3.1 MySQL 아키텍쳐

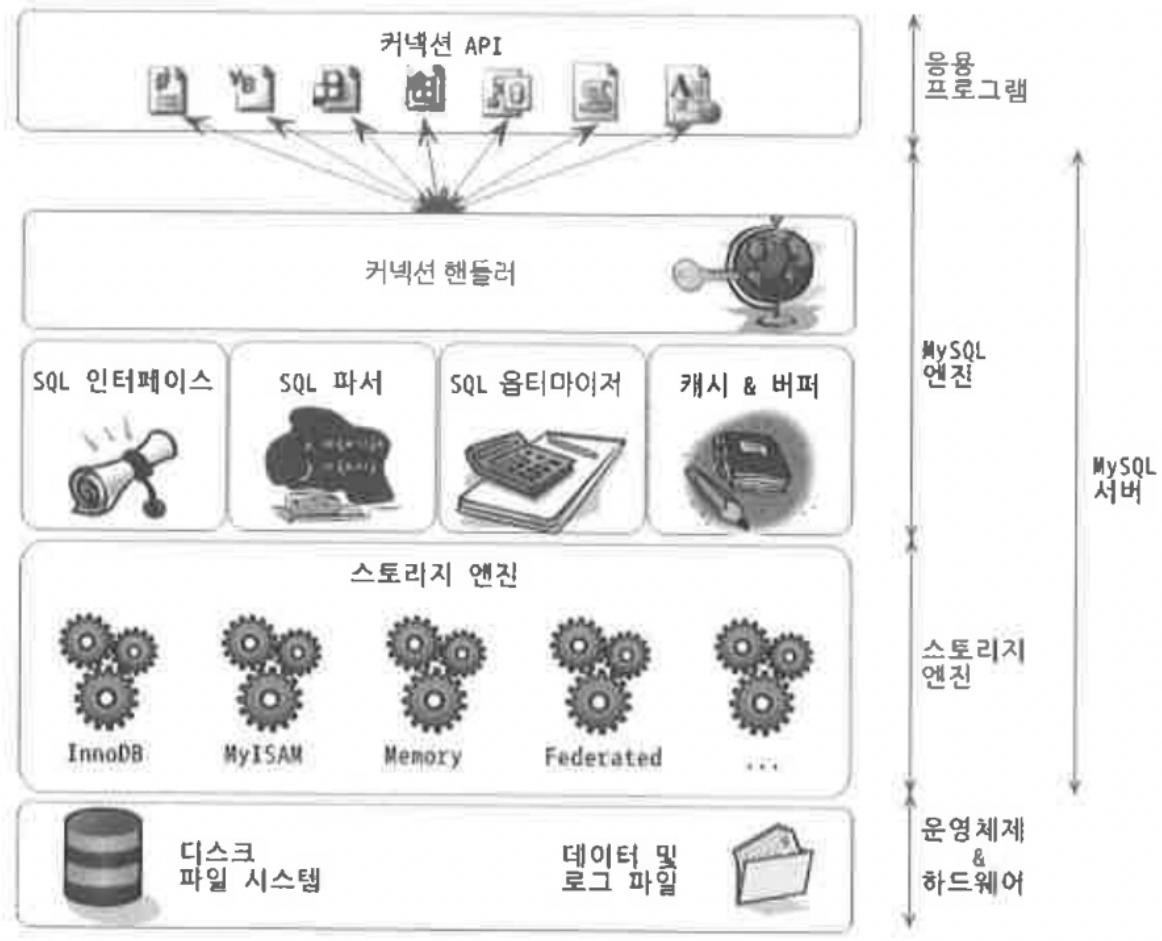
생성일	@2021년 5월 22일 오후 3:36
태그	

MySQL은 다른 DBMS에 비해 구조가 상당히 독특하다

1. MySQL 의 전체 구조

MySQL은 일반 상용 RDBMS에서 제공하는 대부분의 접근법을 모두 지원

- JDBC, ODBC 드라이버
- C/C++, 자바, 파이썬, 루비 .....

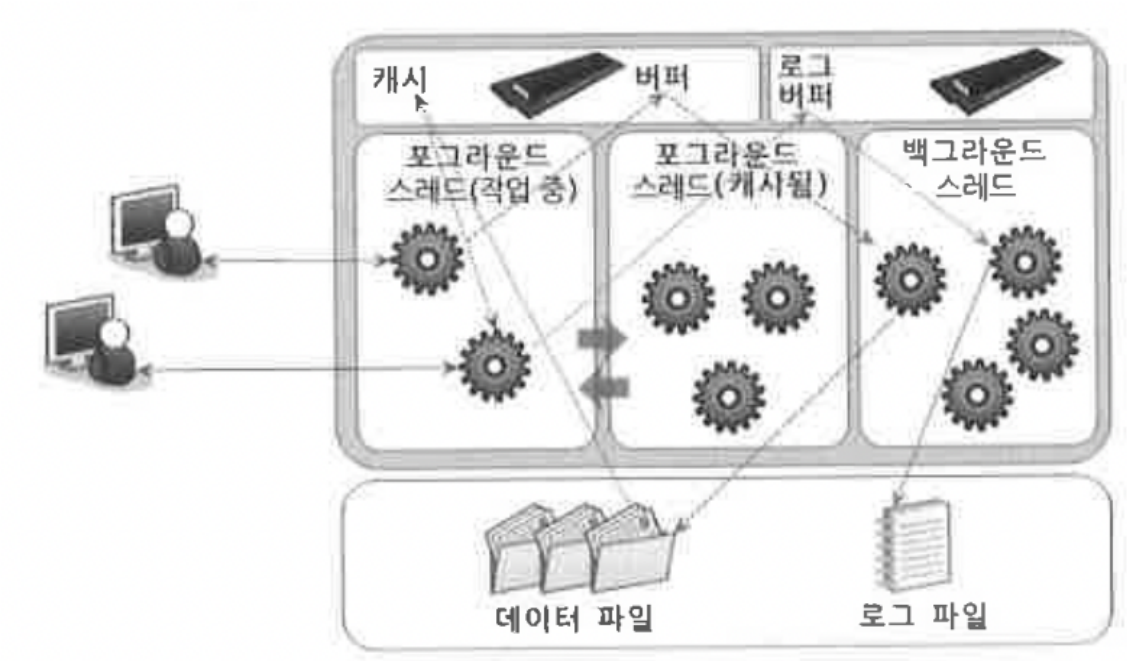


[그림 3-1] MySQL 서버의 전체 구조

MySQL 서버

- MySQL 엔진 : 쿼리 파서, 옵티마이저 등 클라이언트로부터의 접속 및 요청을 처리
- 스토리지 엔진 : 실제 데이터를 디스크 스토리지에 저장하거나 디스크 스토리지로부터 read해오는 부분. 동시에 여러개의 스토리지 엔진 사용가능
  - CREATE TABLE test_table (웅,앵,웅) ENGINE=INNODB (또는 MyISAM)
  - 여기서 InnoDB 스토리지 엔진을 사용하도록 정의
- 핸들러 API : MySQL엔진에서 스토리지엔진에 데이터를 r/w할때 이러한 요청을 핸들러 API를 통해서 데이터를 주고받는다

2. MySQL 스레딩 구조



[그림 3-2] MySQL의 스레딩 모델

MySQL 서버는 프로세스 기반이 아니라 `스레드 기반`으로 작동

포그라운드 스레드 (=클라이언트 스레드)

- 최소한 MySQL 서버에 접속된 클라이언트의 수만큼 존재하며, 주로 각 클라이언트 사용자가 요청하는 쿼리문장을 처리
- 클라이언트가 커넥션 종료시, 해당 커넥션을 담당하던 스레드는 다시 Thread Pool(스레드 캐시)로 되돌아간다
    - 이미 스레드 풀에 일정 개수 이상의 대기 스레드가 있으면 스레드 캐시에 넣지않고 스레드 종료시킴
    - 이렇게 일정하게 스레드 개수 유지시키는 파라미터 ⇒ `thread_cache_size`

백그라운드 스레드 (InnoDB)

- 인서트 버퍼를 병합하는 스레드
- 로그를 디스크로 기록하는 스레드 🌟
- InnoDB 버퍼 풀의 데이터를 디스크에 기록하는 스레드 🌟
- 여러가지 잠금이나 데드락을 모니터링하는 스레드
- → 이 모든 스레드를 총괄하는 메인스레드

🌟Log Thread

🌟Write Thread : 쓰기 스레드는 많은 작업을 백그라운드로 처리하기 때문에 일반적인 내장 디스크를 사용할때는 2-4정도, 다른 DAS, SAN과 같은 스토리지를 사용할때는 4개 이상으로 충분히 설정

> 💡 SQL 처리 도중 Write 작업은 지연처리 가능 (InnoDB)
> → DML 쿼리시, 데이터가 디스크의 데이터 파일로 완전히 저장될 때까지 기다리지 않아도 된다
> → 하지만 Read 작업은 **절대 지연불가**

MyISAM 은 Write작업을 버퍼링해서 일괄처리 하지 않고 사용자 스레드가 쓰기 작업까지 함께 처리하도록 설계되어있다 (일반적인 쿼리는 쓰기 버퍼링 기능 사용불가)

3. 메모리 할당 및 사용 구조

MySQL에서 사용되는 메모리 공간은...

글로벌 메모리 영역

: 모든 스레드에 의해 공유된다

- MySQL 서버가 시작되면서 무조건 OS로부터 할당됨
- 요청된 메모리 공간을 100% 할당해줄 수도 있고, 그 공간만큼 예약해두고 필요할 때 조금씩 할당해주는 경우도 있다. (운영체제의 종류에 따라 다르겠지만)
- 각 운영체제의 메모리 할당 방식은 상당히 복잡. MySQL 서버가 사용하고 있는 정확한 메모리 양을 측정하는것 쉽지 않다

→ MySQL의 파라미터로 설정해 둔 만큼 운영체제로부터 메모리를 할당받는다고 생각하자

로컬 메모리 영역 (= 세션 메모리 영역)

: MySQL 서버상에 존재하는 클라이언트 스레드가 쿼리를 처리하는 데 사용하는 메모리 영역

- 클라이언트가 MySQL 서버에 접속하면 MySQL 서버에서는 클라이언트 커넥션으로부터의 요청을 처리하기 위해 스레드를 하나씩 할당
- 클라이언트 스레드가 사용하는 메모리 공간 → aka 클라이언트 메모리 영역
- 로컬 메모리는 각 클라이언트 스레드별로 독립적으로 할당되며 절대 공유되어 사용되지 않는다
- 쿼리의 용도별로 필요할 때만 공간이 할당되고 필요하지 않은 경우에 MySQL이 메모리 공간을 할당조차도 하지 않을 수 있음
    - sort buffer, join buffer

로컬 메모리 공간은 커넥션이 열려 있는 동안 계속 할당된 상태로 남아 있는 공간도 있고 그렇지 않고 쿼리를 실행하는 순간에만 할당했다가 다시 해제하는 공간도 있다.

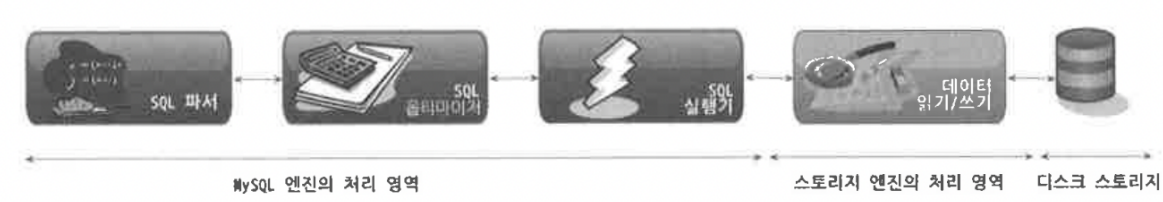
4. 플러그인 스토리지 엔진 모델

MySQL의 독특한 구조 중 대표적인 것이 바로 플러그인 모델.

→ MySQL 5.1부터는 전문 검색 엔진을 위한 검색어 파서도 플러그인 형태로 개발해서 사용할 수 있다.

→ 다양한 스토리지 엔진을 지원한다.

[MySQL에서 쿼리가 실행되는 과정]



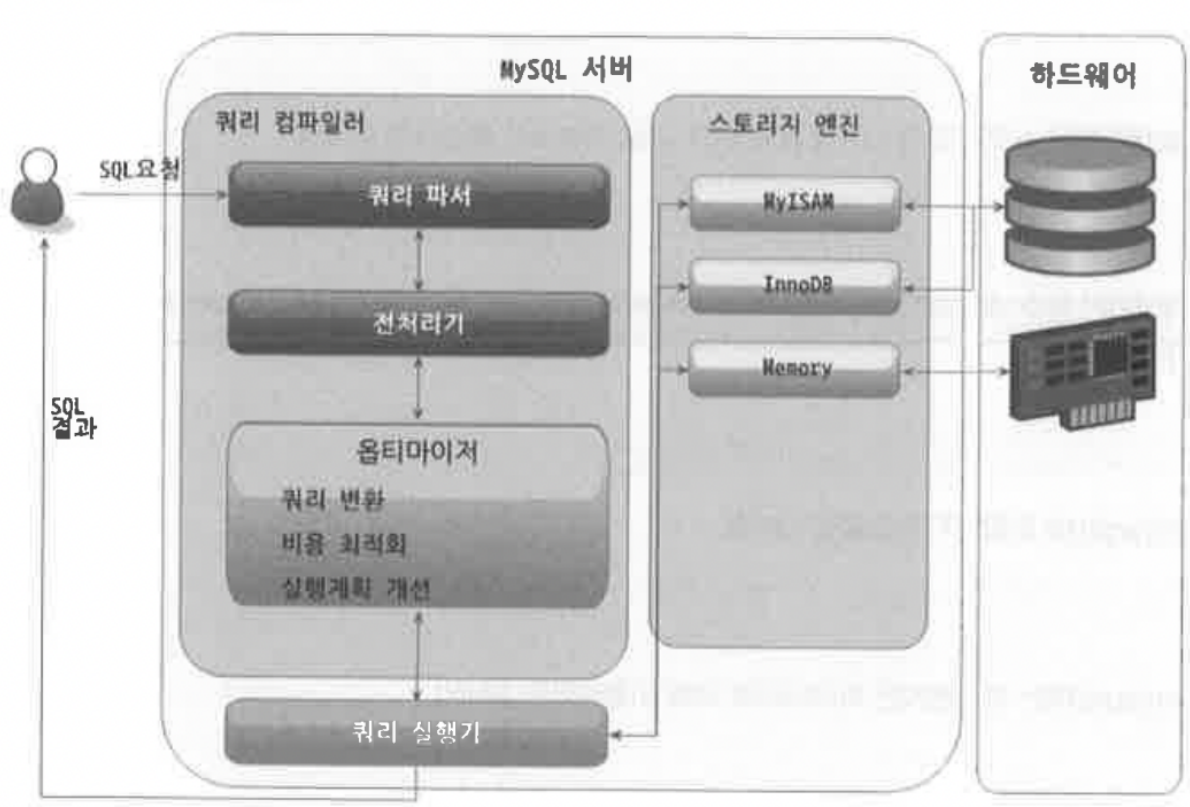
[그림 3-5] MySQL 엔진과 스토리지 엔진의 처리 영역

- 마지막 Data Read/Write 작업만 스토리지 엔진에 의해 처리
- 데이터 읽기/쓰기 작업은 거의 대부분 1건의 레코드 단위로 처리된다

핸들러란

- 어떤 기능을 호출하기 위해 사용하는 운전대와 같은 역할을 하는 객체
- MySQL 엔진이 스토리지 엔진을 조정하기 위해 핸들러라는 것을 사용
- 다른 스토리지 엔진을 사용하는 테이블에 대해 쿼리를 실행하더라도 MySQL 처리는 동일. (마지막 데이터 r/w 영역 처리만 다름)

5. 쿼리 실행 구조

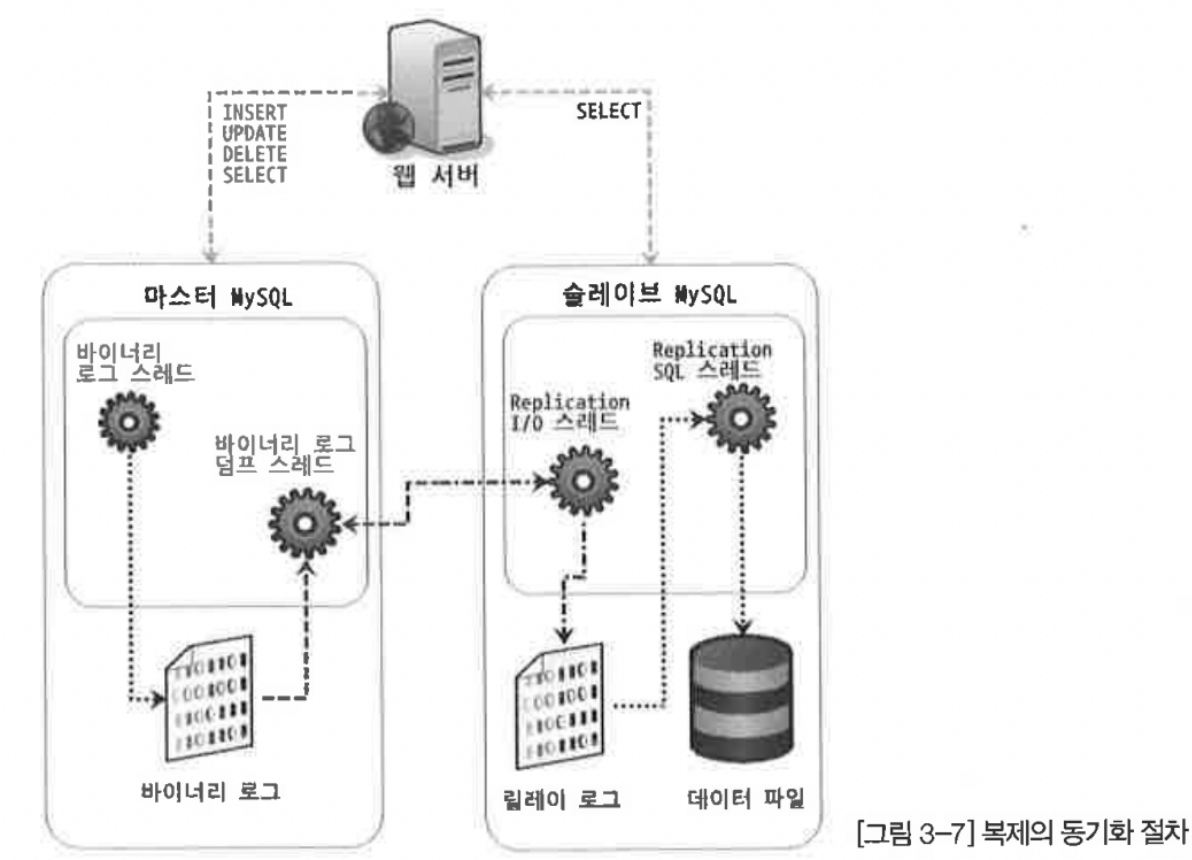


[그림 3-6] 쿼리 실행 구조

쿼리를 실행하는 관점에서 MySQL의 구조를 간략하게 그림으로 표현

1. 파서 : 사용자 요청으로 들어온 쿼리 문장을 토큰으로 분리해 트리 구조로 만듦 → 오류 존재시 exception 메시지 전달
2. 전처리기 : 파서 과정에서 만들어진 파서 트리를 기반으로 **쿼리 문장에 구조적인 문제점이 있는지 확인**. 칼럼 이름 또는 내장 함수와 같은 개체를 매핑해 해당 객체의 존재 여부와 객체의 접근권한 확인
3. 옵티마이저 : 쿼리 문장을 저렴한 비용으로 가장 빠르게 처리할지 결정하는 역할 담당 ⇒ DBMS의 두뇌
4. 실행엔진 : 옵티마이저의 계획대로 각 핸들러에게 요청해서 받은 결과를 또 다른 핸들러 요청의 입력으로 연결하는 역할을 수행 ⇒ DBMS의 중간관리자
5. 핸들러 (=스토리지 엔진) : MySQL 서버의 가장 밑단에서 실행엔진의 요청에 따라 데이터를 디스크로 저장하고 디스크로부터 읽어오는 역할을 담당 ⇒ InnoDB 테이블 조작시 핸들러가 InnoDB 스토리지 엔진이 된다

6. 복제(Replication)



[그림 3-7] 복제의 동기화 절차

확장성(Scalability)을 위해 MySQL에서 제공하는 기술 중 하나

레플리케이션

- 2대 이상의 MySQL 서버가 동일한 데이터를 담도록 실시간으로 동기화 하는 기술
- Master(쓰기 역할) : MySQL의 복제에는 INSERT / UPDATE 같은 데이터 변경 가능한 서버
- Slave (읽기 역할) : SELECT 쿼리로 데이터를 읽기만 할 수 있는 서버

→ MySQL 서버의 복제에서는 마스터 반드시 1개 + 슬레이브는 1개 이상

서버 하나가 마스터이면서 슬레이브 역할 까지 수행하도록 설정하는 것도 가능하다

마스터

- 기술적으로 MySQL의 바이너리 로그가 활성화 되면 어떤 MySQL 서버든 마스터가 될 수 있음
- 데이터 생성/변경/삭제가 되는 시작점
- 데이터 변경 쿼리 문장은 바이너리 로그에 기록됨 → 슬레이브 서버에서 변경 내역을 요청하면 마스터장비는 해당 바이너리 로그를 읽어 슬레이브로 넘김
    - Binlog dump 스레드가 해당 일을 전담하는 스레드임
    - 10개의 슬레이브가 마스터에 연결되어있으면 해당 스레드도 10개

슬레이브

- 데이터(바이너리 로그)를 받아올 마스터 장비의 정보를 가지고 있으면 슬레이브가 된다.
    - 마스터나 슬레이브라고 해서 별도의 빌드 옵션이 필요하거나 프로그램을 별도로 설치 X
- 마스터 서버가 바이너리 로그를 가지고 있으면, 슬레이브는 릴레이 로그를 가지고 있다.
    - 마스터로부터 받아온 변경 내역을 릴레이 로그에 기록
- 슬레이브 서버의 SQL 스레드가 릴레이로그에 기록된 변경내역을 재실행(Replay)해서 슬레이브의 데이터를 마스터와 동일한 상태로 유지

마스터/슬레이브 구조의 특징

- 슬레이브는 하나의 마스터만 설정 가능
- 마스터와 슬레이브의 데이터 동기화를 위해 슬레이브는 읽기 전용으로 설정
- 슬레이브 서버용 장비는 마스터와 동일한 사양이 적합
- 복제가 불필요한 경우에는 바이너리 로그 중지
    - 바이너리 로그는 성능에 큰 영향을 끼침

바이너리 로그와 트랜잭션의 격리 수준은?

- STATEMENT 포맷 방식

: 바이너리 로그 파일에 마스터에서 실행되는 쿼리 문장을 기록하는 방식

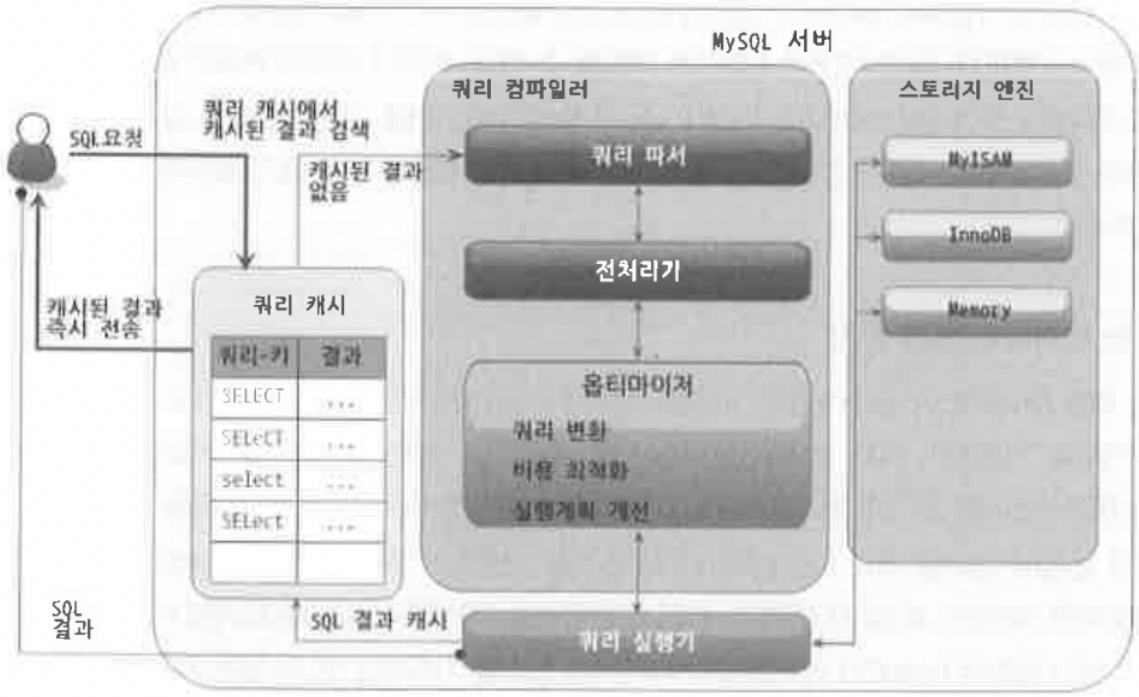
- ROW 포맷 방식

: 마스터에서 실행된 쿼리에 의해 변경된 레코드 값을 기록하는 방식

MySQL 5.0 이하에서는 STATEMENT방식만 제공되었었음 → REPEATABLE READ 격리 수준만 사용 가능

SQL 기반의 복제가 정상적으로 작동하려면 REPEATABLE - READ 이상의 트랜잭션 격리 수준을 사용해야함.

7. 쿼리 캐시



[그림 3-8] 쿼리 캐시 구조

쿼리 캐시는 타 DBMS에 없는 MySQL의 독특한 기능 중 하나 (성능 향상 효과가 큼)

1. 여러가지 복잡한 처리 절차와 고비용을 들여 실행된 쿼리를 메모리에 캐시해둠
2. 동일 쿼리 요청왔을 때 쿼리 캐시에서 찾아서 바로 결과를 내려줌

쿼리 캐시의 구조는 간단한 키와 값의 쌍으로 관리되는 맵과 같은 데이터 구조로 구현됨

- 키 : 쿼리 문장 자체

- 값 : 해당 쿼리의 실행 결과

쿼리 캐시 결과를 내려 보내주기 전에 거쳐야할 확인 절차

1. 요청된 쿼리 문장이 쿼리 캐시에 존재하는가?
    - MySQL의 어떠한 처리보다 앞 단에 위치함
2. 해당 사용자가 결과를 볼 수 있는 권한을 가지고 있는지?
    - 사용자가 해당 테이블의 읽기권한 확인
3. 트랜잭션 내에서 실행된 쿼리인 경우 가시 범위 내에 있는 결과인가?
    - 트랜잭션 ID보다 ID 값이 큰 트랜잭션에서 변경한 작업 내역이나 쿼리 결과는 참조할 수 없음 (트랜잭션 격리 수준을 준수하기 위해)
4. RAND()나 CURRENT_DATE(), SYSDATE() 처럼 호출 시점에 따라 달라지는 요소가 있나?
    - 그럼 사용하지 않는당
5. 프리페어 스테이트먼트의 경우 변수가 결과에 영향을 미치지 않는가?
    - 쿼리 문장 자체에 ? 가 있어서 문장 자체로 쿼리 캐시를 찾을 수 없다
6. 캐시가 만들어지고 난 이후 해당 데이터가 다른 사용자에 의해 변경되지 않았는가?
7. 쿼리에 의해 만들어진 결과가 캐시하기에 너무 크지 않은가?