数据库高性能优化

一、性能分析

1、DQL的查询过程:

1. 客户端发送一条查询给服务器;
2. 服务器通过权限检查之后,先会检查查询缓存，如果命中了缓存，则立即返回存储在缓存中的结果。否则进入下一阶段;
3. 服务器端进行SQL解析、预处理，再由优化器根据该SQL所涉及到的数据表的统计信息进行计算，生成对应的执行计划;
4. MySQL根据查询优化器生成的执行计划，调用存储引擎的API来执行查询;
5. 将结果返回给客户端。

查询优化器

1、写的任何sql,到底是怎么样真正执行的,按照什么条件查询,最后执行的顺序,可能都会有多个执行方案。

2、查询优化器根基对数据表的统计信息(比如索引,有多少条数据),在真正执行一条sql之前,会根据自己内部的数据,进行综合的查询。

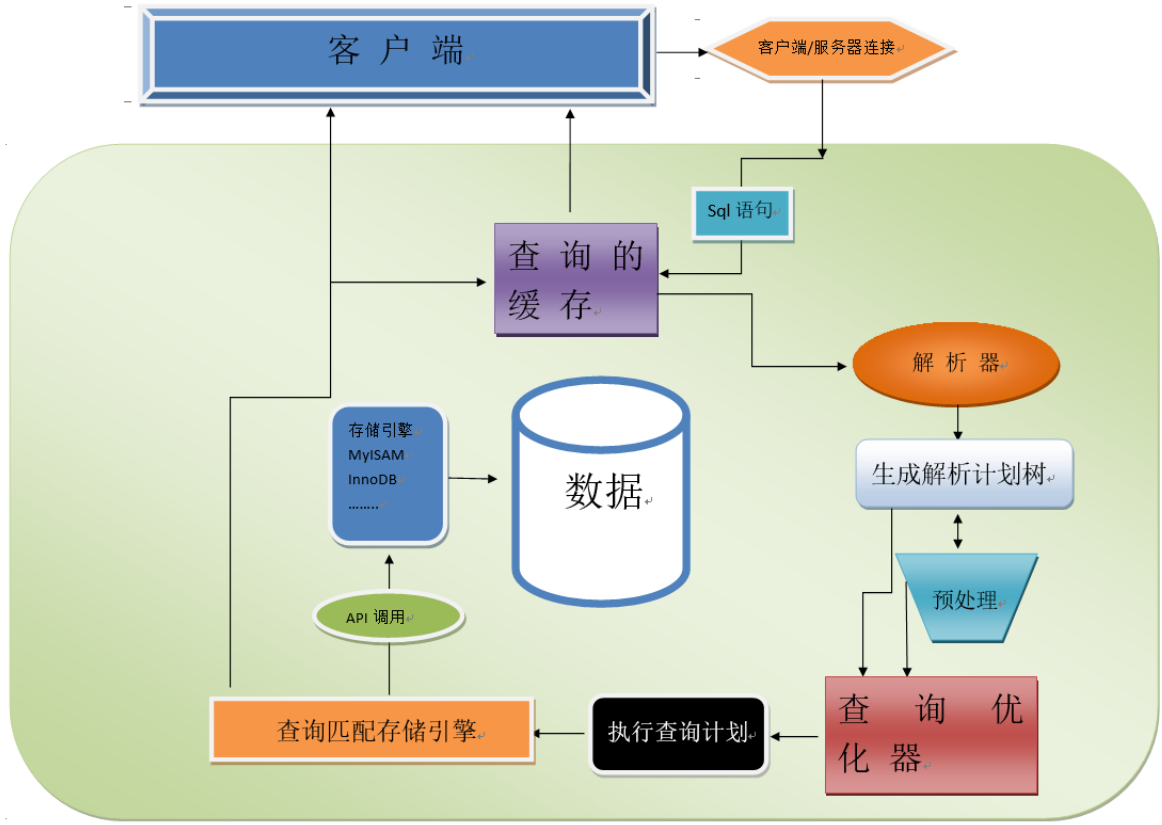
3、根据mysql自身的统计信息, 从多种执行方案当中, 选择一个它认为是最优的执行方案,来去执行。

做优化做什么

- 做优化, 就是想让查询优化器按照我们的想法,帮我们选择最优的执行方案
- 让优化器选择符合程序员计划的执行语句,来减少查询过程中产生的IO

mysql常见的问题

- CPU饱和
- 磁盘I/O读取数据大
- 服务器硬件配置低



2、Explain

查询执行计划：

使用explain关键字,可以模拟优化器执行的SQL语句，从而知道MYSQL是如何处理sql语句的，通过Explain可以分析查询语句或表结构的性能瓶颈。

作用：

- 查看表的读取顺序
- 数据读取操作的操作类型
- 查看哪些索引可以使用
- 查看哪些索引被实际使用
- 查看表之间的引用
- 查看每张表有多少行被优化器执行

使用方法

```
explain sql
##############################################################
explain select * from stuinfo join score using(sid)\G
```

id

- select查询的序号
- 如果上下的id相同，执行顺序由上向下
- 如果id大小不同，一般会是一个子查询，id值越大越会被优先执行

```
explain select * from stuinfo where age in (select
max(age) from stuinfo);
```

- id既有相同的又有不同的

```
explain select * from stuinfo join score using(sid) where age in (select max(age) from stuinfo);

explain select * from stuinfo where sid in (select sid from score where sid in (select max(ch) from score));
```

- 总结：相同的顺序走，不同的大的先走。

select_type

- simple : 简单select查询,查询中不包含子查询或者UNION
- primary : 查询中若包含任何复杂的子查询,最外层查询则被标记为primary
- subquery : 在select或where中包含了子查询
- union : 若第二个select出现的union之后,则被标记为union

```
explain select * from stuinfo union select * from stuinfo;
```

- derived : 若union包含在from子句的子查询中,外层select将被标记为deriver

```
explain select * from ((select * from stuinfo where sid =1 ) union (select * from stuinfo where sid=2)) as a;
```

- union result : 从union表获取结果select,两个UNION合并的结果集在最后

type

- ALL : 将全表进行扫描,从硬盘当中读取数据，如果出现了All 切数据量非常大, 一定要去做优化

```
explain select * from stuinfo;
```

- INDEX :
  - index与All区别为index类型只遍历索引树,通常比All要快,因为索引文件通常比数据文件要小
  - all和index都是读全表,但index是从索引中读取,all是从硬盘当中读取

```
explain select sid from stuinfo;
```

- system： 表中有一行记录(系统表) 这是const类型的特例,平时不会出现
- const： 表示通过索引一次就找到了，const用于比较primary 或者 unique索引. 直接查询主键或者唯一索引，因为只匹配一行数据,所以很快

```
explain select * from stuinfo where sid=1;
```

- range : 根据表统计信息及索引选用情况,大致估算出找到所需的记录所需要读取的行数

```
explain select * from stuinfo where sid>2;
```

- eq_ref 某一种类型的索引,多次被使用过

总结 : 只要不出现ALL(全表硬盘扫描),那么当前的SQL语句的查询速度一定是比较快的

possible_keys

可能自己创建了4个索引,在执行的时候,可能根据内部的自动判断,只使用了3个

判断可能会使用到的索引

key

实际使用到的索引

- 实际使用的索引,如果为NULL,则没有使用索引
- 查询中若使用了覆盖索引 ,则该索引仅出现在key列表中
- possible_keys与key关系 理论应该用到哪些索引 实际用到了哪些索引
- 覆盖索引 查询的字段和建立的字段刚好吻合,这种我们称为覆盖索引

key_len

- 表示索引中使用的字节数,可通过该列计算查询中使用的索引长度 .
- 并不一定十分的准确

ref

索引是否被引入到, 到底引用到了哪几个索引

```
explain select * from stuinfo a,score b where a.sid=b.sid
and a.age=22;
```

row和filtered

row被扫描的越少越好

filtered是索引被应用行的比例，100%是全使用上，是最好的

```
#优化的比较好
explain select * from stuinfo a,score b where a.sid=b.sid
and a.age=22;
#不好
explain select * from stuinfo a,score b where a.sid=b.sid
and a.age=22;
```

Extra

产生的值：

- Using filesort： 说明mysql会对数据使用一个外部的索引排序

```
explain select * from stuinfo a,score b order by age;
```

- Using temporary : 使用了临时表保存中间结果,Mysql在对查询结果排序时, 使用了临时表,常见于排序orderby 和分组查询group by
- using index：查询一个有键的字段

```
explain select sname from stuinfo ;
```

- using where : 查询一个where判断的条件(没有满足的情况)

```
explain select * from stuinfo where sname;
```

二、索引

什么是索引： 帮助Mysql高效获取数据的数据结构，类似新华字典的索引目录,可以通过索引目录快速查到你想要的字，排好序的快速查找数据。

1.为什么要建立索引:

- 提高查询效率，没有排序之前一个一个往后找，通过索引进行排序之后,可以直接定义到想要的位置
- 排好序的快速查找数据结构-->就是索引

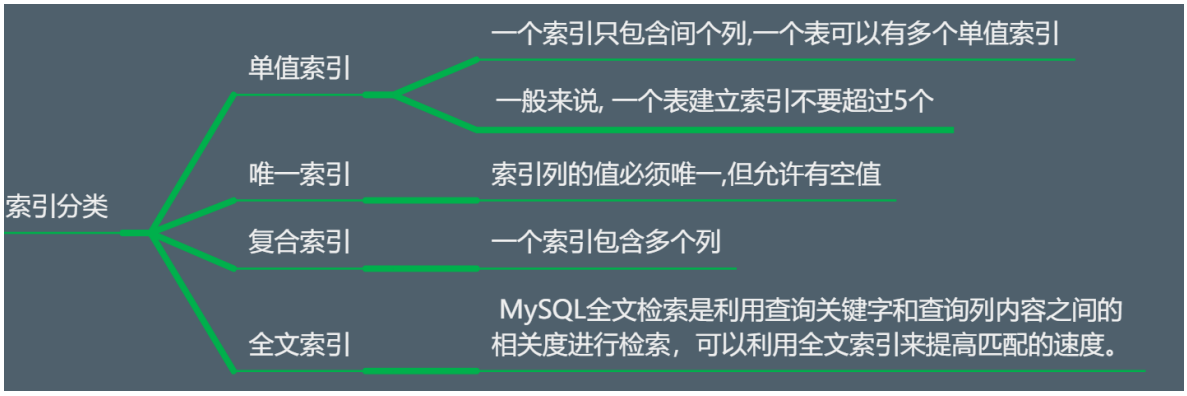
2.优劣势

优势:

- 通过索引对数据项进行排序,降低数据排序成本,降低了CPU的消耗
- 索引类似大学图书馆建立的书目索引,提高数据检索的效率,降低数据库的I/O成本

劣势:

- 一般来说, 索引本身也很大, 索引往往以文件的形式存储到磁盘上
- 虽然索引提高了查询速度,但是会降低更新表的速度
- 会调整因为更新所带来的键值变化后索引的信息



mysql中的索引:

- 主键索引:一张表中只有一个,只有一个字段
- 唯一键索引:一张表中可以有多个,只有一个字段
- 外键索引:一张表中可以有多个,只有一个字段
- 普通键索引(复合索引,联合索引):用的最多的,也是最灵活的

```
alter table `table_name` add index
`index_name`(`filed`,`f2`,`f3`);
```

- 全文键索引:默认不使用(理由:会产生大量的索引,给硬盘带来压力)

3.为什么建立索引查数据会更加快

在我们存数据时, 如果建立索引，数据库系统会维护一个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数据，可以在这些数据结构之上,实现高级查找算法,这种结构就是索引

一般来说, 索引本身也很大, 不可能全部存储在内存中, 因此索引往往以索引文件的形式存储在磁盘上

为了加快数据的查找,可以维护二叉查找树, 每个节点分别包含索引键值和一个指向对应数据记录的物理地址的指针,这样就可以运用二叉查找在一定的复杂度内获取相应的数据,从而快速的检索出符合条件 的记录

除了二叉树还有BTtree索引，我平时所说的索引,如果没有特别指定, 都是指B树结构组织的索引，其中聚焦索引,次要索引,复合索引,前缀索引,唯一默认都是B+树索引。

除B+树索引之外, 还有哈希索引(Hash index)等。

4.二叉查找树

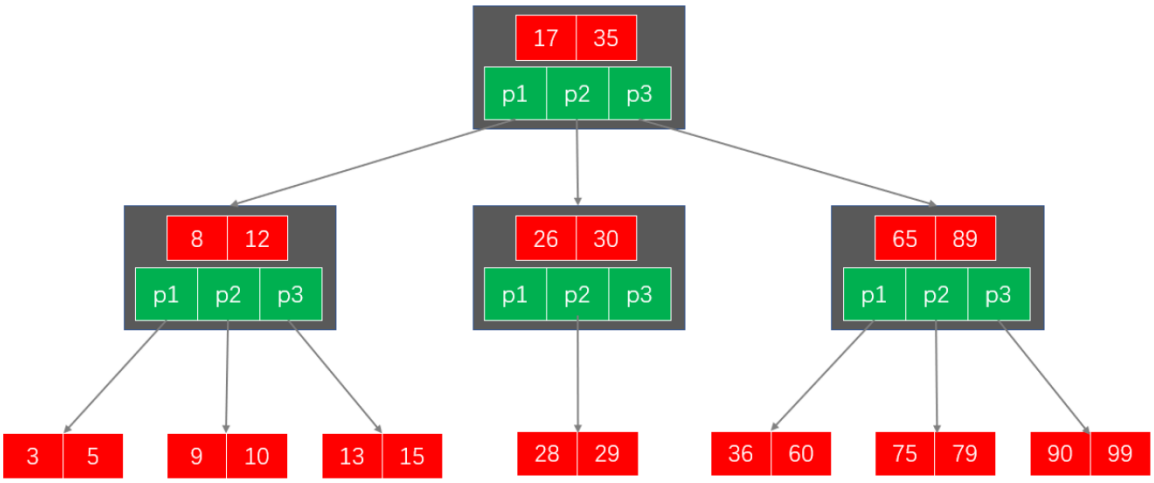
使用折半查找的方式进行快速搜索

5.B-Tree(平衡多路查找树)

m阶的B-Tree满足以下性质：

- （1）每个节点最多拥有m个子树
- （2）根节点最少有2个子树
- （3）分支节点最少拥有m/2棵子树

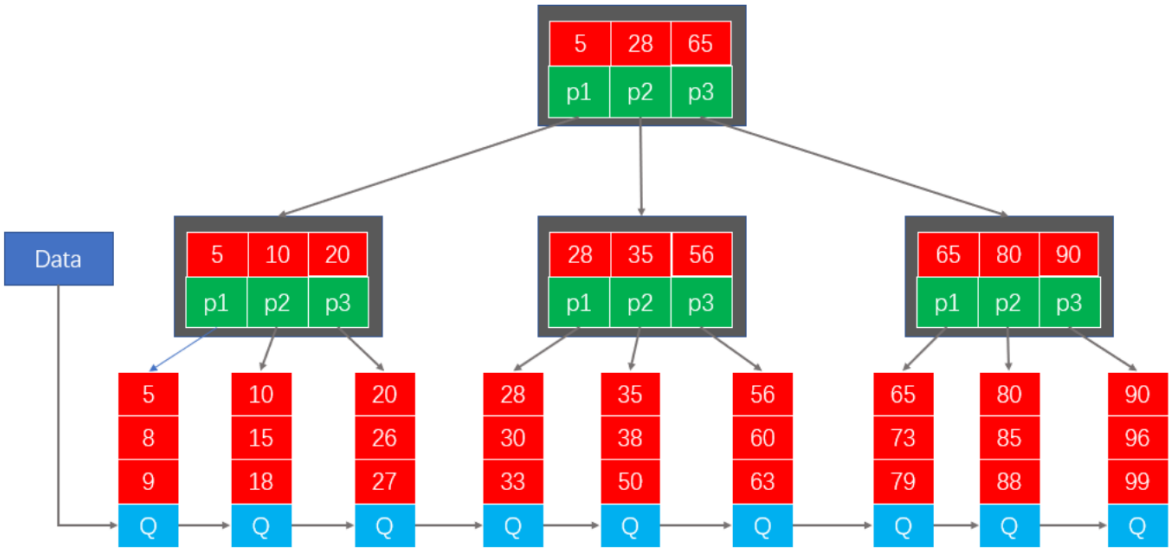
- （4）所有叶节点都在同一层，每个节点最多有m-1个key，并且以升序排列



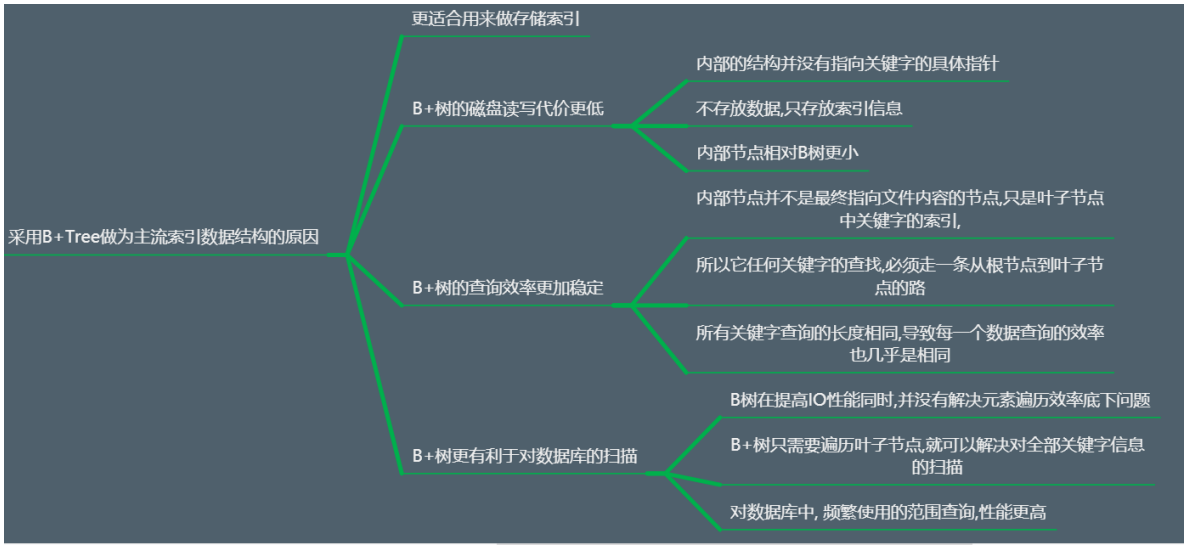
6.B+Tree

B+Tree相对于B-Tree有几点不同：

- 1.非叶子节点只存储键值信息。
- 2.所有叶子节点之间都有一个链指针。
- 3.数据记录都存放在叶子节点中。

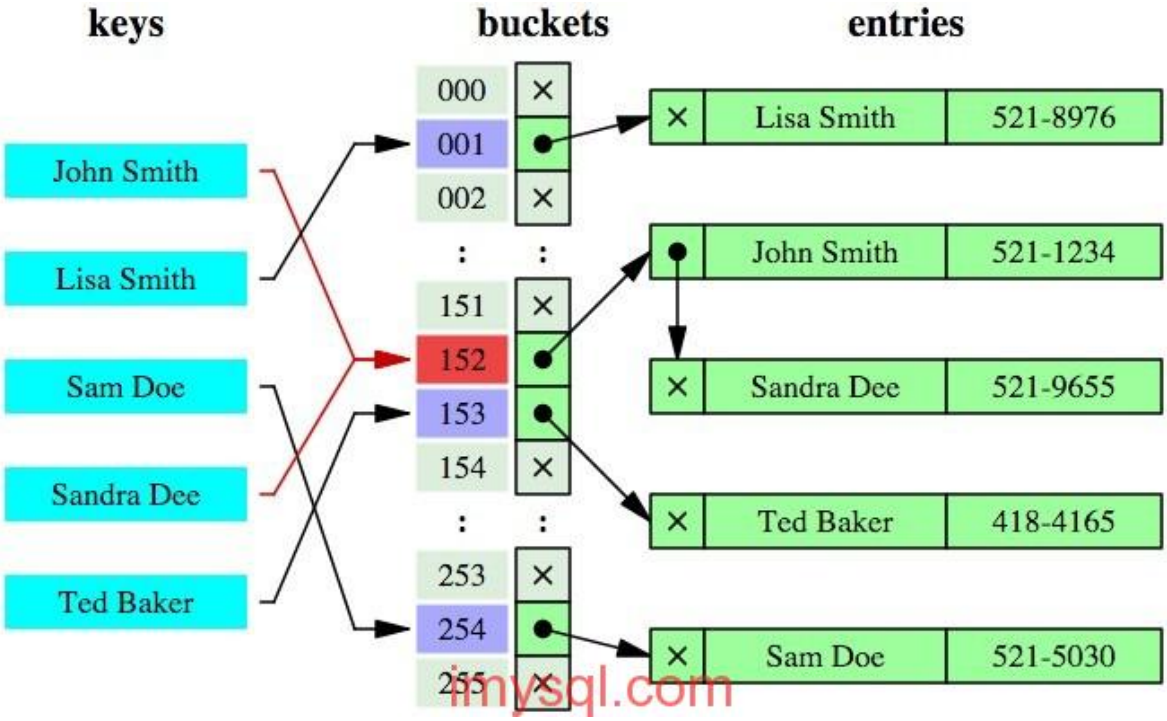


为什么时候B+tree作为默认索引



7.Hash索引

哈希索引比较特殊，时间复杂度为 O(1), 但只适合等值比较方式的查询，不适合范围或大小比较进行查询,在内存中使用的一种索引方式.



8.索引的创建原则

1. 适合用于频繁查找的列
2. 适合经常用于条件判断的列

3. 适合经常由于排序[分组]的列
4. 不适合数据不多的列
5. 不适合很少查询的列

三、数据备份与恢复

1. 备份

```
mysqldump -h localhost -u root -p123456 dbname > dbname.sql
```

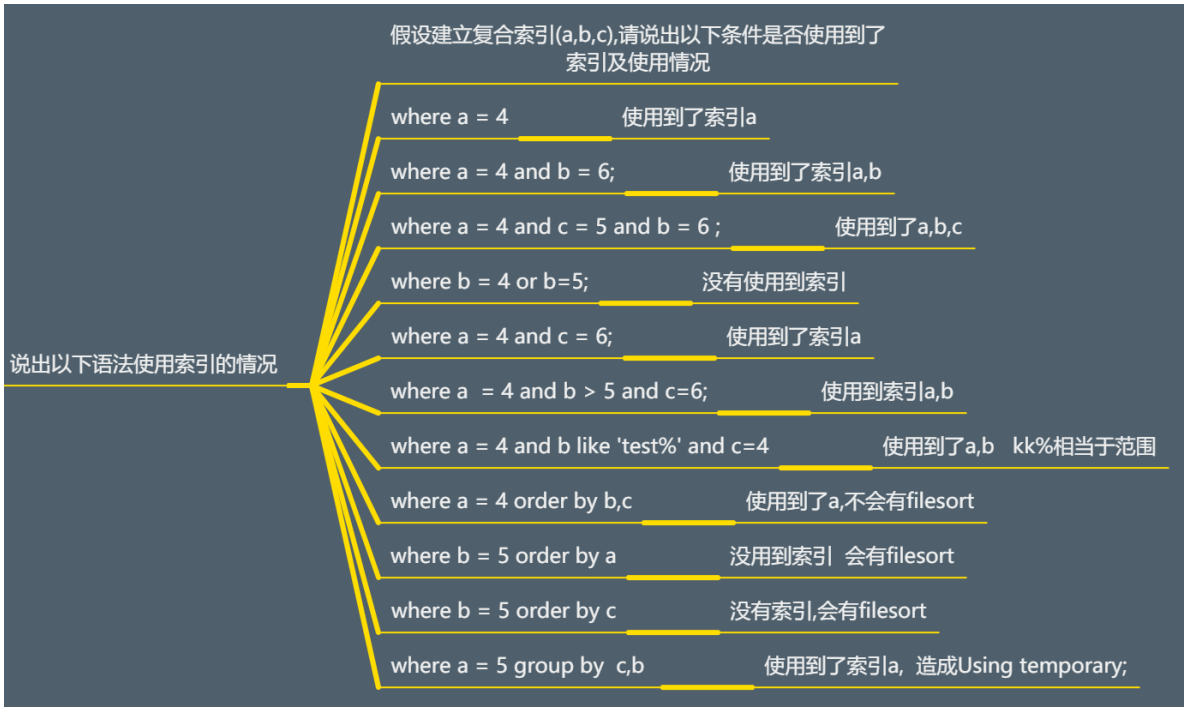
2. 恢复

```
mysql -h localhost -u root -p123456 dbname < ./dbname.sql
```

四、索引面试题

根据左前缀法则来匹配

- 联合索引可以被单个key所使用
- 三个key组成联合索引，如果两两使用的化，没有使用到最左侧索引，或者最左侧索引的排序位置不正确会失效
- 范围查找会导致索引失效
- 普通索引使用or操作会失效



五、其它优化

如果字段的数量太过于庞大:垂直分表(备注:符合三范式)

如果是数据量过于庞大:水平分表(备注:AUTO_INCREMENT=lastid+1)

分库:分机器(分布式架构,主从)

连接产生的问题,查询以后,立马断开

查询数据的时候,经量不要使用*(备注:查询列太多),经量分页(备份:limit)

硬件会影响:固态

六、业务查询

1.将sex 修改成enum('男','女')

ispaid修改为enum('已支付','未支付')

```
alter table `user_info` modify `sex` enum('男','女');

alter table `order_info` modify `ispaid` enum('已支付','未支付');
```

2.birth paidtime时间类型

```
alter table `user_info` modify `birth` datetime;

alter table `order_info` modify `paidtime` datetime;
```

3.删除sex为空的数据

```
delete from `user_info` where `sex` is null or `birth` is null;
```

4.统计不同月份下单的人数

```
select month(paidtime),count(distinct userid) from order_info where ispaid=1 group by month(paidtime);
```

5.统计三月份的下单人数

```
select count(distinct userid) from order_info where ispaid=1 and month(paidtime)=3;
```

6.统计三月份的重复购买率

```
#当月买过一次以上
select count(userid)/(select count(distinct userid) from
order_info where ispaid=1 and month(paidtime)=3)
from (select userid,count(userid) as uc from order_info
where month(paidtime)=3 and ispaid=1 group by userid
having uc>1) as fg;


select concat((count(userid)/54799)*100,'%') from
(select userid,count(userid) as uc from order_info where
month(paidtime)=3 and ispaid=1 group by userid having
uc>1) as fg;
```

7.3月重复购买率和用户数

```
select count(userid) as t,count(if(uc>1,1,null))  as f,
(count(if(uc>1,1,null))/count(userid))  from (select
userid,count(userid) as uc from order_info where
month(paidtime)=3 and ispaid=1 group by userid) as fg;
```

8.求每个月的重复购买率和用户数

```
select m 月份,count(userid) 月总, count(if(uc>1,1,null))
复人,( count(if(uc>1,1,null))/count(userid)) as f  from
(select userid,count(userid) as uc,month(paidtime) as m
from order_info where  ispaid=1 group by userid,m ) as
fg group by m;
```

9.男女的消费频次

```
select sex 性别,avg(ct) 平均 from (select
us.userid,sex,count(us.userid) as ct  from order_info as o
 join (select * from user_info where sex is not null) as us
using(userid) where ispaid=1 group by us.userid,sex) as
b group by sex;
```

```
alter table user_info engine=myisam;
alter table user_info add primary key(`userid`);
alter table user_info add index
`union1`(`sex`,`birth`);
alter table user_info add index
`union2`(`birth`,`sex`);
```

```
alter table order_info engine=myisam;
alter table order_info add primary key(`orderid`);
alter table order_info add index `user`(`userid`);
alter table order_info add index `paid`(`ispaid`);
alter table order_info add index `time`(`paidtime`);
```