SQL语句优化

怎么加快查询速度，优化查询效率，主要原则就是应尽量避免全表扫描，应该考虑在where及order by涉及的列上建立索引。

建立索引不是建的越多越好，原则是：

第一：一个表的索引不是越多越好，也没有一个具体的数字，根据以往的经验，一个表的索引最多不能超过6个，因为索引越多，对update和insert操作也会有性能的影响，涉及到索引的新建和重建操作。

第二：建立索引的方法论为：

1. 多数查询经常使用的列；
2. 很少进行修改操作的列；
3. 索引需要建立在数据差异化大的列上

利用以上的基础我们讨论一下如何优化sql.

1、sql语句模型结构优化指导

a. ORDER BY + LIMIT组合的索引优化

如果一个SQL语句形如：SELECT [column1],[column2],…. FROM [TABLE] ORDER BY [sort] LIMIT [offset],[LIMIT];

这个SQL语句优化比较简单，在[sort]这个栏位上建立索引即可。

b. WHERE + ORDER BY + LIMIT组合的索引优化

如果一个SQL语句形如：SELECT [column1],[column2],…. FROM [TABLE] WHERE [columnX] = [VALUE] ORDER BY [sort] LIMIT [offset],[LIMIT];

这个语句，如果你仍然采用第一个例子中建立索引的方法，虽然可以用到索引，但是效率不高。更高效的方法是建立一个联合索引(columnX,sort)

c. WHERE+ORDER BY多个栏位+LIMIT

如果一个SQL语句形如：SELECT * FROM [table] WHERE uid=1 ORDER x,y LIMIT 0,10;

对于这个语句，大家可能是加一个这样的索引:(x,y,uid)。但实际上更好的效果是(uid,x,y)。这是由MySQL处理排序的机制造成的。

2、复合索引(形如(x,y,uid)索引的索引)

先看这样一条语句这样的：select* from users where area =’beijing’ and age=22;

如果我们是在area和age上分别创建索引的话，由于mysql查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果area，age两列上创建复合索引的话将带来更高的效率。

在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

例如我们建立了一个这样的索引（area,age,salary），那么其实相当于创建了（area,age,salary）,(area,age),(area)三个索引，这样称为最佳左前缀特性。

3、like语句优化

SELECT id FROM A WHERE name like '%abc%'

由于abc前面用了“%”，因此该查询必然走全表查询，除非必要，否则不要在关键词前加%，优化成如下

SELECT id FROM A WHERE name like 'abc%'

4、where子句使用！ = 或 <> 操作符优化

在where子句中使用！ = 或 <>操作符，索引将被放弃使用，会进行全表查询。

如SQL:SELECT id FROM A WHERE ID != 5 优化成：SELECT id FROM A WHERE ID>5 OR ID<5

5、where子句中使用 IS NULL 或 IS NOT NULL 的优化

在where子句中使用 IS NULL 或 IS NOT NULL 判断，索引将被放弃使用，会进行全表查询。

如SQL:SELECT id FROM A WHERE num IS NULL 优化成num上设置默认值0，确保表中num没有null值，然后SQL为：SELECT id FROM A WHERE num=0

6、where子句使用or的优化

很多时候使用union all 或 nuin(必要的时候)的方式替换“or”会得到更好的效果。where子句中使用了or,索引将被放弃使用。

如SQL:SELECT id FROM A WHERE num =10 or num = 20 优化成：SELECT id FROM A WHERE num = 10 union all SELECT id FROM A WHERE num=20

7、where子句使用IN 或 NOT IN的优化

in和not in 也要慎用，否则也会导致全表扫描。

方案一：between替换in

如SQL:SELECT id FROM A WHERE num in(1,2,3) 优化成：SELECT id FROM A WHERE num between 1 and 3

方案二：exist替换in

如SQL:SELECT id FROM A WHERE num in(select num from b) 优化成：SELECT num FROM A WHERE num exists(select 1 from B where B.num = A.num)

方案三：left join替换in

如SQL:SELECT id FROM A WHERE num in(select num from B) 优化成：SELECT id FROM A LEFT JOIN B ON A.num = B.num

8、where子句中对字段进行表达式操作的优化

不要在where子句中的“=”左边进行函数、算数运算或其他表达式运算，否则系统将可能无法正确使用索引。

如SQL:SELECT id FROM A WHERE num/2 = 100 优化成：SELECT id FROM A WHERE num = 100*2

如SQL:SELECT id FROM A WHERE substring(name,1,3) = 'abc' 优化成：SELECT id FROM A WHERE LIKE 'abc%'

如SQL:SELECT id FROM A WHERE datediff(day,createdate,'2016-11-30')=0 优化成：SELECT id FROM A WHERE createdate>='2016-11-30' and createdate<'2016-12-1'

如SQL:SELECT id FROM A WHERE year(addate) <2016 优化成：SELECT id FROM A where addate<'2016-01-01'

9、任何地方都不要用 select * from table ，用具体的字段列表替换"*"，不要返回用不到的字段

10、使用“临时表”暂存中间结果

采用临时表暂存中间结果好处：

（1）避免程序中多次扫描主表，减少程序执行“共享锁”阻塞“更新锁”，减少了阻塞，提高了并发性能。

（2）尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

（3）避免频繁创建和删除临时表，以减少系统资源的浪费。

（4）尽量避免向客户端返回大数据量，若数据量过大，应考虑相应需求是否合理。

11、limit分页优化

当偏移量特别时，limit效率会非常低

SELECT id FROM A LIMIT 1000,10 很快

SELECT id FROM A LIMIT 90000,10 很慢

优化方法：

方法一：select id from A order by id limit 90000,10; 很快，0.04秒就OK。 因为用了id主键做索引当然快

方法二：select id,title from A where id>=(select id from collect order by id limit 90000,1) limit 10;

方法三：select id from A order by id between 10000000 and 10000010;

12、批量插入优化

INSERT into person(name,age) values('A',14) INSERT into person(name,age) values('B',14) INSERT into person(name,age) values('C',14)

可优化为：

INSERT into person(name,age) values('A',14),('B',14),('C',14),

13、利用limit 1 、top 1 取得一行

有时要查询一张表时，你知道只需要看一条记录，你可能去查询一条特殊的记录。可以使用 limit 1 或者 top 1 来终止数据库索引继续扫描整个表或索引。

如SQL：SELECT id FROM A LIKE 'abc%' 优化为：SELECT id FROM A LIKE 'abc%' limit 1

14、尽量不要使用 BY RAND()命令

BY RAND()是随机显示结果，这个函数可能会为表中每一个独立的行执行BY RAND()命令，这个会消耗处理器的处理能力。

如SQL：SELECT * FROM A order by rand() limit 10 优化为：SELECT * FROM A WHERE id >= ((SELECT MAX(id) FROM A)-(SELECT MIN(id) FROM A)) * RAND() + (SELECT MIN(id) FROM

A) LIMIT 10

15、排序的索引问题

Mysql查询只是用一个索引，因此如果where子句中已经使用了索引的话，那么order by中的列是不会使用索引的。因此数据库默认排序可以符合要求情况下不要使用排序操作；

尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

16、尽量用 union add 替换 union

union和union all的差异主要是前者需要将两个（或者多个）结果集合并后再进行唯一性过滤操作，这就会涉及到排序，增加大量的cpu运算，加大资源消耗及延迟。所以当我们可以确认不可能出现重复结果集或者不在乎重复结果集的时候，尽量使用union all而不是union

17、避免类型转换

这里所说的“类型转换”是指where子句中出现column字段的类型和传入的参数类型不一致的时候发生的类型转换。人为的上通过转换函数进行转换，直接导致mysql无法使用索引。如果非要转型，应该在传入参数上进行转换。

例如utime 是datetime类型，传入的参数是“2016-07-23”，在比较大小时通常是 date（utime）>"2016-07-23",可以优化为utime>"2016-07-23 00：00：00"

18、尽可能使用更小的字段

MySQL从磁盘读取数据后是存储到内存中的，然后使用cpu周期和磁盘I/O读取它，这意味着越小的数据类型占用的空间越小，从磁盘读或打包到内存的效率都更好，但也不要太过执着减小数据类型，要是以后应用程序发生什么变化就没有空间了。

修改表将需要重构，间接地可能引起代码的改变，这是很头疼的问题，因此需要找到一个平衡点。

19、Inner join 和 left join、right join、子查询

第一：inner join内连接也叫等值连接是，left/rightjoin是外连接。

SELECT A.id,A.name,B.id,B.name FROM A LEFT JOIN B ON A.id =B.id;

SELECT A.id,A.name,B.id,B.name FROM A RIGHT JOIN ON B A.id= B.id;

SELECT A.id,A.name,B.id,B.name FROM A INNER JOIN ON A.id =B.id;

经过来之多方面的证实inner join性能比较快，因为inner join是等值连接，或许返回的行数比较少。但是我们要记得有些语句隐形的用到了等值连接，如：

SELECT A.id,A.name,B.id,B.name FROM A,B WHERE A.id = B.id;

推荐：能用inner join连接尽量使用inner join连接

第二：子查询的性能又比外连接性能慢，尽量用外连接来替换子查询。

Select* from A where exists (select * from B where id>=3000 and A.uuid=B.uuid);

A表的数据为十万级表，B表为百万级表，在本机执行差不多用2秒左右，我们可以通过explain可以查看到子查询是一个相关子查询(DEPENDENCE SUBQUERY);Mysql是先对外表A执行全表查询，然后根据uuid逐次执行子查询，如果外层表是一个很大的表，我们可以想象查询性能会表现比这个更加糟糕。

一种简单的优化就是用innerjoin的方法来代替子查询，查询语句改为：

Select* from A inner join B ON A.uuid=B.uuid using(uuid) where b.uuid>=3000；这个语句执行测试不到一秒；

第三：使用JOIN时候，应该用小的结果驱动打的结果（left join 左边表结果尽量小，如果有条件应该放到左边先处理，right join同理反向），同时尽量把牵涉到多表联合的查询拆分多个query (多个表查询效率低，容易锁表和阻塞)。如：

Select * from A left join B A.id=B.ref_id where A.id>10;可以优化为：select * from (select * from A wehre id >10) T1 left join B on T1.id=B.ref_id;

20、exist 代替 in

SELECT * from A WHERE idin (SELECT id from B)

SELECT * from A WHERE id EXISTS(SELECT 1 from A.id= B.id)

in 是在内存中遍历比较

exist 需要查询数据库，所以当B的数据量比较大时，exists效率优于in.

in()只执行一次，把B表中的所有id字段缓存起来，之后检查A表的id是否与B表中的id相等，如果id相等则将A表的记录加入到结果集中，直到遍历完A表的所有记录。

In 操作的流程原理如同一下代码

```
List resultSet={}; Array A=(select * from A); Array B=(select id from B); for(int i=0;i<A.length;i++) { for(int j=0;j<B.length;j++) { if(A[i].id==B[j].id) { resultSet.add(A[i]); break; } } } return resultSet;
```

可以看出，当B表数据较大时不适合使用in()，因为它会B表数据全部遍历一次

如：A表有10000条记录，B表有1000000条记录，那么最多有可能遍历10000*1000000次，效率很差。

再如：A表有10000条记录，B表有100条记录，那么最多有可能遍历10000*100次，遍历次数大大减少，效率大大提升。

结论：in()适合B表比A表数据小的情况

exist()会执行A.length()次，执行过程代码如下

```
List resultSet={}; Array A=(select * from A); for(int i=0;i<A.length;i++) { if(exists(A[i].id) { //执行select 1 from B where B.id=A.id是否有记录返回 resultSet.add(A[i]); } } return resultSet;
```

当B表比A表数据大时适合使用exists()，因为它没有那么多遍历操作，只需要再执行一次查询就行。

如：A表有10000条记录，B表有1000000条记录，那么exists()会执行10000次去判断A表中的id是否与B表中的id相等。

如：A表有10000条记录，B表有100000000条记录，那么exists()还是执行10000次，因为它只执行A.length次，可见B表数据越多，越适合exists()发挥效果。

再如：A表有10000条记录，B表有100条记录，那么exists()还是执行10000次，还不如使用in()遍历10000*100次，因为in()是在内存里遍历比较，而exists()需要查询数据库，

我们都知道查询数据库所消耗的性能更高，而内存比较很快。

结论：exists()适合B表比A表数据大的情况

当A表数据与B表数据一样大时，in与exists效率差不多，可任选一个使用。