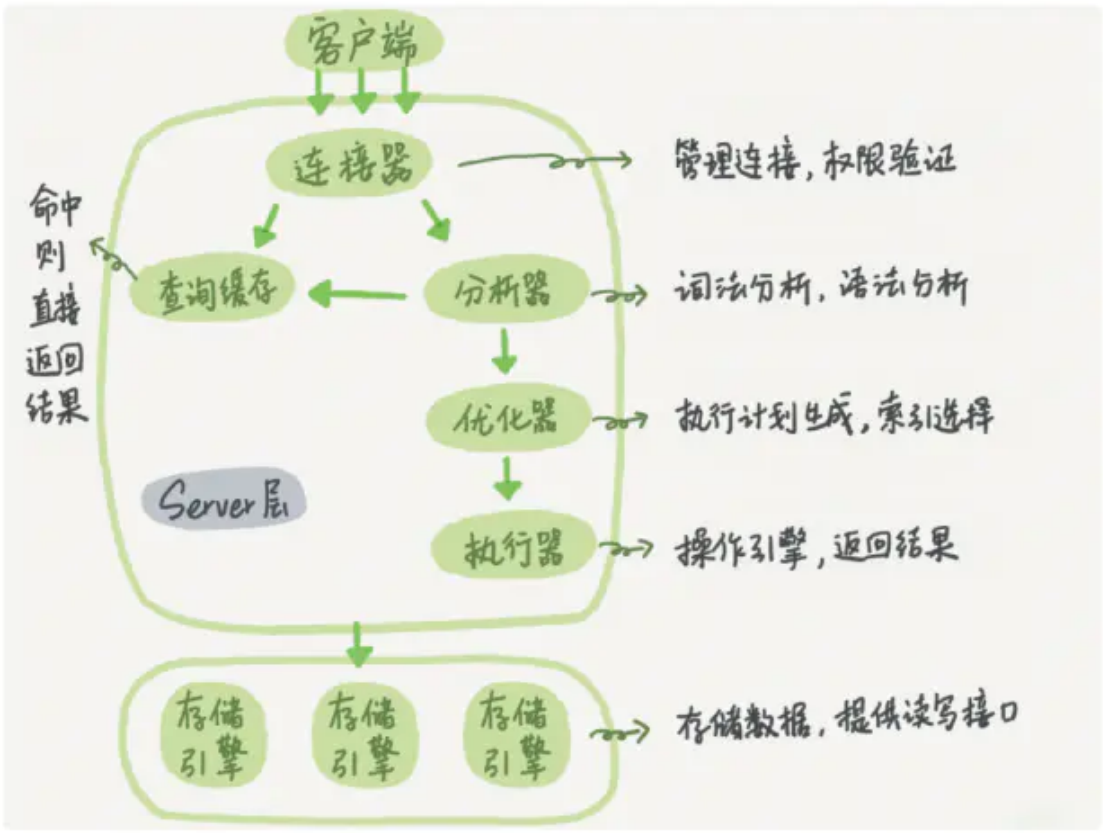
@来源 [快问快答，MySQL面试夺命20问 (qq.com)]

数据库架构

说说MySQL的基础架构图



Mysql逻辑架构图主要分三层：

（1）第一层负责连接处理，授权认证，安全等等

（2）第二层负责编译并优化SQL

（3）第三层是存储引擎。

一条SQL查询语句在MySQL中如何执行的？

> - 先检查该语句**是否有权限**，如果没有权限，直接返回错误信息，如果有权限会先查询缓存(MySQL8.0 版本以前)。
> - 如果没有缓存，分析器进行**词法分析**，提取 sql 语句中 select 等关键元素，然后判断 sql 语句是否有语法错误，比如关键词是否正确等等。
> - 最后优化器确定执行方案进行权限校验，如果没有权限就直接返回错误信息，如果有权限就会**调用数据库引擎接口**，返回执行结果。

SQL优化

日常工作中你是怎么优化SQL的?

可以从这几个维度回答这个问题:

> **1.优化表结构**
>
> （1）尽量使用数字型字段
>
> 若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。
>
> （2）尽可能的使用 varchar 代替 char
>
> 变长字段存储空间小，可以节省存储空间。
>
> （3）当索引列大量重复数据时，可以把索引删除掉
>
> 比如有一列是性别，几乎只有男、女、未知，这样的索引是无效的。
>
> **2.优化查询**
>
> - 应尽量避免在 where 子句中使用!=或<>操作符
> - 应尽量避免在 where 子句中使用 or 来连接条件
> - 任何查询也不要出现select *
> - 避免在 where 子句中对字段进行 null 值判断
>
> **3.索引优化**
>
> - 对作为查询条件和 order by的字段建立索引
> - 避免建立过多的索引，多使用组合索引

怎么看执行计划(explain)，如何理解其中各个字段的含义?

在 select 语句之前增加 explain 关键字，会返回执行计划的信息。

```
mysql> explain select name from student;
+----+-------------+-------+------+---------------+------+---------+------+------+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra |
+----+-------------+-------+------+---------------+------+---------+------+------+-------+
|  1 | SIMPLE      |student| ALL  | NULL          | NULL | NULL    | NULL |    2 | NULL  |
+----+-------------+-------+------+---------------+------+---------+------+------+-------+
```

（1）id 列：是 select 语句的序号，MySQL将 select 查询分为简单查询和复杂查询。

（2）select_type列：表示对应行是是简单还是复杂的查询。

（3）table 列：表示 explain 的一行正在访问哪个表。

（4）type 列：最重要的列之一。表示关联类型或访问类型，即 MySQL 决定如何查找表中的行。从最优到最差分别为：system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL

（5）possible_keys 列：显示查询可能使用哪些索引来查找。

（6）key 列：这一列显示 mysql 实际采用哪个索引来优化对该表的访问。

（7）key_len 列：显示了mysql在索引里使用的字节数，通过这个值可以算出具体使用了索引中的哪些列。

（8）ref 列：这一列显示了在key列记录的索引中，表查找值所用到的列或常量，常见的有：const（常量），func，NULL，字段名。

（9）rows 列：这一列是 mysql 估计要读取并检测的行数，注意这个不是结果集里的行数。

（10）Extra 列：显示额外信息。比如有 Using index、Using where、Using temporary等。

关心过业务系统里面的sql耗时吗？统计过慢查询吗？对慢查询都怎么优化过？

我们平时写Sql时，都要养成用explain分析的习惯。慢查询的统计，运维会定期统计给我们

优化慢查询思路：

> - 分析语句，是否加载了不必要的字段/数据
> - 分析 SQL 执行句话，是否命中索引等
> - 如果 SQL 很复杂，优化 SQL 结构
> - 如果表数据量太大，考虑分表

索引

聚集索引与非聚集索引的区别

可以按以下四个维度回答：

> （1）一个表中只能拥有一个聚集索引，而非聚集索引一个表可以存在多个。
>
> （2）聚集索引，索引中键值的逻辑顺序决定了表中相应行的物理顺序；非聚集索引，索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同。
>
> （3）索引是通过二叉树的数据结构来描述的，我们可以这么理解聚簇索引：索引的叶节点就是数据节点。而非聚簇索引的叶节点仍然是索引节点，只不过有一个指针指向对应的数据块。
>
> （4）聚集索引：物理存储按照索引排序；非聚集索引：物理存储不按照索引排序；

为什么要用B+树，为什么不用普通二叉树?

可以从几个维度去看这个问题，查询是否够快，效率是否稳定，存储数据多少，以及查找磁盘次数，为什么不是普通二叉树，为什么不是平衡二叉树，为什么不是B树，而偏偏是 B+ 树呢?

> （1）**为什么不是普通二叉树?**
>
> 如果二叉树特殊化为一个链表，相当于全表扫描。平衡二叉树相比于二叉查找树来说，查找效率更稳定，总体的查找速度也更快。
>
> （2）**为什么不是平衡二叉树呢?**
>
> 我们知道，在内存比在磁盘的数据，查询效率快得多。如果树这种数据结构作为索引，那我们每查找一次数据就需要从磁盘中读取一个节点，也就是我们说的一个磁盘块，但是平衡二叉树可是每个节点只存储一个键值和数据的，如果是B树，可以存储更多的节点数据，树的高度也会降低，因此读取磁盘的次数就降下来啦，查询效率就快啦。
>
> （3）**为什么不是 B 树而是 B+ 树呢?**
>
> B+ 树非叶子节点上是不存储数据的，仅存储键值，而B树节点中不仅存储键值，也会存储数据。innodb中页的默认大小是16KB，如果不存储数据，那么就会存储更多的键值，相应的树的阶数（节点的子节点树）就会更大，树就会更矮更胖，如此一来我们查找数据进行磁盘的IO次数有会再次减少，数据查询的效率也会更快。
>
> B+ 树索引的所有数据均存储在叶子节点，而且数据是按照顺序排列的，链表连着的。那么 B+ 树使得范围查找，排序查找，分组查找以及去重查找变得异常简单。

Hash索引和B+树索引区别是什么？你在设计索引是怎么抉择的？

- B+ 树可以进行范围查询，Hash 索引不能。
- B+ 树支持联合索引的最左侧原则，Hash 索引不支持。
- B+ 树支持 order by 排序，Hash 索引不支持。
- Hash 索引在等值查询上比 B+ 树效率更高。
- B+ 树使用 like 进行模糊查询的时候，like 后面（比如%开头）的话可以起到优化的作用，Hash 索引根本无法进行模糊查询。

什么是最左前缀原则？什么是最左匹配原则？

最左前缀原则，就是最左优先，在创建多列索引时，要根据业务需求，where 子句中使用最频繁的一列放在最左边。

当我们创建一个组合索引的时候，如 (a1,a2,a3)，相当于创建了（a1）、(a1,a2)和(a1,a2,a3)三个索引，这就是最左匹配原则。

索引不适合那些场景？

- 数据量少的不适合加索引
- 更新比较频繁的也不适合加索引 = 区分度低的字段不适合加索引（如性别）

索引有哪些优缺点？

(1) 优点：

- 唯一索引可以保证数据库表中每一行的数据的唯一性
- 索引可以加快数据查询速度，减少查询时间

(2)缺点：

- 创建索引和维护索引要耗费时间
- 索引需要占物理空间，除了数据表占用数据空间之外，每一个索引还要占用一定的物理空间
- 以表中的数据进行增、删、改的时候，索引也要动态的维护。

锁

MySQL遇到过死锁问题吗，你是如何解决的？

遇到过。我排查死锁的一般步骤是酱紫的：

（1）查看死锁日志

show engine innodb status;

（2）找出死锁Sql

（3）分析sql加锁情况

（4）模拟死锁案发

（5）分析死锁日志

（6）分析死锁结果

说说数据库的乐观锁和悲观锁是什么以及它们的区别?

（1）悲观锁：

悲观锁她专一且缺乏安全感了，她的心只属于当前事务，每时每刻都担心着它心爱的数据可能被别的事务修改，所以一个事务拥有（获得）悲观锁后，其他任何事务都不能对数据进行修改啦，只能等待锁被释放才可以执行。

（2）乐观锁：

乐观锁的"乐观情绪"体现在，它认为数据的变动不会太频繁。因此，它允许多个事务同时对数据进行变动。

实现方式：乐观锁一般会使用版本号机制或CAS算法实现。

MVCC熟悉吗，知道它的底层原理?

MVCC (Multiversion Concurrency Control)，即多版本并发控制技术。

MVCC在MySQL InnoDB中的实现主要是为了提高数据库并发性能，用更好的方式去处理读-写冲突，做到即使有读写冲突时，也能做到不加锁，非阻塞并发读。

事务

MySQL事务的四大特性以及实现原理

- 原子性：事务作为一个整体被执行，包含在其中的对数据库的操作要么全部被执行，要么都不执行。
- 一致性：指在事务开始之前和事务结束以后，数据不会被破坏，假如A账户给B账户转10块钱，不管成功与否，A和B的总金额是不变的。
- 隔离性：多个事务并发访问时，事务之间是相互隔离的，即一个事务不影响其它事务运行效果。简言之，就是事务之间是井水不犯河水的。
- 持久性：表示事务完成以后，该事务对数据库所作的操作更改，将持久地保存在数据库之中。

事务的隔离级别有哪些？MySQL的默认隔离级别是什么?

- 读未提交（Read Uncommitted）
- 读已提交（Read Committed）
- 可重复读（Repeatable Read）
- 串行化（Serializable）

Mysql默认的事务隔离级别是可重复读(Repeatable Read)

什么是幻读，脏读，不可重复读呢?

事务A、B交替执行，事务A被事务B干扰到了，因为事务A读取到事务B未提交的数据，这就是脏读。

在一个事务范围内，两个相同的查询，读取同一条记录，却返回了不同的数据，这就是不可重复读。

事务A查询一个范围的结果集，另一个并发事务B往这个范围中插入/删除了数据，并静悄悄地提交，然后事务A再次查询相同的范围，两次读取得到的结果集不一样了，这就是幻读。

实战

MySQL数据库cpu飙升的话，要怎么处理呢?

排查过程：

（1）使用top 命令观察，确定是mysqld导致还是其他原因。

（2）如果是mysqld导致的，show processlist，查看session情况，确定是不是有消耗资源的sql在运行。

（3）找出消耗高的 sql，看看执行计划是否准确， 索引是否缺失，数据量是否太大。

处理：

（1）kill 掉这些线程(同时观察 cpu 使用率是否下降)，

（2）进行相应的调整(比如说加索引、改 sql、改内存参数)

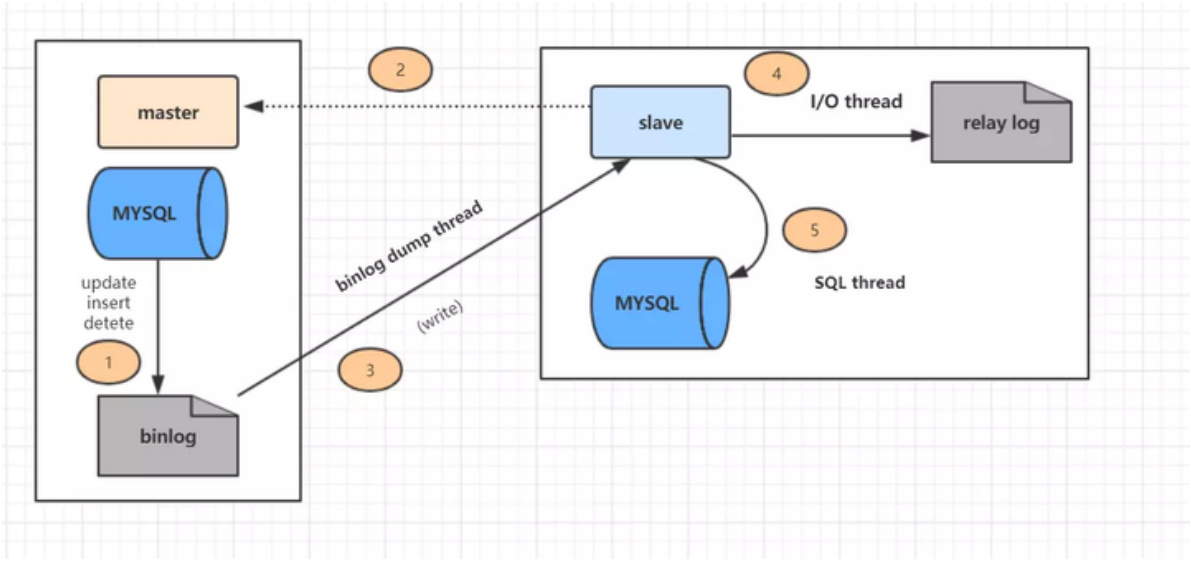
（3）重新跑这些 SQL。

其他情况：

也有可能是每个 sql 消耗资源并不多，但是突然之间，有大量的 session 连进来导致 cpu 飙升，这种情况就需要跟应用一起来分析为何连接数会激增，再做出相应的调整，比如说限制连接数等

MySQL的主从延迟，你怎么解决?

主从复制分了五个步骤进行：（图片来源于网络）



> - 步骤一：主库的更新事件(update、insert、delete)被写到binlog
> - 步骤二：从库发起连接，连接到主库。
> - 步骤三：此时主库创建一个binlog dump thread，把binlog的内容发送到从库。
> - 步骤四：从库启动之后，创建一个I/O线程，读取主库传过来的binlog内容并写入到relay log
> - 步骤五：还会创建一个SQL线程，从relay log里面读取内容，从Exec_Master_Log_Pos位置开始执行读取到的更新事件，将更新内容写入到slave的db

主从同步延迟的原因

一个服务器开放N个链接给客户端来连接的，这样有会有大并发的更新操作, 但是从服务器的里面读取binlog的线程仅有一个，当某个SQL在从服务器上执行的时间稍长 或者由于某个SQL要进行锁表就会导致，主服务器的SQL大量积压，未被同步到从服务器里。这就导致了主从不一致， 也就是主从延迟。

主从同步延迟的解决办法

- 主服务器要负责更新操作，对安全性的要求比从服务器要高，所以有些设置参数可以修改，比如sync_binlog=1，innodb_flush_log_at_trx_commit = 1 之类的设置等。

- 选择更好的硬件设备作为slave。
- 把一台从服务器当度作为备份使用， 而不提供查询， 那边他的负载下来了， 执行relay log 里面的SQL效率自然就高了。
- 增加从服务器喽，这个目的还是分散读的压力，从而降低服务器负载。

如果让你做分库与分表的设计，简单说说你会怎么做?

分库分表方案:

- 水平分库：以字段为依据，按照一定策略（hash、range等），将一个库中的数据拆分到多个库中。
- 水平分表：以字段为依据，按照一定策略（hash、range等），将一个表中的数据拆分到多个表中。
- 垂直分库：以表为依据，按照业务归属不同，将不同的表拆分到不同的库中。
- 垂直分表：以字段为依据，按照字段的活跃性，将表中字段拆到不同的表（主表和扩展表）中。

常用的分库分表中间件:

- sharding-jdbc
- Mycat

分库分表可能遇到的问题

- 事务问题：需要用分布式事务啦
- 跨节点Join的问题：解决这一问题可以分两次查询实现
- 跨节点的count,order by,group by以及聚合函数问题：分别在各个节点上得到结果后在应用程序端进行合并。
- 数据迁移，容量规划，扩容等问题
- ID问题：数据库被切分后，不能再依赖数据库自身的主键生成机制啦，最简单可以考虑UUID
- 跨分片的排序分页问题