链滴

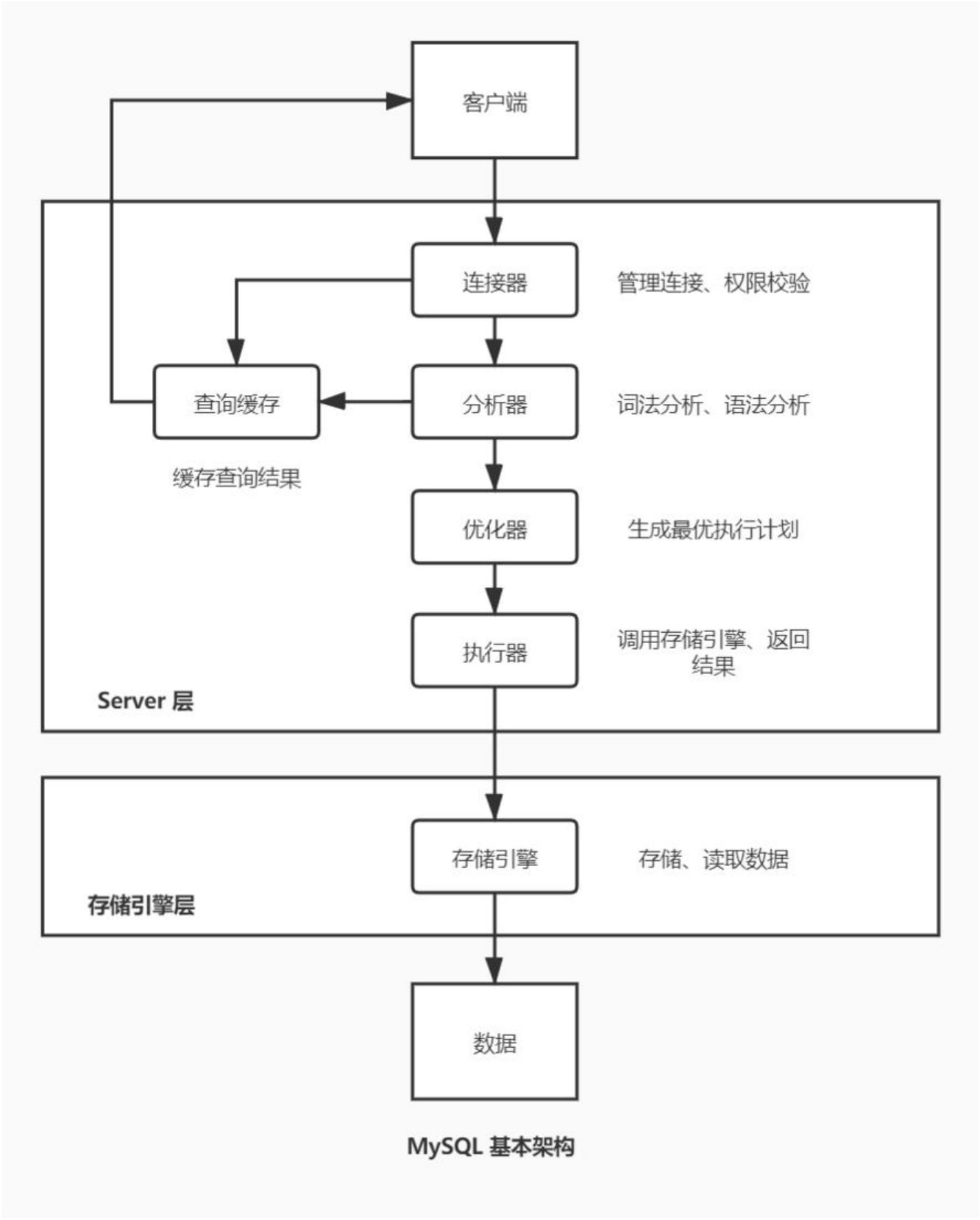
MySQL 面试必备知识点

作者：jingqueyimu

原文链接：https://ld246.com/article/1617980645326

来源网站：链滴

1、MySQL 基本架构



● Server 层：主要包括连接器、查询缓存、分析器、优化器、执行器等，以及通用的 binlog 日志模
。

- 连接器：管理连接，身份、权限验证。
    - 查询缓存：缓存查询结果集，key 为 SQL 语句，value 为结果集。8.0 版本后已移除。
    - 分析器：分析 SQL 语句用途及语法是否正确，包括词法分析、语法分析。没有命中缓存才进入析器。
    - 优化器：选择一条 MySQL 认为最优的执行计划。
    - 执行器：执行语句，调用存储引擎，返回执行数据。
- 存储引擎层：负责数据的存储和读取。插件式架构。支持 MyISAM、InnoDB 等存储引擎，其中 InoDB 自带 redo log 日志模块。

2、SQL 语句执行流程

- 查询语句：权限校验 -> 查询缓存 -> 分析器 -> 优化器 -> 权限校验 -> 执行器 -> 存储引擎。
- 更新语句（增删改）：分析器 -> 权限校验 -> 执行器 -> 存储引擎（写 undo log、存储事务 ID 回滚指针 -> 从磁盘/内存读取数据，修改数据，更新内存，异步刷盘 -> 写 redo log（prepare 状态 -> 写 binlog，刷盘 -> redo log 刷盘（commit 状态））。

3、MySQL 存储引擎

- MyISAM：5.5 版本之前默认的存储引擎，只有表级锁，不支持事务、外键、MVCC，崩溃后无法全恢复。
- InnoDB：5.5 版本之后默认的存储引擎，支持行级锁（默认）、表级锁，支持事务、外键、MVCC 具有崩溃修复能力。

4、事务四大特性（ACID）

- 原子性（Atomicity）：事务不允许分割，一组操作要么全部成功，要么全部失败。
- 一致性（Consistency）：事务执行前后，数据保持一致，多个事务对同一数据的读取结果是相同。
- 隔离性（Isolation）：并发访问数据库时，一个事务不被其他事务所干扰。
- 持久性（Durability）：事务被提交后，它对数据的改变是持久的。

5、如何保证 ACID

- 原子性（A）：由 undo log（回滚日志） 保证，它记录了需要回滚的日志信息。
- 一致性（C）：数据库层面的一致性由 A、I、D 保证，C 是目的，A、I、D 是手段，是为了保证 C 应用层面的一致性由代码控制，通过代码判断数据是否有效，然后决定是提交还是回滚。
- 隔离性（I）：由锁和 MVCC 保证。
- 持久性（D）：由 redo log 保证，当修改数据时，会在 redo log 记录该次操作。提交事务时，会将 redo log 中的数据进行刷盘；宕机时，会将 redo log 中的数据恢复到数据库。

6、并发事务会导致什么问题

- 脏读（Dirty read）：一个事务读取到了另一个事务未提交的更新。

● 丢失更新（Lost update）：一个事务的更新被另一个事务的更新所覆盖。

● 不可重复读（Unrepeatable read）：一个事务读取到了另一个事务已提交的更新或删除，即在一事务内多次读取时，上次读取到的那几条数据发生了变化（变了或没了）。

● 幻读（Phantom read）：一个事务读取到了另一个事务的插入，即在一个事务内多次读取时，出了之前没有的数据。

7、四个隔离级别

● 读取未提交（READ-UNCOMMITTED）：允许读取未提交的数据，最低的隔离级别。

● 读取已提交（READ-COMMITTED）：允许读取并发事务已提交的数据。

● 可重复读（REPEATABLE-READ）：事务内多次读取返回的结果相同，除非事务本身自己修改。

● 串行化（SERIALIZABLE）：所有事务依次逐个执行，最高的隔离级别，完全服从 ACID。

隔离级别	**脏读**	**不可重复读**	**读**
READ-UNCOMMITTED	√	√	√
READ-COMMITTED	×	√	√
REPEATABLE-READ	×	×	√
SERIALIZABLE	×	×	×

8、MVCC

多版本并发控制（Multiversion Concurrency Control），通过维持一份数据的多个版本来实现并发制。只在 REPEATABLE-READ（RR）、READ-COMMITTED（RC） 隔离级别下工作。

● MVCC 是一种用来解决读写冲突的无锁并发控制，它的实现依赖于版本链和读视图（Read View）。

● 版本链：事务对记录的每次修改，都会在 undo log 中保存一份修改前的记录。当多个事务对同一录进行修改时，undo log 中就会存在该记录的多个版本。

　● 记录中包含事务 ID 和回滚指针。事务 ID 为修改该记录的事务 ID，它会一直递增；回滚指针则向记录的上一版本。

　● undo log 中记录的多个版本通过回滚指针串联成一个版本链。

● 读视图：当事务进行快照读时，会生成一份当前时刻的数据快照，并且记录当前活跃事务 ID（未提的事务 ID）。读视图根据可见性算法，顺着版本链判断哪个版本是当前事务可见的，直到找到一份可版本为止。

　● 如果某个版本的事务 ID 等于当前事务 ID，说明该版本是当前事务创建的，因此该版本对当前务可见。

　● 如果某个版本的事务 ID 小于活跃事务 ID 列表中的最小 ID，说明生成该版本的事务在生成读视前已经提交了，因此该版本对当前事务可见。

　● 如果某个版本的事务 ID 不在活跃事务 ID 列表中，并且小于活跃事务 ID 列表中的最大 ID，则明生成该版本的事务在生成读视图前已经提交了，因此该版本对当前事务可见。

● RR 和 RC 生成读视图的时机不同。

　● RR：只在同一事务的第一次快照读，生成读视图，之后都使用该读视图。

- RC：同一事务的每次快照读，都会生成一份新的读视图。

9、当前读、快照读

- 当前读：读取的是记录的最新版本，读取时其他并发事务不能修改当前记录。会对读取的记录加锁。
- 快照读：读取的是记录的可见版本（有可能是历史版本）。不会对记录进行加锁。串行级别下会退为当前读。快照读的实现基于 MVCC。

10、索引

- 哈希索引：底层为哈希表，适合单条记录查询。
- BTree 索引：有序，适合多条记录查询（范围查询），MySQL 中使用的是 B+Tree。

11、MyISAM、InnoDB 对 B+Tree 索引的实现

- MyISAM：非聚簇索引，索引文件和数据文件分离。
    - 主索引：叶节点的 data 域存放的是数据记录的地址，key 必须唯一；索引检索时，根据 key 取 data 域的地址值，根据地址值读取相应的数据记录。
    - 辅助索引：结构上与主索引一样，key 可以重复。
- InnoDB：聚簇索引，数据文件本身就是索引文件。
    - 主索引：表数据文件本身就是按 B+Tree 组织的一个索引结构，叶节点的 data 域保存了完整的据记录，key 为表的主键。
    - 辅助索引：叶节点的 data 域存放的是主键，检索时需要先获得主键，再用主键到主索引中检索得记录（回表）。

12、覆盖索引

- 一次查询中，一个索引（一般是联合索引）覆盖（包含）所有需要查询的字段。
- 通过索引即可直接获取查询结果，而不用回表查询。

13、锁类型

- 共享锁（读锁）：其他事务可以读，但不能写。
- 排他锁（写锁）：其他事务即不能读，也不能写。
    - 表锁：锁定整张表。开销小，加锁快，不会出现死锁；锁粒度大，冲突概率高，并发度低。
    - 行锁：锁定一行记录。开销大，加锁慢，会出现死锁；锁粒度小，冲突概率低，并发度高。可为乐观锁、悲观锁，乐观锁可通过版本号实现。

14、InnoDB 的三种锁算法

- Record lock：单个行记录上的锁。
- Gap lock：间隙锁，锁定一个范围，不包括记录本身。

- Next-key lock：Record + Gap，锁定一个范围，包括记录本身。

15、关于 MySQL 的锁

- InnoDB 默认支持的隔离级别是 REPEATABLE-READ。该隔离级别下，对于快照读，MySQL 使用 VCC 来避免幻读；而对于当前读，MVCC 依然会出现幻读，需要使用加锁读才可避免幻读。加锁读用的是 Next-key lock 锁算法。
- 当查询的索引含有唯一属性时，Next-key lock 降级为 Record lock。

16、大表优化

- 限定数据范围：查询时带上限制数据范围的条件。
- 读写分离：主库负责写，从库负责读。
- 垂直拆分：列的拆分，把一张列比较多的表拆分为多张表。
- 水平拆分：行的拆分，把一张表的数据拆成多张表来存放，最好分库。

17、分库分表后如何保证 ID 唯一性

- UUID：无序，查询效率低，不推荐。
- 自增 ID 设定步长：使 ID 落到不同的表上。
- Redis 生成 ID：利用 Redis 的 incr 命令实现 ID 原子性自增。
- 分布式 ID 算法：比如 SnowFlake（雪花）算法。

Reference

[1] https://snailclimb.gitee.io/javaguide-interview/#/./docs/d-1-mysql

[2] https://www.cnblogs.com/wyq178/p/11576065.html

[3] https://blog.csdn.net/SnailMann/article/details/94724197

[4] https://zhuanlan.zhihu.com/p/343134817

[5] 《高性能 MySQL》