

### 1. 插入orders表的插入方式操作描述，时间截图

在保存有data1.txt的目录下进入MySQL CLI，然后执行load file local infile 'data1.txt' into table orders;

```
mysql> load data local infile 'data1.txt' into table orders;
Query OK, 5000000 rows affected, 65535 warnings (25.15 sec)
Records: 5000000 Deleted: 0 Skipped: 0 Warnings: 5000000
```

### 2. 插入products表的插入方式操作描述，时间截图

在保存有data2.txt的目录下进入MySQL CLI，然后执行load file local infile 'data2.txt' into table products;

```
mysql> load data local infile 'data2.txt' into table products;
Query OK, 10000 rows affected (0.10 sec)
Records: 10000 Deleted: 0 Skipped: 0 Warnings: 0
```

### 3. 问题1：在 orders 表中找出购买人年龄小于20岁的order列表。

SQL: select \* from orders where age < 20;

建立索引方式：在age列上建立B+树索引，create index age\_index on orders(age);

```
mysql> create index age_index on orders(age);
Query OK, 0 rows affected (12.48 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

理由：此为检索范围值的查询，可以在age上执行B+索引以加快此SQL查询的速度。（通过实践，发现最终结论是不应建立索引。原因请见本部分最后的分析和结论。）

建立索引前后执行效率截图：

前：

执行结果：

```
571196 rows in set (3.41 sec)
```

执行计划

```
mysql> explain select * from orders where age < 20;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | ALL | NULL | NULL | NULL | NULL | 4987295 | 33.33 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

可以看到type是ALL，即遍历表查询。

后：

执行结果：

```
571196 rows in set (3.64 sec)
```

## 执行计划

```
mysql> explain select * from orders where age < 20;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | ALL | age_index | NULL | NULL | NULL | 4987295 | 25.09 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

## 分析和结论：

创建索引前后执行结果并没有大区别，从explain可以看到查询的方法还是使用ALL遍历查询，key为NULL，即是说本例并没有采用之前建立的索引。即是说本例不应建立索引。

探索其原因，发现age列的选择性（即不同的取值个数和总数的比值）过低（如下图）。这样，如果通过索引去查询，也需要访问大量的（在这个例子里是571196个）索引来访问数据。由于程序局部性和cache的存在，当需要访问大量的索引的时候，通过索引访问数据可能并不能直接全表查询的速度更快。所以在这个例子中，数据库还是采取了直接全表查询的策略。

```
mysql> select count(*) from orders;
+-----+
| count(*) |
+-----+
| 5000000 |
+-----+
1 row in set (1.56 sec)

mysql> select count(distinct(age)) from orders;
+-----+
| count(distinct(age)) |
+-----+
| 70 |
+-----+
1 row in set (0.03 sec)
```

4. 问题2：在 orders 表中找出所有姓王的人的order列表。

SQL: `select * from orders where name like '王%';`

建立索引方式：在name列上建立B+树索引，`create index name_index on orders(name);`

```
mysql> create index name_index on orders(name);
Query OK, 0 rows affected (17.88 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

原因：此为前缀查询。

建立索引前后执行效率截图：

前：

执行结果：

```
11160 rows in set (3.18 sec)
```

执行计划：

```
mysql> explain select * from orders where name like '王%';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | ALL | NULL | NULL | NULL | NULL | 4987295 | 11.11 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

可以看到type是ALL，即遍历表查询。

后：

执行结果：

```
11160 rows in set (2.74 sec)
```

执行计划：

```
mysql> explain select * from orders where name like '王%';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | range | name_index | name_index | 303 | NULL | 20630 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

分析：

从执行计划可以看到，此查询可以利用name上的B+索引。从执行结果可以看到，此索引加快了查询速度。

和第一个问题进行比对，可以看到name列的选择性较高（34772290 vs 70），所以此索引能够有效地增加查询效率。

```
mysql> select count(distinct(name)) from orders
+-----+
| count(distinct(name)) |
+-----+
| 3477290 |
+-----+
1 row in set (3.59 sec)
```

5. 问题3：统计 orders 表中所有男性的人的数量。

SQL: `select count(*) from orders where sex = '男';`

建立索引方式：在sex列上建立索引，`create index sex_index on orders(sex);`。

```
mysql> create index sex_index on orders(sex);
Query OK, 0 rows affected (10.65 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

原因：本例是查询count(\*)而不是select \*，省去了通过索引访问数据的开销，只需要统计索引的个数即可。所以即使sex的选择性比较低，其二叉树搜索的方式仍然可以避免全表查询，提供查询效率。

建立索引前后执行效率截图：

前：

执行结果:

```
mysql> select count(*) from orders where sex = '男';
+-----+
| count(*) |
+-----+
| 2499997 |
+-----+
1 row in set (2.37 sec)
```

执行计划:

```
mysql> explain select count(*) from orders where sex = '男';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | ALL | NULL | NULL | NULL | NULL | 4987295 | 50.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

后:

执行结果:

```
mysql> select count(*) from orders where sex = '男';
+-----+
| count(*) |
+-----+
| 2499997 |
+-----+
1 row in set (1.04 sec)
```

执行计划:

```
mysql> explain select * from orders where sex = '男';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | ref | sex_index | sex_index | 2 | const | 2493647 | 100.00 | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

6. 问题4: 在 orders 表中计算女性, 姓张, 年龄大于50, 且消费小于100的人数。

SQL: `select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;`

建立索引方式: 依次在sex, name, amount, age列上建立B+复合索引, `create index orders_index on orders(sex, name, amount, age);;`

```
mysql> create index orders_index on orders(sex, name, amount, age);
Query OK, 0 rows affected (21.07 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

原因: 此可以建立复合索引。首先选出完全匹配的列 (sex), 然后把剩下三个范围匹配的列根据选择性从高到低 (在此例子中, 各列的选择性见下图) 进行排序, 以增加对此查询的索引效率。

```
mysql> select count(distinct(name)), count(distinct(amount)), count(distinct(age)), count(distinct(sex)) from orders;
+-----+-----+-----+-----+
| count(distinct(name)) | count(distinct(amount)) | count(distinct(age)) | count(distinct(sex)) |
+-----+-----+-----+-----+
| 3477290 | 100001 | 70 | 2 |
+-----+-----+-----+-----+
1 row in set (22.21 sec)
```

建立索引前后执行效率截图：

前：

执行结果：

```
mysql> select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;
+-----+
| count(*) |
+-----+
|      258 |
+-----+
1 row in set (3.52 sec)
```

执行计划：

```
mysql> explain select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | ALL | NULL | NULL | NULL | NULL | 4987295 | 0.62 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

后：

执行结果：

```
mysql> select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;
+-----+
| count(*) |
+-----+
|      258 |
+-----+
1 row in set (0.00 sec)
```

执行计划：

```
mysql> explain select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | range | orders_index | orders_index | 310 | NULL | 10282 | 11.11 | Using where; Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

结论和补充：

在已经有orders\_index的基础上，我还测试了只按照选择性排序建立索引create index orders\_index\_1 on orders(name, amount, age, sex);，结果显示系统显示仍然使用orders\_index进行查询优化。

```
mysql> create index orders_index_1 on orders(name, amount, age, sex);
Query OK, 0 rows affected (19.82 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | range | orders_index,orders_index_1 | orders_index | 310 | NULL | 10282 | 11.11 | Using where; Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

删除orders\_index，仅使用orders\_index\_1索引时，仍然有速度提升，但是仍然不如orders\_index。

```
mysql> drop index orders_index on orders;
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> explain select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | range | orders_index_1 | orders_index_1 | 308 | NULL | 21932 | 5.55 | Using where; Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> select count(*) from orders where sex = '女' and name like '张%' and age > 50 and amount < 100;
+-----+
| count(*) |
+-----+
| 258 |
+-----+
1 row in set (0.01 sec)
```

这证明了在选择如何排序复合索引的列时，需要首先考虑完全匹配和范围查询，然后再考虑选择性。

7. 问题5: 统计 orders 表中姓名为三个字的人数。

SQL: `select count(*) from orders where name like '___';`

建立索引方式: 在name列上建立B+树索引, `create index name_three_index on orders(name);`

```
mysql> create index name_three_index on orders(name);
Query OK, 0 rows affected (16.43 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

原因: 此为范围查询, 且name列选择性比较高。

前:

执行结果:

```
mysql> select count(*) from orders where name like '___';
+-----+
| count(*) |
+-----+
| 2501252 |
+-----+
1 row in set (2.68 sec)
```

执行计划:

```
mysql> explain select count(*) from orders where name like '___';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | ALL | NULL | NULL | NULL | NULL | 4987295 | 11.11 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

后:

执行结果:

```
mysql> select count(*) from orders where name like '___';
+-----+
| count(*) |
+-----+
| 2501252 |
+-----+
1 row in set (2.36 sec)
```

执行计划:

```
mysql> explain select count(*) from orders where name like '___';
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | orders | NULL | index | NULL | name_three_index | 303 | NULL | 4987295 | 11.11 | Using where; Using index |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

分析:

在这个例子中可以看出, 虽然索引对查询结果有一定的提升, 但是实际提升效果不大(300ms级别)。推测可能是因为通过索引查询到的索引数量过多(2501252, 超过了50%), 造成通过索引访问数据时的时间花费。

8. 问题6: 在 products 表中查找库存大于150的product列表。

SQL: `select * from products where nums>150;`

建立索引方式: B+树索引, `create index nums_index on products(nums);`

```
mysql> create index nums_index on products(nums);
Query OK, 0 rows affected (0.07 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

原因: 是个范围查询。(通过实践, 发现最终结论是不应建立索引。原因请见本部分最后的分析和结论。)

建立索引前后执行效率截图:

前:

执行结果:

```
2534 rows in set (0.02 sec)
```

执行计划:

```
mysql> explain select * from products where nums > 150;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | products | NULL | ALL | NULL | NULL | NULL | NULL | 9963 | 33.33 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

后:

执行结果:

```
2534 rows in set (0.01 sec)
```

执行计划:

```
mysql> explain select * from products where nums > 150;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | products | NULL | ALL | nums_index | NULL | NULL | NULL | 9963 | 25.43 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

结论:

可以看到对于此查询, nums\_index索引并没有被使用。经过测试, 在n>=167的时候才开始采用索引。

```
mysql> explain select * from products where nums > 167;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | products | NULL | range | nums_index | nums_index | 5 | NULL | 1671 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from products where nums > 166;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | products | NULL | ALL | nums_index | NULL | NULL | NULL | 9963 | 17.29 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

论其原因，可能和在问题1遇到的问题是一样的：当查询结果索引量比较大的时候，通过索引访问数据的开销将会大于使用索引所节省的时间。所以在 $n \geq 167$ 的时候，数据量够小，才开始使用索引来查找。