最近一周接连处理了2个由于int向varchar转换无法使用索引，从而引发的慢查询。

```
CREATE TABLE `appstat_day_prototype_201305` (
  `day_key` date NOT NULL DEFAULT '1900-01-01',
  `appkey` varchar(20) NOT NULL DEFAULT '',
  `user_total` bigint(20) NOT NULL DEFAULT '0',
  `user_activity` bigint(20) NOT NULL DEFAULT '0',
  `times_total` bigint(20) NOT NULL DEFAULT '0',
  `times_activity` bigint(20) NOT NULL DEFAULT '0',
  `incr_login_daily` bigint(20) NOT NULL DEFAULT '0',
  `unbind_total` bigint(20) NOT NULL DEFAULT '0',
  `unbind_activitys` bigint(20) NOT NULL DEFAULT '0',
  PRIMARY KEY (`appkey`,`day_key`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8

mysql> explain SELECT * from appstat_day_prototype_201305 where appkey =
xxxxx and day_key between '2013-05-23' and '2013-05-30';
+----+-------------+-------------------------------+------+--------------
-+------+---------+------+----------+-------------+
| id | select_type | table                         | type | possible_keys
| key  | key_len | ref  | rows     | Extra       |
+----+-------------+-------------------------------+------+--------------
-+------+---------+------+----------+-------------+
|  1 | SIMPLE      | appstat_day_prototype_201305 | ALL  | PRIMARY
| NULL | NULL    | NULL | 19285787 | Using where |
+----+-------------+-------------------------------+------+--------------
-+------+---------+------+----------+-------------+
1 row in set (0.00 sec)

mysql> explain SELECT * from appstat_day_prototype_201305 where appkey =
'xxxxx' and day_key between '2013-05-23' and '2013-05-30';
+----+-------------+-------------------------------+-------+-------------
--+---------+---------+------+------+-------------+
| id | select_type | table                         | type  |
possible_keys | key     | key_len | ref  | rows | Extra       |
+----+-------------+-------------------------------+-------+-------------
--+---------+---------+------+------+-------------+
|  1 | SIMPLE      | appstat_day_prototype_201305 | range | PRIMARY
| PRIMARY | 65      | NULL |    1 | Using where |
+----+-------------+-------------------------------+-------+-------------
--+---------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

从上面可以很明显的看到由于appkey是varchar，而在where条件中不加''，会引发全表查

询，加了就可以用到索引，这扫描的行数可是天差地别，对于服务器的压力和响应时间自然也是天差地别的。
我们再看另外一个例子：

```
*************************** 1. row ***************************
       Table: poll_joined_151
Create Table: CREATE TABLE `poll_joined_151` (
  `poll_id` bigint(11) NOT NULL,
  `uid` bigint(11) NOT NULL,
  `item_id` varchar(60) NOT NULL,
  `add_time` int(11) NOT NULL DEFAULT '0',
  `anonymous` tinyint(1) NOT NULL DEFAULT '0',
  `sub_item` varchar(1200) NOT NULL DEFAULT '',
  KEY `idx_poll_id_uid_add_time` (`poll_id`,`uid`,`add_time`),
  KEY `idx_anonymous_id_addtime` (`anonymous`,`poll_id`,`add_time`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8

SELECT * FROM poll_joined_151 WHERE poll_id = '2348993' AND anonymous =0
ORDER BY add_time DESC LIMIT 0 , 3

*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: poll_joined_151
         type: ref
possible_keys: idx_poll_id_uid_add_time,idx_anonymous_id_addtime
          key: idx_anonymous_id_addtime
      key_len: 9
          ref: const,const
         rows: 30240
        Extra: Using where
```

从上面的例子看，虽然poll_id的类型为bigint，但是SQL中添加了''，但是这个语句仍然用到了索引，虽然扫描行数也不少，但是能用到索引就是好SQL。

那么一个小小的''为什么会有这么大的影响呢？根本原因是因为MySQL在对文本类型和数字类型进行比较的时候会进行隐式的类型转换。以下是5.5官方手册的说明：

```
If both arguments in a comparison operation are strings, they are
compared as strings.
```
两个参数都是字符串，会按照字符串来比较，不做类型转换。
```
If both arguments are integers, they are compared as integers.
```
两个参数都是整数，按照整数来比较，不做类型转换。
```
Hexadecimal values are treated as binary strings if not compared to a
```

number.
十六进制的值和非数字做比较时，会被当做二进制串。
If one of the arguments is a TIMESTAMP or DATETIME column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. Note that this is not done for the arguments to IN()! To be safe, always use complete datetime, date, or time strings when doing comparisons. For example, to achieve best results when using BETWEEN with date or time values, use CAST() to explicitly convert the values to the desired data type.
有一个参数是 TIMESTAMP 或 DATETIME，并且另外一个参数是常量，常量会被转换为 timestamp
If one of the arguments is a decimal value, comparison depends on the other argument. The arguments are compared as decimal values if the other argument is a decimal or integer value, or as floating-point values if the other argument is a floating-point value.
有一个参数是 decimal 类型，如果另外一个参数是 decimal 或者整数，会将整数转换为 decimal 后进行比较，如果另外一个参数是浮点数，则会把 decimal 转换为浮点数进行比较
In all other cases, the arguments are compared as floating-point (real) numbers.
所有其他情况下，两个参数都会被转换为浮点数再进行比较

根据以上的说明，当where条件之后的值的类型和表结构不一致的时候，MySQL会做隐式的类型转换，都将其转换为浮点数在比较。

对于第一种情况：
比如where string = 1;
需要将索引中的字符串转换成浮点数，但是由于'1',' 1','1a'都会比转化成1,故MySQL无法使用索引只能进行全表扫描，故造成了慢查询的产生。

```
mysql> SELECT CAST('  1' AS SIGNED)=1;
+-------------------------+
| CAST('  1' AS SIGNED)=1 |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT CAST('  1a' AS SIGNED)=1;
+--------------------------+
| CAST('  1a' AS SIGNED)=1 |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> SELECT CAST('1' AS SIGNED)=1;
+-----------------------+
| CAST('1' AS SIGNED)=1 |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

同时需要注意一点，由于都会转换成浮点数进行比较，而浮点数只有53bit，故当超过最大值的时候，比较会出现问题。

对于第二种情况：
由于索引建立在int的基础上，而将纯数字的字符串可以百分百转换成数字，故可以使用到索引，虽然也会进行一定的转换，消耗一定的资源，但是最终仍然使用了索引，不会产生慢查询。

```
mysql> select CAST( '30' as SIGNED) = 30;
+-----------------------------+
| CAST( '30' as SIGNED) = 30 |
+-----------------------------+
|                           1 |
+-----------------------------+
1 row in set (0.00 sec)
```

参考阅读：http://dev.mysql.com/doc/refman/5.5/en/type-conversion.html