1.使用explain语句去查看分析结果

如explain select * from test1 where id=1;
会出现：id selecttype table type possible_keys key key_len ref rows extra各列。

SHOW PROFILE Result (ordered by duration)

state	duration (summed) in sec	percentage
starting	0.00014	25.45455
checking privileges on cached	0.00010	18.18182
sending cached result to clien	0.00009	16.36364
checking query cache for query	0.00009	16.36364
logging slow query	0.00007	12.72727
cleaning up	0.00006	10.90909
Total	0.00055	100.00000

EXPLAIN Result

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	sso_user_acct	const	PRIMARY	PRIMARY	4	const	1	100.00	

EXPLAIN EXTENDED Information

Level: Note Code: 1003

Message

```
select
  '200017364'                          AS `uid`,
  '5ab2fbe598bc7e8e541e3e2207536fed' AS `pass`,
  '1359446499'                         AS `mtime`,
  '1359446499'                         AS `ctime`,
  '1'                                  AS `pass_level`
from `sso20`.`sso_user_acct`
where 1
```


其中，
type=const表示通过索引一次就找到了；
key=primary的话，表示使用了主键；
type=all,表示为全表扫描；
key=null表示没用到索引。
type=ref，因为这时认为是多个匹配行，在联合查询中，一般为REF。

2.MYSQL中的组合索引

假设表有id,key1,key2,key3,把三者形成一个组合索引，则如：

```
[sql] view plain copy
where key1=....
where key1=1 and key2=2
where key1=3 and key2=3 and key3=2
```

根据最左前缀原则，这些都是可以使用索引的，如from test where key1=1 order by key3，用explain分析的话，只用到了normal_key索引，但只对where子句起作用，而后面的order by需要排序。

3.使用慢查询分析（实用）

```
在my.ini中：
long_query_time=1
log-slow-queries=d:\mysql5\logs\mysqlslow.log
把超过1秒的记录在慢查询日志中
可以用mysqlsla来分析之。也可以在mysqlreport中，有如
DMS分别分析了select ,update,insert,delete,replace等所占的百分比
```

4.MYISAM和INNODB的锁定

```
myisam中，注意是表锁来的，比如在多个UPDATE操作后，再SELECT时，会发现SELECT操作被锁定了，
必须等所有UPDATE操作完毕后，再能SELECT
innodb的话则不同了，用的是行锁，不存在上面问题。
```

5.MYSQL的事务配置项

```
innodb_flush_log_at_trx_commit=1
表示事务提交时立即把事务日志flush写入磁盘，同时数据和索引也更新，很费性能。
innodb_flush_log_at_trx_commit=0
事务提交时，不立即把事务日志写入磁盘，每隔1秒写一次，MySQL挂了可能会丢失事务的数据。
innodb_flush_log_at_trx_commit=2 ，在整个操作系统 挂了时才可能丢数据，一般不会丢失超过
1-2秒的更新。
事务提交时，立即写入磁盘文件（这里只是写入到系统内核缓冲区，但不立即刷新到磁盘，而是每隔1
秒刷新到磁盘，同时更新数据和索引），
这种方案是不是性价比好一些，当然如何配置，决定于你对系统数据安全性的要求。
```

explain用法详解

```
EXPLAIN tbl_name或: EXPLAIN [EXTENDED] SELECT select_options
```

前者可以得出一个表的字段结构等等，后者主要是给出相关的一些索引信息，而今天要讲述的重点是后者。 举例

```
[sql] view plain copy
mysql> explain select * from event;
+--+-----+---+---+-----+---+---+---+---+---+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
Extra |
+--+-----+---+---+-----+---+---+---+---+---+
| 1 | SIMPLE | event | ALL | NULL | NULL | NULL | NULL | 13 | |
+--+-----+---+---+-----+---+---+---+---+---+
1 row in set (0.00 sec)
各个属性的含义
```

```
+----+-------------+-------+------+---------------+------+---------+------+------+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra |
+----+-------------+-------+------+---------------+------+---------+------+------+-------+
```

id :select查询的序列号

select_type:select查询的类型，主要是区别普通查询和联合查询、子查询之类的复杂查询。
a.SIMPLE：查询中不包含子查询或者UNION
b.查询中若包含任何复杂的子部分，最外层查询则被标记为：PRIMARY
c.在SELECT或WHERE列表中包含了子查询，该子查询被标记为：SUBQUERY
d.在FROM列表中包含的子查询被标记为：DERIVED（衍生）
e.若第二个SELECT出现在UNION之后，则被标记为UNION；若UNION包含在 FROM子句的子查询中，外层SELECT将被标记为：DERIVED
f.从UNION表获取结果的SELECT被标记为：UNION RESULT

table :输出的行所引用的表。

type :联合查询所使用的类型，表示MySQL在表中找到所需行的方式，又称“访问类型”。

```
+------+--------+-------+-----+--------+---------------+------+
| ALL  | index  | range | ref | eq_ref | const,system | NULL |
+------+--------+-------+-----+--------+---------------+------+
```

type显示的是访问类型，是较为重要的一个指标，结果值从好到坏依次是：
system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL ，
一般来说，得保证查询至少达到range级别，最好能达到ref。
ALL：扫描全表
index：扫描全部索引树
range：扫描部分索引，索引范围扫描，对索引的扫描开始于某一点，返回匹配值域的行，常见于between、<、>等的查询
ref：非唯一性索引扫描，返回匹配某个单独值的所有行。常见于使用非唯一索引即唯一索引的非唯一前缀进行的查找
eq_ref：唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描
const，system：当MySQL对查询某部分进行优化，并转换为一个常量时，使用这些类型访问。
如将主键置于where列表中，MySQL就能将该查询转换为一个常量。system是const类型的特例，当查询的表只有一行的情况下， 使用system。
NULL：MySQL在优化过程中分解语句，执行时甚至不用访问表或索引。
如下所示：

```
mysql> explain extended select * from t1 where id = (select min(id) from t2);
+----+-------------+-------+-------+---------------+---------+---------+-------+------+----------+--------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra  |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+----------+--------+
|  1 | PRIMARY     | t1    | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 |        |
|  2 | SUBQUERY    | NULL  | NULL  | NULL          | NULL    | NULL    | NULL  | NULL |     NULL |        |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+----------+--------+
2 rows in set, 1 warning (0.00 sec)

mysql> show warnings;
+-------+------+-----------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                 |
+-------+------+-----------------------------------------------------------------------------------------+
| Note  | 1003 | select '1' AS `ID`,'hu' AS `col1`,'dba' AS `col2` from `shared`.`t1` where 1 |
+-------+------+-----------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

possible_keys:指出MySQL能使用哪个索引在该表中找到行。查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用。
如果是空的，没有相关的索引。这时要提高性能，可通过检验WHERE子句，看是否引用某些字段，或者检查字段不是适合索引。

key :显示MySQL实际决定使用的键。如果没有索引被选择，键是NULL。

key_len:显示MySQL决定使用的键长度。表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度。
如果键是NULL，长度就是NULL。文档提示特别注意这个值可以得出一个多重主键里mysql实际使用了哪一部分。

注：key_len显示的值为索引字段的最大可能长度，并非实际使用长度，即key_len是根据表定义计算而得，不是通过表内检索出的。
ref:显示哪个字段或常数与key一起被使用。

rows:这个数表示mysql要遍历多少数据才能找到，表示MySQL根据表统计信息及索引选用情况，估算的找到所需的记录所需要读取的行数，在innodb上可能是不准确的。

Extra:包含不适合在其他列中显示但十分重要的额外信息。
Only index，这意味着信息只用索引树中的信息检索出的，这比扫描整个表要快。

using where是使用上了where限制，表示MySQL服务器在存储引擎受到记录后进行“后过滤”（Post-filter），如果查询未能使用索引，
Using where的作用只是提醒我们MySQL将用where子句来过滤结果集。

impossible where 表示用不着where，一般就是没查出来啥。

Using filesort（MySQL中无法利用索引完成的排序操作称为“文件排序”）当我们试图对一个没有索引的字段进行排序时，就是filesoft。它跟文件没有任何关系，实际上是内部的一个快速排序。

Using temporary（表示MySQL需要使用临时表来存储结果集，常见于排序和分组查询），使用filesort和temporary的话会很吃力，WHERE和ORDER BY的索引经常无法兼顾，
如果按照WHERE来确定索引，那么在ORDER BY时，就必然会引起Using filesort，这就要看是先过滤再排序划算，还是先排序再过滤划算。

- 最后，再看一个查询计划的例子：

```
mysql> explain select d1.name, (select id from t3) d2
    -> from (select id, name from t1 where other_column = '') d1
    -> union
    -> (select name, id from t2);
+------+--------------+------------+--------+---------------+---------+---------+------+------+-------------+
| id   | select_type  | table      | type   | possible_keys | key     | key_len | ref  | rows | Extra       |
+------+--------------+------------+--------+---------------+---------+---------+------+------+-------------+
|  1   | PRIMARY      | <derived3> | system | NULL          | NULL    | NULL    | NULL |    1 |             |
|  3   | DERIVED      | t1         | ALL    | NULL          | NULL    | NULL    | NULL |    1 | Using where |
|  2   | SUBQUERY     | t3         | index  | NULL          | PRIMARY | 4       | NULL |    1 | Using index |
|  4   | UNION        | t2         | ALL    | NULL          | NULL    | NULL    | NULL |    1 |             |
| NULL | UNION RESULT | <union1,4> | ALL    | NULL          | NULL    | NULL    | NULL | NULL |             |
+------+--------------+------------+--------+---------------+---------+---------+------+------+-------------+
5 rows in set (0.01 sec)
```

> 第一行：id列为1，表示第一个select，select_type列的primary表示该查询为外层查询，table列被标记为，表示查询结果来自一个衍生表，其中3代表该查询衍生自第三个select查询，即id为3的select。[select d1.name.....]

> 第二行：id为3，表示该查询的执行次序为2（4→3），是整个查询中第三个select的一部分。因查询包含在from中，所以为derived。[select id,name from t1 where other_column='']

> 第三行：select列表中的子查询，select_type为subquery，为整个查询中的第二个select。[select id from t3]

> 第四行：select_type为union，说明第四个select是union里的第二个select，最先执行。[select name,id from t2]

> 第五行：代表从union的临时表中读取行的阶段，table列的表示用第一个和第四个select的结果进行union操作。[两个结果union操作]

关于MySQL执行计划的局限性：

```
EXPLAIN不会告诉你关于触发器、存储过程的信息或用户自定义函数对查询的影响情况
EXPLAIN不考虑各种Cache
EXPLAIN不能显示MySQL在执行查询时所作的优化工作
部分统计信息是估算的，并非精确值
EXPALIN只能解释SELECT操作，其他操作要重写为SELECT后查看。
```

备注：

```
filesort是通过相应的排序算法,将取得的数据在内存中进行排序。
```

```
MySQL需要将数据在内存中进行排序，所使用的内存区域也就是我们通过sort_buffer_size 系统变量所设置的排序区。这个排序区是每个Thread 独享的，所以说可能在同一时刻在MySQL 中可能存在多个sort buffer 内存区域。
```

```
在MySQL中filesort 的实现算法实际上是有两种：
双路排序：是首先根据相应的条件取出相应的排序字段和可以直接定位行数据的行指针信息，然后在sort buffer 中进行排序。
单路排序：是一次性取出满足条件行的所有字段，然后在sort buffer中进行排序。
```

> 在MySQL4.1版本之前只有第一种排序算法双路排序，第二种算法是从MySQL4.1开始的改进算法，主要目的是为了减少第一次算法中需要两次访问表数据的IO操作，将两次变成了一次，但相应也会耗用更多的sortbuffer 空间。当然，MySQL4.1开始的以后所有版本同时也支持第一种算法。
>
> MySQL主要通过比较我们所设定的系统参数 max_length_for_sort_data的大小和Query 语句所取出的字段类型大小总和来判定需要使用哪一种排序算法。
>
> 如果 max_length_for_sort_data更大，则使用第二种优化后的算法，反之使用第一种算法。所以如果希望 ORDER BY 操作的效率尽可能的高，一定要注意max_length_for_sort_data 参数的

> 设置。如果filesort过程中，由于排序缓存的大小不够大，那么就可能会导致临时表的使用。==max_length_for_sort_data的默认值是1024。==