Mysql索引

1、索引（index）

1.1、什么是索引?

索引是在数据库表的字段上添加的，是为了提高查询效率存在的一种机制。
一张表的一个字段可以添加一个索引，当然，多个字段联合起来也可以添加索引。
索引相当于一本书的目录，是为了缩小扫描范围而存在的一种机制。

索引的执行过程是：sql查询语句中的查询部分(如：where子句)，在查找列值的时候，如果发现要查询的列上有索引，就会用索引去查。如果发现没有索引，就将表中的这个列全部进行查找。

对于一本字典来说，查找某个汉字有两种方式：
第一种方式：一页一页挨着找，直到找到为止，这种查找方式属于全字典扫描。
效率比较低。
第二种方式：先通过目录（索引）去定位一个大概的位置，然后直接定位到这个
位置，做局域性扫描，缩小扫描的范围，快速的查找。这种查找方式属于通过
索引检索，效率较高。

```
t_user

id(idIndex)     name(nameIndex)     email(emailIndex)
address(emailAddressIndex)
-------------------------------------------------------------------------------
-

1               zhangsan...
2               lisi
3               wangwu
4               zhaoliu
5               hanmeimei
6               jack
```

select * from t_user where name = 'jack';

以上的这条SQL语句会去name字段上扫描，为什么?
因为查询条件是：name='jack'

如果name字段上没有添加索引（目录），或者说没有给name字段创建索引，
MySQL会进行全表扫描，会将name字段上的每一个值都比对一遍。效率比较低。

MySQL在查询方面主要就是两种方式：
第一种方式：全表扫描。全表扫描：是指将表中的查询条件列全部进行扫描，而不是指将表中全部的列进行扫瞄。
第二种方式：根据索引检索。

注意：
在实际中，汉语字典前面的目录是排序的，按照a b c d e f....排序，
为什么排序呢？因为只有排序了才会有区间查找这一说！（缩小扫描范围
其实就是扫描某个区间罢了！）

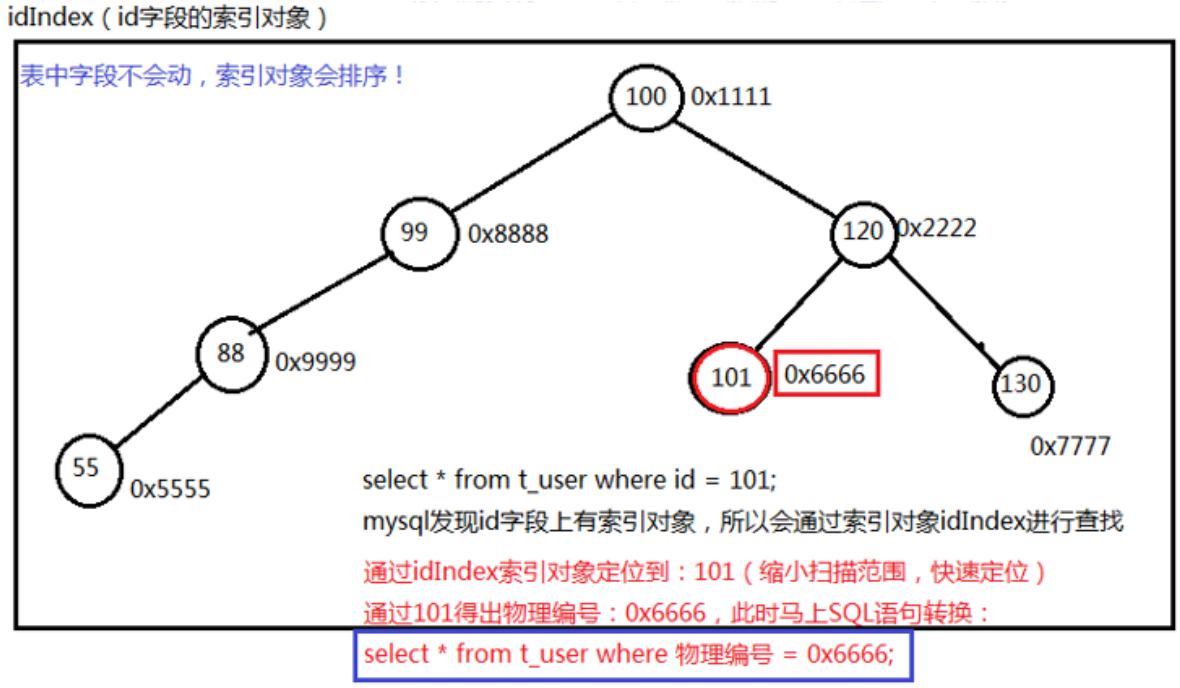
在mysql数据库当中索引也是需要排序的，并且这个所以的排序和TreeSet数据结构相同。TreeSet（TreeMap）底层是一个自平衡的二　叉树！在mysql当中索引是一个B-Tree数据结构。也就是说索引的排序是将索引排成了一个B树的形式。

遵循左小又大原则存放。采用中序遍历方式遍历取数据。

1.2、索引的实现原理：

假设有一张用户表：t_user

```
id(PK)                  name                    每一行记录在硬盘上都有物理存储编号
-------------------------------------------------------------------------------
-
100                     zhangsan                0x1111
120                     lisi                    0x2222
99                      wangwu                  0x8888
88                      zhaoliu                 0x9999
101                     jack                    0x6666
55                      lucy                    0x5555
130                     tom                     0x7777
```



然后就取到了一整条记录！

提醒1：
在任何数据库当中主键上都会自动添加索引对象，id字段上自动有索引，因为id是PK。另外在mysql当中，一个字段上如果有unique约　　束的话，也会自动创建索引对象。

提醒2：
在任何数据库当中，任何一张表的任何一条记录在硬盘存储上都有一个硬盘的物理存储编号。

提醒3：
在mysql当中，索引是一个单独的对象，不同的存储引擎以不同的形式存在，在MyISAM存储引擎中，索引存储在一个.MYI文件中。
在InnoDB存储引擎中索引存储在一个逻辑名称叫做tablespace的当中。
在MEMORY存储引擎当中索引被存储在内存当中。
不管索引存储在哪里，索引在mysql当中都是一个树的形式存在。（自平衡二叉树：B-Tree）

索引的执行过程是：

sql查询语句中的查询部分(如：where子句)，在查找列值的时候，如果发现要查询的列上有索引，就会用索引去查。如果发现没有索引，就将表中的这个列全部进行查找。

1.3、在mysql当中，主键上，以及unique字段上都会自动添加索引的！！！

什么条件下，我们会考虑给字段添加索引呢?

条件1：数据量庞大（到底有多么庞大算庞大，这个需要测试，因为每一个硬件环境不同）
条件2：该字段经常出现在where的后面，以条件的形式存在，也就是说这个字段总是被扫描。
条件3：该字段很少的DML(insert delete update)操作。（因为DML之后，索引需要重新排序。）

建议不要随意添加索引，因为索引也是需要维护的，太多的话反而会降低系统的性能。
建议通过主键查询，建议通过unique约束的字段进行查询，效率是比较高的。

1.4、索引怎么创建？怎么删除？语法是什么？

```
创建索引：
    mysql> create index emp_ename_index on emp(ename);
    给emp表的ename字段添加索引，起名：emp_ename_index

删除索引：
    mysql> drop index emp_ename_index on emp;
    将emp表上的emp_ename_index索引对象删除。
```

1.5、在mysql当中，怎么查看一个SQL语句是否使用了索引进行检索?

```
mysql> explain select * from emp where ename = 'KING';
+----+-------------+-------+------+---------------+------+---------+------+------
+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------
+-------------+
|  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |   14
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+------
+-------------+
扫描14条记录：说明没有使用索引。type=ALL

mysql> create index emp_ename_index on emp(ename);

mysql> explain select * from emp where ename = 'KING';
+----+-------------+-------+------+-----------------+-----------------+---------
+-------+------+-------------+
| id | select_type | table | type | possible_keys   | key             | key_len |
ref   | rows | Extra       |
+----+-------------+-------+------+-----------------+-----------------+---------
+-------+------+-------------+
```

```
|  1 | SIMPLE      | emp   | ref  | emp_ename_index | emp_ename_index | 33      |
const |    1 | Using where |
+----+-------------+-------+------+-----------------+-----------------+---------
+-------+------+-------------+
```

1.6、索引有失效的时候，什么时候索引失效呢?

失效的第1种情况:

```
select * from emp where ename like '%T';
```

ename上即使添加了索引，也不会走索引，为什么?
原因是因为模糊匹配当中以“%”开头了!
尽量避免模糊查询的时候以“%”开始。
这是一种优化的手段/策略。

```
    mysql> explain select * from emp where ename like '%T';
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
    | id | select_type | table | type | possible_keys | key  | key_len | ref  |
rows | Extra       |
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
    |  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |
14 | Using where |
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
```

失效的第2种情况:

使用or的时候会失效，如果使用or那么要求or两边的条件字段都要有索引，才会走索引，如果其中一边有一个字段没有索引，那么另　一个字段上的索引也会失效。所以这就是为什么不建议使用or的原因。

```
    mysql> explain select * from emp where ename = 'KING' or job = 'MANAGER';
    +----+-------------+-------+------+-----------------+------+---------+------
+------+-------------+
    | id | select_type | table | type | possible_keys   | key  | key_len | ref  |
rows | Extra       |
    +----+-------------+-------+------+-----------------+------+---------+------
+------+-------------+
    |  1 | SIMPLE      | emp   | ALL  | emp_ename_index | NULL | NULL    | NULL |
  14 | Using where |
    +----+-------------+-------+------+-----------------+------+---------+------
+------+-------------+
```

所以建议用union而不是用or。

失效的第3种情况:

使用复合索引的时候，没有使用建立这个复合索引时的左侧的列查找，索引失效
什么是复合索引?
两个字段，或者更多的字段联合起来添加一个索引，叫做复合索引。

```
    create index emp_job_sal_index on emp(job,sal);

    mysql> explain select * from emp where job = 'MANAGER';
+----+-------------+-------+------+------------------+------------------+------
---+-------+------+----------- --+
| id | select_type | table | type | possible_keys     | key              |
key_len | ref   | rows | Extra       |
+----+-------------+-------+------+------------------+------------------+------
---+-------+------+-------------+
|  1 | SIMPLE      | emp   | ref  | emp_job_sal_index | emp_job_sal_index | 30
   | const |    3 | Using where |
+----+-------------+-------+------+------------------+------------------+------
---+-------+------+-------------+

    mysql> explain select * from emp where sal = 800;
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
    | id | select_type | table | type | possible_keys | key  | key_len | ref  |
rows | Extra       |
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
    |  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |
14 | Using where |
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
```

```
create index emp_job_sal_index on emp(job,sal);
                                  左侧的列      右侧的列
```

失效的第4种情况：

在where当中索引列参加了运算，索引失效。

```
    mysql> create index emp_sal_index on emp(sal);

    explain select * from emp where sal = 800;
    +----+-------------+-------+------+---------------+---------------+---------
+-------+------+-------------+
    | id | select_type | table | type | possible_keys | key           | key_len |
ref   | rows | Extra       |
    +----+-------------+-------+------+---------------+---------------+---------
+-------+------+-------------+
    |  1 | SIMPLE      | emp   | ref  | emp_sal_index | emp_sal_index | 9       |
const |    1 | Using where |
    +----+-------------+-------+------+---------------+---------------+---------
+-------+------+-------------+

    mysql> explain select * from emp where sal+1 = 800;
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
    | id | select_type | table | type | possible_keys | key  | key_len | ref  |
rows | Extra       |
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
    |  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    | NULL |
14 | Using where |
```

```
    +----+-------------+-------+------+---------------+------+---------+------+--
----+-------------+
```

失效的第5种情况：

在where当中索引列使用了函数

```
     explain select * from emp where lower(ename) = 'smith';
        +----+-------------+-------+------+---------------+------+---------+----
--+------+-------------+
        | id | select_type | table | type | possible_keys | key  | key_len | ref
| rows | Extra       |
        +----+-------------+-------+------+---------------+------+---------+----
--+------+-------------+
        |  1 | SIMPLE      | emp   | ALL  | NULL          | NULL | NULL    |
NULL |   14 | Using where |
        +----+-------------+-------+------+---------------+------+---------+----
--+------+-------------+
```

1.7、索引是各种数据库进行优化的重要手段。优化的时候优先考虑的因素就是索引。

索引的分类：

单一索引：一个字段上添加索引。
复合索引：在两个字段或者更多的字段上联合起来添加的一个索引。
主键索引：主键上添加索引。
唯一性索引：具有unique约束的字段上添加索引。
.....

哪些字段上添加索引用处不大？：

注意：唯一性比较弱的字段上添加索引用处不大。字段唯一性比较弱的意思是：字段中存在大量重复数据，在这种字段上添加索引，索引不会起太大作用。字段的唯一性越强，在它上面建索引起到的作用越大。