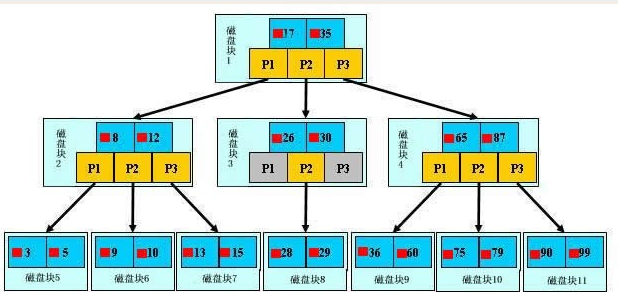
1. Mysql索引原理？

索引的目的是提高查询效率，本质上是存储引擎用于快速定位记录的一种数据结构。

mysql采用的是B+数数据结构作为索引的存储。

介绍之前需要先知道磁盘IO和预读：磁盘IO是很昂贵的，局部预读性说明当计算机访问一个地址的数据的时候，与其相邻的数据也会被读到内存中。每次IO读取的单元是page一般是4k/8k，这就是一次IO，索引原理中一定要尽可能尽可能低的IO读取次数。



上图是B+数，浅蓝色代表数据块，可以认为是page页（不规范），深蓝色存储的是包含的数据项，黄色是指针，上图中磁盘块1包含数据项17和35，包含指针p1、p2、p3，其中p1表示小于17数据项的磁盘块，p2表示在17和35之间数据的数据块，p3表示大于35的磁盘块。真是数据全部储存于叶子结点3 5 9.....。非叶子节点不存储真是数据，只存储指引搜索方向的数据项，如17和35并不是真是的数据。

B+树查找流程：

如果要查找29，那么首先把磁盘块1加载到内存，此时发生了一次IO，在内存中用二分查找确定29在17-35之间，锁定p2指向的磁盘块3，将磁盘块3加载到内存中，此时发生第二次IO，同样的29在26和30之间，锁定磁盘块8，再发生第三次IO就把数据所有的数据找到了。

B+树性质：

①索引字段尽量要小，因为磁盘块大小一次IO最大是一个page，所以如果字段过大，树的高度越大，IO次数将越多

②索引的最左匹配，比如多个索引（name,age,sex），首先会根据name来决定下一步搜索方向，如果name缺失，将不知道走哪里，将会遍历所有记录了，所以如果有符合索引，查询的时候一个顶要按照顺序加where条件。

最左匹配有个特别之处就是当where中只有=和in的时候可以乱序，mysql自动识别来优化索引。

Mysql索引分类：

①普通索引index：加速查找

②唯一索引：

主键索引：primary key：加速查找+约束（唯一且不为空）

唯一索引：unique：加速查找+约束（唯一）

③联合索引：

-primary key(id, name)：联合主键索引

-unique(id, name)：联合唯一索引

-index(id, name)：联合普通索引

④全文索引full text：用于搜索很长一篇文章时效果好

⑤空间索引spatial：几乎不用

注意区分联合索引和索引合并的区别

索引的类型：

①hash索引

查询单条块，范围查询慢

②btree索引

创建索引原则：

①最左匹配原则，需要注意一点就是当遇到范围查询字段的时候索引匹配结束，所以where后面把范围条件放到最后去

②=和in可以乱序

③尽量选择区分度高的字段做索引count(distinct col1)/count(\*)

④所以列不能参与计算

索引无法命中的情况：

①like ‘%xx’ 左模糊

②索引使用函数

③or：如果or两边都是索引的话没事

④类型不一致，比如string字段查询时不加引号也会不走索引

⑤普通索引的!=和范围查询不会命中索引，特例：字段类型是int的时候会走索引，

⑥排序字段为索引时select 的字段也需要是索引字段

⑦组合索引

⑧组合索引最左前缀原则(name, age)的连联合索引，where name=..and agel=...会使用索引，where name = ...会使用，但是where age = ...不使用索引

最后再说一下覆盖索引：

在where条件中用到了索引并且命中了，如果此时select中的字段含有这个索引字段，那么这就叫做覆盖索引，会提升查询速度的！！

慢查询优化的基本步骤：

①先看看是不是真的慢，注意设置SQL\_NO\_CACHE

②where条件单标查，锁定最小返回记录表。意思就是把查询语句的where都应用到表中返回的记录数最小的表开始查起，单表每个字段分别查询，看哪个字段的区分度最高

③explain查看执行计划，看是都与②中预期的一样（从锁定记录最少的表开始查询）

④order by limit 形式的sql语句让排序的表优先查

⑤加索引参照上述的索引规则

1. Mysql锁机制？

锁分类：

①按照对数据操作的类型分

共享锁（读锁）：多个读操作互不影响

排它锁（写锁）：当前写操作没有完成之前会阻断其他的写锁和读锁（注意：这里到底是锁哪里呢，是整个表还是某个行要搞清，因为假如只是行锁的话是不影响其他数据的读和写的！）

②按照对数据操作的粒度分

表级锁

行级锁

页级锁

表级锁：偏向MyISAM存储引擎，开销小、加锁快、无死锁；锁定粒度大、发生冲突概率大，并发度低。

行级锁：偏向InnoDB存储引擎，开销大、加锁慢、会出现死锁；锁定粒度小，发生冲突概率低，并发度高。

页面所：开销和加锁时间介于表锁和行锁之间、会出现死锁；锁定粒度也是介于之间。并发度一般。

MyISAM与InnoDB比较：

MyISAM存储引擎由于自身不支持事务，所以用的少，对于锁也相对比较简单：

表级共享锁：不影响其他用户对数据的读取，但是会阻塞其他用户对数据的写！

表级排它锁：阻塞所有对该表的操作！

所以适合读。

InnoDB存储引擎比较复杂，下面进行详细介绍。

InnoDB行锁的实现方式？

InnoDB行锁是通过给索引上的索引项加锁来实现的，只有通过索引条件检索数据，InnoDB才是用行级锁，否则InnoDB将使用表级锁，在实际应用中，要特别主要InnoDB行锁这一特性，不然的话，可能导致大量的锁冲突，从而影响并发性能。

从这里可以看出意向锁存在的意义！

①在不通过索引条件查询的时候，InnoDB确实使用的是表锁，而不是行锁。

②由于InnoDB的行锁是针对索引加的锁，不是针对记录加的锁，所以虽然访问不同行的记录，但是如果是使用相同的索引建，会出现锁冲突。

③当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，另外，不论是使用主键索引唯一索引、普通索引，InnoDB都会使用行锁来对数据加锁。

④即便在条件中使用了索引字段，但是否使用索引来检索数据是有Mysql通过判断不同执行计划的代价来决定的，如果mysql任务全表扫描相率更高，比如对很小的表，它就不会使用索引，这种情况下InnoDB将使用表锁，而不是行锁。因此在分析锁冲突的时候，别忘了检查sql的执行计划，以确定是否真正使用了索引。

所以：这就是为什么查询语句最好要有where 索引条件！！！这样才会提高效率。

同时表的建立可以没有主键但是索引一定要建立！！！！

什么事间隙锁？

当我们用范围条件而不是相等条件检索数据，并请求共享或者排它锁是，InnoDB会给符合条件的已有数据记录的索引加锁；对于键值在条件范围但并不存在的记录，叫做“间隙GAP”，InnoDB也会对这个“间隙”加锁，这种锁机制就是所谓的间隙锁（Next-key锁）。

间隙锁虽然可以防止幻读，但是也会造成无辜锁定导致其他操作等待。

死锁问题？

当两个事务都需要对方释放资源时发生死锁，InnoDB自带检查机制，会将影响记录少的事务回滚，但是最好设置InnoDB\_lock\_wait\_timeout，这个值不仅适用于死锁，当大量查询并发等待挂起时可以减少压力。

如何避免死锁？

①在应用中，如果不同的程序会并发存储多个表，应尽量约定一相同的顺序来访问表，这样可以大大降低产生死锁的机会。

②在程序以批量方式处理数据的时候，如果实现对数据排序，保证每个线程按固定顺序来处理记录，也可以大大降低死锁的可能。

③在事务中，如果要更新记录，应该直接申请足够的级别锁，即排它锁而不是先申请共享锁，更新时再申请排它锁，因为当用户申请排他锁是，其他事务可能又已经获得相同记录的共享锁。从而造成锁冲突，甚至死锁。

④在repeatable-read隔离级别下，如果两个线程同时对相同条件记录用select for update加排它锁，在没有符合条件记录情况下，两个线程都会加锁成功，程序发现记录不存在就试图插入一条新的记录，如果两个线程都这么做就会出现死锁。这种情况下，将隔离级别改为read-commitred就可以避免

⑤当隔离级别为read-committed时，如果两个线程都先执行了select ... fro update,判断是否存在符合条件的记录，如果没有就插入记录，此时，只有一个线程会插入成功，另一个线程会出现锁等待，当一个线程提交之后，第二个线程会因主键重出错，但索然这个线程出错了却获得一个排它锁，这是如果有第三个线程又来申请排它锁，也会出现死锁。对于这种情况，可以直接做插入操作，然后再捕获主键重异常或者在遇到主键重错误时总是执行rollback释放获得的排它锁。

InnoDB也可以应用表级锁：

①当涉及表大部分数据都需要改变时

②当事务涉及多个表、比较复杂、可能引起死锁时

1. Mysql事务隔离你了解多少？