MySql面试重点

一、MySql 索引原理

1、MySQL 索引

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引

优势

- 类似大学图书馆建书目索引，提高数据检索的效率，降低数据库的IO成本

- 通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗

劣势

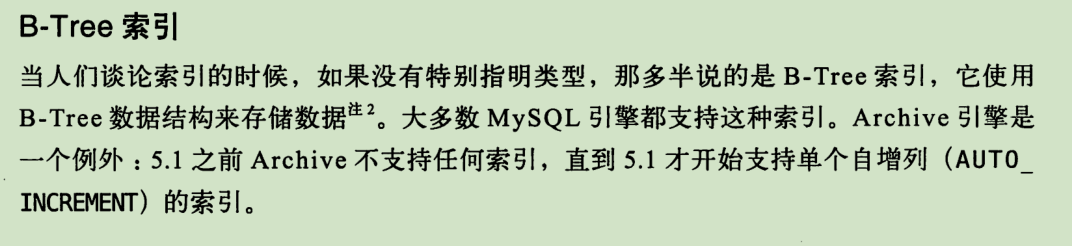
- 虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE。 因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件每次更新添加了索引列的字段， 都会调整因为更新所带来的键值变化后的索引信息

- 实际上索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引列也是要占用空间的

2、mysql索引类型

在MySQL中，索引是在存储引擎层而不是服务器层实现的。

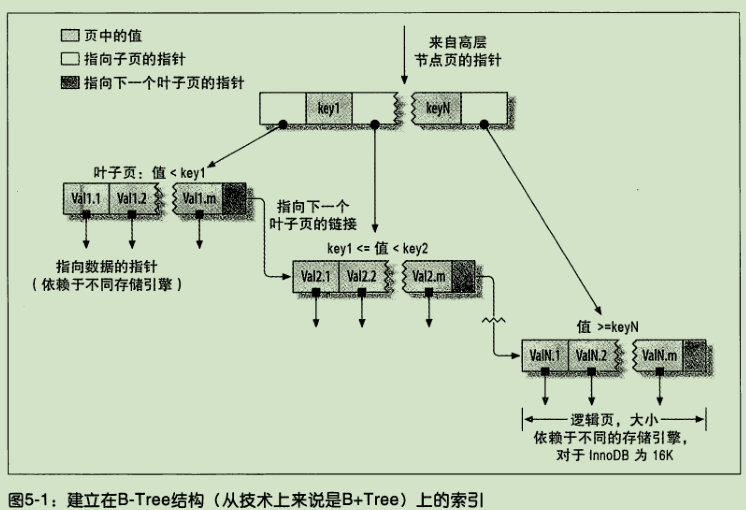
2.1 B-Tree索引



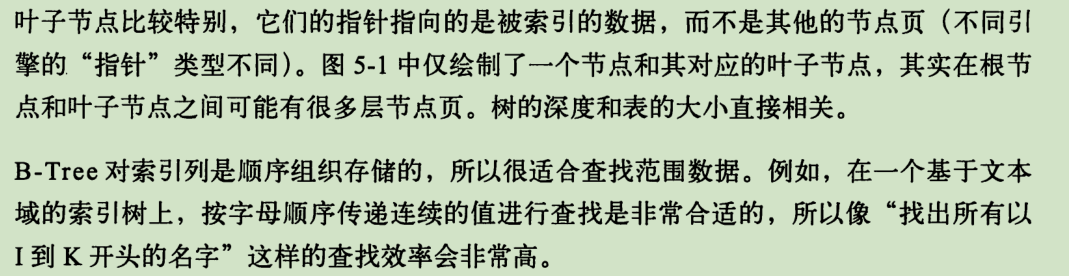
底层的存储引擎也可能使用不同的存储结构，例如，NDB集群存储引擎内部实际 上使用了 T-Tree结构存储这种索引，即使其名字是BTREE , InnoDB则使用的是B+Tree,。

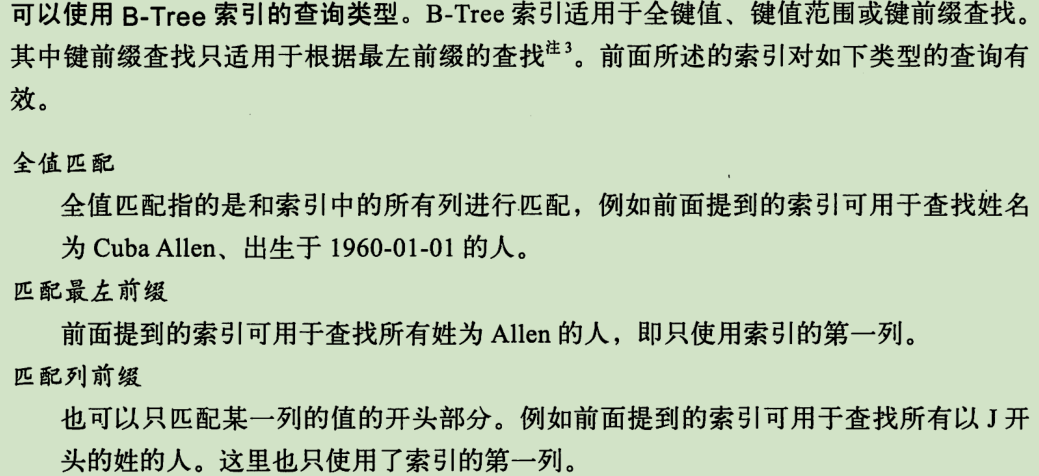
存储引擎以不同的方式使用B-Tree索引，性能也各有不同，各有优劣。例如，MylSAM 使用前缀压缩技术使得索引更小，但 InnoDB则按照原数据格式进行存储。再如 MylSAM索引通过数据的物理位置引用被索引的行，而InnoDB则根据主键引用被索引 的行。

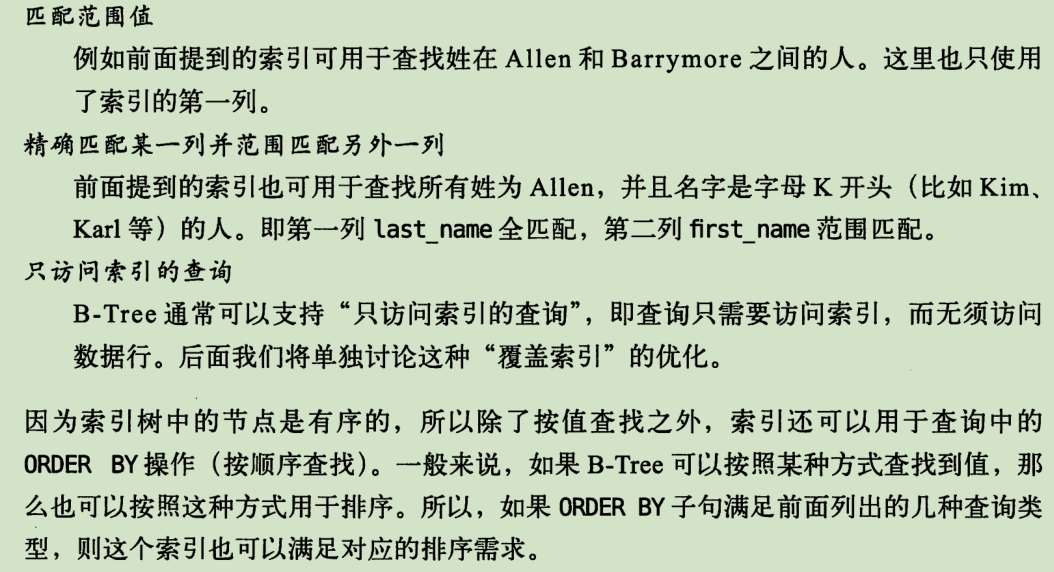
**B-Tree通常意味着所有的值都是按顺序存储的，并且每一个叶子页到根的距离相同**。下图 B-Tree索引的抽象表示，大致反映了InnoDB索引是如何工作的，MylSAM 使用的结构有所不同，但基本思想是类似的。

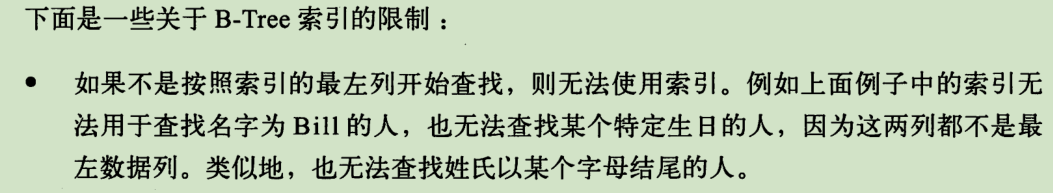


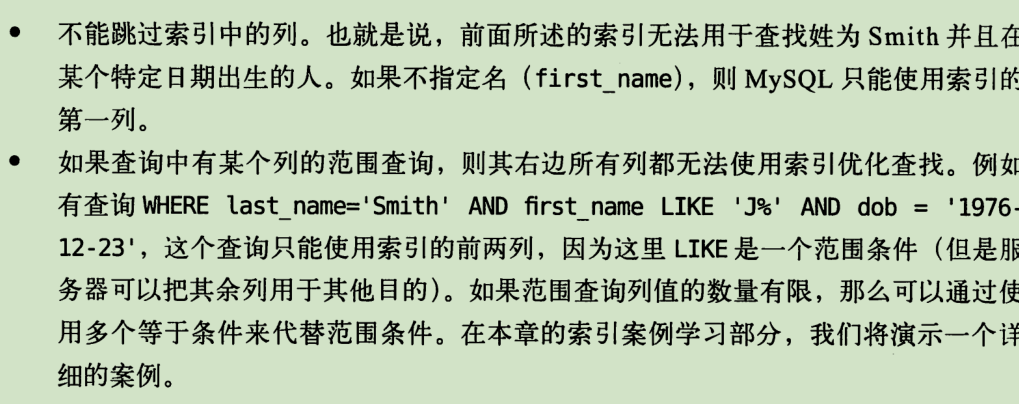
**B-Tree索引能够加快访问数据的速度，因为存储引擎不再需要进行全表扫描来获取需要 的数据，取而代之的是从索引的根节点（图示并未画出）开始进行搜索。**根节点的槽中存放了指向子节点的指针，存储引擎根据这些指针向下层査找。通过比较节点页的值和要査找的值可以找到合适的指针进入下层子节点，这些指针实际上定义了子节点页中值 的上限和下限。最终存储引擎要么是找到对应的值，要么该记录不存在。











2.2 B-Tree和B+Tree的区别

1）B-树的关键字和记录是放在一起的，叶子节点可以看作外部节点，不包含任何信息；B+树的非叶子节点中只有关键字和指向下一个节点的索引，记录只放在叶子节点中。

2）在B-树中，越靠近根节点的记录查找时间越快，只要找到关键字即可确定记录的存在；而B+树中每个记录的查找时间基本是一样的，都需要从根节点走到叶子节点，而且在叶子节点中还要再比较关键字。从这个角度看B-树的性能好像要比B+树好，而在实际应用中却是B+树的性能要好些。因为B+树的非叶子节点不存放实际的数据，这样每个节点可容纳的元素个数比B-树多，树高比B-树小，这样带来的好处是减少磁盘访问次数。尽管B+树找到一个记录所需的比较次数要比B-树多，但是一次磁盘访问的时间相当于成百上千次内存比较的时间，因此实际中B+树的性能可能还会好些，而且B+树的叶子节点使用指针连接在一起，方便顺序遍历（例如查看一个目录下的所有文件，一个表中的所有记录等），这也是很多数据库和文件系统使用B+树的缘故。

2.3 为什么说B+树比B-树更适合实际应用中操作系统的文件索引和数据库索引？

1) B+树的磁盘读写代价更低

　　B+树的内部结点并没有指向关键字具体信息的指针。因此其内部结点相对B 树更小。如果把所有同一内部结点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多。一次性读入内存中的需要查找的关键字也就越多。相对来说IO读写次数也就降低了。

2) B+树的查询效率更加稳定

由于非终结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引。所以任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当。

总结：

B树

B-tree（多路搜索树，并不是二叉的）是一种常见的数据结构。使用B-tree结构可以显著减少定位记录时所经历的中间过程，从而加快存取速度。按照翻译，B 通常认为是Balance的简称。这个数据结构一般用于数据库的索引，综合效率较高。

B树的特征：

1. 根节点至少有两个孩子

2. 每个非根节点有[ ,M]个孩；

3. 每个非根节点有[ -1,M-1]个关键字，并且以升序排列

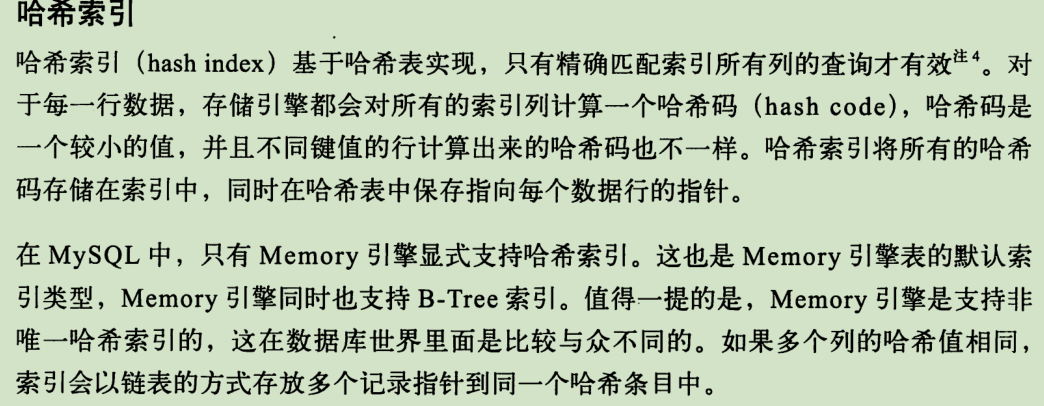
4. key[i]和key[i+1]之间的孩子节点的值介于key[i]、key[i+1]之间

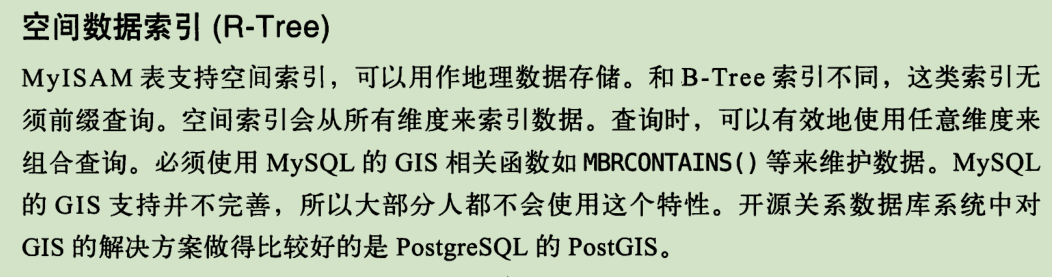
5. 所有的叶子节点都在同一层

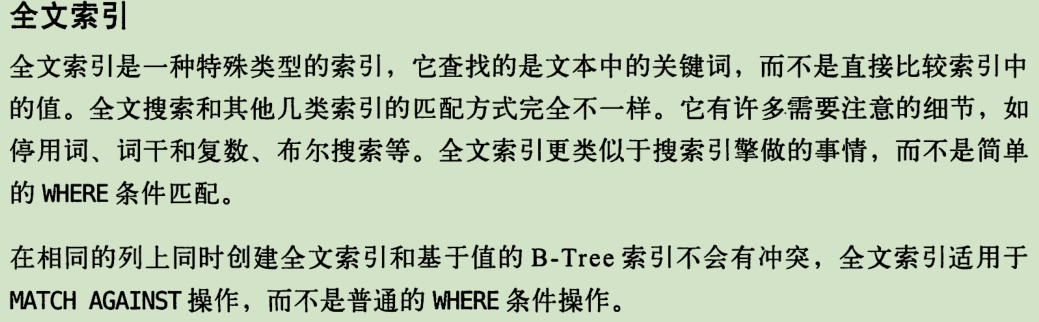
B+树

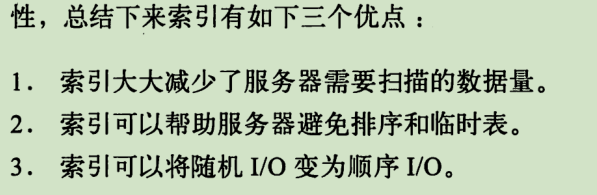
与B-Tree相比，B+Tree有以下不同点：

* 每个节点的指针上限为2d而不是2d+1。
* 内节点不存储data，只存储key；叶子节点不存储指针。



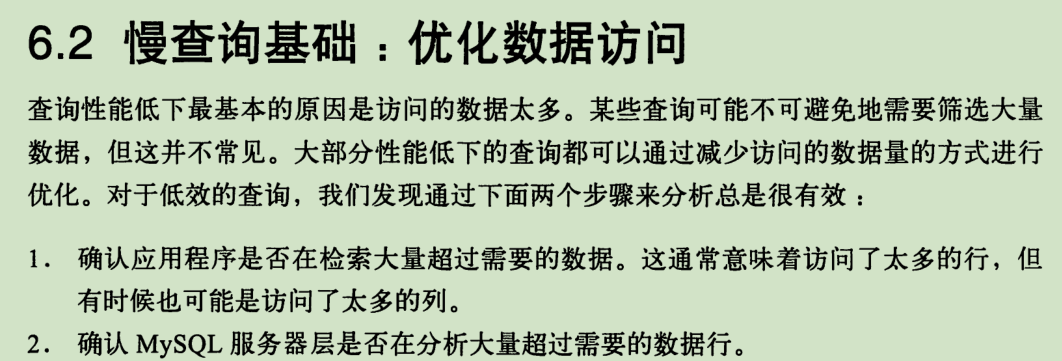






二、MySQL的优化策略







数据库表优化，索引优化

mysql优化策略：慢查询，建索引，分库分表，sharding

1、聚簇索引

聚簇索引并不是一种单独的索引类型，而是一种数据存储方式。

术语‘聚簇’表示数据行和相邻的键值聚簇的存储在一起。

2、覆盖索引

简单说就是，select 到 from 之间查询的列 <=使用的索引列+主键

3、Explain

type显示的是访问类型，是较为重要的一个指标，结果值从最好到最坏依次是：

|  |
| --- |
| system > const > eq\_ref > ref > fulltext > ref\_or\_null > index\_merge > unique\_subquery > index\_subquery > range > index > ALL |

一般来说，得保证查询至少达到range级别，最好能达到ref。

|  |
| --- |
| system>const>eq\_ref>ref>range>index>ALL |

三、MySQL引擎机制(InnoDB, Myisam)

|  |  |  |
| --- | --- | --- |
| 对比项 | MyISAM | InnoDB |
| 外键 | 不支持 | 支持 |
| 事务 | 不支持 | 支持 |
| 行表锁 | 表锁，即使操作一条记录也会锁住整个表，不适合高并发的操作 | 行锁,操作时只锁某一行，不对其它行有影响，适合高并发的操作 |
| 缓存 | 只缓存索引，不缓存真实数据 | 不仅缓存索引还要缓存真实数据，对内存要求较高，而且内存大小对性能有决定性的影响 |
| 关注点 | 读性能 | 并发写、事务、资源 |
| 默认使用 | N | Y |
| 自带系统表使用 | Y | N |

四、详述MySQL主从复制原理及配置主从的完整步骤

主从复制的原理如下：

主库开启binlog功能并授权从库连接主库，从库通过change master得到主库的相关同步信息,然后连接主库进行验证，主库IO线程根据从库slave线程的请求，从master.info开始记录的位置点向下开始取信息，同时把取到的位置点和最新的位置与binlog信息一同发给从库IO线程，从库将相关的sql语句存放在relay-log里面，最终从库的sql线程将relay-log里的sql语句应用到从库上，至此整个同步过程完成，之后将是无限重复上述过程。

完整步骤如下：

|  |
| --- |
| 1、主库开启binlog功能，并进行全备，将全备文件推送到从库服务器上  2、show master status\G 记录下当前的位置信息及二进制文件名  3、登陆从库恢复全备文件  4、执行change master to 语句  5、执行start slave and show slave status\G |

总结：

MySQL读写分离原理

读写分离就是在主服务器上修改，数据会同步到从服务器，从服务器只能提供读取数据，不能写入，实现备份的同时也实现了数据库性能的优化，以及提升了服务器安全。

一般来说都是通过 主从复制（Master-Slave）的方式来同步数据，再通过读写分离（MySQL-Proxy）来提升数据库的并发负载能力 这样的方案来进行部署与实施的。

五、MySQL密码丢了，请找回？

1、先关闭mysql服务 pkill mysql

2、使用--skip-grant-tables启动mysql，忽略授权登陆验证，

|  |
| --- |
| mysqld\_safe --defaults-file=/etc/my.cnf --skip-grant-tables & |

此时再登陆，已经不需要密码了

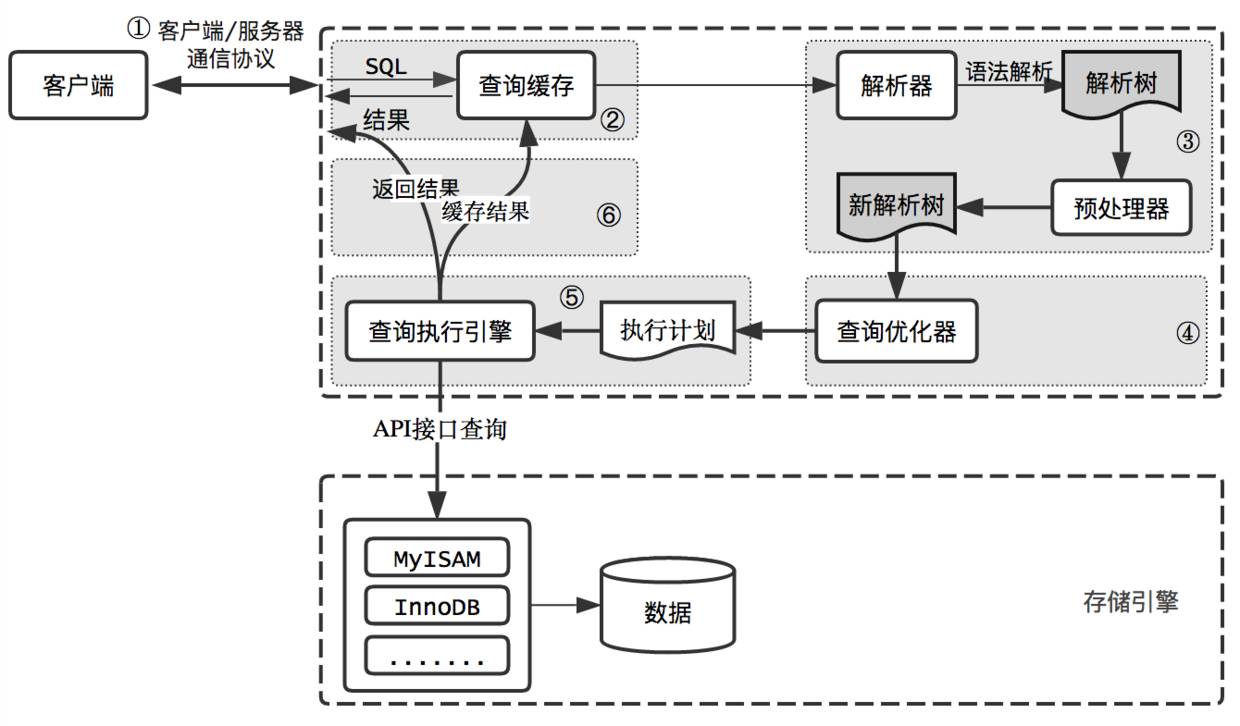
3、设置新的密码

|  |
| --- |
| mysql> update mysql.user set password=password('abc123') where user='root' and host="localhost"; |

4、再次用新设置的密码登陆即可

|  |
| --- |
| flush privileges;  mysql -uroot -pabc123 |

六、mysql的查询流程大致是：



mysql客户端通过协议与mysql服务器建连接，发送查询语句，先检查查询缓存，如果命中，直接返回结果，否则进行语句解析,也就是说，在解析查询之前，服务器会先访问查询缓存(query cache)——它存储SELECT语句以及相应的查询结果集。如果某个查询结果已经位于缓存中，服务器就不会再对查询进行解析、优化、以及执行。它仅仅将缓存中的结果返回给用户即可，这将大大提高系统的性能。

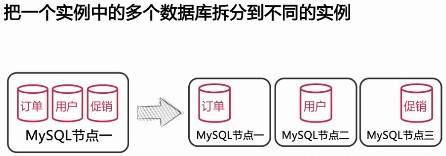
语法解析器和预处理：首先mysql通过关键字将SQL语句进行解析，并生成一颗对应的“解析树”。mysql解析器将使用mysql语法规则验证和解析查询；预处理器则根据一些mysql规则进一步检查解析数是否合法。

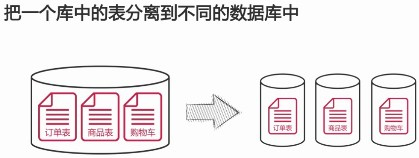
查询优化器当解析树被认为是合法的了，并且由优化器将其转化成执行计划。一条查询可以有很多种执行方式，最后都返回相同的结果。优化器的作用就是找到这其中最好的执行计划。。

然后，mysql默认使用的BTREE索引，并且一个大致方向是:无论怎么折腾sql，至少在目前来说，mysql最多只用到表中的一个索引。

七、数据库的分库分表

7.11.数据库分库分表的几种方式





2.数据库分片前的准备



如何选择分区键

-分区键要能尽量避免跨分片査询的发生

-分区键要能尽使各个分片中的数据平均

如何存储无需分片的表

-每个分片中存储一份相同的数据

-使用额外的节点统一存储

如何在节点上部署分片

-每个分片使用单一数据库，并且数据库名也相同

-将多个分片表存储在一库中，并在表名上 加入分片号后缀

-在一个节点中部署多个数据库，每个数据库包含 4分片

如何分配分片中的数据

-按分区键的Hash值取模来分配分片数据

–按分区键的范围来分配分片数据

-利用分区键和分片的映射表来分配分片数据

如何生成全局唯一ID

-使用 auto\_increment\_incrementS]auto\_increment\_offset 参数

-使用全局节点来生成ID

-在Redis等缓存服务器中创建全局ID

八、数据库中事务的四大特性（ACID）

如果一个数据库声称支持事务的操作，那么该数据库必须要具备以下四个特性：

⑴ 原子性（Atomicity）

　　原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，这和前面两篇博客介绍事务的功能是一样的概念，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

⑵ 一致性（Consistency）

　　一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

　　拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

⑶ 隔离性（Isolation）

　　隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

　　即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

　　关于事务的隔离性数据库提供了多种隔离级别，稍后会介绍到。

⑷ 持久性（Durability）

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

九、事务的四种隔离级别

设置数据库的隔离级别一定要是在开启事务之前！

MySQL数据库为我们提供的四种隔离级别：

① Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

② Repeatable read (可重复读)：可避免脏读、不可重复读的发生。

③ Read committed (读已提交)：可避免脏读的发生。

④ Read uncommitted (读未提交)：最低级别，任何情况都无法保证。

在MySQL数据库中，支持上面四种隔离级别，默认的为Repeatable read (可重复读)；而在Oracle数据库中，只支持Serializable (串行化)级别和Read committed (读已提交)这两种级别，其中默认的为Read committed级别。

在MySQL数据库中查看当前事务的隔离级别：

|  |
| --- |
| select @@tx\_isolation; |

在MySQL数据库中设置事务的隔离 级别：

|  |
| --- |
| set [glogal | session] transaction isolation level 隔离级别名称;  set tx\_isolation=’隔离级别名称;’ |