**索引的添加:**

1. **Create** [UNIQUE|FULLTEXT|SPATIAL] INDEX 索引的名

ON 表名（字段名[长度]）

https://www.2cto.com/database/201801/713186.html

1. **Alter table 表名 add 索引类型 [索引的名] （索引名（长度））**

**索引的删除:**

1. **Drop index 索引的名 ON 表名**
2. **Alter table表名 drop index 索引的名**

**查看索引信息**

1. **Show index from 表名；**

**查看表状态命令:**

1. **show table status like ‘user’\G;**

**查看创建表的命令的命令:**

1. **show create table user \G;**

**查看表索引  
 5. Show index from 表名\G**

**新手学习索引网站**

**https://blog.csdn.net/u012954706/article/details/81241049**

1. **索引优化**

**索引是在存储引擎层实现的**

**Btree索引 和 Hash 索引**

1. **B-tree** 
   * 1. **采用B+tree结构**

**B+tree 结构的叶子节点存放了下一个叶子节点的指针。方便遍历**

**B+tree结构在不同引擎上的表现方式**

1. **Innodb**

**在innodb中btree索引分两种。**

**第一种: 聚集索引**

**聚集索引 就是 主键索引**

**表数据文件本身就是按照btree组织的一个索引结构。存行数据**

**第二种 二级索引**

**存 索引列的值 和 主键信息**

1. **Myisam**

**叶子节点存的是行指针。**

**适合什么场景?**

1. **全值匹配的查询**

**Order\_name = “order”**

1. **匹配最左前缀的查询**

**复合索引中最左列必须存在**

1. **匹配列前缀查询**

**Order\_name = “abc%”**

1. **范围查询**

**Order\_sn > 100**

1. **只访问索引的查询 （覆盖索引）**
2. **排序查询**

**什么时候索引失效?**

1. **查询的的数据过大。会失效**
2. **如果不是最左列查找则无法使用到**
3. **复合索引是三列的话。如果第二列没有用到 则第一列能用到索引，第三列用不到**
4. **Not in 和 <> 会失效**
5. **某列使用范围查询 则 右边所有列都会失效**

**2. 哈希索引**

**使用场景:**

**等值查找（精确查找）；**

**限制:**

1. **不能范围查找，排序。**
2. **不支持部分索引查找**

**索引带来的好处:**

1. **减少了存储引擎需要扫描的数据量**

**如innodb是以数据页为单位 大小为16K 那么索引文件相对于数据文件是非常小的。在以页为单位的查询中所查询的效率是非常高的。次数也是少了。**

1. **索引可以帮助我们排序避免使用临时表**
2. **索引可以把随机I/0变为顺序I/O**

**数据是以随机I/O存储的。而索引是以顺序I/O存储的。通过索引查数据的话可以大大提高I/O的性能**

**索引多 带来的坏处:**

1. **增加写操作的成本，添加数据的时候要添加相对的索引**
2. **太多索引让查询优化器不知道用那个了，给人搞蒙了。选择也是需要时间的。**

**二 安装数据库文件**

**网址: http://downloads.mysql.com/docs/sakila-db.tar.gz**

**三 索引优化策略**

1. **索引列上不能使用表达式和函数**

**针对日期的函数:**

**to\_days() 返回从0年的天数**

**date\_add(date,INTERVAL expr type);**

**参数date : 日期表达式**

**参数 expr 时间间隔**

1. **前缀索引和选择性**

**Innodb 索引不能大于756字节（255） myisam 不能大于1000**

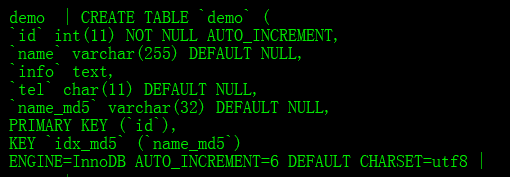
**所以我们想要字符串长的列上使用索引就要用到前缀索引了**

**Create index index\_name on table(col\_name(n));**

**还有一种办法:**

**就是用b+tree模拟一下hash结构**

**如下面一张表**



**Name列为255 等同于 755个字节了**

**所以太长了不适合也不能加索引了。那我们可以这样做**

**在添加一个字段。**

**Alter table demo add name\_md5 varchar(32);**

**给字段一个索引。**

**Create index idx\_md5 on demo (name\_md5);**

**给这个字段数据。（数据是MD5后的name）**

**Update demo set name\_md5 = md5(name);**

**然后想用索引查找指定的值就能够这样用:**

**Select \* from demo where name\_md5=md5（你要查的名字） and name = “你要查的名字”；**

**这样根据索引将name\_md5 = MD5(你要查的名字) 一行数据查出来 然后在对比 name = “你要查的名字”就实现了不用加前缀索引。**

**索引的选择性:**

**就是不重复的索引和表的记录数的比值。如主键和唯一索引就很高，所以加前缀索引要考虑大小也要考虑选择性**

1. **联合索引**

**什么时候失效?**

1. **当不是最左定义的时候**
2. **当前面有排序的时候**

**怎么选择索引列的顺序**

* 1. **经常被用到的列**
  2. **选择性高的列**
  3. **宽度小的列 。小则快**

**解释一下宽度小的列的好处:**

**因为innodb是以数据页为单位读取。所以呢，宽度越小则存储的数据页越少，所需要的磁盘I/O操作越少（索引文件也是在磁盘上），所以就快，性能就好。**

1. **覆盖索引**

**优点:**

1. **可以优化缓存，减少I/O**
2. **可以减少随机I/O，变随机为顺序**
3. **可以避免二次查询**

**无法使用覆盖索引的情况**

**查询中使用了太多的列**

**使用了双%号的like查询**

**5. 索引排序:**

**1. 索引的列顺序和Order By 字句的顺序完全一致。**

**2. 索引中所有列的方向(desc , asc ) 和orderby 字句一致**

**3. orderby 字段在关联表中 必须在第一张表中。**

**6. 索引优化innodb行锁**

1. **当开启事务后会开启锁。**

**当你要修改name = “张三”的字段的时候。**

**如果name不加索引那么整个表中所有行都会锁住相当于锁了整个表了;**

**如果name加了索引那么只会锁住name=“张三”的行，因为过滤掉了哪些不是name=张三的行；**

**7. 删除重复和冗余索引  
 见到这样的索引就要去除掉；**

**1. 什么是重复的？**

**一个字段 上 加了多个索引**

1. **. 什么是冗余索引**

**就是在两个索引所覆盖的列有重叠**

**如（aid,bid）和（bid，aid）列重叠 但顺序不一样，是冗余**

**还有 主键id 和 （id,aid）也是冗余的；要知道innodb索引叶子节点上是有id的 所以都不用和id关联，只要关联就不行。**

**3. 怎么查出来有重复和冗余索引呢?**

**用工具呗: pt-duplicate-key-checker**

**8. 索引的维护和优化**

**1. 更新索引统计信息**

**Mysql 内置的查询优化器是根据索引统计信息进行选择索引进行查询的；**

**所以，索引统计信息出错会导致查询优化器判断错误**

**所以我们要定时更新它:**

**工具命令 : analyze table table\_name**

1. **Myisam**
   1. **每次更新的时候都是进行锁表对全索引进行扫描（完全统计法）**
2. **Innodb** 
   1. **随机抽取扫描不会锁表（抽样调查法）**

**索引的扫描更新会造成索引碎片。同样表也会有一些碎片。**

**所以我们要根据这个命令 定时清理:**

**Optimize table table\_name // 可能会锁表**

**四、sql语句优化**

**1. 获取存在性能问题的sql**

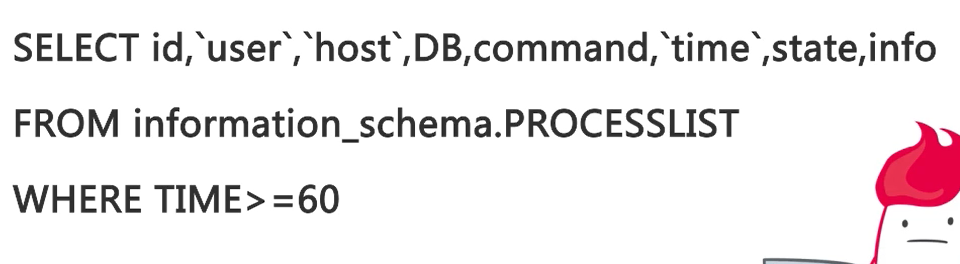
**1) 用户反馈**

**如: 用户反馈哪个页面相应慢，那个影响人家体验了；**

**2) 慢查询日志**

****

**3) 实时获取存在性能问题的sql（没成功）**



**2. 面对大表数据的修改**

**分批进行修改。在每批修改的时候最好间隔一段时间给主从复制同步的时间。**

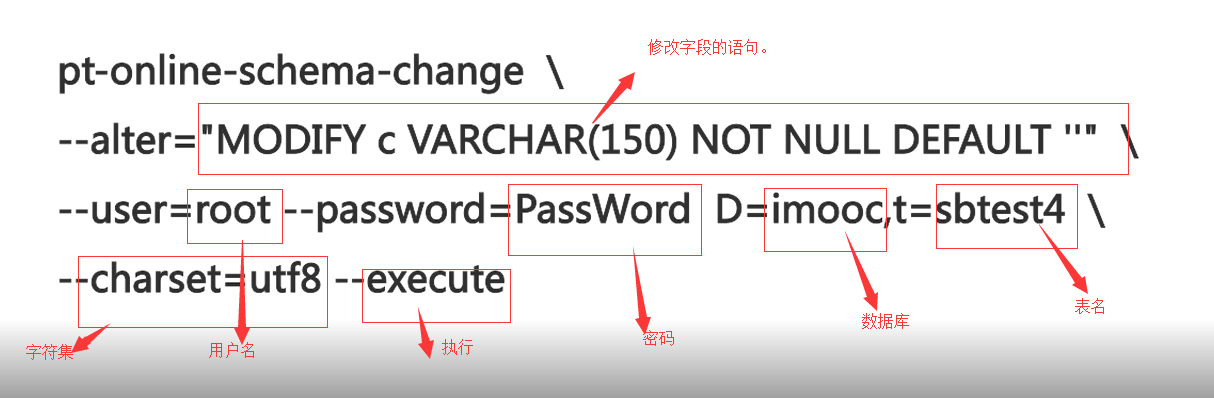
**3. 修改大表的表结构**

**Innodb修改表结构的时候会锁表。也无法解决主从延迟**

**解决:**

1. **先在从服务器你修改 ，然后主从切换（不推荐）**
2. **建立一个新表。将老表数据给新表。然后在老表身上加触发器直到新老同步。然后将老表加一个锁。给新表重命名成老表的名字后删掉老表。**

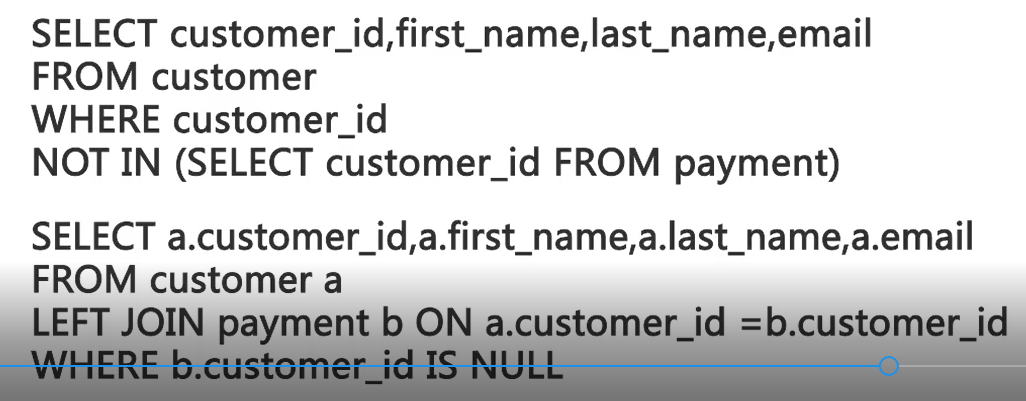
**对于第二点有个工具(pt-online-schema-change)**



**4. 优化not in 和 <> 查询**

**因为优化器对这个优化不了。**

**所以我们要手动的给他改成关联查询。**



**5. 使用汇总表优化查询**

**如: select count(\*) form product\_comment where product\_id =999;**

**面对上面的语句我们这样做**

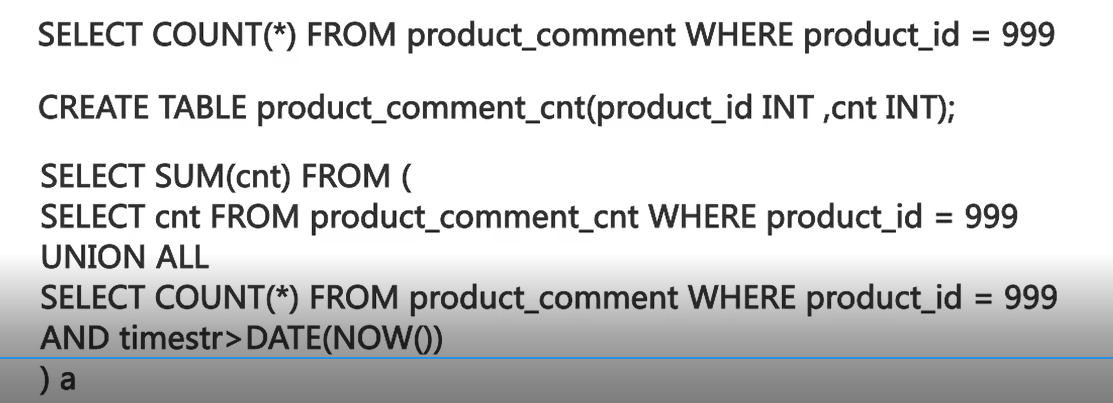
**1. 建立以个汇总表；**

**如。Create table product\_count (product\_id int,cnt int);**

**2. 在每天的一个时间点里面进行对汇总表的更新。**

**3. 当我们要查汇总结果的时候。上个时间点的数量。然后在拿到今天所有的数量。一相加就OK了。**

**实例sql:**

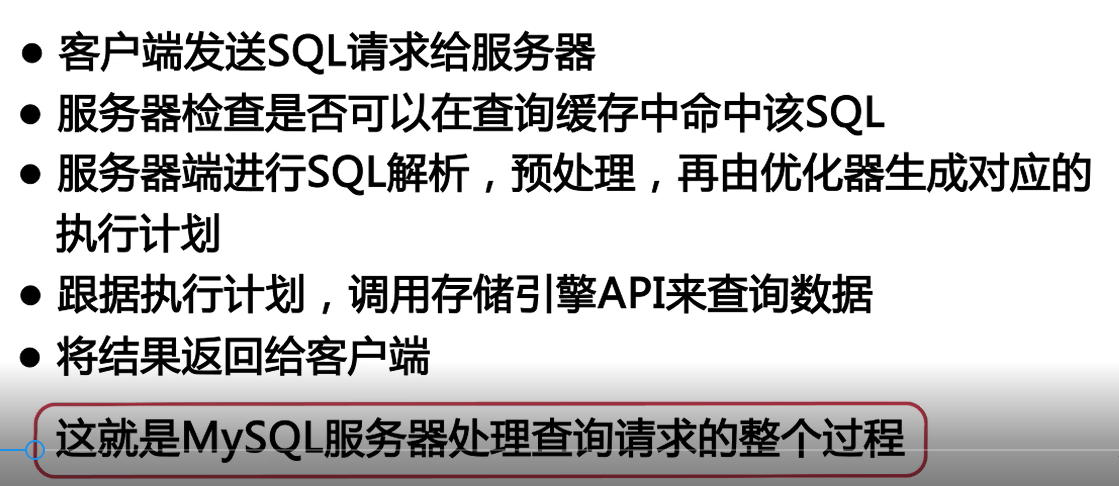


**如上: 在product\_comment上的timestr列上加个索引。**

**然后通过索引覆盖来做。快的很**

**五、mysql 对sql 处理流程 再讲解**

**1. 查询为什么慢？**



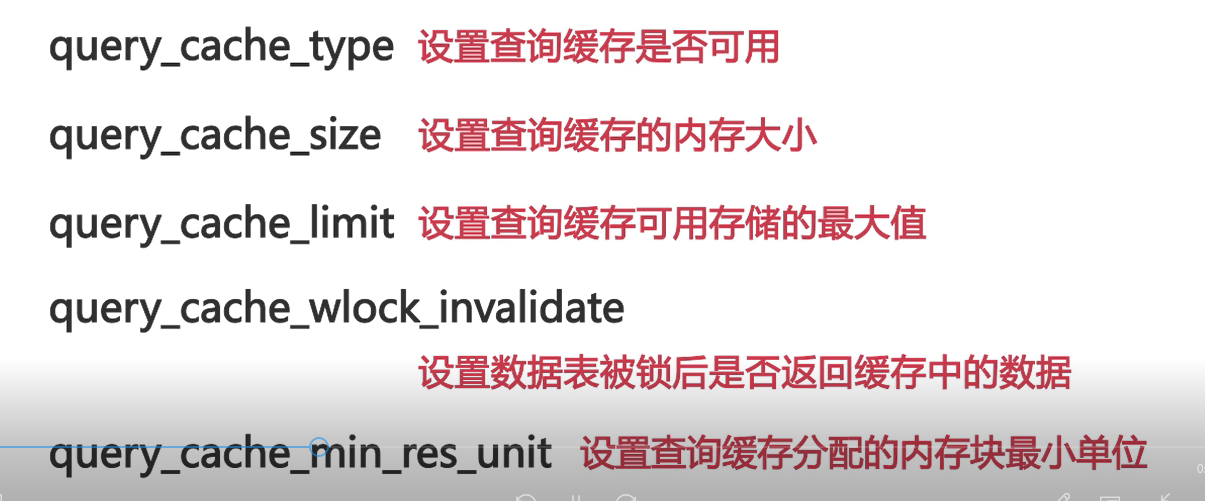
**首先从查询缓存说。**

**在缓存打开情况下。Mysql先会检查查询是否命中查询缓存中的数据。是通过hash查找的。我们也了解过hash的特性，它需要全值匹配，哪怕一个空格，一个符号 出现误差都会匹配失效。**

**听着挺好!那它慢在了哪里？**

**慢就慢在太难命中。**

1. **全值匹配到才能获取到值**
2. **缓存中所对应的表一旦修改则数据就失效**
3. **而且它还给缓存加锁，读写频繁的时候那效率。。低**



**下一个。Sql解析验证**

**这一步呢就是针对sql中的关键字生成一个解析树。通过mysql语法规则检测1. 是否使用了正确关键字。2. 关键字的顺序。等验证。**

**下一个。预处理**

**将根据mysql 规则进一步检查解析树是否合法。**

**比如 在挑挑毛病。看你要用的table\_name 是否存在**

**数据列存在不存在。**

**下一个。生成执行计划（explain）**

**生成一个最优的查询计划。**

**这一步慢在哪里？**

**1. 比如索引多了。Mysql优化器就会选择时间非常长**

**2. 统计信息不准确。导致选择最优的偏差**

**3. 人家优化器和你认为的最优不一致。如人家考虑成本，你考虑查的时间。**

**2. Mysql优化器做了什么工作？**

1. **重新定义表的关联顺序。**
2. **将外链接 改 为内连接 （当外链接等价于内连接的结果）**
3. **使用等价变换规则**

**（1=1 and a>5） 改为 a>5**

**覆盖索引。因为在索引中找到了数据。就不用查表了**

1. **优化 count() min() max();**

**在索引上。Min在最左边 max在最右边**

1. **将表达式转化为常数**
2. **子查询优化（子查询转化为关联查询）**

**7．提前终止查询（如:查 主键id = -1 就会直接停止）**

**8. 对in的优化（根据二叉树）**

**3. 记录查询在各个阶段消耗的时间:**





**官方推荐工具(performance\_schema)**