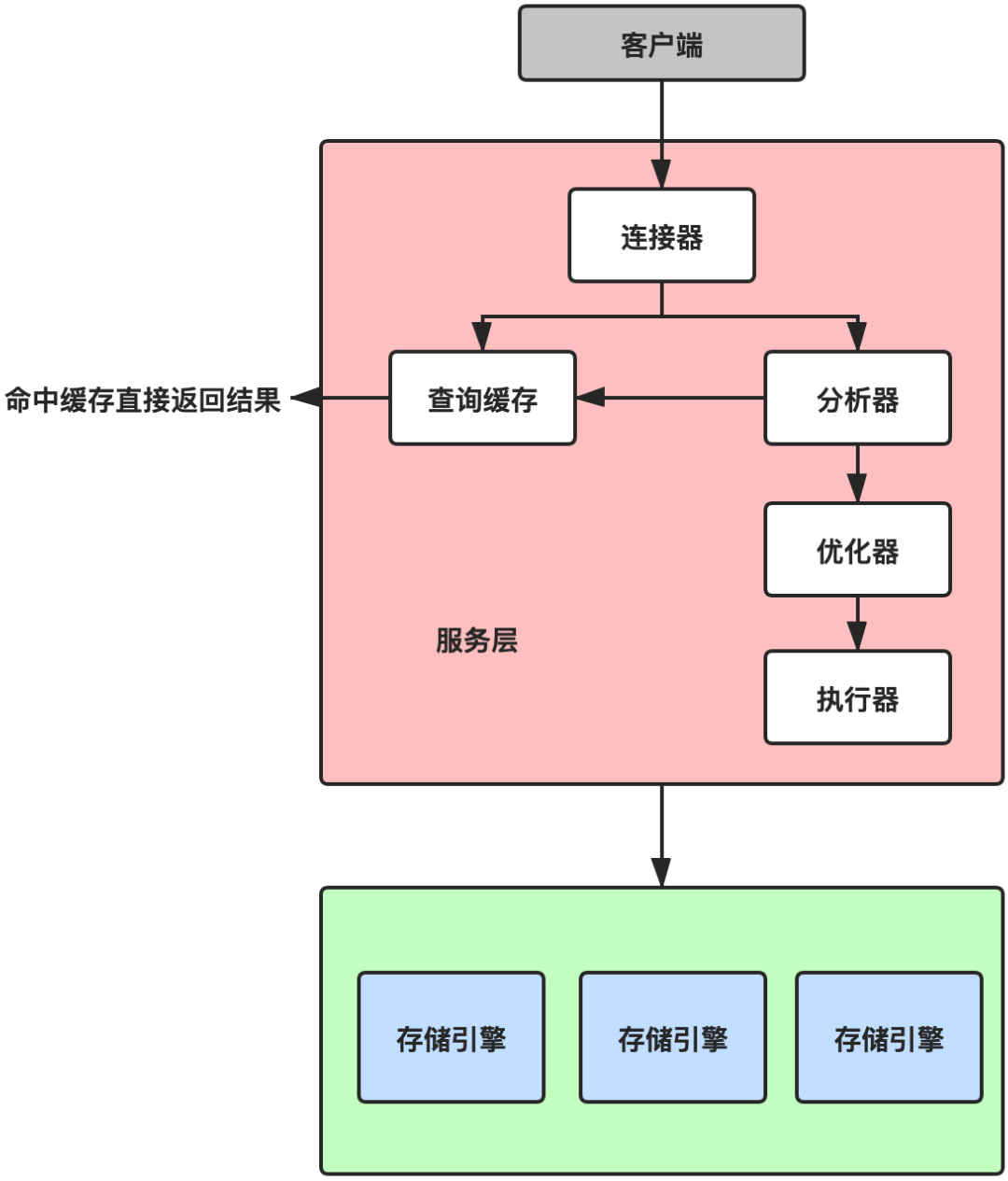
数据库相关

1. MySQl执行一条语句的步骤



MySQL内部分为服务层和数据引擎层两部分

- **服务层**：包括**连接器、查询缓存、分析器、优化器、执行器**等，所有跨存储引擎的功能都在这一层实现
- **数据引擎**：负责数据的存储和提取，其架构是插件式的，支持InnoDB、MyISAM、Memory等多个，最常用的是InnoDB

Server执行SQL语句的步骤

客户端请求

- 连接器：验证用户身份，给予权限
- 查询缓存：存在缓存直接返回，否则继续
- 分析器：对SQL语句进行词法和语法分析操作
- 优化器：选择最优的SQL优化方案执行
- 执行器：确认用户是否有权限，有的话使用引擎提供的接口
- 数据引擎层返回结果：如果开启缓存则会缓存查询结果

2. 并发事务带来的问题

脏读

> 第一个事务首先读取变量为50，接着准备更新为100的时，并未提交，第二个事务已经读取为100，此时第一个事务做了回滚。

丢弃修改

> T1 和 T2 两个事务都对一个数据进行修改，T1 先修改，T2 随后修改，**T2 的修改覆盖了 T1 的修改**

不可重复读

> T2 读取一个数据，T1 对该数据做了修改并提交。如果 T2 再次读取这个数据，此时读取的结果和第一次读取的结果不同。

幻读

> T1 读取某个范围的数据，T2 在这个范围内插入新的数据，T1 再次读取这个范围的数据，此时读取的结果和第一次读取的结果不同。（和不可重复读的区别：一个是变量变化，一个是范围变化）

3. 数据库的隔离级别

> InnoDB的默认隔离级别是**REPEATABLE-READ**（可重读）;与SQL标准不同的地方在于InnoDB 存储引擎在 REPEATABLE-READ（可重读）事务隔离级别 下使用的是**Next-Key Lock 锁**算法，因此可以避免幻读的产生，已经可以完全保证事务的隔离性要求，即达到了 SQL标准的SERIALIZABLE(可串行化)隔离级别。

READ_UNCOMMITED (未提交读)

> 事务中发生了修改，即使没有提交，其他事务也是可见的,**可能会导致脏读、幻读或不可重复读**

READ_COMMITED提交读

> 对于一个事务从开始直到提交之前，所做的任何修改对于其他事务都是不可见的。**可以阻止脏读，但是幻读和不可重复读仍有可能发生**

REPEATABLE-READ可重复读

> 就是对一个记录读取多次的记录是相同的**可以阻止脏读和不可重复读，但幻读仍有可能发生**

SERIALIZABLE可串行读

> 在并发情况下，和串行化的读取的结果是一致的，没有什么不同；**该级别可以防止脏读、不可重复读以及幻读**

4. ACID

> 分别是原子性、一致性、隔离性、持久性

原子性

> 一个事务是不可分割的工作单位，**其中的操作要么都做，要么都不做**

一致性

> 事物执行前后，数据处于一种合法的状态，这种状态是语义上的而不是语法上的。**满足所定义的状态，数据就是一致的，不满足这个状态，数据就是不一致的了**

隔离性

> 多个事务并发执行的时候，事务内部的操作与其他事务是隔离的

持久性

> 事务一旦提交，它对数据库的改变就是永久性的

5. InnoDB和MyISAM的区别

InnoDB

- MySQL默认的事务型存储引擎
- 实现了四个隔离级别，默认是REPEATABLE_READ，通过**多版本并发控制（MVCC）+（Next-Key Locking）防止幻读**
- 主索引是聚簇索引，在索引中保存了数据，从而避免直接读取磁盘，性能很大提升
- 内部做了很多优化，包括从磁盘读取数据时采用的**可预测性读**、能够加快读操作并且自动创建**自适应哈希索引**、能够加速插入操作的**插入缓冲区**等。
- 支持真正的**在线热备份。**

MyISAM

- 设计简单，数据以**紧密格式存储。**对于只读数据，或者表比较小、可以容忍修复操作，则依然可以使用它
- 提供了大量的特性，包括**压缩表、空间数据索引**等
- **不支持事物**
- **不支持行级锁，只能对整张表加锁**

5. MySQL的日志文件

> undolog，redolog，binlog

undolog回滚日志文件

> 主要用于事务中执行失败，进行回滚，以及MVCC中对于数据历史版本的查看。由**引擎层的InnoDB引擎实现**,是**逻辑日志**

redolog重做日志文件

> **记录数据修改之后的值，用于持久化到磁盘中。**redo log包括两部分：**一是内存中的日志缓冲(redo log buffer)，该部分日志是易失性的**；二是**磁盘上的重做日志文件(redo log file)**，该部分日志是持久的。由**引擎层的InnoDB引擎实现**,是**物理日志**

binlog逻辑日志文件

> Mysql的Server层实现,是**逻辑日志**,记录的是sql语句的原始逻辑.**事务提交的时候,一次性将事务中的sql语句,按照一定的格式记录到binlog中。用于复制和恢复在主从复制中，从库利用主库上的binlog进行重播(执行日志中记录的修改逻辑),实现主从同步。业务数据不一致或者错了，用binlog恢复。**

binlog和redolog的区别

- binlog是Server实现的逻辑日志，redolog是存储引擎产生的物理日志
- **binlog日志只在事务提交完成后进行一次写入。**而**innodb存储引擎的redolog在事务进行中不断地被写入，并日志不是随事务提交的顺序进行写入的。**
- binlog不是循环使用，在写满或重启之后，会生成新的binlog文件，redolog是循环使用
- **binlog可以作为恢复数据使用，主从复制搭建，redolog作为异常宕机或者介质故障后的数据恢复使用。**

6. MVCC多版本并发控制的缺点

> 大多数情况下代替了行锁，缺点是每行记录**都需要额外的存储空间，需要做更多的维护和检查工作**

7. 索引类类型

FULLTEXT

> 全文索引，只有MyISAM引擎支持

Hash

> 由于HASH的唯一及类似键值对的形式，很适合作为索引。 HASH索引可以一次定位，不需要像树形索引那样逐层查找,因此具有极高的效率

B Tree

> BTREE索引就是一种将索引值按一定的算法，存入一个树形的数据结构中（二叉树），每次查询都是从树的入口root开始，依次遍历node，获取leaf。这是MySQL里默认和最常用的索引类型。

R Tree

> TREE：RTREE在MySQL很少使用，仅支持geometry数据类型

8. 索引种类

- 普通索引：仅加速查询
- 唯一索引：加速查询 + 列值唯一（可以有null）
- 主键索引：加速查询 + 列值唯一（不可以有null）+ 表中只有一个
- 组合索引：多列值组成一个索引，专门用于组合搜索，其效率大于索引合并
- 全文索引：对文本的内容进行分词，进行搜索
- 索引合并：使用多个单列索引组合搜索

- 覆盖索引：select的数据列只用从索引中就能够取得，不必读取数据行，换句话说查询列要被所建的索引覆盖
- 聚簇索引：表数据是和主键一起存储的，主键索引的叶结点存储行数据(包含了主键值)，二级索引的叶结点存储行的主键值。使用的是B+树作为索引的存储结构，非叶子节点都是索引关键字，但非叶子节点中的关键字中不存储对应记录的具体内容或内容地址。叶子节点上的数据是主键与具体记录(数据内容)

9. 索引结构

MyISAM

- 索引文件和数据文件时分离的，索引文件仅保存数据记录的地址，同样使用B+树作为索引结构，叶节点的data域存放数据记录的地址
- 主和辅助索引没有区别，只是主索引要求key唯一，辅助索引不用

InnoDB

- 数据文件本身就是索引文件，树的叶节点data域保存了保证的数据记录（聚集索引）
- 辅助索引data域存储相应记录主键的值而不是地址

10. 索引最左

举例子:

如果索引列分别为A，B，C，顺序也是A，B，C:

- 那么查询的时候，如果查询【A】【A，B】【A，B，C】，那么可以通过索引查询
- 如果查询的时候，采用【A，C】，那么C这个虽然是索引，但是由于中间缺失了B，因此C这个索引是用不到的，只能用到A索引
- 如果查询的时候，采用【B】【B，C】【C】，由于没有用到第一列索引，不是最左前缀，那么后面的索引也是用不到了
- 如果查询的时候，采用范围查询，并且是最左前缀，也就是第一列索引，那么可以用到索引，但是范围后面的列无法用到索引（比如，a>= 3 and b = 4 and c = 5; A走索引，bc不走）（比如，a = 3 and b >= 4 and c = 5; a和b走，c不走）
  组合索引的底层其实按照第一个索引排序，从排序里面查第二个索引，以此类推。如果第一个索引失效，或者没有经过第一个索引，后面没发在前面的基础上查询。

11. 为什么使用索引

- 通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性
- 大幅加快数据的检索速度
- 帮助服务器避免排序和临时表
- 将随机IO变为顺序IO
- 加速表和表之间的连接

索引的缺点

- 需要动态维护
- 占物理空间
- 创建和维护索引需要时间

12. MySQL的锁

MyISAM

> 只有表锁，表锁又分为共享读锁和独占写锁
> 读写操作、写操作之间是串行的

InnoDB

> 杭锁：共享锁、排它锁

13. MySQL调优

> 分为索引优化和结构优化

三范式

- 第一范式： 第一范式为原子性，字段不可再分割
- 第二范式： 确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关
- 第三范式： 所有主键和非主键之间不能产生传递依赖

反范式设计

> 为了性能和读取效率的考虑而适当的对数据库设计范式的要求进行违反，而允许存在少量的数据冗余