. 어댑티브 해시 인덱스

어댑티브 해시 인덱스는 사용자가 수동으로 생성하는 인덱스가 아닌 InnoDB 스토리지 엔진에서 사용자가 자주 요청하는 데이터에 대해 자동으로 생성하는 인덱스다.

B-Tree 검색 시간을 줄여주기 위해 도입된 기능으로 자주 읽히는 데이터 페이지의 키 값을 이용해 해시 인덱스를 만들고, 필요할 때마다 어댑티브 해시 인덱스를 검색해서 레코드가 저장된 데이터 페이지를 즉시 찾아갈 수 있다.
B-Tree를 루트 노드부터 리프 노드까지 찾아가는 비용이 없어지고 그만큼 CPU는 적은 일을 하지만 쿼리의 성능은 빨라진다.

하지만 다음과 같은 경우 성능 향상에 크게 도움이 되지 않기 때문에 의도적으로 비활성화하는 경우도 많다.

- 디스크 읽기가 많은 경우
- 특정 패턴의 쿼리가 많은 경우(조인이나 LIKE 패턴 검색)

- 매우 큰 데이터를 가진 테이블의 레코드를 폭넓게 읽는 경우

다음과 같은 경우 성능 향상에 많은 도움이 된다.

- 디스크의 데이터가 InnoDB 버퍼 풀 크기와 비슷한 경우(디스크 읽기가 많지 않은 경우)
- 동등 조건 검색(동등 비교와 IN 연산자)이 많은 경우
- 쿼리가 데이터 중에서 일부 데이터에만 집중되는 경우

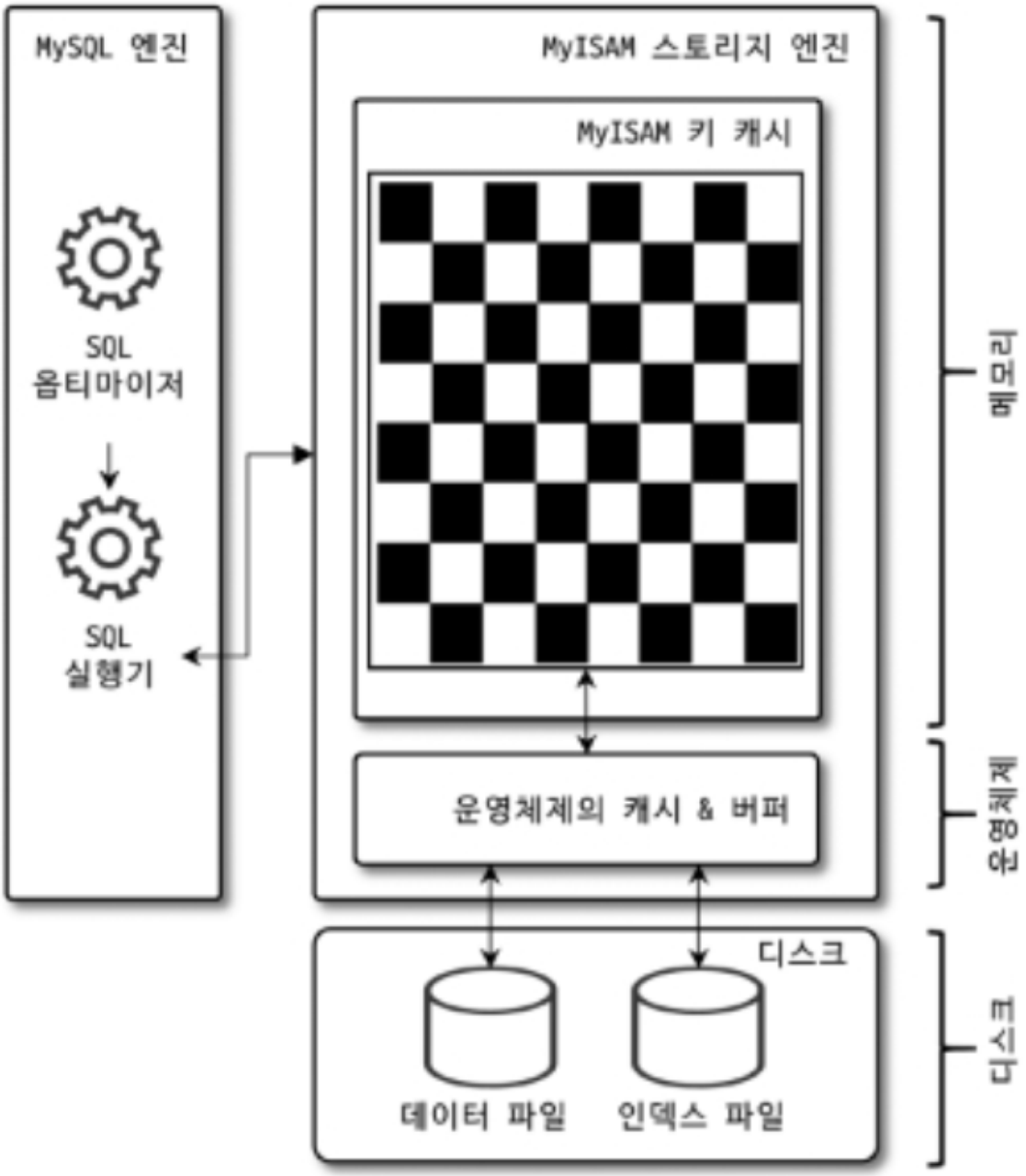
어댑티브 해시 인덱스는 테이블의 삭제 작업에도 많은 영향을 미친다. 어댑티브 해시 인덱스의 도움을 많이 받을수록 테이블 삭제 또는 변경 작업은 더 치명적인 작업이 되는 것이다.

13. InnoDB와 MyISAM, MEMORY 스토리지 엔진 비교

지금까지는 MyISAM이 기본 스토리지 엔진으로 사용되는 경우가 많았지만 MySQL 5.5부터는 InnoDB 스토리지 엔진이 기본으로 채택되었다. (시스템 테이블은 여전히 MyISAM 테이블 사용)
MySQL 8.0으로 업그레이드 되면서 MySQL 서버의 모든 시스템 테이블이 InnoDB 스토리지 엔진으로 교체되었고, 공간 좌표 검색이나 전문 검색 기능이 모두 InnoDB 스토리지 엔진을 지원하도록 개선됐다.

---

3. MyISAM 스토리지 엔진 아키텍처



1. 키 캐시

InnoDB의 버퍼 풀과 비슷한 역할을 하지만 인덱스만을 대상으로 작동하며, 인덱스의 디스크 쓰기 작업에 대해서만 부분적으로 버퍼링 역할을 한다.

2. 운영체제의 캐시 및 버퍼

MyISAM 테이블의 데이터에 대해서는 디스크로부터의 I/O를 해결해 줄 만한 어떠한 캐시나 버퍼링 기능도 MyISAM 스토리지 엔진은 가지고 있지 않다.

3. 데이터 파일과 프라이머리 키(인덱스) 구조

MyISAM 테이블은 프라이머리 키에 의한 클러스터링 없이 데이터 파일이 힙(Heap) 공간처럼 활용된다.

즉, 프라이머리 키 값과 무관하게 INSERT되는 순서대로 데이터 파일에 저장된다.

---

4. MySQL 로그 파일

로그 파일을 이용하면 MySQL의 상태나 부하를 일으키는 원인을 쉽게 찾아서 해결할 수 있다.

1. 에러 로그 파일

MySQL이 실행되는 도중에 발생하는 에러나 경고 메시지가 출력되는 로그 파일이다.

- MySQL이 시작하는 과정과 관련된 정보성 및 에러 메시지
- 마지막으로 종료할 때 비정상적으로 종료된 경우 나타나는 InnoDB의 트랜잭션 복구 메시지
- 쿼리 처리 도중에 발생하는 문제에 대한 에러 메시지
- 비정상적으로 종료된 커넥션 메시지
- InnoDB의 모니터링 또는 상태 조회 명령의 결과 메시지
- MySQL의 종료 메시지

2. 제너럴 쿼리 로그 파일

쿼리 로그 파일에는 시간 단위로 실행됐던 쿼리의 내용이 모두 기록된다. 슬로우 쿼리 로그와는 다르게 MySQL이 쿼리 요청을 받으면 바로 기록하기 때문에 실행 중에 에러가 발생해도 일단 로그 파일에 기록된다.

`general_log_file`이라는 이름의 파라미터에 설정돼 있다.

3. 슬로우 쿼리 로그

서비스에서 사용되는 쿼리 중에서 어떤 쿼리가 문제인지를 판단하는 데 슬로우 쿼리 로그가 상당히 많은 도움이 된다.

시스템 변수에 설정한 시간 이상의 시간이 소요된 쿼리가 모두 기록된다. 즉, 실제 소요된 시간을 기준으로 기록할지 여부를 판단하기 때문에 쿼리가 정상적으로 실행되었을 때만 기록된다.