MySQL 개념 구조

학습목표

1. MySQL의 기본 구조를 이해합니다.

1. 서론
2. RDBMS 기본 개념 구조
3. MySQL RDBMS 개념 구조
4. 동작 방식
5. 엔진 타입

이 자료는 MySQL RDBMS의 개념 구조의 이해를 돕기 위해 만들어진 자료 입니다.

자료는 지금까지 개발자와 아키텍트들에 의해 밝혀진 사실,
직관적으로 판단할 수 있는 기능 등을 토대로 만들어 졌습니다.
(transaction과 recovery functionality 만을 포함합니다)

그럼에도 불구 하고 공식적인 구조도가 릴리스된 적이 없기 때문에
이 개념 자료가 정확하다고 판단할 수 없습니다.

자료는 MySQL을 직접 개발하지 않는 사람들이 MySQL의 개념 구조를 대략적으로
이해 하기 위한 용도로서만 활용 부탁 드립니다.

이 자료에서 MySQL이 어떤 식으로 모듈화 되어 있는지 확인할 수 있습니다.

우선 RDBMS의 기본 개념 구성을 알아본 후
기본 개념을 통해 MySQL의 특화된 개념 구성을 확인하도록 하겠습니다.

특화된 개념 구성에서는 각 계층의 시스템들의 성격을 구분하고
각 계층의 시스템들이 어떤 식으로 상호 작용을 하는지 알아 보겠습니다.

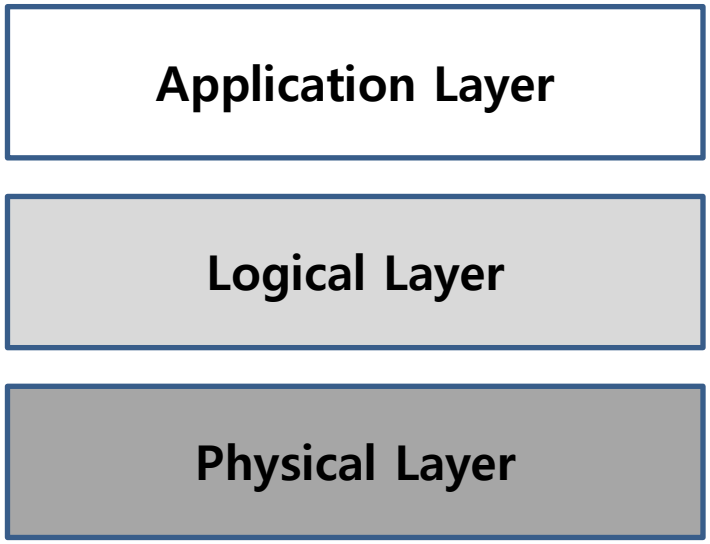
마지막으로는 diagram과 케이스별 동작 시나리오를 통해서
components 간의 동작 구조를 확인하겠습니다.

또한 각 계층에 대한 설명 시에
MySQL의 확장성 및 개발 유연성에 대해 설명하도록 하겠습니다.

참고 문헌

1. Atkinson, Leon. Core MySQL: The Serious Developer's Guide. New Jersey: Prentice Hall Publishing,2002.
2. Date, C.J. An Introduction to Database Systems. Menlo Park: Addison-Wesley Publishing Company, Inc,1986.
3. Dubois, Paul. MySQL. New York: New Riders Publishing, 2000.
4. Frost, R.A. Database Management Systems. New York: Granada Technical Books, 1984.
5. Garcia-Molina, Hector. Database System Implementation. New Jersey: Prentice Hall, 2000.
6. Garlan, David. Shaw, Mary. "An Introduction to Software Architecture". Pittsburgh, PA USA: School of Computer Science, Carneigie Mellon University, 1994.
7. Kruchten, Phillippe. "Architectural Blueprints – The 4+1 View Model of Software Architecture". Rational Software Corp, 1995.
8. Silberschatz, Abraham et al. Database System Concepts. New York: McGraw- Hill, c1997.
9. U.S. Department of Commerce. "An Architecture for Database Management Standards". Washington: U.S. Government Printing Office, 1982.
10. "http://www.mysql.com/". MySQL AB, 2002.
11. Yarger, Randy Jay MySQL & mSQL. Sebastopol: O'Reilly & Associates, 1999.
12. Ryan Bannon. Alvin Chin. Faryaaz Kassam. Andrew Roszko. "MySQL Conceptual Architecture", 2002

MySQL의 개념 구조를 이해하기 전에 RDBMS 기본 구조를 대략 적으로 설명합니다.



RDBMS의 기본 구조는 세가지 구성으로 나누어 집니다.

Application layer

Logical layer

Physical layer

Application layer

Application layer는 RDBMS가 외부와 상호작용을 하기 위한 중요한 interface 입니다.

다양한 사용자를 충족시키기 위해 그 interface 역시 다양하게 존재하는데,
주요 사용자들은 네 가지 그룹으로 나눌 수 있습니다.

Sophisticated users : 어떠한 형식에 구애 받지 않고 DBMS에 직접 database query language를 사용하여 질의하는 사용자들

Specialized users : 전통적인 data-processing framework를 사용하지 않고 특정화된 application에서 질의를 하는 application 개발자들

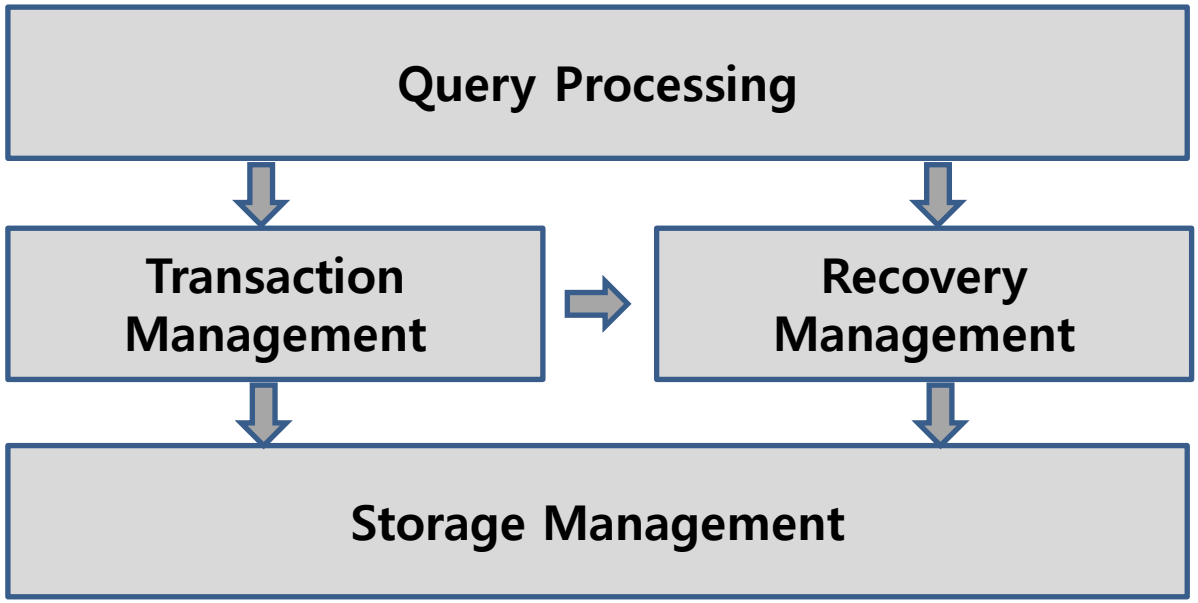
Naïve users : Sophisticated users와는 반대로 고정적인 application을 통해야만 질의가 가능한 사용자들

Database Administrators : DBMS에 대해 모든 권한과 책임을 가지며 관리하는 사용자들

Logical Layer

Logical Layer는 RDBMS의 CORE 기능이며,
각 제품의 벤더들이 다양한 로직으로 구현을 합니다.

하지만, 다양한 database management 매뉴얼을 확인해 보면
몇몇 가지의 core functionality로 추상화가 가능합니다.



Physical layer

RDBMS는 storage에 대한 여러 가지 정보를
Storage manager를 통해 secondary storage에 저장 합니다.

저장되는 데이터의 종류는 다음과 같습니다.

Data files : 사용자 데이터

Data dictionary : database의 구조를 담는 metadata

Indices : 빠른 데이트 사용을 위한 특정적인 데이터

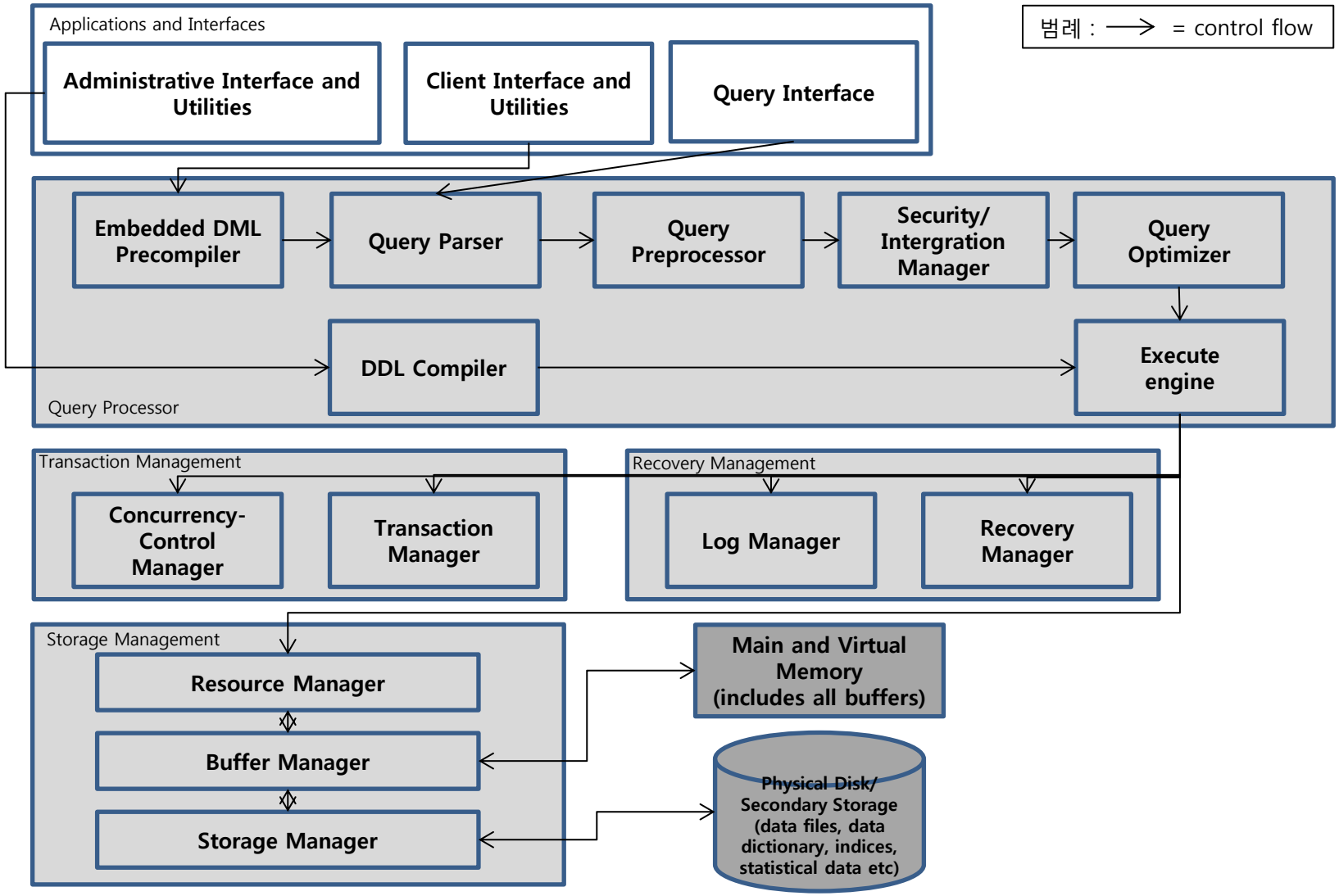
Statistical Data : database 내에 데이터에 대한 통계성 데이터

Log information : system crash 발생 시 recovery manager를 통한 데이터 복구에 사용되는 수행된 쿼리문의 집합

위에서는 RDBMS의 기본적인 개념 구조를 소개 하였으며,
이를 토대로 벤더에서 어떠한 형태로 logical layer를 구체화를 하였는지 알아 봅니다.

다음 장은 MySQL RDBMS의 개념 구조 입니다.

MySQL RDBMS의 개념 구조



Application layer

MySQL 개념 구조를 보면 총 3개의 components가 Application layer에 존재합니다.
이 components는 각기 다른 형태의 사용자들과 상호작용을 하며,
이 사용자들은 administrators, clients, query users로 나눌 수 있습니다.

Administrators : Administrative Interface and Utilities : *mysqladmin, myisamchk, etc*
Clients : Client Interface and Utilities : *C API, DBI API for Perl, PHP API, etc*
Query users : Query Interface : *mysql*

Logical Layer

Logical Layer 의 경우 여러 가지 components 로 이루어 지는데
하나씩 그 기능을 확인하도록 합니다.

Query Processor

사용자들이 원하는 데이터 조작을 위해서는 data-manipulation language(ie SQL)
을 parsing 하고 optimize를 할 수 있는 기능이 필요하며,
이 기능을 Query Processor가 일련의 components 를 통해 처리합니다.

이 과정에 pipeline, filter architecture가 사용되며,
이 architecture는 MySQL을 유연하게 합니다.

Query Processor

Embedded DML Precompiler : client가 request를 보냈을 경우, client API에 존재하는 SQL statements를 찾아내서, MySQL이 수행할 수 있는 형태의 SQL로 변환하는 기능입니다.

DDL Compiler : Administrator가 request를 보냈을 경우, request를 받아 수행하는 기능입니다. Administrator와 administrative utilities는 interface를 노출하지 않기 때문에 MySQL 서버와 직접 통신합니다.(SQL 형태) 이에 따라 Embedded DML Precompiler는 동작하지 않습니다.

Query Parser : client 또는 administrative request에서 적합한 형태의 SQL이 추출이 되면, 다른 이후 components 가 이해 할 수 있도록 SQL을 parse tree 구조로 전환 하는 기능입니다.

Query Processor

Query Preprocessor : Query Parser에 의해 생성된 parse tree를 토대로 SQL syntax에 대한 validation을 처리하는 기능입니다. 부적합한 SQL의 경우 이후 pipeline을 통하지 않고 사용자에게 error로 처리해 줍니다.

Security/Intergration Manager : SQL syntax가 정확한 경우, request의 사용자가 적합한 Database안의 테이블 레코드를 사용할 수 있는 권한이 있는지 확인하는 기능입니다.

Query Optimizer : 적합한 권한은 사용자가 가지고 있다면, 해당 SQL은 optimization 대상이 됩니다. Query Optimizer는 indexes와 analyzed data를 통해 가능한 가장 빠른 쿼리 수행을 시킬 수 있도록 처리 합니다.

Execution Engine : Optimized 된 쿼리를 가지고 Physical Layer에 access 합니다. DDL compiler에서 받은 administrative task(repair, recovery, copy, backup) 도 수행합니다.

Transaction Management

Transaction manager : Transaction은 하나 또는 하나 이상의 SQL 문으로 구성 됩니다. Transaction Manger의 역할은 이런 Transaction이 저장(logged)되고 수행(executed)되는 과정이 자동으로 발생 할 수 있도록 합니다.(이 과정은 Log Manager와 Concurrency-Control Manager가 일부 담당하고 있습니다.)
또한 deadlock를 해소하는 기능 및 COMMIT/ROLLBACK SQL에도 관여하고 있습니다.

Concurrency-Control Manager : Transactions 이 각기 별도로 수행될 수 있도록 처리하는 역할을 담당합니다(acquiring locks). Lock이 설정 되면 Lock를 획득한 하나의 Transaction 만이 데이터를 변경할 수 있으며, 다른 Transaction이 접근할 경우 Concurrency-Control Manager가 차단합니다.

Recovery Management

Log manager : Buffer Manager를 통해 RDBMS 모든 operations 를 기록 하는 역할 입니다. 모든 operation은 SQL commands 형태로 기록 됩니다. 따라서 만약 시스템이 crash 된다 하여도, 기록된 log를 수행하여 사고 이전 상태로 복원합니다(Recovery).

Recovery Manager : 기록된 log를 통해 데이터베이스를 복원하는 기능을 담당합니다. Log는 Buffer Manager에 의해 database's life의 시작점 부터 기록되는 전체로그로서 log file에서 부터 복원됩니다.

Storage Management

Storage는 물리적 환경으로 secondary storage 타입으로 분류 됩니다.
물리적 작업은 여러 buffers 에 의해 처리 됩니다.
Buffer Manager에 의해 main 및 virtual memory 사이즈가 변경됩니다.

이 management 작업은 Resource Manager, Buffer Manager 및 Store Manager 라는 세 개의 속성으로 구성 됩니다.

Storage Management

Resource Manager : Execution Engine으로 부터 받는 request를 Buffer Manager에서 보내 주며, 발생한 데이터를 Buffer Manager로 부터 받아 상위 layer로 보내는 역할을 합니다.

Buffer Manager : 사용자가 데이터를 보거나 변경을 하기 위한 memory allocation resource 관리를 담당합니다. Request 당 할당되는 buffer의 개수와 buffer당 할당 되는 memory allocation size를 관리하며, request는 Resource Manager에 의해 발생합니다.

Storage Manager : 가장 낮은 단계에서 처리를 하는 component입니다. 역할은 Buffer Manager와 secondary storage 사이에서 상호작용을 하는데, disk controller 또는 operating system과 직접 작용하며, 데이터를 물리적 디스크에서 찾아내서 Buffer Manager에게 전달하는 역할을 합니다.

동작 방식 1 : updating table scenario

1) Query Processor의 pipeline을 통과하며, SQL은 parse tree로 변경되고, syntax 및 권한을 검사 받고, optimized 처리가 되어 Execution Exgine에게 전달 됩니다.
2) Update는 transaction이기 때문에 Transaction Manager에게 전달됩니다.
3) Transaction Manager는 Log Manger를 호출하여 logging를 합니다.
4) Log Manager는 Resource Manger(Buffer/Storage Mananger)를 호출하여 logging 합니다.
5) Transaction을 serialization 처리를 위해 Transaction Manager는 Concurrency Control Manager(A.K.A 스케줄러)를 호출하여 적합한 lock을 획득합니다.
6) 스케줄러가 Resource Manager에게 변경 data file, record size, index를 명시합니다.
7) Resource Manager는 이 정보를 page화 하고 Buffer Manager에게 전달하고 결과를 전달 받습니다.
8) 데이터가 메모리 상에 존재하게 되면, 스케줄러가 정상적으로 데이터를 변경합니다.
9) commit command가 수행되면 스케줄러는 lock를 해제 합니다.

동작 방식 2 : crash scenario

1) Application Layer에서 isamchk 또는 myisamchk utilities를 통해 crash recovery를 수행합니다.
2) 이 crash recovery utility는 Query Processor에 의해 MySQL이 이해할 수 있는 상태로 전환됩니다. DDL Compiler를 통해 전환이 되며, 이 결과는 Execution Engine에 의해 수행됩니다.
3) Execution Engine는 통제를 하위 Layer(Recovery Management)에 존재하는 Recovery Manger에게 전달합니다.
4) Recovery Manager는 log file를 읽어 들이기 위해 하위 Layer(Storage Management)의 Resource Manger를 호출 합니다.
5) Buffer Manager가 Storage Manager를 통해 log file를 읽어 다시 Resource Manger에게 전달하고, Resource Manger는 Recovery Manager에서 이 결과를 다시 전달합니다.
6) Recovery Manager는 log의 시작점 부터 읽어 상위 Layer(Query processor)에서 전달합니다.
7) 각각의 SQL command(log)는 Execution Engine에 의해 수행됩니다.

Pluggable Storage Engine Architecture

MySQL Pluggable Storage Engine Architecture는 application 별로 특성화된 storage engine을 사용할 수 있도록 지원해주며 각개의 engine은 완벽히 별도로 분리 되어있습니다.
Low-Level 구성을 숨겨 둠으로써 항상 동일하고 간단한 API 모델을 유지합니다.

각각의 engine 저마다 다른 특성이 있기 때문에 사용이 구분이 필요합니다.
각각의 engine은 서로 다른 component 이며, physical sever level에서 동작합니다.
이런 modular 구성은 application에서 동일한 interface와 service로 각기 다른 성격의 작업 needs(DSS, OLTP, LOGGING etc)를 충족 시킬 수 있습니다.

Pluggable Storage Engine Architecture

Engine	Limit	transaction	Locking granularity	B-tree index	Hash index	Compressed data	Foreign key Support
MyISAM	256TB	NO	TABLE	YES	NO	YES	NO
MEMORY	RAM	NO	TABLE	YES	NO	NO	NO
CSV	N/A	N/A	N/A	N/A	N/A	N/A	N/A
ARCHIVE	N/A	NO	TABLE	NO	NO	YES	NO
BLACKHOLE	N/A	N/A	N/A	N/A	N/A	N/A	N/A
MERGE	N/A	NO	TABLE	YES	NO	N/A	NO
FEDERATED	N/A	N/A	N/A	N/A	N/A	N/A	N/A
INNODB	64TB	YES	ROW	YES	NO	YES	YES

Q&A