《数据库原理》实验报告

实验	名称	数据查询分析实验
班	级	2014211304
学	号	2014211218
姓	名	

实验六 数据查询分析实验

一、 实验目的

通过对不同情况下查询语句的执行分析,巩固和加深对查询和查询优化相关理论知识的理解,提高优化数据库系统的实践能力,熟悉了解 kingbase 中查询分析器的使用,并进一步提高编写复杂查询的 SOL 程序的能力。

二、实验环境

MySQL 5.7 on win10 x64

三、 实验内容

1、索引对查询的影响

- 对结果集只有一个元组的查询分三种情况进行执行
- 对结果集中有多个元组的查询(例如查看某门成绩的成绩表)分类似(1)的三种情况进行执行比较
- 对查询条件为一个连续的范围的查询(例如查看学号在某个范围内的学生的选课情况)分类似(1)的三种情况进行执行比较、注意系统处理的选择
- 有索引和无索引的情况下插入数据(例如在选课情况表 SC 上插入数据),比较插入的执行效率

2、对相同查询功能不同查询语句的执行比较分析

3、查询优化

除了建立适当索引,对 SQL 语句重写外,还有其他手段来进行查询调优,例如调整缓冲区大小,事先建立视图等。

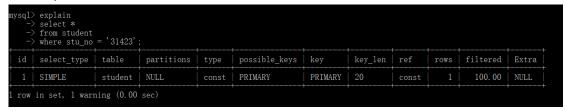
四、 实验步骤及结果分析

1、索引对查询的影响

由于 MySQL 没有聚集索引和非聚集索引的概念,因此在这里只比较没有索引和添加了聚集索引(主索引)之后的结果。

由于在创建 table 时 MySQL 已经为 table 添加了主索引 (如果有主码的话, 在 student 表里, 主码是 stu_no), 因此 student 表里已经带有了索引 stu_no, 且为主索引。

利用 explain 语句进行查询有如下结果:



结果表示这是一个简单的查询,仅用常数值比较就可以完成,使用索引的长度为20,

且仅仅考察了1列就返回了结果,效率(返回结果和需要查询的行数之比)为100%。

随后利用删除语句删除索引,再进行同样的查询,结果如下:

```
mysql> alter table student drop PRIMARY KEY;
Query OK, 54 rows affected (0.19 sec)
Records: 54 Duplicates: 0 Warnings: 0

mysql> explain
    -> select *
    -> from student
    -> where stu_no = '31423'
    -> ;

    id select_type table partitions type possible_keys key key_len ref rows filtered Extra

    1 SIMPLE student NULL ALL NULL NULL NULL S4 10.00 Using where
1 row in set, 1 warning (0.01 sec)
```

结果表示, 完成这个查询需要从头至尾扫描全表, 共扫描了 54 行, 效率仅为 10%。 为了进行接下来的实验, 用 create 语句恢复原来的索引。

```
mysql> create unique index index_stu_no on student(stu_no);
Query OK, O rows affected (0.10 sec)
Records: O Duplicates: O Warnings: O
```

对 sel 表进行多结果查询,不使用索引:

->	explain select * from where course ;		01'								
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	sel	NULL	ALL	NULL	NULL	NULL	NULL	142	10. 00	Using where
1 row	in set, 1 warr	ning (0.0	00 sec)			+					

使用索引:

```
mysql> create index index_course_no on sel(course_no)
-> ;
Query OK, 0 rows affected (0.04 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain
-> select * from sel
-> where course_no = 'CO1'
-> ;

id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |

1 | SIMPLE | sel | NULL | ref | index_course_no | index_course_no | 11 | const | 25 | 100.00 | NULL |

1 row in set, 1 warning (0.00 sec)
```

可见 type 由 ALL(全表扫描)改为了 ref (跟某个参考值相比较), 效率也由 10%提升到 100%。

对于范围索引,先验证有索引的情况下效率高,如下图:

-	> explain > select * from > where stu_no > and stu_no <	> '3141									
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	sel	NULL	range	PRIMARY	PRIMARY	20	NULL	19	100.00	Using where
1 row	in set, 1 warı	ning (0.	01 sec)								

再删掉在 stu_no 的索引后,使用同样的查找,结果如下:

```
mysql> ALTER TABLE sel DROP PRIMARY KEY;
Query OK, 142 rows affected (0.10 sec)
Records: 142 Duplicates: 0 Warnings: 0

mysql> explain
-> select * from sel
-> where stu_no > '31410'
-> and stu_no < '31418';

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
| 1 | SIMPLE | sel | NULL | ALL | NULL | NULL | NULL | NULL | 7 | 14.29 | Using where |
| 1 row in set, 1 warning (0.00 sec)
```

同样地,可见 type 由 range (在一定范围内做比对)变成了 ALL,效率也显著降低。接下来对比插入效率。

首先确保 sel 表中不带任何索引:

mysql> show index from sel; Empty set (0.00 sec)

执行插入语句, 得到结果如下:

nysq1	explain inser	rt into	sel values ('S	99999',	'AAA', '99');						
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	INSERT	sel	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
+ 1 row	in set (0.00 s	 sec)		+		+				l	

在 stu_no 添加索引后,结果如下:

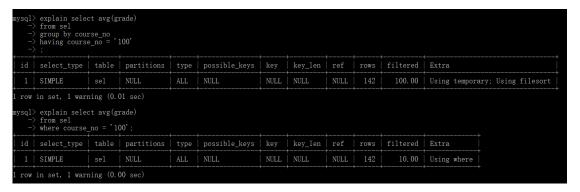
mysq1	explain inser	rt into	sel values ('S	99999',	'AAA', '99');						
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	INSERT	sel	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
1 row	in set (0.00 s	sec)									

却没有任何的变化。

2、对相同查询功能不同查询语句的执行比较分析

先做(1)

对于是否有 group by, 我们进行如下实验:



首先使用 group by 时间更长,这是因为 group by 要求先分组,having 子句筛选的是各个分组而不是元组,至少应该扫描全部进行分组后,再进行筛选工作;where 子句真正是针对元组的,因此后者只需要将表全部扫描一次就可完成筛选和选择工作。 之后做(2)。

由于缺少 age 字段,我们改用查找每个课程中取得最高分的学生的 ID 来代替。为了保证两种查找的等价性,我们先做如下测试:

```
mysql> select stu_no, course_no, grade
    -> from sel sl
    -> where grade =
    -> ( select max(grade)
    -> from sel s2
    -> where s1.course_no = s2.course_no
 stu_no
           course_no
                        grade
 30208
                           97
           C05
 30212
                           91
           C04
 31404
                           98
           C03
 31411
           C01
                           97
  31417
           C01
                           97
  31426
           C02
                           96
 rows in set (0.02 sec)
```

```
mysql> create table tmp
-> select course_no, max(grade) as max_grade
-> from sel
-> group by course_no;
Query OK, 5 rows affected (0.05 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

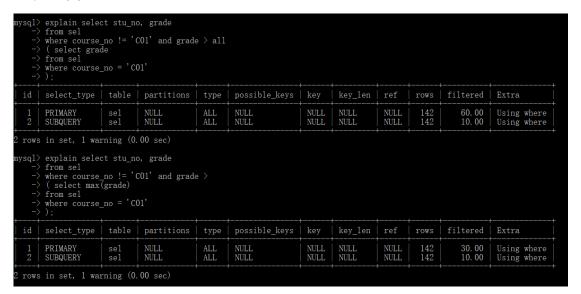
```
mysql> select stu_no, sel.course_no, grade
    -> from sel, tmp
-> where sel.grade = tmp.max_grade and sel.course_no = tmp.course_no;
  stu_no
            course no
                          grade
  30208
            C05
                             97
  30212
            C04
                             91
  31404
            C03
                             98
                             97
  31411
            C01
  31417
                             97
            C01
  31426
            C02
                             96
 rows in set (0.01 sec)
```

可见两者结果确实相同,在语句块之前那加入 explain 语句结果如下:

id select_type table partitions type possible_keys key key_len ref rows filtered Extra 1 SIMPLE tmp NULL ALL NULL NULL NULL NULL 5 100.00 NULL 1 SIMPLE sel NULL ALL NULL NULL NULL NULL 142 1.00 Using where; Using join buffer (Block Nested Loop)	->	explain selection from sel, tmp				ade course_no = tmp.	course_i	10;				
1 SIMPLE tmp NULL ALL NULL NULL NULL NULL 5 100.00 NULL 1 SIMPLE sel NULL ALL NULL NULL NULL NULL 142 1.00 Using where; Using join buffer (Block Nested Loop)	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
		SIMPLE SIMPLE	tmp sel	NULL NULL	ALL ALL	NULL NULL	NULL NULL	NULL NULL	NULL NULL	5 142	100. 00 1. 00	NULL Using where; Using join buffer (Block Nested Loop)

可见,后者仅查找了 5+142 个元组,由于前者存在嵌套,进行了两遍轮询,共查找了 142+142 个元组。因此后者的效率更高(在 tmp 表已经存在的情况下,即忽略因建立 tmp 表而带来的开销)。

再来做(3)。

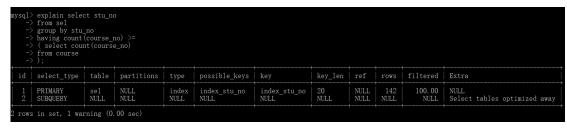


可以看出前者的 filtered 效率更高,可见 all 关键字的使用是优于聚集函数 max 的。 打开 profiles 也可以看到运行时间的差别:

3、查询优化

(1) 查找选修了每一门课的学生

由于学生不能选两次同样的课,因此我们可以根据总课程数目做数值比较:



执行结果为:

```
mysql> select stu_no
   -> from sel
   -> group by stu_no
   -> having count(course_no) >=
   -> ( select count(course_no)
   -> from course
   -> );
Empty set (0.00 sec)
```

详细时间参数为:

(2) 查找至少选修了课程数据库原理和操作系统的学生的学号

我们选用先找出选择其中一门课的学号, 再从这些学号中筛出选择了另一门课的同学。

```
mysql> select stu_no
  -> );
 stu_no
 31401
 31403
 31404
 31406
 31407
 31408
 31409
 31411
 31412
 31414
 31415
 31417
 31418
 31419
31420
31422
 31423
 31424
 31426
 31428
20 rows in set (0.01 sec)
```

时间统计结果为:

-> ; 	++
Status	Duration
starting checking permissions checking permissions checking permissions checking permissions Opening tables init System lock optimizing statistics preparing executing Sending data end query end removing tmp table query end closing tables freeing items cleaning up	0. 004054 0. 000006 0. 000007 0. 000001 0. 000005 0. 000828 0. 000064 0. 000012 0. 000019 0. 000752 0. 000419 0. 000490 0. 000003 0. 000003 0. 000009 0. 000005 0. 000001 0. 000008 0. 000072 0. 000023

五、 实验小结

本次实验主要完成了查询分析和查询优化。让我们知道在 SQL 中有各种各样的手段去完成优化,并利用 profile 信息或 explain 语句来做查询效率分析,其中各个统计值的含义在 MySQL 手册上均可以查到,其中也不乏一些晦涩难懂的参数。

了解并学习各种 SQL 优化方法和原理,可以在处理大型查询和复杂查询时,加快查询效率,更快更高效地得出结果。此外,对索引的理解和使用也是提高数据库查询效率的重要手段,其主要思想就是类似于书本目录的方式,使得一个查询过程未必需要遍历全表,而是可以利用一些搜索码进行快速定位和查找。