DB实验1

本次实验在java + jdbc拼sql完成实验中的导入部分，在索引部分使用命令行排除jdbc的干扰和可能的优化[项目地址](#)，截图为在mac OSX 16G环境下的运行时间

步骤二

jdbc直接从txt导入到数据库表，对应语句

```
 private static String LOAD_ORDERS_DATA =
        "load data local infile \"/Users/py/Downloads/索引作业数据/data1.txt\"\n" +
        "into table orders(id, name, age, sex, amount);";

private static String LOAD_PRODUCTS_DATA =
        "load data local infile \"/Users/py/Downloads/索引作业数据/data2.txt\"\n" +
        "into table products(id, pid, nums);";

private void loadOrdersData() {
        try {
            Statement statement = connection.createStatement();
            long startTime=System.currentTimeMillis();
            statement.execute(LOAD_ORDERS_DATA);
            long endTime=System.currentTimeMillis();
            System.out.println("Load orders data costs: " + (endTime - startTime)/
1000 + " s.");
        }catch (Exception ex) {
            ex.printStackTrace();
        }
    }

private void loadProductsData() {
    try {
        Statement statement = connection.createStatement();
        long startTime=System.currentTimeMillis();
        statement.execute(LOAD_PRODUCTS_DATA);
        long endTime=System.currentTimeMillis();
        System.out.println("Load products data costs: " + (endTime - startTime)/10
00 + " s.");
    }catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java ...
Succeed to connect...
Load orders data costs: 29 s.
Load products data costs: 0 s.

Process finished with exit code 0
```

步骤三

问题1:在 orders 表中找出购买人年龄小于20岁的order列表。

未加入索引前sql语句:

```
select * from orders where age < 20
```

```
571196 rows in set (1.96 sec)
```

加入索引:

```
create index age_idx on orders(age)
```

```
mysql> create index age_idx on orders(age)
    ->
    ->
    ->
    -> ;

Query OK, 0 rows affected (9.86 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

再次执行查询:

```
select * from orders where age < 20
```

```
+---------+-----------+------+------+--------+
571196 rows in set (1.89 sec)

mysql> 
```

发现在age列建立B+树索引并没有对查询效率有明显帮助

```
explain select * from orders where age < 20;
```

```
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_idx       | NULL | NULL
 | NULL | 4982784 |    33.33 | Using where |
+----+-------------+--------+------------+------+---------------+------+--------
-+------+---------+----------+-------------+
1 row in set, 2 warnings (0.00 sec)
```

发现这依旧是一个全表查询，type为ALL

究其原因在查询过程中集合index value的大小十分重要，总结来说是当列越长，越少的值能被放入B+树的结点，因此树的深度会变大，当树的深度越大时，需要更多磁盘访问，磁盘访问越多效率越低

这是援引ovasitariq的一则回答翻译理解过来

在这个例子中，age列集合显然不会超过200与整个order表无法相比，此时磁盘开销巨大，需要更多的索引，而这些具体的开销高昂以至于MySQL直接全表查询

问题2:在 orders 表中找出所有姓王的人的order列表。

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

```
+---------+-----------+------+------+--------+
11160 rows in set (1.62 sec)
```

继续建B+索引在name上

```
create index name_idx on orders(name);
```

```
mysql> create index name_index on orders(name);
Query OK, 0 rows affected (10.83 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

继续查询

```
| 4999928 | 王亿       | 42   | 女   |      72 |
+---------+----------+------+------+--------+
11160 rows in set (0.11 sec)
```

效率大大增加

可以理解，因为名字集合本身很大，统计一下

```
select count(distinct(name)) from orders;
```

```
mysql> select count(distinct(name)) from orders
    -> ;
+-----------------------+
| count(distinct(name)) |
+-----------------------+
|               3477290 |
+-----------------------+
1 row in set (1.84 sec)
```

总: 3477290确实很巨大，考量表的其他列，名字的区分性确实让本身建立索引查询占优

问题3:统计 orders 表中所有男性的人的数量。

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

```
+----------+
|  2499997 |
+----------+
1 row in set (1.17 sec)
```

心里有数然而继续实践一下

```
create index sex_idx on orders(sex);
```

```
mysql> create index sex_idx on orders(sex);
Query OK, 0 rows affected (6.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

```
mysql> select count(*) from orders where sex = '男'
+----------+
| count(*) |
+----------+
|  2499997 |
+----------+
1 row in set (0.44 sec)
```

因为是单纯计数而不需要访问本身，其索引建立的意义得到体现

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

```
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       | ref  | sex_idx       | sex_idx | 2
    | const | 2491392 |   100.00 | Using index |
+----+-------------+--------+------------+------+---------------+---------+-----
----+-------+---------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

此时是索引查询

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。

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

```
mysql> select count(*) from orders where sex = '女' and name like '张%' and age
> 50 and amount < 100;
+----------+
| count(*) |
+----------+
|     2586 |
+----------+
1 row in set (0.10 sec)
```

沿用上面两题建立对sex, name, amount, age的复合索引而在计数不需要担心全部访问的尴尬

```
create index mul_idx on orders(sex, name, amount, age);
```

```
mysql> create index mul_idx on orders(sex, name, amount, age);
Query OK, 0 rows affected (13.42 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Once again

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

```
mysql> select count(*) from orders where sex = '女' and name like '张%' and age
> 50 and amount < 100;
+----------+
| count(*) |
+----------+
|     2586 |
+----------+
1 row in set (0.01 sec)
```

Stack Overflow对于复合列索引有一个类比电话本的解释我很喜欢，写一下:

1. 如果你找一个姓潘的人，你可以很容易找到因为电话本按姓排列
2. 如果你找一个名羽的人，那没办法了，因为电话本又链接不到名，你得查找全部😭
3. 如果你找一个姓潘名羽的肥宅，电话本会很有帮助，因为你能找到潘以后按名的顺序找到羽

以上解释了查找准确值，但是如果很不幸你想找**一个范围**，比如名是羽姓以p开头，这时候你得先找到那些名羽的人，只有先这样你才能group

我记下来的原因是因为这对理解复合列的索引很有帮助，**不要以为你做完了，你只是运气好吧sex放第一个，那么如果你运气不好呢，尝试一把**

```
create index test_idx0 on orders(name, amount, age, sex);
```

```
mysql> create index test_idx0 on orders(name, amount, age, sex);
Query OK, 0 rows affected (12.85 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> 
```

忽略那些令人尴尬的warning，对multi-index都有数，MySQL还是选择了我第一把建的mul_index，这是有道理的

```
---------------+---------+---------+------+------+----------+-----------------
--------+
|  1 | SIMPLE      | orders | NULL       | range | age_idx,name_index,sex_idx,mu
l_idx,test_idx0 | mul_idx | 140      | NULL | 5715 |    11.11 | Using where; Usin
g index |
+----+-------------+--------+------------+-------+-------------------------------
---------------+---------+---------+------+------+----------+-----------------
--------+
1 row in set, 4 warnings (0.00 sec)

mysql> 
```

此时心中应该规定优先级 exact > range > like

问题5:统计 orders 表中姓名为三个字的人数。

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

```
mysql> select count(*) from orders where name like '___';
+----------+
| count(*) |
+----------+
|  2501252 |
+----------+
1 row in set (1.18 sec)
```

```
create index name_idx on orders(name);
```

```
mysql> create index name_idx on orders(name);
Query OK, 0 rows affected, 1 warning (10.04 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

好了，这个索引好像**没用**

```
ysql> select count(*) from orders where name like '___';
----------+
 count(*) |
----------+
  2501252 |
----------+
 row in set (1.16 sec)
```

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

```
id | select_type | table  | partitions | type  | possible_keys
y_len | ref  | rows     | filtered | Extra                      |
---+------------+-------+-----------+------+-------------
-----+-----+--------+---------+------------------------+
 1 | SIMPLE      | orders | NULL       | index | NULL
8      | NULL | 4982784 |    11.11 | Using where; Using index |
---+------------+-------+-----------+------+-------------
-----+-----+--------+---------+------------------------+
```

其实还是用了的，然而这个索引之所以没有提升也不能理解，**索引本身的数量**决定了性能，而对于name列来说这本身就是一个巨大的挑战，MySQL拿出资源来维护索引本身也是**高昂的开销**

问题6:在 products 表中查找库存大于150的product列表。

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

```
+------+--------------+------+
2534 rows in set (0.00 sec)
```

其实我已经不想建索引了，0s我也看不出来，但没办法，Once again

```
create index nums_idx on products(nums);
```

```
mysql> create index nums_idx on products(nums);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

其实你**建了也没用**

还是那句话，索引本身是有代价的，MySQL会在资源和查询中作出权衡，具体视数据而定，这道题不应该建