Benwin 's blog PHP|APACHE|MYSQL|别以为你很厉害，大牛多的是!

on 2012-6-1 14:51 Friday by Benwin 分类: mysql

由浅入深探究mysql索引结构原理、性能分析与优化

摘要:

(2) Fds optimize、Analyze、check、repair维护操作

(3) 表结构的更新与维护

第四部分：图说mysql查询执行流程

第一部分：基础知识：

索引

官方介绍索引是帮助MySQL高效获取数据的数据结构。笔者理解索引相当于一本书的目录，通过目录就知道要的资料在哪里，不用一页一页查阅找出需要的资料。关键字index

---

唯一索引

强调唯一，就是索引值必须唯一，关键字unique index

创建索引：

1、create **unique index** 索引名 on 表名(列名);

2、alter table 表名 add **unique index** 索引名 (列名);

删除索引：

1、 drop index 索引名 on 表名;

2、 alter table 表名 drop index 索引名;

主键

主键就是唯一索引的一种，主键要求建表时指定，一般用auto_increatment列，关键字是primary key

主键创建：

creat table test2 (id int not null **primary key** auto_increment);

全文索引

InnoDB不支持，Myisam支持性能比较好，一般在 CHAR、VARCHAR 或 TEXT 列上创建。

Create table 表名(id int not null primary anto_increment,title varchar(100),**FULLTEXT**(title))type=**myisam**

单列索引与多列索引

索引可以是单列索引也可以是多列索引（也叫复合索引）。按照上面形式创建出来的索引是单列索引，现在先看看创建多列索引：

create table test3 (id int not null primary key auto_increment,uname char (8) not null default '',password char(12) not null,**INDEX**(uname,password))type =myisam;

注意：INDEX(a, b, c)可以当做a或(a, b)的索引来使用，但和b、c或(b,c)的索引来使用这是一

个**最左前缀**的优化方法，在后面会有详细的介绍，你只要知道有这样两个概念

聚集索引

一种索引，该索引中键值的逻辑顺序决定了表中相应行的物理顺序。 聚集索引确定表中数据的物理顺序。Mysql中myisam表是没有聚集索引的，innodb有（主键就是聚集索引），聚集索引在下面介绍innodb结构的时有详细介绍。

查看表的索引

通过命令：Show index from 表名

如：

```
mysql> show index from test3;
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Card
Packed | Null | Index_type | Comment |
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+----+
| test3 |          0 | PRIMARY  |            1 |    id       |     A     |   0
         |     NULL |
NULL   |      | BTREE      |         |
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+-------+
```

Table：表名

Key_name：什么类型索引（这了是主键）

Column_name：索引列的字段名

Cardinality：索引基数，很关键的一个参数，平均数值组=索引基数/表总数据行，平均数值组越接近1就越有可能利用索引

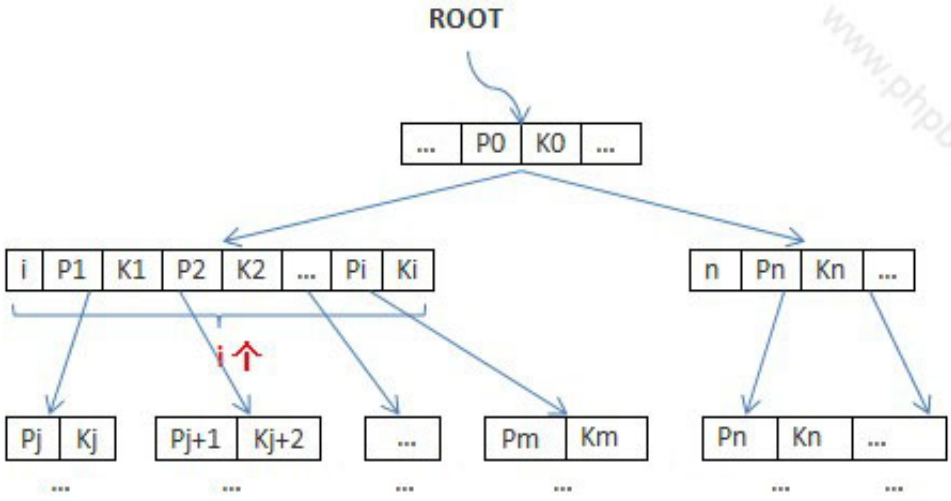
Index_type：如果索引是全文索引，则是fulltext,这里是b+tree索引，b+tre也是这篇文章研究的重点之一

其他的就不详细介绍，更多：

第二部分：MYISAM和INNODB索引结构

1、简单介绍B-tree B+ tree树

B-tree结构视图



一棵m阶的B-tree树，则有以下性质

（1）Ki表示关键字值，上图中，k1<k2<...<ki<k0<Kn(可以看出，一个节点的左子节点关键字值<该关键字值<右子节点关键字值)

（2）Pi表示指向子节点的指针，左指针指向左子节点，右指针指向右子节点。即是：p1[指向值]<k1<p2[指向值]<k2.....

（3）所有关键字必须唯一值（这也是创建myisam 和innodb表必须要主键的原因），每个节点包含一个说明该节点多少个关键字，如上图第二行的i和n

（4）节点：

- 每个节点最可以有m个子节点。
- 根节点若非叶子节点，至少2个子节点，最多m个子节点
- 每个非根，非叶子节点至少[m/2]子节点或叫子树（[]表示向上取整），最多m个子节点

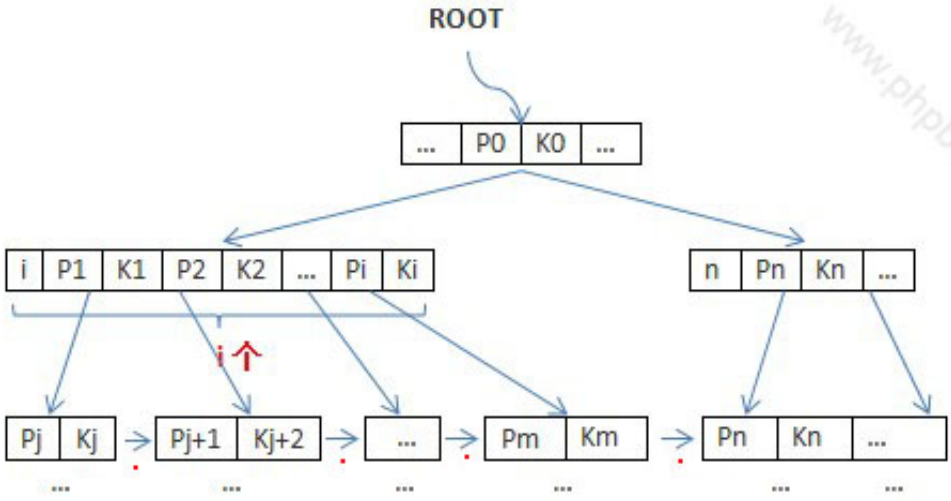
（5）关键字：

- 根节点的关键字个数1~m-1
- 非根非叶子节点的关键字个数[m/2]-1~m-1,如m=3，则该类节点关键字个数：2-1~2

（6）关键字数k和指向子节点个数指针p的关系：

- k+1=p ，注意根据储存数据的具体需求，左右指针为空时要有标志位表示没有

B+tree结构示意图如下：



B+树是B-树的变体，也是一种多路搜索树：

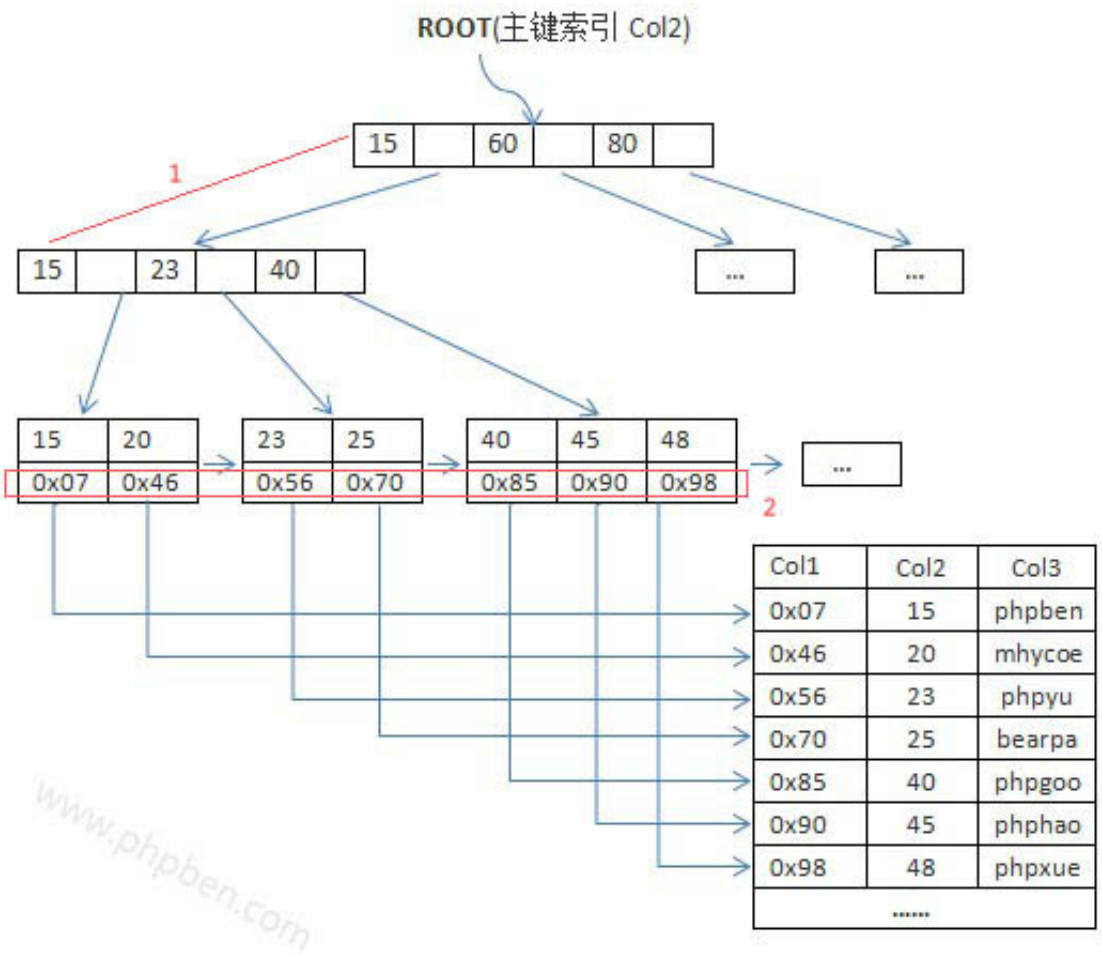
- 非叶子结点的子树指针与关键字个数相同
- 为所有叶子结点增加一个链指针（红点标志的箭头）

B+树是B-树的变体，也是一种多路搜索树：

- 非叶子结点的子树指针与关键字个数相同
- 为所有叶子结点增加一个链指针（红点标志的箭头）

2、 MyisAM索引结构

MyisAM索引用的B+tree来储存数据，MyisAM索引的指针指向的是键值的地址，地址存储的是数据，如下图：



（1）结构讲解：上图3阶树，主键是Col2，Col值就是改行数据保存的物理地址，其中红色部分是说明标注。

l 1标注部分也许会迷惑，前面不是说关键字15右指针的指向键值要大于15，怎么下面还有15关键字？因为B+tree的所以叶子节点包含所有关键字且是按照升序排列（主键索引唯一，辅助索引可以不唯一），所以等于关键字的数据值在右子树

l 2标注是相应关键字存储对应数据的物理地址，注意这也是之后和InnoDB索引不同的地方之一

l 2标注也是一个所说MyiAM表的索引和数据是分离的，索引保存在”表名.MYI”文件内，而数据保存在“表名.MYD”文件内，2标注的物理地址就是“表名.MYD”文件内相应数据的物理地址。（InnoDB表的索引文件和数据文件在一起）

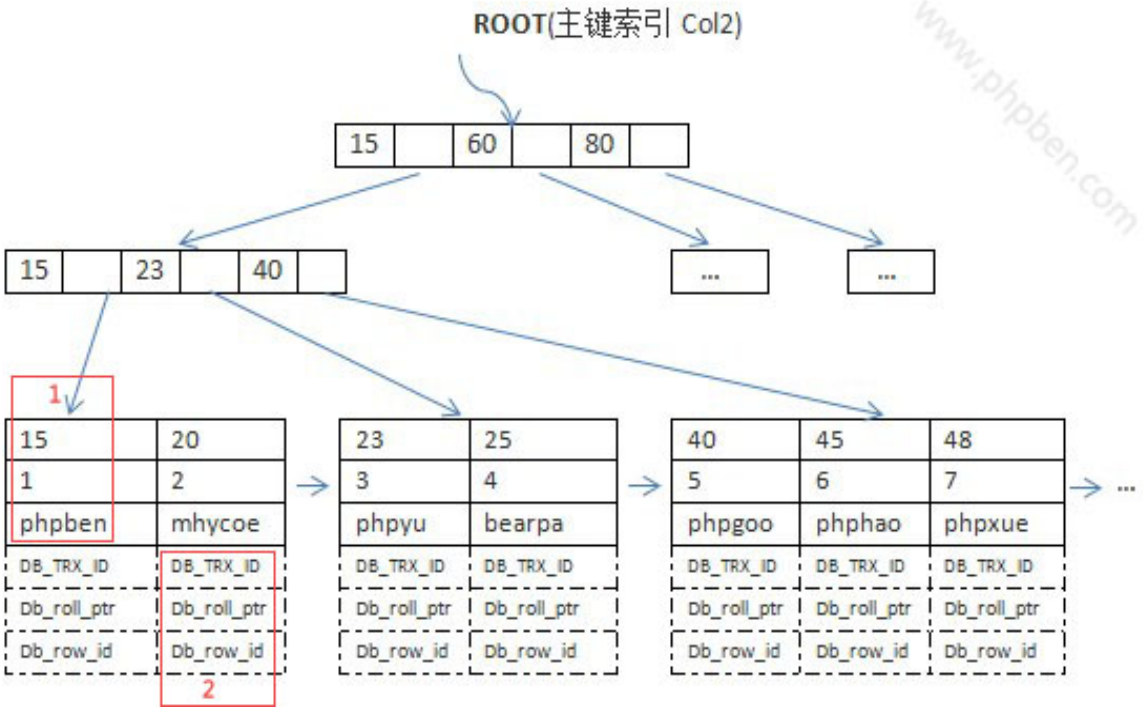
l 辅助索引和主键索引没什么大的区别，辅助索引的索引值是可以重复的（但InnoDB辅助索引和主键索引有很明显的区别，这里先提醒注意一下）

3、 Annode索引结构

（1）首先有一个表，内容和主键索引结构如下两图：

Col1	Col2	Col3
1	15	phpben
2	20	mhycoe
3	23	phpyu
4	25	bearpa

5	40	phpgoo
6	45	phphao
7	48	phpxue
.....		

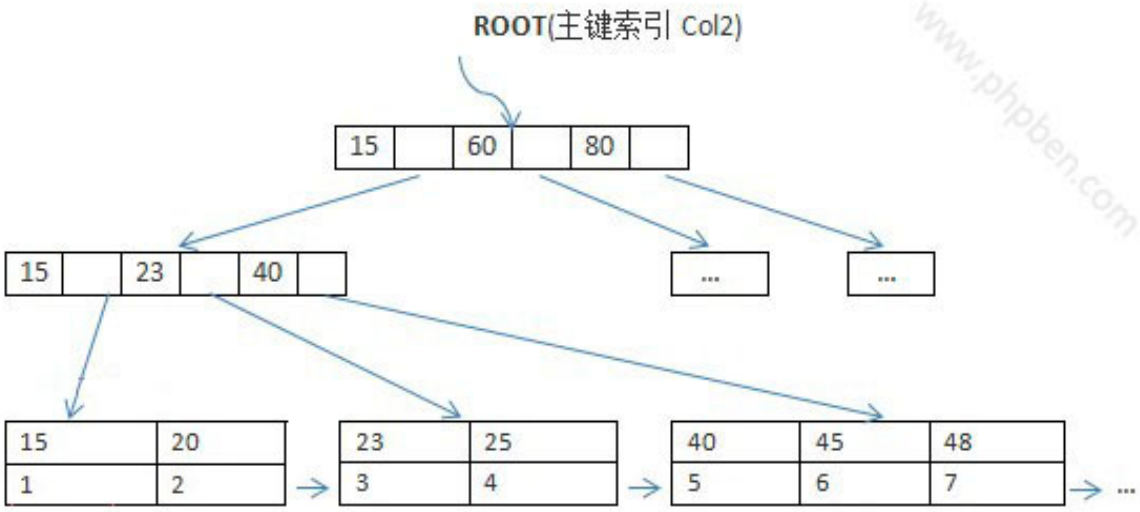


结构上：由上图可以看出InnoDB的索引结构很MyisAM的有很明显的区别

1 MyisAM表的索引和数据是分开的，用指针指向数据的物理地址，而InnoDB表中索引和数据是储存在一起。看红框1可一看出一行数据都保存了。

1 还有一个上图多了三行的隐藏数据列（虚线表），这是因为MyisAM不支持事务，InnoDB处理事务在性能上并发控制上比较好，看图中的红框2中的DB_TRX_ID是事务ID，自动增长；db_roll_ptr是回滚指针，用于事务出错时数据回滚恢复；db_row_id是记录行号，这个值其实在主键索引中就是主键值，这里标出重复是为了容易介绍，还有的是若不是主键索引（辅助索引），db_row_id会找表中unique的列作为值，若没有unique列则系统自动创建一个。关于InnoDB跟多事务MVCC点此：http://www.phpben.com/?post=72

（2）加入上表中Col1是主键（下图**标错**），而Col2是辅助索引，则相应的辅助索引结构图：



可以看出InnoDB辅助索引并没有保存相应的所有列数据，而是保存了主键的键值（图中1、2、3....）这样做利弊也是很明显：

- 在已有主键索引，避免数据冗余，同时在修改数据的时候只需修改辅助索引值。
- 但辅助索引查找数据事要检索两次，先找到相应的主键索引值然后在去检索主键索引找到对应的数据。这也是网上很多mysql性能优化时提到的“主键尽可能简短”的原因，主键越长辅助索引也就越大，当然主键索引也越大。

4、 MyisAM索引与InnoDB索引相比较

- MyisAM支持全文索引（FULLTEXT）、压缩索引，InnoDB不支持
- AnnoDB支持事务，MyisAM不支持
- MyisAM顺序储存数据，索引叶子节点保存对应数据行地址，辅助索引很主键索引相差无几；AnnoDB主键节点同时保存数据行，其他辅助索引保存的是主键索引的值
- MyisAM键值分离，索引载入内存（key_buffer_size）,数据缓存依赖操作系统；InnoDB键值一起保存，索引与数据一起载入InnoDB缓冲池
- MyisAM主键（唯一）索引按升序来存储存储，InnoDB则不一定
- MyisAM索引的基数值（Cardinality，show index 命令可以看见）是精确的，InnoDB则是估计值。这里涉及到信息统计的知识，MyisAM统计信息是保存磁盘中，在alter表或Analyze table操作更新此信息，而InnoDB则是在表第一次打开的时候估计值保存在缓存区内
- MyisAM处理字符串索引时用增量保存的方式，如第一个索引是‘preform’，第二个是‘preformence’，则第二个保存是‘7，ance‘，这个明显的好处是缩短索引，但是缺陷就是不支持倒序提取索引，必须顺序遍历获取索引

第三部分：MYSQL优化

mysql优化是一个重大课题之一，这里会重点详细的介绍mysql优化，包括表数据类型选择，sql语句优化，系统配置与维护优化三类。

1、 表数据类型选择

（1）能小就用小。表数据类型第一个原则是：使用能正确的表示和存储数据的最短类型。这样可以减少对磁盘空间、内存、cpu缓存的使用。

（2）避免用NULL，这个也是网上优化技术博文传的最多的一个。理由是额外增加字节，还有使索引，索引统计和值更复杂。很多还忽略一个count(列)的问题，count(列)是不会统计列值为null的行数。更多关于NULL可参考：http://www.phpben.com/?post=71

（3）字符串如何选择char和varchar？一般phper能想到就是char是固定大小，varchar能动态储存数据。这里整理一下这两者的区别：

属性	Char	Varchar
值域大小	最长**字符数**是255（不是字节），不管什么编码，超过此值则自动截取255个字符保存并没有报错。	65535个字节，开始两位存储长度，超过255个字符，用2位储存长度，否则1位，具体字符长度根据编码来确定，如utf8则字符最长是21845个
如何处理字符串末尾空格	去掉末尾空格，取值出来比较的时候自动加上进行比较	Version<=4.1，字符串末尾空格被删掉，version>5.0则保留
储存空间	固定空间，比喻char(10)不管字符串是否有10个字符都分配10个字符的空间	Varchar内节约空间，但更新可能发生变化，若varchar(10),开始若储存5个字符，当update成7个时有myisam可能把行拆开，innodb可能分页，这样开销就增大
适用场合	适用于存储很短或固定或长度相似字符，如MD5加密的密码char(33)、昵称char(8)等	当最大长度远大于平均长度并且发生更新的时候。

注意当一些英文或数据的时候，最好用每个字符用字节少的类型，如latin1

（4）整型、整形优先原则

Tinyint、smallint、mediumint、int、bigint，分别需要8、16、24、32、64。

值域范围：-2^(n-1)~ 2^(n-1)-1

很多程序员在设计数据表的时候很习惯的用int，压根不考虑这个问题

笔者建议：能用tinyint的绝不用smallint

误区：int(1) 和int(11)是一样的，唯一区别是mysql客户端显示的时候显示多少位。

整形优先原则：能用整形的不用其他类型替换，如ip可以转换成整形保存，如商品价格‘50.00元’则保存成50

（5）精确度与空间的转换。在存储相同数值范围的数据时，浮点数类型通常都会比DECIMAL类型使用更少的空间。FLOAT字段使用4字节存储

数据。DOUBLE类型需要8 个字节并拥有更高的精确度和更大的数值范围，DECIMAL类型的数据将会转换成DOUBLE类型。

2、 sql语句优化

```
mysql> create table one (
id smallint(10) not null auto_increment primary key,
username char(8) not null,
password char(4) not null,
`level` tinyint (1) default 0,
last_login char(15) not null,
index(username,password,last_login))engine=innodb;
```

这是test表，其中id是主键，多列索引(username,password,last_login),里面有10000多条数据.

```
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Card
 Index_type | Comment |
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+------+
| one   |          0 | PRIMARY  |            1 | id          | A         |20242 |
 BTREE     |          |
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+------+
| one   |          1 | username |            1 | username    | A         |10121 |
BTREE     |          |
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+------+
| one   |          1 | username |            2 | password    | A         |10121 |
 BTREE     |          |
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+------+
| one   |          1 | username |              3 | last_login  | A         |20242
 BTREE     |          |
+-------+------------+----------+--------------+-------------+-----------+----
---------+----------+--------+------+
```

(1) 最左前缀原则

定义：最左前缀原则指的的是在sql where 字句中一些条件或表达式中出现的列的顺序要保持和多索引的一致或以多列索引顺序出现，只要出现非顺序出现、断层都无法利用到多列索引。

举例说明：上面给出一个多列索引(username,password,last_login)，当三列在where中出现的顺序如(username,password,last_login)、(username,password)、(username)才能用到索引，如下面几个顺序(password,last_login)、(passwrod)、(last_login)---这三者不从username开始，(username,last_login)---断层，少了password，都无法利用到索引。

因为B+tree多列索引保存的顺序是按照索引创建的顺序，检索索引时按照此顺序检索

测试：以下测试不精确，这里只是说明如何才能正确按照最左前缀原则使用索引。还有的是以下的测试用的时间0.00sec看不出什么时间区别，因为数据量只有20003条，加上没有在实体机上运行，很多未可预知的影响因素都没考虑进去。当在大数据量，高并发的时候，最左

前缀原则对与提高性能方面是不可否认的。

Ps：最左前缀原则中where字句有or出现还是会遍历全表

(1.1)能正确的利用索引

l Where子句表达式 顺序是(username)

```
mysql> explain select * from one where username='abgvwfnt';
+----+-------------+-------+------+---------------+----------+---------+------
-+------+-------------+
| id | select_type | table | type | possible_keys | key      | key_len | ref

+----+-------------+-------+------+---------------+----------+---------+------
-+------+-------------+
|  1 | SIMPLE      | one   | ref  | username      | username | 24      | const

+----+-------------+-------+------+---------------+----------+---------+------
-+------+-------------+
1 row in set (0.00 sec)
```

l Where子句表达式 顺序是(username,password)

```
mysql> explain select * from one where username='abgvwfnt' and password='123456
+----+-------------+-------+------+---------------+----------+---------+------
-------+------+-------------+
| id | select_type | table | type | possible_keys | key      | key_len | ref | 
+----+-------------+-------+------+---------------+----------+---------+------
-------+------+-------------+
|  1 | SIMPLE      | one   | ref  | username      | username | 43      | const,
+----+-------------+-------+------+---------------+----------+---------+------
-------+------+-------------+
1 row in set (0.00 sec)
```

l Where子句表达式 顺序是(username,password, last_login)

```
mysql> explain select * from one where username='abgvwfnt' and password='123456
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-------------+
| id | select_type | table | type | possible_keys | key      | key_len | ref| r
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-------------+
|  1 | SIMPLE    | one   | ref  | username      | username | 83      | const,cons
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-------------+
1 row in set (0.00 sec)
```

上面可以看出type=ref 是多列索引，key_len分别是24、43、83，这说明用到的索引分别是(username), (username,password), (username,password, last_login);row分别是5、1、1检索的数据行都很少，因为这三个查询都按照索引前缀原则，可以利用到索引。

(1.2)不能正确的利用索引

l Where子句表达式 顺序是(password, last_login)

```
mysql> explain select * from one where password='123456'and last_login='1338251
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
```

```
| id | select_type | table | type | possible_keys | key  | key_len | ref  | row
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
|  1 | SIMPLE      | one   | ALL  | NULL          | NULL | NULL    | NULL | 201
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
1 row in set (0.00 sec)
```

l Where 子句表达式顺序是(last_login)

```
mysql> explain select * from one where last_login='1338252525';
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | row
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
|  1 | SIMPLE      | one   | ALL  | NULL          | NULL | NULL    | NULL | 201
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
1 row in set (0.00 sec)
```

以上的两条语句都不是以username开始，这样是用不了索引，通过type=all（全表扫描），key_len=null，rows都很大20146

Ps：one表里只有20003条数据，为什么出现20146，这是优化器对表的一个估算值，不精确的。

l Where 子句表达式虽然顺序是(username,password, last_login)或(username,password)但第一个是有范围’<’、’>’，’<=’，’>=’等出现

```
mysql> explain select * from one where username>'abgvwfnt' and password ='12345
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | row
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
|  1 | SIMPLE      | one   | ALL  | username      | NULL | NULL    | NULL | 201
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
1 row in set (0.00 sec)
```

这个查询很明显是遍历所有表，一个索引都没用到，非第一列出现范围（password列或last_login列），则能利用索引到首先出现范围的一列，也就是“where username='abgvwfnt' and password >'123456'and last_login='1338251170';”或则“where username='abgvwfnt' and password >'123456'and last_login<'1338251170';” 索引长度ref_len=43, 索引检索到password列，所以考虑多列索引的时候把那些查询语句用的比较的列放在最后（或非第一位）。

l 断层，即是where顺序(username, last_login)

```
mysql> explain select * from one where username='abgvwfnt' and last_login='1338
+----+-------------+-------+------+---------------+---------+---------+------
-+------+-------------+
| id | select_type | table | type | possible_keys | key     | key_len | ref
+----+-------------+-------+------+---------------+---------+---------+------
```

```
-+------+-------------+
|  1 | SIMPLE    | one    | ref  | username    | username | 24      | const |5 | Us
+----+-------------+-------+------+---------------+----------+---------+------
-+------+-------------+
1 row in set (0.00 sec)
```

注意这里的key_len=24=8*3(8是username的长度，3是utf8编码)，rows=5，和下面一条sql语句搜索出来一样

```
mysql>  select * from one where username='abgvwfnt';
+-------+----------+----------+-------+------------+
| id    | username | password | level | last_login |
+-------+----------+----------+-------+------------+
|  3597 | abgvwfnt | 234567   |     0 | 1338251420 |
|  7693 | abgvwfnt | 456789   |     0 | 1338251717 |
| 11789 | abgvwfnt | 456789   |     0 | 1338251992 |
| 15885 | abgvwfnt | 456789   |     0 | 1338252258 |
| 19981 | abgvwfnt | 456789   |     0 | 1338252525 |
+-------+----------+----------+-------+------------+
5 rows in set (0.00 sec)

mysql>  select * from one where username='abgvwfnt' and last_login='1338252525'
+-------+----------+----------+-------+------------+
| id    | username | password | level | last_login |
+-------+----------+----------+-------+------------+
| 19981 | abgvwfnt | 456789   |     0 | 1338252525 |
+-------+----------+----------+-------+------------+
1 row in set (0.00 sec)
```

这个就是要的返回结果，所以可以知道断层(username,last_login)，这样只用到username索引，把用到索引的数据再重新检查last_login条件，这个相对全表查询来说还是有性能上优化，这也是很多sql优化文章中提到的where 范围查询要放在最后（这不绝对，但可以利用一部分索引）

(1.3)如果一个查询where子句中确实不需要password列，那就用“补洞”。

```
mysql> select distinct(password) from one;
+----------+
| password |
+----------+
| 234567   |
| 345678   |
| 456789   |
| 123456   |
+----------+
4 rows in set (0.08 sec)
```

可以看出password列中只有这几个值，当然在现实中不可能密码有这么多一样的，再说数据也可能不断更新，这里只是举例说明补洞的方法

```
mysql> explain select * from one where username='abgvwfnt' and password in('123
and last_login='1338251170';
+----+-------------+-------+-------+---------------+----------+---------+-----
-+------+-------------+
| id | select_type | table | type  | possible_keys | key      | key_len | ref
+----+-------------+-------+-------+---------------+----------+---------+-----
-+------+-------------+
|  1 | SIMPLE    | one | range | username    | username| 83      | NULL |4 | Us
```

```
+----+-------------+-------+-------+--------------+---------+---------+-----
-+------+-------------+
1 row in set (0.00 sec)
```

可以看出ref=83 所有的索引都用到了，type=range是因为用了in子句。

这个被“补洞”列中的值应该是有限的，可预知的，如性别，其值只有男和女（加多一个不男不女也无妨）。

“补洞”方法也有瓶颈，当很多列，且需要补洞的相应列（可以多列）的值虽有限但很多（如中国城市）的时候，优化器在优化时组合起来的数量是很大，这样的话就要做好基准测试和性能分析，权衡得失，取得一个合理的优化方法。

(1.4)like

```
mysql> explain select * from one where username like 'abgvwfnt%';
+----+-------------+-------+-------+--------------+----------+---------+-----
-+------+-------------+
| id | select_type | table | type  | possible_keys | key      | key_len | ref
 rows | Extra       |
+----+-------------+-------+-------+--------------+----------+---------+-----
-+------+-------------+
|  1 | SIMPLE      | one   | range | username      | username | 24      | NULL
    5 | Using where |
+----+-------------+-------+-------+--------------+----------+---------+-----
-+------+-------------+
1 row in set (0.00 sec)
mysql> explain select * from one where username like '%abgvwfnt%';
+----+-------------+-------+------+--------------+------+---------+------+---
----+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | row
+----+-------------+-------+------+--------------+------+---------+------+---
----+-------------+
|  1 | SIMPLE      | one   | ALL  | NULL          | NULL | NULL    | NULL | 202
+----+-------------+-------+------+--------------+------+---------+------+---
----+-------------+
1 row in set (0.01 sec)
```

对比就知道like操作abgvwfnt%能用到索引，%abgvwfnt%用不到

---

(2) Order by 优化

(2.1)filesort优化算法.

在mysql version()<4.1之前，优化器采用的是filesort第一种优化算法，先提取键值和指针，排序后再去提取数据，前后要搜索数据两次，第一次若能使用索引则使用，第二次是随机读（当然不同引擎也不同）。mysql version()>=4.1,更新了一个新算法，就是在第一次读的时候也把selcet的列也读出来，然后在sort_buffer_size中排序（不够大则建临时表保存排序顺序），这算法只需要一次读取数据。所以有这个广为人传的一个优化方法，那就是增大sort_buffer_size。Filesort第二种算法要用到更的空间，sort_buffer_size不够大反而会影响速度，所以mysql开发团队定了个变量max_length_for_sort_data，当算法中读出来的需要列的数据的大小超过该变量的值才使用，所以一般性能分析的时候会尝试把max_length_for_sort_data改小。

(2.2)单独order by 用不了索引，索引考虑加where 或加limit

先建一个索引(last_login),建的过程就不给出了

```
mysql> explain select * from one order by last_login desc;
+----+-------------+-------+------+---------------+------+---------+------+---
----+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | row
  | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+---
----+----------------+
|  1 | SIMPLE      | one   | ALL  | NULL          | NULL | NULL    | NULL | 204
3 | Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+---
----+----------------+
1 row in set (0.00 sec)

mysql> explain select * from one order by last_login desc limit 10;
+----+-------------+-------+-------+---------------+-----------+---------+---
---+------+-------+
| id | select_type | table | type  | possible_keys | key       | key_len | ref
 | rows | Extra |
+----+-------------+-------+-------+---------------+-----------+---------+---
---+------+-------+
|  1 | SIMPLE      | one   | index | NULL          | last_login | 4       | NULL
 |   10 |       |
+----+-------------+-------+-------+---------------+-----------+---------+---
---+------+-------+
1 row in set (0.00 sec)
```

开始没limit查询是遍历表的，加了limit后，索引可以使用，看key_len 和key

(2.3)where + orerby 类型，where满足最左前缀原则，且orderby的列和where子句用到的索引的列的子集。即是(a,b,c)索引，where满足最左前缀原则且order by中列a、b、c的任意组合

```
mysql> explain select * from one where username='abgvwfnt' and password ='12345
' and last_login='1338251001' order by password desc,last_login desc;

+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-------------+
| id | select_type | table | type | possible_keys | key      | key_len | ref

           | rows | Extra       |
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-------------+
|  1 | SIMPLE      | one   | ref  | username      | username | 83      | const,
onst,const |    1 | Using where |
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-------------+
1 row in set (0.00 sec)

mysql> explain select * from one where username='abgvwfnt' and password ='12345
' and last_login='1338251001' order by password desc,level desc;
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-----------------------------+
| id | select_type | table | type | possible_keys | key      | key_len | ref| r
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-----------------------------+
|  1 | SIMPLE      | one   | ref  | username      | username | 83      | const,
onst,const |    1 | Using where; Using filesort |
+----+-------------+-------+------+---------------+----------+---------+------
-------------+------+-----------------------------+

1 row in set (0.00 sec)
```

上面两条语句明显的区别是多了一个非索引列level的排序，在extra这列对了Using filesort

笔者测试结果：where满足最左前缀且order by中的列是该多列索引的子集时（也就是说orerby中没最左前缀原则限制），不管是否有asc ,desc混合出现，都能用索引来满足order by。

笔者测试过，因为篇幅比较大，这里就不一一列出。

Ps:很优化博文都说order by中的列要where中出现的列（是索引）的顺序一致，笔者认为不够严谨。

(2.3) where + orerby+limit

这个其实也差不多，只要where最左前缀，orderby也正确，limit在此影响不大

(2.4)如何考虑order by来建索引

这个回归到创建索引的问题来，在比较常用的oder by的列和where中常用的列建立多列索引，这样优化起来的广度和扩张性都比较好，当然如果要考虑UNION、JOIN、COUNT、IN等进来就复杂很多了

(3) 隔离列

隔离列是只查询语句中把索引列隔离出来，也就是说不能在语句中把列包含进表达式中，如id+1=2、inet_aton('210.38.196.138')---ip转换成整数、convert(123,char(3))---数字转换成字符串、date函数等mysql内置的大多函数。

非隔离列影响性能很大甚至是致命的，这也就是赶集网石展的《三十六军规》中的一条，虽然他没说明是隔离列。

以下就测试一下：

首先建立一个索引(last_login)，这里就不给出建立的代码了，且把last_login改成整型（这里只是为了方便测试，并不是影响条件）

```
01. mysql> explain select * from one where last_login = 8388605;
02. +----+-------------+-------+------+---------------+------------+---------+----
    ---+-------+-------------+
03. | id | select_type | table | type | possible_keys | key        | key_len | ref
04. +----+-------------+-------+------+---------------+------------+---------+----
    ---+-------+-------------+
05. |  1 | SIMPLE      | one   | ref  | last_login    | last_login | 3       | cons
06.  | 1 | Using where |
07. +----+-------------+-------+------+---------------+------------+---------+----
    ---+-------+-------------+
08. 1 row in set, 1 warning (0.00 sec)
```

容易看出建的索引已起效

```
01. mysql> explain select * from one where last_login +1= 8388606 ;
02. +----+-------------+-------+------+---------------+------+---------+------+---
    ----+-------------+
03. | id | select_type | table | type | possible_keys | key  | key_len | ref  | row
04.   | Extra       |
05. +----+-------------+-------+------+---------------+------+---------+------+---
    ----+-------------+
06. |  1 | SIMPLE      | one   | ALL  | NULL          | NULL | NULL    | NULL | 204
07. 7 | Using where |
08. +----+-------------+-------+------+---------------+------+---------+------+---
    ----+-------------+
09. 1 row in set (0.00 sec)
```

last_login +1=8388608非隔离列的出现导致查找的列20197，说明是遍历整张表且索引不能使用。

这是因为这条语句要找出所有last_login的数据，然后+1再和20197比较，优化器在这方面比较差，性能很差。

所以要尽可能的把列隔离出来，如last_login +1= 8388606改成login_login=8388607,或者把计算、转换等操作先用php函数处理过再传递给mysql服务器

(4) OR、IN、UNION ALL，可以尝试用UNION ALL

(4.1)or会遍历表就算有索引

```
mysql> explain select * from one where username = 'abgvwfnt' or password='12345
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | row
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
|  1 | SIMPLE      | one   | ALL  | username      | NULL | NULL    | NULL | 20259 | Us
+----+-------------+-------+------+---------------+------+---------+------+---
----+-------------+
1 row in set (0.00 sec)
```

(4.2)对于in，这个是有争议的，网上很多优化方案中都提到尽量少用in，这不全面，其实在in里面如果是常量的话，可一大胆的用in，这个也是赶集网石展、阿里hellodab的观点（笔者从微博中获知）。应用hellodab一句话“MySQL用IN效率不好，通常是指in中嵌套一个子查询，因为MySQL的查询重写可能会产生一个不好的执行计划，而如果in里面是常量的话，我认为性能没有任何问题，可以放心使用”---------当然对于这个比较的话，没有实战数据的话很难辩解，就算有，影响性能的因素也很多，也许会每个dba都有不同的测试结果.这也签名最左前缀中“补洞”一个方法

(4.3)UNION All 直接返回并集，可以避免去重的开销。之所说“尝试”用UNION All 替代 OR来优化sql语句，因为这不是一直能优化的了，这里只是作为一个方法去尝试。

(5) 索引选择性

索引选择性是不重复的索引值也叫基数（cardinality）表中数据行数的比值，索引选择性=基数/数据行，基数可以通过“show index from 表名”查看。

高索引选择性的好处就是mysql查找匹配的时候可以过滤更多的行，唯一索引的选择性最佳，值为1。

那么对于非唯一索引或者说要被创建索引的列的数据内容很长，那就要选择索引前缀。这里就简单说明一下：

```
mysql> select count(distinct(username))/count(*)  from one;
+-----------------------------------+
| count(distinct(username))/count(*) |
+-----------------------------------+
|                            0.2047 |
+-----------------------------------+
1 row in set (0.09 sec)
```

count(distinct(username))/count(*)就是索引选择性的值，这里0.2太小了。

假如username列数据很长，则可以通过

select count(distinct(concat(first_name, left(last_name, N))/count(*) from one; 测试出接近1的索引选择性，其中N是索引的长度，穷举法去找出N的值，然后再建索引。

(6) 重复或多余索引

很多phper开始都以为建索引相对多点性能就好点，压根没考虑到有些索引是重复的，比如建一个(username),(username,password), (username,password,last_login),很明显第一个索引是重复的，因为后两者都能满足其功能。

要有个意识就是，在满足功能需求的情况下建最少索引。对于INNODB引擎的索引来说，每次修改数据都要把主键索引，辅助索引中相应索引值修改，这可能会出现大量数据迁移，分页，以及碎片的出现。

3、系统配置与维护优化

(1) 重要的一些变量

- key_buffer_size索引块缓存区大小, 针对MyISAM存储引擎,该值越大,性能越好.但是超过操作系统能承受的最大值,反而会使mysql变得不稳定. ----这是很重要的参数
- sort_buffer_size 这是索引在排序缓冲区大小，若排序数据大小超过该值，则创建临时文件，注意和myisam_sort_buffer_size的区别----这是很重要的参数
- read_rnd_buffer_size当排序后按排序后的顺序读取行时，则通过该缓冲区读取行，避免搜索硬盘。将该变量设置为较大的值可以大大改进ORDER BY的性能。但是，这是为每个客户端分配的缓冲区，因此你不应将全局变量设置为较大的值。相反，只为需要运行大查询的客户端更改会话变量
- join_buffer_size用于表间关联(join)的缓存大小
- tmp_table_size缓存表的大小
- table_cache允许 MySQL 打开的表的最大个数，并且这些都cache在内存中
- delay_key_write针对MyISAM存储引擎,延迟更新索引.意思是说,update记录时,先将数据up到磁盘,但不up索引,将索引存在内存里,当表关闭时,将内存索引,写到磁盘

更多参数查看http://www.phpben.com/?post=70

(2) optimize、Analyze、check、repair维护操作

- optimize 数据在插入，更新，删除的时候难免一些数据迁移，分页，之后就出现一些碎片，久而久之碎片积累起来影响性能，这就需要DBA定期的优化数据库减少碎片，这就通过optimize命令。

  如对MyisAM表操作：optimize table 表名

  对于InnoDB表是不支持optimize操作，否则提示“Table does not support optimize, doing recreate + analyze instead”，当然也可以通过命令：alter table one type=innodb; 来替代。

- Analyze 用来分析和存储表的关键字的分布，使得系统获得准确的统计信息，影响 SQL 的执行计划的生成。对于数据基本没有发生变化的表，是不需要经常进行表分析的。但是如果表的数据量变化很明显，用户感觉实际的执行计划和预期的执行计划不 同的时候，执行一次表分析可能有助于产生预期的执行计划。

Analyze table 表名

- Check检查表或者视图是否存在错误，对 MyISAM 和 InnoDB 存储引擎的表有作用。对于 MyISAM 存储引擎的表进行表检查，也会同时更新关键字统计数据
- Repair optimize需要有足够的硬盘空间，否则可能会破坏表，导致不能操作，那就要用上repair，注意INNODB不支持repair操作

以上的操作出现的都是如下这是check

```
+----------+-------+--------------+-------------+
| Table    | Op    | Msg_type| Msg_text |
+----------+-------+--------------+-------------+
| test.one | check | status   | OK      |
+----------+-------+--------------+-------------+
```

其中op是option 可以是repair check optimize，msg_type 表示信息类型，msg_text 表示信息类型，这里就说明表的状态正常。如在innodb表使用repair就出现note | The storage engine for the table doesn't support repair

注意：以上操作最好在数据库访问量最低的时候操作，因为涉及到很多表锁定，扫描，数据迁移等操作，否则可能导致一些功能无法正常使用甚至数据库崩溃。

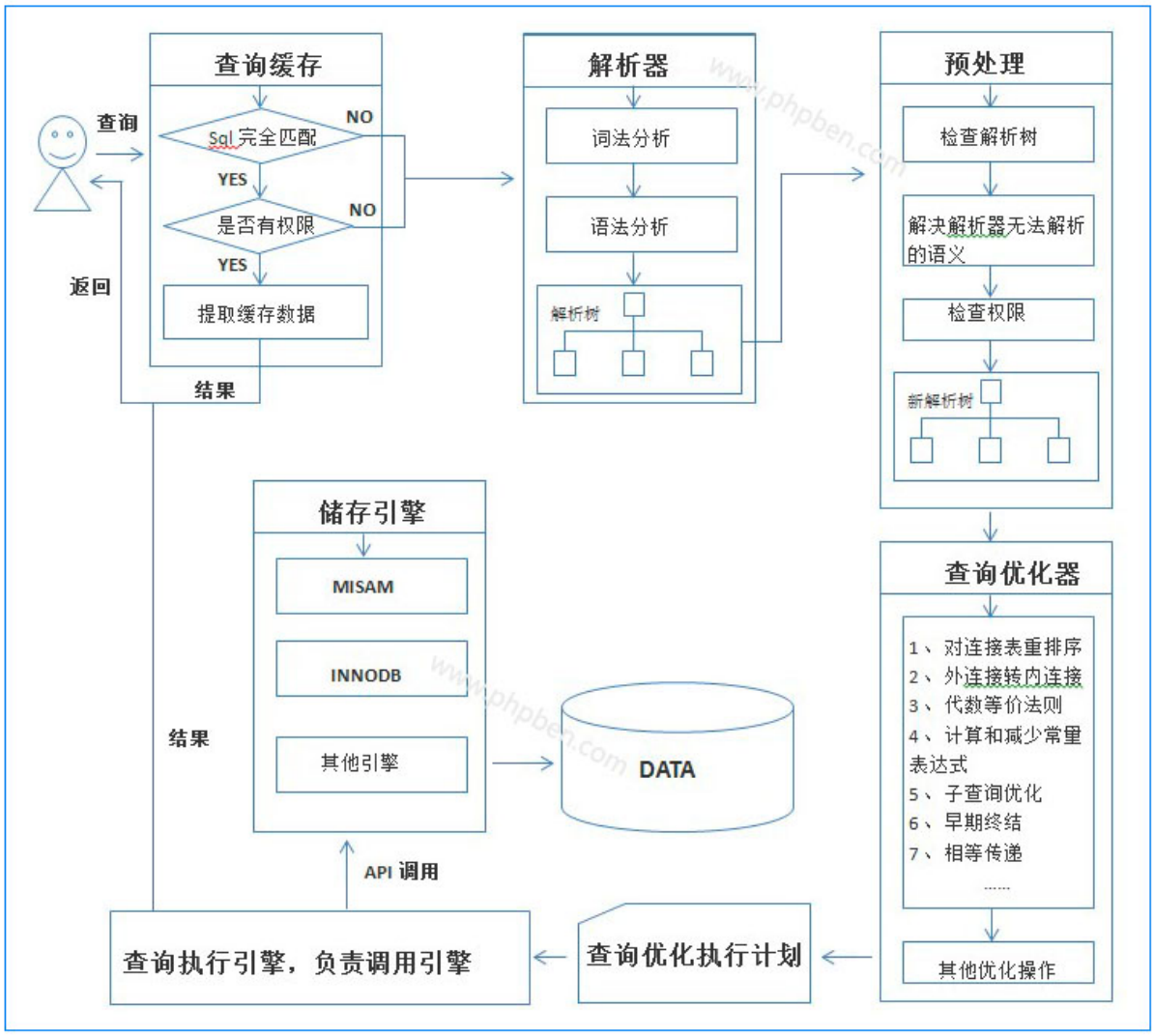
(3)表结构的更新与维护

- 改表结构。当要在数据量千万级的数据表中使用alter更改表结构的时候，这是一个棘手问题。一种方法是在低并发低访问量的时候用平常的alter更改表。另外一种就是建另一个与要修改的表，这个表除了要修改的结构属性外其他的和原表一模一样，这样就能得到一个相应的.frm文件，然后用flush with read lock 锁定读，然后覆盖用新建的.frm文件覆盖原表的.frm，最后unlock table 释放表。
- 建立新的索引。一般方法这里不说。
  1、 创建没索引的a表，导入数据形成.MYD文件。
  2、 创建包括索引b表，形成.FRM和.MYI文件
  3、 锁定读写
  4、 把b表的.FRM和.MYI文件改成a表名字
  5、 解锁
  6、 用repair创建索引。

  这个方法对于大表也是很有效的。这也是为什么很多dba坚持说“先导数据库在建索引，这样效率更快”
- 定期检查mysql服务器

定期使用show status、show processlist等命令检查数据库。这里就不细说，这说起来也篇幅是比较大的，笔者对这个也不是很了解

第四部分：图说mysql查询执行流程



1、 查询缓存，判断sql语句是否完全匹配，再判断是否有权限，两个判断为假则到解析器解析语句，为真则提取数据结果返回给用户。

2、 解析器解析。解析器先词法分析，语法分析，检查错误比如引号有没闭合等，然后生成解析树。

3、 预处理。预处理解决解析器无法决解的语义，如检查表和列是否存在，别名是否有错，生成新的解析树。

4、 优化器做大量的优化操作。

5、 生成执行计划。

6、 查询执行引擎，负责调度引擎获取相应数据

7、 返回结果。

这篇博文准备，写，将用了一个月时间！终于写完，但真的学了很多东西！ 有纰漏请联系：benwin(bw@7bus.net/445235728@qq.com)

参考：

http://www.cnblogs.com/hustcat/archive/2009/10/28/1591648.html

http://www.cnblogs.com/oldhorse/archive/2009/11/16/1604009.html
http://blog.csdn.net/zuiaituantuan/article/details/5909334
http://www.codinglabs.org/html/theory-of-mysql-index.html
http://isky000.com/database/mysql_order_by_implement
http://dev.mysql.com/doc/refman/5.0/en/server-system-variables.html
http://www.docin.com/p-211669085.html
标签: mysql索引结构 inndbo b-树 sql优化

« 各种浏览器审查、监听http头工具介绍

查看表的存储引擎结构“show table status like ‘表名’”»

Powered by benwin 

Entries RSS