4.1 MySQL 엔진 아키텍쳐

❗ MySQL 엔진 영역과 스토리지 엔전 영역의 차이

❗ 하나의 쿼리 작업에 대한 각 하위 작업이 어느 영역에서 처리되는지 구분

Chapter 4. 아키텍쳐

MySQL 서버

- MySQL 엔진: 머리
- 스토리지 엔진: 손발. 핸들러 API를 만족하면 누구든 스토리지 엔진을 구현해서 MySQL 서버에 추가해서 사용 가능하다. 아래 두 엔진은 MySQL 서버에서 기본으로 제공된다.
  - InnoDB 스토리지 엔진
  - MyISAM 스토리지 엔진

비교 항목	InnoDB	MyISAM
트랜잭션 지원	O (지원)	X (미지원)
잠금 방식	행 단위 잠금	테이블 단위 잠금
외래 키(Foreign Key)	O (지원)	X (미지원)
충돌 복구(Crash Recovery)	O (지원)	X (미지원)
읽기 속도	느릴 수 있음	빠름
쓰기 속도	빠름 (동시 처리 가능)	느릴 수 있음 (전체 테이블 잠김)
사용 사례	금융, 쇼핑몰, 회원 관리 등 데이터 무결성이 중요한 경우	게시판, 로그 저장 등 읽기 위주의 작업

4.1 MySQL 엔진 아키텍쳐

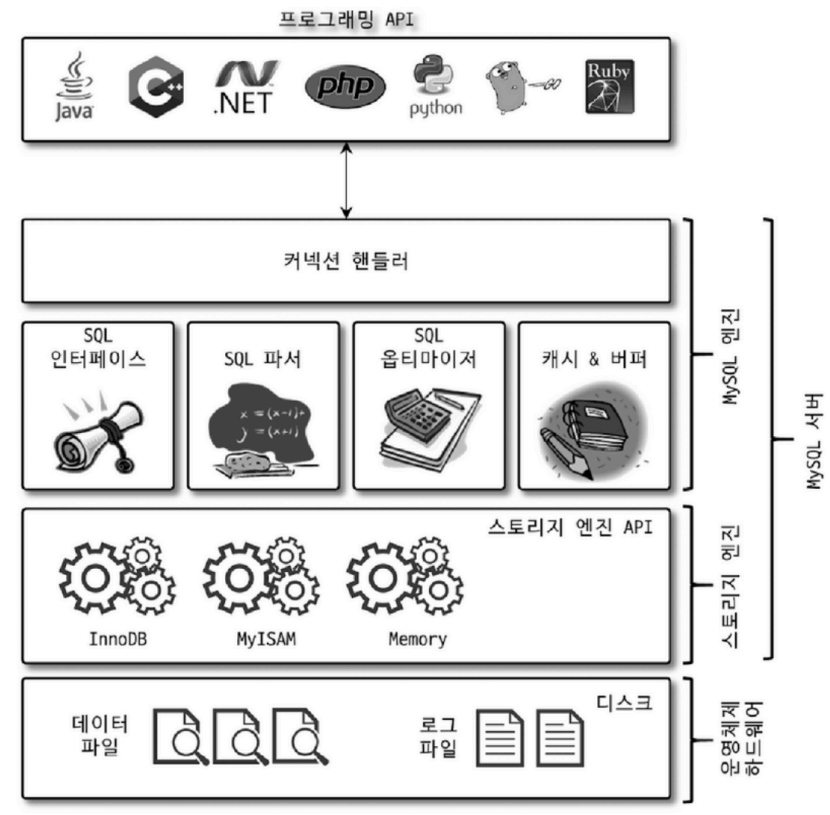


그림 4.1 MySQL 서버의 전체 구조

MySQL 서버에서 MySQL 엔진은 하나지만 스토리지 엔진은 여러 개를 동시에 사용할 수 있다.

MySQL 엔진

요청된 SQL문을 분석 및 최적화 하는 처리를 수행

(MySQL은 표준 SQL (ANSI SQL) 문법을 지원하기 때문에 표준 문법에 따라 작성된 쿼리는 타 DBMS와도 호환된다.)

- **핸들러 (Handler)* 요청**

  : 쿼리 실행기에서 데이터를 쓰거나 읽어야 하는 경우, 각 스토리지 엔진에 쓰기 또는 읽기 작업을 요청하는 것.

  MySQL 엔진은 스토리지 엔진을 조작하기 위해 핸들러를 사용해야 한다. 즉, MySQL 엔진이 각 스토리지 엔진에게 데이터 읽기 또는 저장 명령을 하려면 반드시 핸들러를 통해야 한다.

  * 핸들러: 프로그래밍 언어에서 어떤 기능을 호출하기 위해 사용하는 운전대와 같은 역할을 하는 객체.

- **핸들러 API**

  : 핸들러 요청에 사용되는 API.

  InnoDB 스토리 엔진 또한 핸들러 API를 통해 MySQL 엔진과 데이터를 주고 받는다.

스토리지 엔진

실제 데이터를 디스크 스토리지에 저장 또는 디스크 스토리지로부도 데이터를 읽어오는 작업 수행

> CREATE TABLE test (fd1 INT, fd2 INT) ENGINE = INNODB; //해당 테이블의 모든 읽기, 변경 작업은 정의된 스토리지 엔진이 처리하게 된다.

각 스토리지 엔진은 성증 향상을 위해 다음 기능을 내장하고 있다.

- 키 캐시 (MyISAM 스토리지 엔진)
- InnoDB 버퍼풀 (InnoDB 스토리지 엔진)

MySQL 스레딩 구조

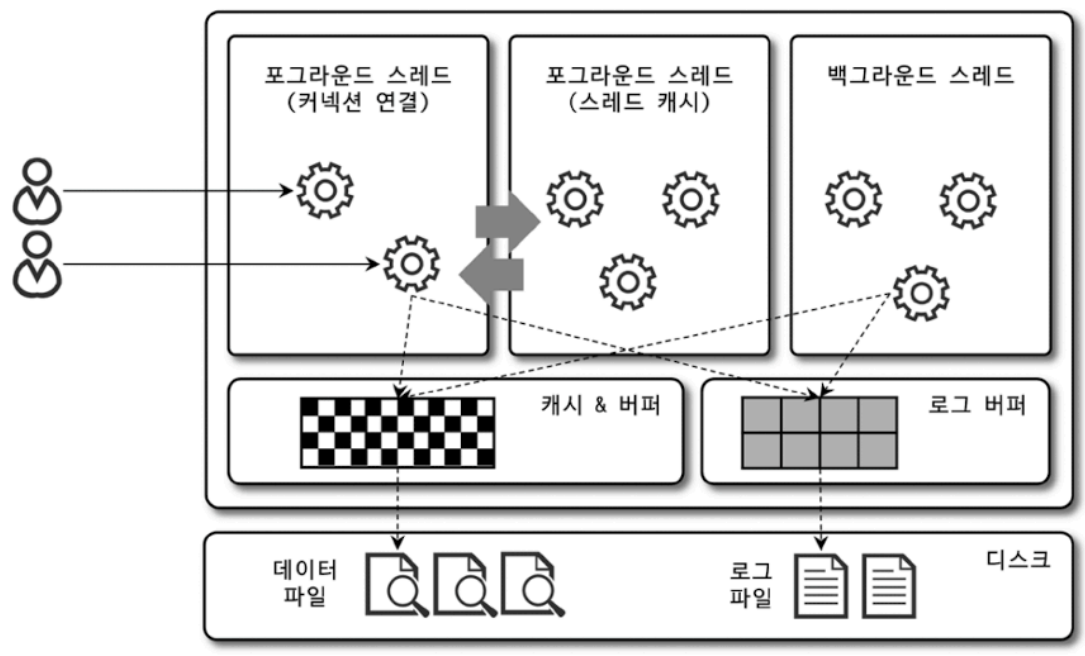


그림 4.2 MySQL의 스레딩 모델

MySQL 서버는 프로세스 기반이 아니라 스레드 기반으로 작동한다.

- **Foreground Thread, 클라이언트 스레드**

  실제 사용자 쿼리 요청을 처리하는 스레드

  최소 MySQL 서버에 접속한 클라이언트 수만큼 존재한다.

  클라이언트 커넥션이 종료되면 스레드 캐시로 돌아가며, 스레드 캐시에 일정 개수 이상의 대기중인 스레드가 있으면 캐시에 넣지 않고 스레드를 종료시켜 최대 스레드 수를 유지한다.

- **Background Thread**

InnoDB에서 처리되는 백그라운드 작업들 (MyISAM은 백그라운드 작업이 별로 없음)

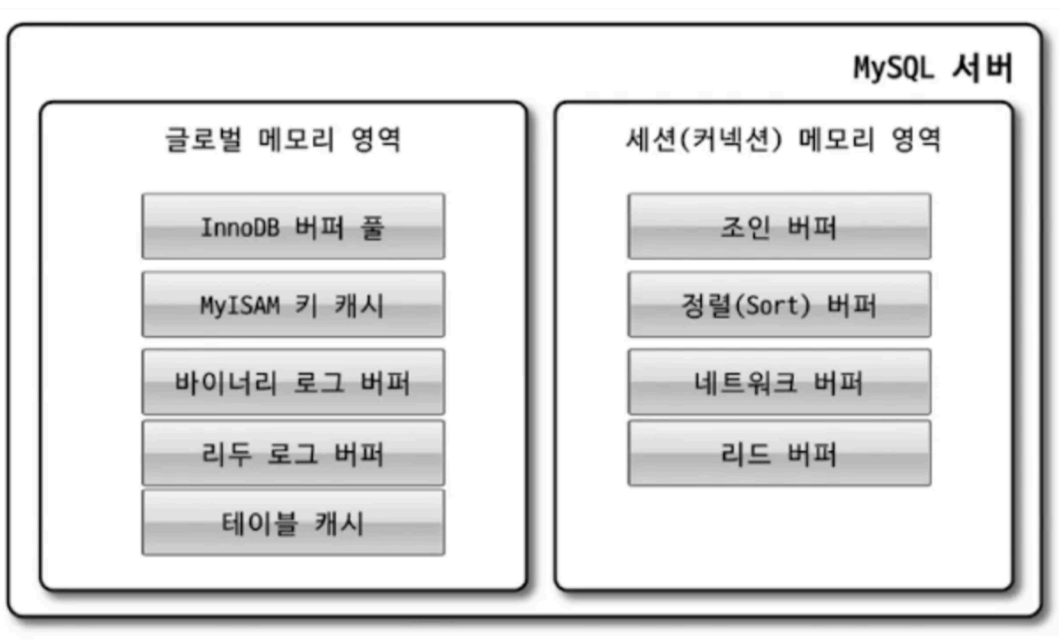
- Insert Buffer 병합 스레드
- 로그를 디스크에 기록하는 스레드
- InnoDB 버퍼 풀의 데이터를 디스크에 기록하는 스레드
- 데이터를 버퍼로 읽어오는 스레드
- lock과 deadlock을 모니터링하는 스레드

MySQL 5.5버전부터 데이터 쓰기/읽기 스레드 개수를 2개 이상 지정 가능

InnoDB에서 데이터를 읽는 작업은 주로 클라이언트 스레드에서 처리되기 때문에 읽기 스레드는 많이 설정할 필요 없음. 그러나 쓰기 스레드는 아주 많은 작업을 백그라운드로 처리하기 때문에 일반적인 내장 디스크를 사용할 때는 2~4 정도로 두는 것이 좋다.

> 💡 **InnoDB와 MyISAM의 쓰기 작업**
>
> 사용자의 요청을 처리하는 도중 **데이터의 쓰기 작업은 지연 (버퍼링) 되어 처리될 수 있지만, 데이터의 읽기 작업은 절대 지연될 수 없다.**
>
> 그래서 일반적인 DBMS에서는 **대부분 쓰기 작업을 버퍼링**해서 일괄 처리하도록 하며, InnoDB 또한 이런식으로 동작한다. 반면 **MyISAM은 사용자 스레드가 쓰기 작업까지 함께 처리**하도록 설계되어 있다.
>
> → InnoDB에서는 데이터 변경 요청 시 데이터가 디스크에 완전히 기록될 때까지 기다리지 않아도 된다. 하지만 MyISAM에서 일반적인 쿼리는 쓰기 버퍼링을 사용할 수 없다.

메모리 할당 및 사용 구조



MySQL 서버 내에 존재하는 많은 스레드들이 공유해서 하용하는 공간인지 여부에 따라 글로벌 메모리 영역, 로컬 메모리 영역으로 구분된다.

- **글로벌 메모리 영역**

  일반적으로 클라이언트 스레드의 수와 무관하게 하나의 메모리 공간만 할당된다.

  생성된 글로벌 영역이 N 개라 하더라도 모든 스레드에 의해 공유된다.

  ex) 테이블 캐시, InnoDB 버퍼 풀, InnoDB 어댑티브 해시 인덱스, InnoDB 리두 로그 버퍼

- **로컬 메모리 영역 (= 세션* 메모리 영역, 클라이언트 메모리 영역)**

  클라이언트 스레드가 쿼리를 처리하는데 사용하는 메모리 영역

  각 클라이언트 스레드별로 독립적으로 할당되며 절대 공유되어 사용되지 않는다.

커넥션이 열려 있는 동안 계속 할당된 상태로 남아 있는 공간도 있고 쿼리를 실행하는 순간에만 할당했다가 다시 해제하는 공간도 있다.

ex) Connection 버퍼, Sort 버퍼, Join 버퍼, 바이너리 로그 캐시, 네트워크 버퍼

* 세션: 클라이언트와 MySQL 서버와의 커넥션

- **플러그인 스토리지 엔진 모델**

  : 기본적으로 제공되는 스토리지 엔진 이외에 부가적인 기능을 가진 스토리지 엔진을 플러그인 하여 사용할 수 있다.

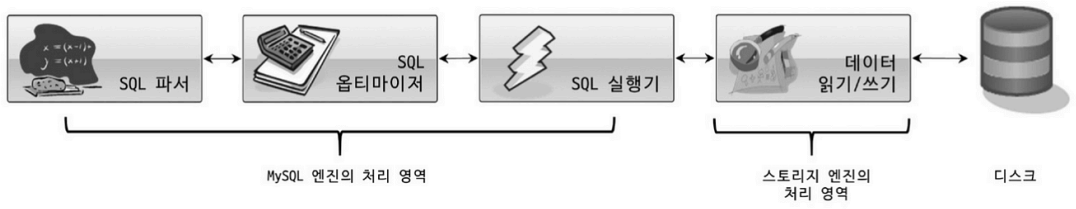


그림 4.5 MySQL 엔진과 스토리지 엔진의 처리 영역

  위의 경우 DBMS 전체 기능이 아닌 데이터 읽기/쓰기라는 일부 기능만 수행하는 엔진을 작성할 수 있다.

  MySQL 서버는 스토리지 엔진 뿐만 아니라 다양한 기능을 플러그인 형태로 지원한다.

- **컴포넌트**

  MySQL 8.0부터 기존의 플러그인 아키텍처를 대체하기 위해 컴포넌트 아키텍처를 지원한다.

  컴포넌트 아키텍처는 아래와 같은 단점을 보완하여 구현되었다.

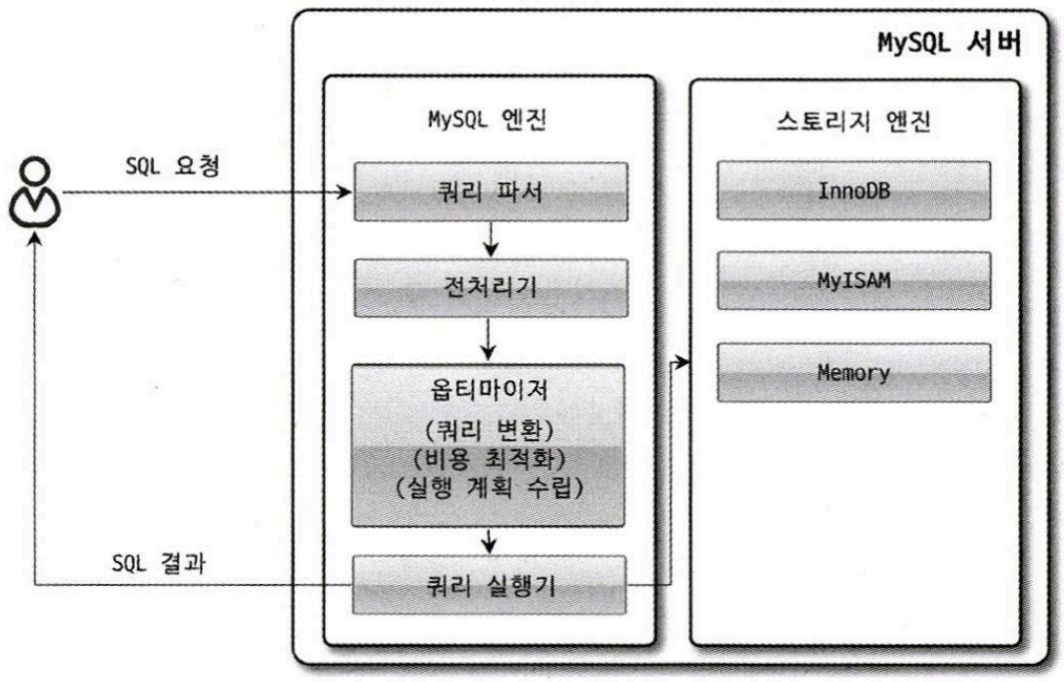
  - MySQL 서버 플러그인의 단점

    1) 플러그인 은 오직 MySQL 서버와 인터페이스할 수 있고, 플러그인끼리는 통신할 수 없다.

    2) 플러그인은 MySQL 서버의 변수나 함수를 직접 호출하기 때문에 안전하지 않다. (캡슐화되지 않음)

    3) 플러그인은 상호 의존 관계를 설정할 수 없어서 초기화가 어렵다.

쿼리 실행 구조



- **쿼리 파서**

  : 사용자 요청 쿼리 문장을 토큰으로 분리해 트리 형태의 구조로 만들어내는 작업

  쿼리 문장의 문법 오류를 검출하고 사용자에게 오류 메시지를 전달한다.

- **전처리기**

  : 파서에서 만들어진 파서 트리를 기반으로 쿼리 문장에서 구조적인 문제점이 있는지 확인한다.

  각 토큰을 테이블 명, 칼럼 명, 내장 함수 등의 개체를 매핑하고 해당 객체의 존재 여부와 접근 권한 등을 확인한다.

- **옵티마이저**

  : 쿼리 문장을 저렴한 비용으로 가장 빠르게 처리할 수 있도록 최적의 실행 계획을 결정한다.

- **실행 엔진**

  : 옵티마이저가 생성한 계획대로 핸들러에게 요청을 보내고, 요청 결과를 다른 핸들러의 요청 입력에 연결한다.

- **핸들러 (스토리지 엔진)**

  : MySQL 서버의 가장 밑단에서 MySQL 실행 엔진의 요청에 따라 데이터를 디스크로 저장하고 읽어온다.

  MyISAM 테이블을 조작하는 경우 MyISAM 스토리지 엔진이, InnoDB 테이블을 조작하는 경우 InnoDB 스토리지 엔진이 핸들러가 된다.

- **복제 (Replication) →** 16장 참고

- **쿼리 캐시**

  : SQL의 실행 결과를 메모리에 캐시하고 동일한 쿼리가 실행되면 테이블을 읽지 않고 즉시 결과를 반환하기 때문에 매우 빠른 성능을 보인다.

  하지만 테이블의 데이터가 변경되면 캐싱된 결과 중 변경된 테이블과 관련된 데이터는 모두 삭제 (Invalidate)해야 했다. 이는 심각한 동시 처리 성능 저하와 많은 버그의 원인이 되었다.

  이런 이유로 MySQL 8.0부터 쿼리 캐시와 관련된 기능은 완전히 제거되었다.

- **스레드 풀**

  스레드 풀을 통해 내부적으로 사용자의 요청을 처리하는 스레드 개수를 제한한다. CPU가 제한된 개수의 스레드 처리만 관리할 수 있도록 하여 서버의 자원 소모를 줄이는 것을 목적으로 한다.

  (MySQL 서버 엔터프라이즈 에디션은 프로그램에 내장되어 제공되지만, 커뮤니티 에디션은 동일 버전의 Percona 서버의 스레스 풀 플러그인 라이브러리를 커뮤니티 에디션 서버에 설치해서 사용할 수 있다.)

- **트랜젝션 지원 메타데이터**

  DB 서버에서 메타 데이터(= 데이터 딕셔너리)는 테이블의 구조, stored 프로그램의 정보 등을 말한다.

  MySQL 5.7버전까지는 이런 메타데이터를 파일 기반으로 관리했으며, 이는 생성 및 변경 작업이 트랜젝션을 지원하지 않는다. 따라서 테이블 생성 또는 변경 도중 MySQL 서버가 비정상적으로 종료되면 데이터가 일관되지 않는 상태로 남는 문제가 있었다.

  MySQL 8.0 버전부터는 이런 문제점을 해결하기 위해 테이블의 구조 정보나 stored 프로그램의 코드 관련 정보를 모두 InnoDB의 테이블에 저장하도록 개선되었다. 시스템 테이블과 데이터 딕셔너리 정보는 모두 `mysql` DB에 저장하고, `mysql` DB는 `mysql.ibd`라는 이름의 테이블 스페이스에 저장된다.

  InnoDB 스토리지 엔진을 사용하는 테이블은 메타 정보가 InnoDB 테이블 기반의 딕셔너리에 저장되지만 이외 MyISAM, CSV 등과 같은 스토리지 엔진을 사용하는 테이블의 메타 정보는 SDI (Serialized Dictionary Information) 파일을 사용한다. 이 파일은 기존의 FRM 파일과 동일한 역할을 한다.