# 1.server层

## 字符集

### 一、字符集和校验规则

字符集是一套符合和编码，校验规则（collation）是在字符集内用于比较字符的一套规则，即字符集的排序规则。MySQL可以使用对种字符集和检验规则来组织字符。

MySQL服务器可以支持多种字符集，在同一台服务器，同一个数据库，甚至同一个表的不同字段都可以指定使用不同的字符集，相比oracle等其他数据库管理系统，在同一个数据库只能使用相同的字符集，MySQL明显存在更大的灵活性。

每种字符集都可能有多种校对规则，并且都有一个默认的校对规则，并且每个校对规则只是针对某个字符集，和其他的字符集么有关系。

在MySQL中，字符集的概念和编码方案被看做是同义词，一个字符集是一个转换表和一个编码方案的组合。

Unicode（Universal Code）是一种在计算机上使用的字符编码。Unicode 是为了解决传统的字符编码方案的局限而产生的，它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。Unicode存在不同的编码方案，包括Utf-8，Utf-16和Utf-32。Utf表示Unicode Transformation Format。

### 二、查看mysql字符集方法

#### 1、查看mysql服务器支持的字符集

mysql> show character set;

mysql> select \* from information\_schema.character\_sets;

mysql> select character\_set\_name, default\_collate\_name, description, maxlen from

information\_schema.character\_sets;

#### 2、查看字符集的校对规则

mysql> show collation;

mysql> show collation like 'utf8';

mysql> select \* from information\_schema.collations where collation\_name like 'utf8%';

#### 3、查看当前数据库的字符集

mysql> show variables like 'character%';

+--------------------------+----------------------------------+

| Variable\_name | Value |

+--------------------------+----------------------------------+

| character\_set\_client | utf8 |

| character\_set\_connection | utf8 |

| character\_set\_database | latin1 |

| character\_set\_filesystem | utf8 |

| character\_set\_results | utf8 |

| character\_set\_server | utf8 |

| character\_set\_system | utf8 |

| character\_sets\_dir | /usr/local/mysql/share/charsets/ |

+--------------------------+----------------------------------+

8 rows in set (0.00 sec)

名词解释：

    character\_set\_client：客户端请求数据的字符集  
    character\_set\_connection：客户机/服务器连接的字符集  
    character\_set\_database：默认数据库的字符集，无论默认数据库如何改变，都是这个字符集；如果没有默认数据库，那就使用 character\_set\_server指定的字符集，这个变量建议由系统自己管理，不要人为定义。  
    character\_set\_filesystem：把os上文件名转化成此字符集，即把 character\_set\_client转换character\_set\_filesystem， 默认binary是不做任何转换的

    character\_set\_results：结果集，返回给客户端的字符集  
    character\_set\_server：数据库服务器的默认字符集  
    character\_set\_system：系统字符集，这个值总是utf8，不需要设置。这个字符集用于数据库对象（如表和列）的名字，也用于存储在目录表中的函数的名字。

#### 4、查看当前数据库的校对规则

mysql> show variables like 'collation%';

+----------------------+-------------------+

| Variable\_name | Value |

+----------------------+-------------------+

| collation\_connection | utf8\_general\_ci |

| collation\_database | latin1\_swedish\_ci |

| collation\_server | utf8\_general\_ci |

+----------------------+-------------------+

3 rows in set (0.01 sec)

名词解释：

collation\_connection 当前连接的字符集。  
collation\_database    当前日期的默认校对。每次用USE语句来“跳转”到另一个数据库的时候，这个变量的值就会改变。如果没有当前数据库，这个变量的值就是collation\_server变量的值。  
collation\_server 服务器的默认校对。

排序方式的命名规则为：字符集名字\_语言\_后缀，其中各个典型后缀的含义如下：  
1）\_ci：不区分大小写的排序方式  
2）\_cs：区分大小写的排序方式  
3）\_bin：二进制排序方式，大小比较将根据字符编码，不涉及人类语言，因此\_bin的排序方式不包含人类语言

### 三、MySQL字符集的设置

#### 1、概述

MySQL字符集设置分为两类：

1）创建对象的默认值。

2）控制server和client端交互通信的配置。

##### 1、创建对象的默认值

字符集合校对规则有4个级别的默认设置：

1）服务器级别；

2）数据库级别；

3）表级别、列级别；

4）连接级别。

更低级别的设置会集成高级别的设置。

这里有一个通用的规则：先为服务器或者数据库选择一个合理的字符集，然后根据不同的实际情况，让某个列选择自己的字符集。

##### 2、控制server和client端交互通信的配置

大部分MySQL客户端都不具备同时支持多种字符集的能力，每次都只能使用一种字符集。

客户和服务器之间的字符集转换工作是由如下几个MySQL系统变量控制的。

1）character\_set\_server：mysql server默认字符集。

2）character\_set\_database：数据库默认字符集。

3）character\_set\_client：MySQL server假定客户端发送的查询使用的字符集。

4）character\_set\_connection：MySQL Server接收客户端发布的查询请求后，将其转换为character\_set\_connection变量指定的字符集。

5）character\_set\_results：mysql server把结果集和错误信息转换为character\_set\_results指定的字符集，并发送给客户端。

6）**character\_set\_system**：系统元数据(字段名等)字符集

还有以**collation\_**开头的同上面对应的变量，用来描述字符序。

注意事项：

• my.cnf中的default\_character\_set设置只影响mysql命令连接服务器时的连接字符集，不会对使用libmysqlclient库的应用程序产生任何作用！

• 对字段进行的SQL函数操作通常都是以内部操作字符集进行的，不受连接字符集设置的影响。

• SQL语句中的裸字符串会受到连接字符集或introducer设置的影响，对于比较之类的操作可能产生完全不同的结果，需要小心！

#### 3、默认情况下字符集选择规则

(1)编译MySQL 时，指定了一个默认的字符集，这个字符集是 latin1;  
(2)安装MySQL 时，可以在配置文件 (my.cnf) 中指定一个默认的的字符集，如果没指定，这个值继承自编译时指定的;  
(3)启动mysqld 时，可以在命令行参数中指定一个默认的的字符集，如果没指定，这个值继承自配置文件中的配置,此时character\_set\_server被设定为这个默认的字符集;  
(4)当创建一个新的数据库时，除非明确指定，这个数据库的字符集被缺省设定为character\_set\_server;  
(5)当选定了一个数据库时，character\_set\_database被设定为这个数据库默认的字符集;  
(6)在这个数据库里创建一张表时，表默认的字符集被设定为character\_set\_database，也就是这个数据库默认的字符集;  
(7)当在表内设置一栏时，除非明确指定，否则此栏缺省的字符集就是表默认的字符集;

#### 2、分述

##### 2.1、为列分配字符集

属于同一个表的不同列可以有不同的字符集，如果没有为一个列显示的定义字符集就使用默认字符集。创建一个表的时候，若显示的为列指定字符集，则字符集作为数据类型选项包含在其中，要放在数据类型后面及空指定和主键前面。

例如：

create table column\_charset(

c1 char(10) character set utf8 not null,

c2 char(10) char set utf8,

c3 varchar(10) charset utf8,

c4 varchar(10)) engine=innodb;

**注意：character set可以简写为char set和charset**

使用show create table table\_name;命令查看column\_charset建表语句：

mysql> show create table column\_charset\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: column\_charset

Create Table: CREATE TABLE `column\_charset` (

`c1` char(10) CHARACTER SET utf8 NOT NULL,

`c2` char(10) CHARACTER SET utf8 DEFAULT NULL,

`c3` varchar(10) CHARACTER SET utf8 DEFAULT NULL,

`c4` varchar(10) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1

1 row in set (0.01 sec)

ERROR:

No query specified

插入数据，感受一下效果：

mysql> insert into column\_charset(c1,c2,c3,c4) value("图灵","图灵","图灵","chavin");

Query OK, 1 row affected (0.01 sec)

mysql> select \* from column\_charset;

+--------+--------+--------+--------+

| c1 | c2 | c3 | c4 |

+--------+--------+--------+--------+

| 图灵 | 图灵 | 图灵 | chavin |

+--------+--------+--------+--------+

1 row in set (0.00 sec)

##### 2.2、为表分配字符集

create table table\_charset(

c1 varchar(10),

c2 varchar(10))engine=innodb default charset=utf8;

**注意：为表指定字符集可以使用以下几种方式：**

**default charset=utf8;**

**charset=utf8;**

**default character set=utf8;**

**character set=utf8;**

**default char set=utf8;**

**char set=utf8;**

检查建表语句：

mysql> show create table table\_charset\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: table\_charset

Create Table: CREATE TABLE `table\_charset` (

`c1` varchar(10) DEFAULT NULL,

`c2` varchar(10) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1 row in set (0.00 sec)

测试：

mysql> insert into table\_charset(c1,c2) values('图灵','图灵');

Query OK, 1 row affected (0.01 sec)

mysql> select \* from table\_charset;

+--------+--------+

| c1 | c2 |

+--------+--------+

| 图灵 | 图灵 |

+--------+--------+

1 row in set (0.00 sec)

##### 2.3、为数据库指定字符集

创建的每个数据库都有一个默认字符集，如果没有指定，就用latin1。

create database dbking charset=utf8;

**注意：创建数据库分配字符集可以采用以下几种子句：**

**charset=utf8;**

**default charset=utf8;**

**charset utf8;**

**default charset utf8;**

**char set=utf8;**

**default char set=utf8;**

**char set utf8;**

**default char set utf8;**

**character set=utf8;**

**default character set=utf8;**

**character set utf8;**

**default character set utf8;**

使用show create database db\_name;命令查看数据库创建语句：

mysql> show create database dbking;

+----------+-----------------------------------------------------------------+

| Database | Create Database |

+----------+-----------------------------------------------------------------+

| dbking | CREATE DATABASE `dbking` /\*!40100 DEFAULT CHARACTER SET utf8 \*/ |

+----------+-----------------------------------------------------------------+

1 row in set (0.00 sec)

##### 2.4、为列分配校对规则

每个列都应该有一个校对，如果没有显示指定，MySQL就使用属于该字符集的默认校对。如果指定了一个字符集和一个校对，字符集应该放在前面。

create table column\_collate(

c1 varchar(10) charset utf8 collate utf8\_romanian\_ci not null,

c2 varchar(10) charset utf8 collate utf8\_spanish\_ci)engine=innodb;

查看表的校验规则信息：

mysql> select table\_name,column\_name,collation\_name

from information\_schema.columns

where table\_name='column\_collate';

+----------------+-------------+------------------+

| table\_name | column\_name | collation\_name |

+----------------+-------------+------------------+

| column\_collate | c1 | utf8\_romanian\_ci |

| column\_collate | c2 | utf8\_spanish\_ci |

+----------------+-------------+------------------+

2 rows in set (0.04 sec)

注意：字符集和校对在处理字符表达式的过程中扮演着重要角色。我们不能比较两个属于不同校对的不同字符值。例如：

mysql> insert into column\_collate(c1,c2) values('A','A');

Query OK, 1 row affected (0.22 sec)

mysql> select \* from column\_collate;

+----+------+

| c1 | c2 |

+----+------+

| A | A |

+----+------+

1 row in set (0.00 sec)

mysql> select \* from column\_collate where c1=c2;

ERROR 1267 (HY000): Illegal mix of collations (utf8\_romanian\_ci,IMPLICIT) and (utf8\_spanish\_ci,IMPLICIT) for operation '='

##### 2.5、为表指定校对规则

create table table\_collate(

c1 varchar(10),

c2 varchar(10))engine=innodb default charset utf8 collate utf8\_romanian\_ci;

检查表的校对规则：

mysql> select table\_name,column\_name,collation\_name from information\_schema.columns where table\_name='table\_collate';

+---------------+-------------+------------------+

| table\_name | column\_name | collation\_name |

+---------------+-------------+------------------+

| table\_collate | c1 | utf8\_romanian\_ci |

| table\_collate | c2 | utf8\_romanian\_ci |

+---------------+-------------+------------------+

2 rows in set (0.00 sec)

##### 2.6、为数据库指定校对规则

create database dbking102 default charset utf8 collate utf8\_romanian\_ci;

查看数据库定义语句：

mysql> show create database dbking102\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Database: dbking102

Create Database: CREATE DATABASE `dbking102` /\*!40100 DEFAULT CHARACTER SET utf8 COLLATE utf8\_romanian\_ci \*/

1 row in set (0.00 sec)

##### 2.7、字符直接量字符集

如果没有显示指定，那么字符直接量的字符集就是数据库的默认字符集。如果要显示分配另一个字符集，需要把字符集的名字放在直接量前面，并且要在字符集前面加上下划线。

mysql> select \_utf8'语言 Language 言語 язык';

+---------------------------------+

| 语言 Language 言語 язык     |

+---------------------------------+

| 语言 Language 言語 язык     |

+---------------------------------+

##### 2.8、修改和设置MySQL服务器级别字符集

MySQL服务器支持众多不同的字符集，这类字符集可在编译时和运行时指定。

###### 1） 编译时指定

编译时可指定默认字符集和默认校对规则，要想同时更改默认字符集和校对规则，要同时使用--with-charset和--with-collation选项。校对规则必须是字符集的合法校对规则。

./configure -- with-charset=CHARSET --with-collation=COLLATION

通过configure选项--with-extra-charsets=LIST，可以定义在服务器中再定义增加字符集。LIST 指下面任何一项：  
    a.空格间隔的一系列字符集名  
    b.complex -，以包括不能动态装载的所有字符集  
    c.all –，以将所有字符集包括进二进制

./configure -- with-charset=CHARSET --with-collation=COLLATION --with-extra-charsets=all

###### 2） 在参数文件my.cnf中指定

[mysqld]

character\_set\_server=utf8

    --影响参数：character\_set\_server 和 character\_set\_database

    --注意：修改后要重启数据库才能生效。

[client]

default-character-set=utf8

    --影响参数：character\_set\_client，character\_set\_connection 和character\_set\_results。

    --注意：修改后无需重启数据库。  
3） 在启动参数前指定

./mysqld --character-set-server=utf8 &

    --影响参数：character\_set\_server 和 character\_set\_database  
4）在mysql客户端登陆时通过--default-character-set指定

mysql -uroot -pmysql --default-character-set=utf8

--影响参数：set character\_set\_client，set character\_set\_connection，set character\_set\_results。

###### 5）临时指定

a）分别指定

mysql> SET character\_set\_client = utf8;

mysql> SET character\_set\_connection = utf8;

mysql> SET character\_set\_database = utf8;

mysql> SET character\_set\_results = utf8;

mysql> SET character\_set\_server = utf8;

b）mysql客户端使用：set names utf8;

等同于

set character\_set\_client=utf8;

set character\_set\_connection=utf8;

set character\_set\_results=utf8;

c）set character set utf8;

等同于

set character\_set\_client=utf8;

set character\_set\_results=utf8;

set collation\_connection=@@collation\_database;

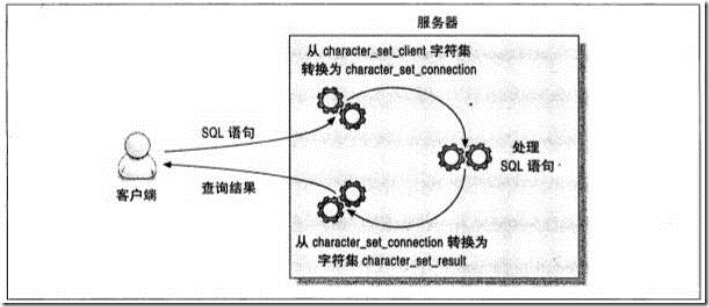
#### 3、总结

下面介绍下几个MYSQL命令：  
1）show character set;或show char set;  
查看数据库支持的所有字符集  
    2）status;或\s;  
查看当前状态 里面包括当然的字符集设置  
    3）show variables like 'char%';  
查看系统字符集设置，包括所有的字符集设置  
    4）show table status from sqlstudy like '%countries%';  
查看sqlstudy数据库中表的字符集设置  
    5）show full columns from countries;  
查看表列的字符集设置，关键是在同一个表中，每列可以设置成不同的字符集  
知道怎么查看字符集了，下面我来说下如何设置这些字符集  
    1.修改服务器级  
        a. 临时更改：  
            mysql>SET GLOBAL character\_set\_server=utf8;  
       b. 永久更改：  
修改my.cnf文件  
          [mysqld]  
          character-set-server=utf8  
   2.修改数据库级  
         a. 临时更改：  
             mysql>SET GLOBAL character\_set\_database=utf8;  
        b. 永久更改：  
改了服务器级就可以了  
  3.修改表级  
         mysql>ALTER TABLE table\_name DEFAULT CHARSET utf8;  
更改了后永久生效  
  4.修改列级  
修改示例：  
         mysql>alter table `products` change `products\_model` `products\_model` varchar( 20 )  
        character set  utf8 collate utf8\_general\_ci null default null;  
更改了后永久生效  
     5.更改连接字符集  
          a. 临时更改：  
              mysql> set names utf8;  
         b. 永久更改：  
修改my.cnf文件  
在[client]中增加：  
              default-character-set=utf8  
执行SQL语句时信息的路径是这样的  
信息输入路径：client→connection→server;  
信息输出路径：server→connection→results.

### 四、MySQL数据库中字符集转换流程

1、MySQL Server收到请求时将请求数据从character\_set\_client转换为character\_set\_connection；  
2、进行内部操作前将请求数据从character\_set\_connection转换为内部操作字符集，其确定方法如下：  
使用每个数据字段的CHARACTER SET设定值；  
若上述值不存在，则使用对应数据表的DEFAULT CHARACTER SET设定值(MySQL扩展，非SQL标准)；  
若上述值不存在，则使用对应数据库的DEFAULT CHARACTER SET设定值；  
若上述值不存在，则使用character\_set\_server设定值。  
    3、将操作结果从内部操作字符集转换为character\_set\_results。

下图源自于《高性能MySQL》中关于字符集转换的图解：

[](http://images2015.cnblogs.com/blog/669905/201705/669905-20170529003503344-1694516152.jpg)

### 五、MySQL数据库乱码原因解析及案例

1、产生乱码的根本原因  
    1）客户机没有正确地设置client字符集，导致原先的SQL语句被转换成connection所指字符集，而这种转换，是会丢失信息的，如果client是utf8格式，那么如果转换成gb2312格式，这其中必定会丢失信息，反之则不会丢失。一定要保证connection的字符集大于client字符集才能保证转换不丢失信息。  
    2）数据库字体没有设置正确，如果数据库字体设置不正确，那么connection字符集转换成database字符集照样丢失编码，原因跟上面一样。

#### 2、乱码或数据丢失

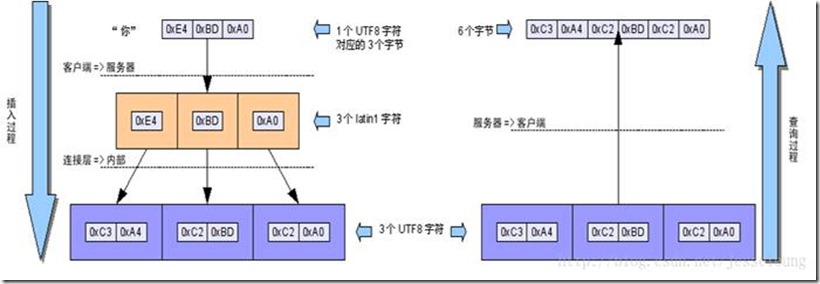
    character\_set\_client:我们要告诉服务器，我给你发送的数据是什么编码？  
    character\_set\_connection:告诉字符集转换器，转换成什么编码？  
    character\_set\_results:查询的结果用什么编码？  
如果以上三者都为字符集N,可简写为set names 'N';

##### 2.1 乱码问题

模拟情景1：

向默认字符集为utf8的数据表插入utf8编码的数据前连接字符集设置为latin1，查询时设置连接字符集为utf8。

插入时根据MySQL服务器的默认设置，character\_set\_client、character\_set\_connection和character\_set\_results均为latin1；  
插入操作的数据将经过latin1=>latin1=>utf8的字符集转换过程，这一过程中每个插入的汉字都会从原始的3个字节变成6个字节保存；  
查询时的结果将经过utf8=>utf8的字符集转换过程，将保存的6个字节原封不动返回，产生乱码……

**[](http://images2015.cnblogs.com/blog/669905/201705/669905-20170529003506250-240859631.jpg)**

**例如：**

mysql> set names latin1;

mysql> create table temp(name varchar(10)) charset utf8;

mysql> insert into temp values('中国');

mysql> select \* from temp;

+--------+

| name |

+--------+

| 中国 |

+--------+

mysql> set names utf8;

mysql> select \* from temp;

+---------------+

| name |

+---------------+

| ä¸­å›½ |

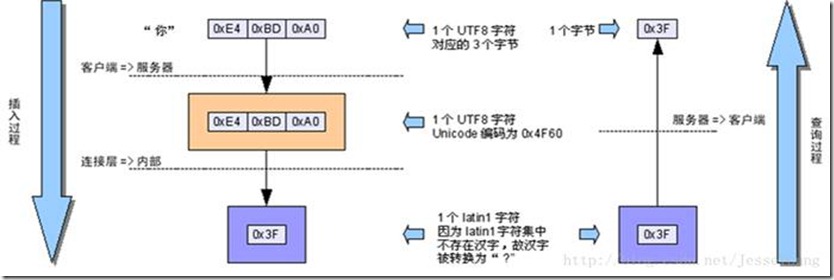
+---------------+

注意：存储字符集编码比插入时字符集大时，如果原封不动返回数据会出现乱码，不过可通过修改查询字符集，避免乱码，即不会丢失数据。

##### 2.2 数据丢失问题

模拟情景1：

向默认字符集为latin1的数据表插入utf8编码的数据前设置了连接字符集为utf8  
插入时根据连接字符集设置，character\_set\_client、character\_set\_connection和character\_set\_results均为utf8；  
插入数据将经过utf8=>utf8=>latin1的字符集转换，若原始数据中含有\u0000~\u00ff范围以外的Unicode字 符，会因为无法在latin1字符集中表示而被转换为“?”(0×3F)符号，以后查询时不管连接字符集设置如何都无法恢复其内容了。

[](http://images2015.cnblogs.com/blog/669905/201705/669905-20170529003508360-587956587.jpg)

例如：

mysql> set names utf8;

mysql> create table temp(name varchar(10)) charset latin1;

mysql> insert into temp values('中国');

mysql> select \* from temp;

+------+

| name |

+------+

| ?? |

+------+

mysql> set names latin1;

mysql> select \* from temp;

+------+

| name |

+------+

| ?? |

+------+

数据不完整了，且无法恢复。

#### 3、 乱码终极解决方案

        1）首先要明确你的客户端时候何种编码格式，这是最重要的（IE6一般用utf8，命令行一般是gbk，一般程序是gb2312)  
        2）确保你的数据库使用utf8格式，很简单，所有编码通吃。  
        3）一定要保证connection字符集大于等于client字符集，不然就会信息丢失，比如： latin1 < gb2312 < gbk < utf8，若设置set character\_set\_client = gb2312，那么至少connection的字符集要大于等于gb2312，否则就会丢失信息  
        4）以上三步做正确的话，那么所有中文都被正确地转换成utf8格式存储进了数据库，为了适应不同的浏览器，不同的客户端，你可以修改character\_set\_results来以不同的编码显示中文字体，由于utf8是大方向，因此web应用是我还是倾向于使用utf8格式显示中文的。

## **执行计划**

### 参考一

MySql提供了EXPLAIN语法用来进行查询分析，在SQL语句前加一个"EXPLAIN"即可。比如我们要分析如下SQL语句：

explain select \* from table where table.id = 1 

运行上面的sql语句后你会看到，下面的表头信息：

table | type | possible\_keys | key | key\_len | ref | rows | Extra

EXPLAIN列的解释

table   
显示这一行的数据是关于哪张表的

type   
这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为const、eq\_reg、ref、range、indexhe和ALL

说明：不同连接类型的解释（按照效率高低的顺序排序）

system：表只有一行：system表。这是const连接类型的特殊情况。

const ：表中的一个记录的最大值能够匹配这个查询（索引可以是主键或惟一索引）。因为只有一行，这个值实际就是常数，因为MYSQL先读这个值然后把它当做常数来对待。

eq\_ref：在连接中，MYSQL在查询时，从前面的表中，对每一个记录的联合都从表中读取一个记录，它在查询使用了索引为主键或惟一键的全部时使用。

ref：这个连接类型只有在查询使用了不是惟一或主键的键或者是这些类型的部分（比如，利用最左边前缀）时发生。对于之前的表的每一个行联合，全部记录都将从表中读出。这个类型严重依赖于根据索引匹配的记录多少—越少越好。

range：这个连接类型使用索引返回一个范围中的行，比如使用>或<查找东西时发生的情况。

index：这个连接类型对前面的表中的每一个记录联合进行完全扫描（比ALL更好，因为索引一般小于表数据）。

ALL：这个连接类型对于前面的每一个记录联合进行完全扫描，这一般比较糟糕，应该尽量避免。

possible\_keys   
显示可能应用在这张表中的索引。如果为空，没有可能的索引。可以为相关的域从WHERE语句中选择一个合适的语句

key   
实际使用的索引。如果为NULL，则没有使用索引。很少的情况下，MYSQL会选择优化不足的索引。这种情况下，可以在SELECT语句中使用USE INDEX（indexname）来强制使用一个索引或者用IGNORE INDEX（indexname）来强制MYSQL忽略索引

key\_len   
使用的索引的长度。在不损失精确性的情况下，长度越短越好

ref   
显示索引的哪一列被使用了，如果可能的话，是一个常数

rows   
MYSQL认为必须检查的用来返回请求数据的行数

Extra   
关于MYSQL如何解析查询的额外信息。将在表4.3中讨论，但这里可以看到的坏的例子是Using temporary和Using filesort，意思MYSQL根本不能使用索引，结果是检索会很慢

说明：extra列返回的描述的意义

Distinct ：一旦mysql找到了与行相联合匹配的行，就不再搜索了。

Not exists ：mysql优化了LEFT JOIN，一旦它找到了匹配LEFT JOIN标准的行，就不再搜索了。

Range checked for each Record（index map:#） ：没有找到理想的索引，因此对从前面表中来的每一个行组合，mysql检查使用哪个索引，并用它来从表中返回行。这是使用索引的最慢的连接之一。

**Using filesort ：看到这个的时候，查询就需要优化了。**mysql需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行。

Using index ：列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的，这发生在对表的全部的请求列都是同一个索引的部分的时候。

**Using temporary ：看到这个的时候，查询需要优化了。**这里，mysql需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上。

Where used ：使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行，并且连接类型ALL或index，这就会发生，或者是查询有问题。

因此，弄明白了explain语法返回的每一项结果，我们就能知道查询大致的运行时间了，如果查询里没有用到索引、或者需要扫描的行过多，那么可以感到明显的延迟。因此需要改变查询方式或者新建索引。mysql中的explain语法可以帮助我们改写查询，优化表的结构和索引的设置，从而最大地提高查询效率。当然，在大规模数据量时，索引的建立和维护的代价也是很高的，往往需要较长的时间和较大的空间，如果在不同的列组合上建立索引，空间的开销会更大。因此索引最好设置在需要经常查询的字段中。

### 参考二

1）、id列数字越大越先执行，如果说数字一样大，那么就从上往下依次执行，id列为null的就表是这是一个结果集，不需要使用它来进行查询。

2）、select\_type列常见的有：

A：simple：表示不需要union操作或者不包含子查询的简单select查询。有连接查询时，外层的查询为simple，且只有一个

B：primary：一个需要union操作或者含有子查询的select，位于最外层的单位查询的select\_type即为primary。且只有一个

C：union：union连接的两个select查询，第一个查询是dervied派生表，除了第一个表外，第二个以后的表select\_type都是union

D：dependent union：与union一样，出现在union 或union all语句中，但是这个查询要受到外部查询的影响

E：union result：包含union的结果集，在union和union all语句中,因为它不需要参与查询，所以id字段为null

F：subquery：除了from字句中包含的子查询外，其他地方出现的子查询都可能是subquery

G：dependent subquery：与dependent union类似，表示这个subquery的查询要受到外部表查询的影响

H：derived：from字句中出现的子查询，也叫做派生表，其他数据库中可能叫做内联视图或嵌套select

3）、table

显示的查询表名，如果查询使用了别名，那么这里显示的是别名，如果不涉及对数据表的操作，那么这显示为null，如果显示为尖括号括起来的<derived N>就表示这个是临时表，后边的N就是执行计划中的id，表示结果来自于这个查询产生。如果是尖括号括起来的<union M,N>，与<derived N>类似，也是一个临时表，表示这个结果来自于union查询的id为M,N的结果集。

4）、type

依次从好到差：system，const，eq\_ref，ref，fulltext，ref\_or\_null，unique\_subquery，index\_subquery，range，index\_merge，index，ALL，除了all之外，其他的type都可以使用到索引，除了index\_merge之外，其他的type只可以用到一个索引

A：system：表中只有一行数据或者是空表，且只能用于myisam和memory表。如果是Innodb引擎表，type列在这个情况通常都是all或者index

B：const：使用唯一索引或者主键，返回记录一定是1行记录的等值where条件时，通常type是const。其他数据库也叫做唯一索引扫描

C：eq\_ref：出现在要连接过个表的查询计划中，驱动表只返回一行数据，且这行数据是第二个表的主键或者唯一索引，且必须为not null，唯一索引和主键是多列时，只有所有的列都用作比较时才会出现eq\_ref

D：ref：不像eq\_ref那样要求连接顺序，也没有主键和唯一索引的要求，只要使用相等条件检索时就可能出现，常见与辅助索引的等值查找。或者多列主键、唯一索引中，使用第一个列之外的列作为等值查找也会出现，总之，返回数据不唯一的等值查找就可能出现。

E：fulltext：全文索引检索，要注意，全文索引的优先级很高，若全文索引和普通索引同时存在时，mysql不管代价，优先选择使用全文索引

F：ref\_or\_null：与ref方法类似，只是增加了null值的比较。实际用的不多。

G：unique\_subquery：用于where中的in形式子查询，子查询返回不重复值唯一值

H：index\_subquery：用于in形式子查询使用到了辅助索引或者in常数列表，子查询可能返回重复值，可以使用索引将子查询去重。

I：range：索引范围扫描，常见于使用>,<,is null,between ,in ,like等运算符的查询中。

J：index\_merge：表示查询使用了两个以上的索引，最后取交集或者并集，常见and ，or的条件使用了不同的索引，官方排序这个在ref\_or\_null之后，但是实际上由于要读取所个索引，性能可能大部分时间都不如range

K：index：索引全表扫描，把索引从头到尾扫一遍，常见于使用索引列就可以处理不需要读取数据文件的查询、可以使用索引排序或者分组的查询。

L：all：这个就是全表扫描数据文件，然后再在server层进行过滤返回符合要求的记录。

5）、possible\_keys

查询可能使用到的索引都会在这里列出来

6）、key

查询真正使用到的索引，select\_type为index\_merge时，这里可能出现两个以上的索引，其他的select\_type这里只会出现一个。

7）、key\_len

用于处理查询的索引长度，如果是单列索引，那就整个索引长度算进去，如果是多列索引，那么查询不一定都能使用到所有的列，具体使用到了多少个列的索引，这里就会计算进去，没有使用到的列，这里不会计算进去。留意下这个列的值，算一下你的多列索引总长度就知道有没有使用到所有的列了。要注意，mysql的ICP特性使用到的索引不会计入其中。另外，key\_len只计算where条件用到的索引长度，而排序和分组就算用到了索引，也不会计算到key\_len中。

8）、ref

如果是使用的常数等值查询，这里会显示const，如果是连接查询，被驱动表的执行计划这里会显示驱动表的关联字段，如果是条件使用了表达式或者函数，或者条件列发生了内部隐式转换，这里可能显示为func

9）、rows

这里是执行计划中估算的扫描行数，不是精确值

10）、extra

这个列可以显示的信息非常多，有几十种，常用的有

A：distinct：在select部分使用了distinc关键字

B：no tables used：不带from字句的查询或者From dual查询

C：使用not in()形式子查询或not exists运算符的连接查询，这种叫做反连接。即，一般连接查询是先查询内表，再查询外表，反连接就是先查询外表，再查询内表。

D：using filesort：排序时无法使用到索引时，就会出现这个。常见于order by和group by语句中

E：using index：查询时不需要回表查询，直接通过索引就可以获取查询的数据。

F：using join buffer（block nested loop），using join buffer（batched key accss）：5.6.x之后的版本优化关联查询的BNL，BKA特性。主要是减少内表的循环数量以及比较顺序地扫描查询。

G：using sort\_union，using\_union，using intersect，using sort\_intersection：

using intersect：表示使用and的各个索引的条件时，该信息表示是从处理结果获取交集

using union：表示使用or连接各个使用索引的条件时，该信息表示从处理结果获取并集

using sort\_union和using sort\_intersection：与前面两个对应的类似，只是他们是出现在用and和or查询信息量大时，先查询主键，然后进行排序合并后，才能读取记录并返回。

H：using temporary：表示使用了临时表存储中间结果。临时表可以是内存临时表和磁盘临时表，执行计划中看不出来，需要查看status变量，used\_tmp\_table，used\_tmp\_disk\_table才能看出来。

I：using where：表示存储引擎返回的记录并不是所有的都满足查询条件，需要在server层进行过滤。查询条件中分为限制条件和检查条件，5.6之前，存储引擎只能根据限制条件扫描数据并返回，然后server层根据检查条件进行过滤再返回真正符合查询的数据。5.6.x之后支持ICP特性，可以把检查条件也下推到存储引擎层，不符合检查条件和限制条件的数据，直接不读取，这样就大大减少了存储引擎扫描的记录数量。extra列显示using index condition

J：firstmatch(tb\_name)：5.6.x开始引入的优化子查询的新特性之一，常见于where字句含有in()类型的子查询。如果内表的数据量比较大，就可能出现这个

K：loosescan(m..n)：5.6.x之后引入的优化子查询的新特性之一，在in()类型的子查询中，子查询返回的可能有重复记录时，就可能出现这个

除了这些之外，还有很多查询数据字典库，执行计划过程中就发现不可能存在结果的一些提示信息

11）、filtered

使用explain extended时会出现这个列，5.7之后的版本默认就有这个字段，不需要使用explain extended了。这个字段表示存储引擎返回的数据在server层过滤后，剩下多少满足查询的记录数量的比例，注意是百分比，不是具体记录数

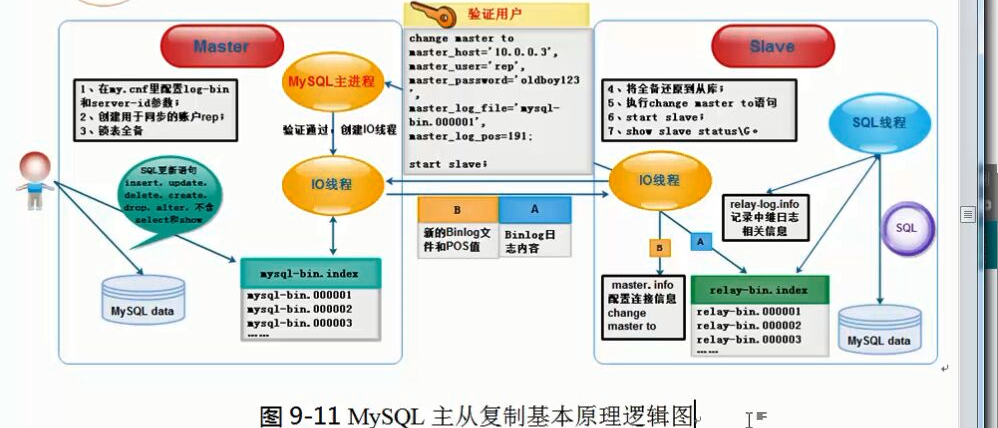
## binlog

## 主从复制

### 原理

MySQL的主从复制是一个异步的复制过程（虽然一般情况下感觉是实时的），数据将从一个Mysql数据库（我们称之为Master）复制到另一个Mysql数据库（我们称之为Slave），在Master与Slave之间实现整个主从复制的过程是由三个线程参与完成的。其中有两个线程（SQL线程和IO线程）在Slave端，另一个线程（I/O线程）在Master端。

  要实现MySQL的主从复制，首先必须打开Master端的binlog记录功能，否则就无法实现。因为整个复制过程实际上就是Slave从aster端获取binlog日志，然后再在Slave上以相同顺序执行获取的binlog日志中的记录的各种SQL操作





画图：

1）在Slave 服务器上执行sart slave命令开启主从复制开关，开始进行主从复制。

2）此时，Slave服务器的IO线程会通过在master上已经授权的复制用户权限请求连接master服务器，并请求从执行binlog日志文件的指定位置（日志文件名和位置就是在配置主从复制服务时执行change  
master命令指定的）之后开始发送binlog日志内容

3）Master服务器接收到来自Slave服务器的IO线程的请求后，其上负责复制的IO线程会根据Slave服务器的IO线程请求的信息分批读取指定binlog日志文件指定位置之后的binlog日志信息，然后返回给Slave端的IO线程。返回的信息中除了binlog日志内容外，还有在Master服务器端记录的IO线程。返回的信息中除了binlog中的下一个指定更新位置。

4）当Slave服务器的IO线程获取到Master服务器上IO线程发送的日志内容、日志文件及位置点后，会将binlog日志内容依次写到Slave端自身的Relay Log（即中继日志）文件（Mysql-relay-bin.xxx）的最末端，并将新的binlog文件名和位置记录到master-info文件中，以便下一次读取master端新binlog日志时能告诉Master服务器从新binlog日志的指定文件及位置开始读取新的binlog日志内容

5）Slave服务器端的SQL线程会实时检测本地Relay Log 中IO线程新增的日志内容，然后及时把Relay LOG 文件中的内容解析成sql语句，并在自身Slave服务器上按解析SQL语句的位置顺序执行应用这样sql语句，并在relay-log.info中记录当前应用中继日志的文件名和位置点

### 主从复制条件

1、开启Binlog功能

2、主库要建立账号

3、从库要配置master.info（CHANGE MASTER to…相当于配置密码文件和Master的相关信息）

4、start slave 开启复制功能

### 知识点

1.3个线程，主库IO，从库IO和SQL及作用

2.master.info（从库）作用

3.relay-log 作用

4.异步复制

5.binlog作用（如果需要级联需要开启Binlog）

### 小结：

主从复制是异步的逻辑的SQL语句级的复制

复制时，主库有一个O/O线程，从库有两个线程，I/O和SQL线程

实现主从复制的必要条件是主库要开启记录binlog功能

作为复制的所有Mysql节点的server-id都不能相同

binlog文件只记录对数据库有更改的SQL语句（来自主库内容的变更），不记录任何查询（select，show）语句

# 2.索引实现原理

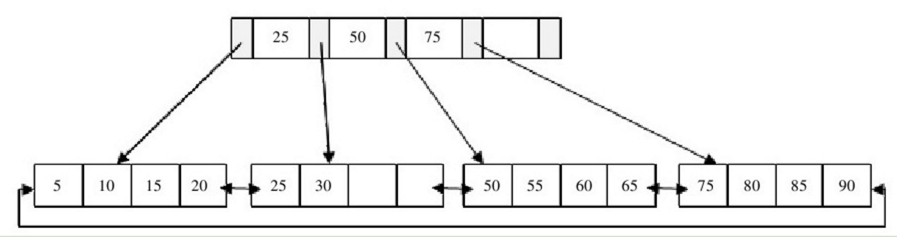
## B+树

### 原理

B+树和二叉树、平衡二叉树一样，都是经典的数据结构。B+树由B树和索引顺序访问方法（ISAM，是不是很熟悉？对，这也是MyISAM引擎最初参考的数据结构）演化而来，但是在实际使用过程中几乎已经没有使用B树的情况了。

B+树的定义十分复杂，因此只简要地介绍B+树：B+树是为磁盘或其他直接存取辅助设备而设计的一种平衡查找树，在B+树中，所有记录节点都是按键值的大小顺序存放在同一层的叶节点中，各叶节点指针进行连接。

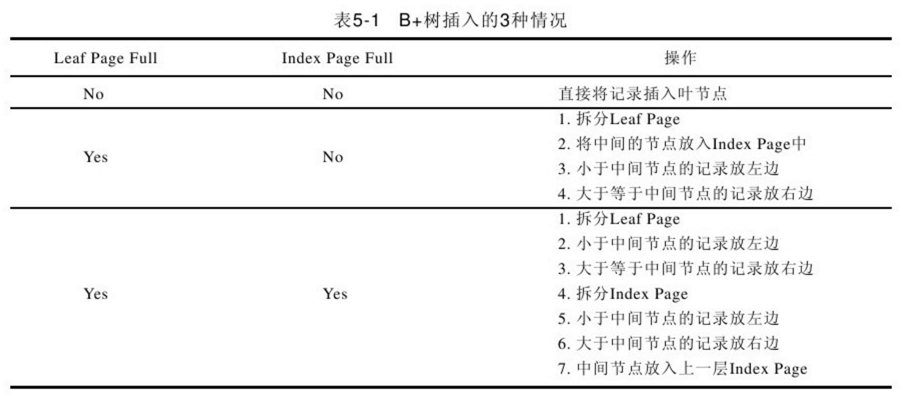
我们先来看一个B+树，其高度为2，每页可存放4条记录，扇出（fan out）为5。



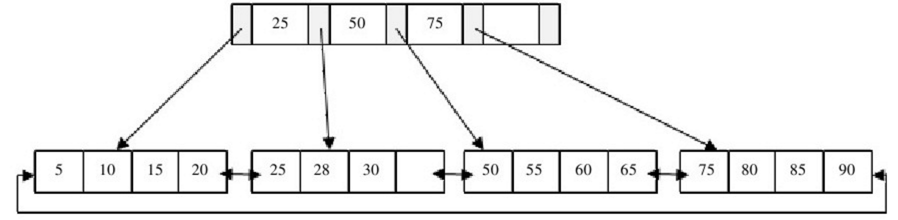
可以看出，所有记录都在叶节点中，并且是顺序存放的，如果我们从最左边的叶节点开始顺序遍历，可以得到所有键值的顺序排序：5、10、15、20、25、30、50、55、60、65、75、80、85、90。

### B+树的插入操作

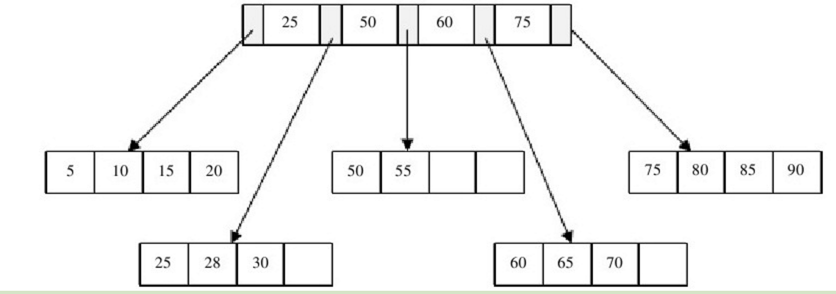
B+树的插入必须保证插入后叶节点中的记录依然排序，同时需要考虑插入B+树的三种情况，每种情况都可能会导致不同的插入算法，如表5-1所示。



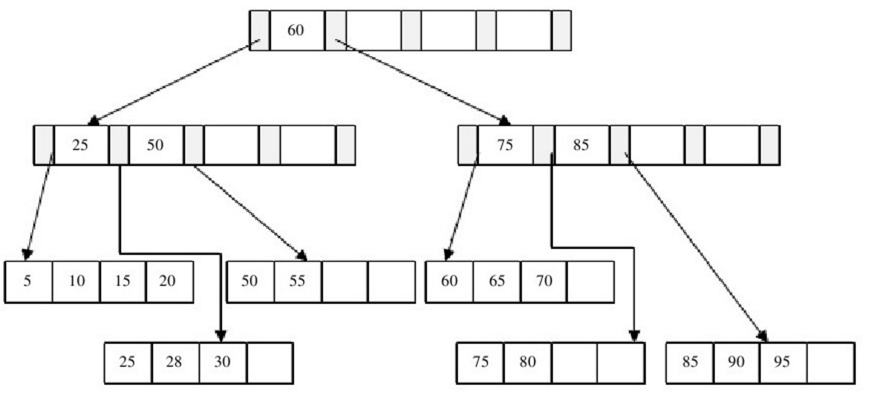
我们用实例来分析B+树的插入，我们插入28这个键值，发现当前Leaf Page和Index Page都没有满，我们直接插入就可以了。



这次我们再插入一条70这个键值，这时原先的Leaf Page已经满了，但是Index Page还没有满，符合表5-1的第二种情况，这时插入Leaf Page后的情况为50、55、60、65、70。我们根据中间的值60拆分叶节点。

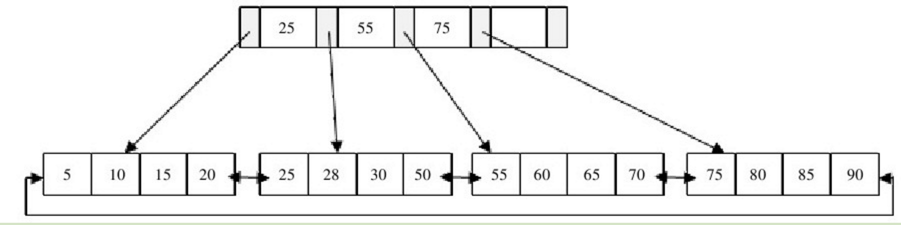


因为图片显示的关系，这次我没有能在各叶节点加上双向链表指针。最后我们来插入记录95，这时符合表5-1讨论的第三种情况，即Leaf Page和Index Page都满了，这时需要做两次拆分。



可以看到，不管怎么变化，B+树总是会保持平衡。但是为了保持平衡，对于新插入的键值可能需要做大量的拆分页（split）操作，而B+树主要用于磁盘，因此页的拆分意味着磁盘的操作，应该在可能的情况下尽量减少页的拆分。因此，B+树提供了旋转（rotation）的功能。

旋转发生在Leaf Page已经满了、但是其左右兄弟节点没有满的情况下。这时B+树并不会急于去做拆分页的操作，而是将记录移到所在页的兄弟节点上。通常情况下，左兄弟被首先检查用来做旋转操作，这时我们插入键值70，其实B+树并不会急于去拆分叶节点，而是做旋转，50，55，55旋转。



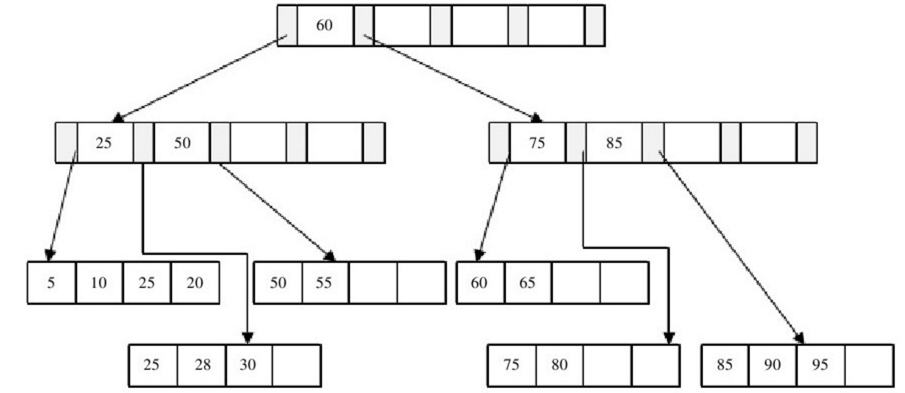
可以看到，采用旋转操作使B+树减少了一次页的拆分操作，而这时B+树的高度依然还是2。

### B+树的删除操作

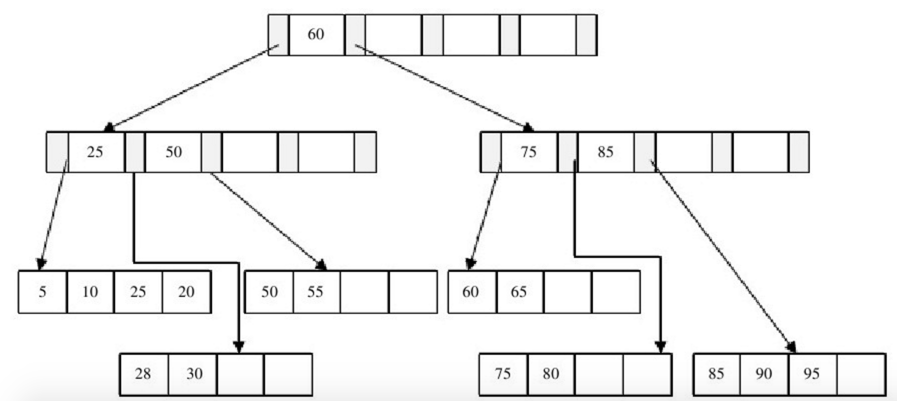
B+树使用填充因子（fill factor）来控制树的删除变化，50%是填充因子可设的最小值。B+树的删除操作同样必须保证删除后叶节点中的记录依然排序，同插入一样，B+树的删除操作同样需要考虑如表5-2所示的三种情况，与插入不同的是，删除根据填充因子的变化来衡量。



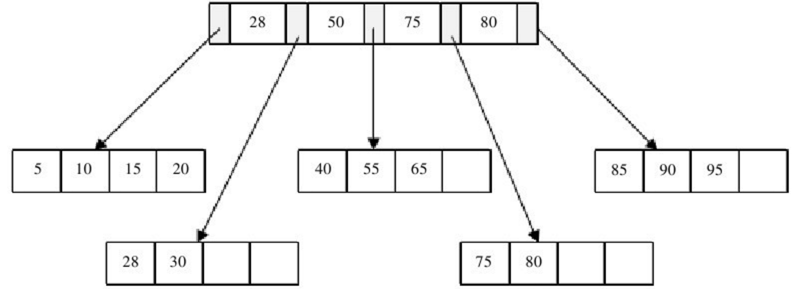
首先，删除键值为70的这条记录，该记录符合表5-2讨论的第一种情况，删除后。



接着我们删除键值为25的记录，这也是表5-2讨论的第一种情况，但是该值还是Index Page中的值，因此在删除Leaf Page中25的值后，还应将25的右兄弟节点的28更新到Page Index中，最后可得到图。



最后我们来看删除键值为60的情况，删除Leaf Page中键值为60的记录后，填充因子小于50%，这时需要做合并操作，同样，在删除Index Page中相关记录后需要做Index Page的合并操作，最后得到图。



### 补充

#### B-树

##### B-树是一种多路搜索树（并不一定是二叉的）

1970年，R.Bayer和E.mccreight提出了一种适用于外查找的树，它是一种平衡的多叉树，称为B树（或B-树、B\_树）。

##### 一棵m阶B树(balanced tree of order m)是一棵平衡的m路搜索树。它或者是空树，或者是满足下列性质的树：

1、根结点至少有两个子女；

2、每个非根节点所包含的关键字个数 j 满足：┌m/2┐ - 1 <= j <= m - 1；

3、除根结点以外的所有结点（不包括叶子结点）的度数正好是关键字总数加1，故内部子树个数 k 满足：┌m/2┐ <= k <= m ；

4、所有的叶子结点都位于同一层。

##### 特点：

是一种多路搜索树（并不是二叉的）：

1.定义任意非叶子结点最多只有M个儿子；且M>2；

2.根结点的儿子数为[2, M]；

3.除根结点以外的非叶子结点的儿子数为[M/2, M]；

4.每个结点存放至少M/2-1（取上整）和至多M-1个关键字；（至少2个关键字）

5.非叶子结点的关键字个数=指向儿子的指针个数-1；

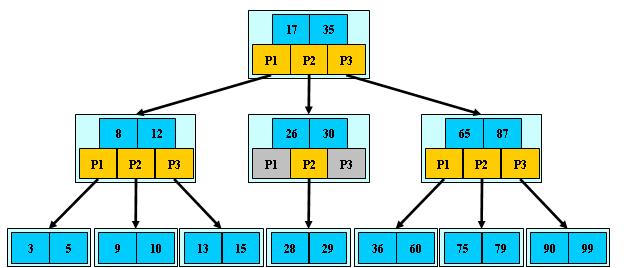
6.非叶子结点的关键字：K[1], K[2], …, K[M-1]；且K[i] < K[i+1]；

7.非叶子结点的指针：P[1], P[2], …, P[M]；其中P[1]指向关键字小于K[1]的

子树，P[M]指向关键字大于K[M-1]的子树，其它P[i]指向关键字属于(K[i-1], K[i])的子树；

8.所有叶子结点位于同一层；

###### 如：（M=3）



B-树的搜索，从根结点开始，对结点内的关键字（有序）序列进行二分查找，如果

命中则结束，否则进入查询关键字所属范围的儿子结点；重复，直到所对应的儿子指针为

空，或已经是叶子结点；

##### B-树的特性：

1.关键字集合分布在整颗树中；

2.任何一个关键字出现且只出现在一个结点中；

3.搜索有可能在非叶子结点结束；

4.其搜索性能等价于在关键字全集内做一次二分查找；

5.自动层次控制；

#### B+树

B+ 树是一种树数据结构，是一个n叉树，每个节点通常有多个孩子，一棵B+树包含根节点、内部节点和叶子节点。根节点可能是一个叶子节点，也可能是一个包含两个或两个以上孩子节点的节点。

##### 用途：

B+ 树通常用于[数据库](http://lib.csdn.net/base/14)和操作系统的文件系统中。NTFS, ReiserFS, NSS, XFS, JFS, ReFS 和BFS等文件系统都在使用B+树作为元数据索引。B+ 树的特点是能够保持数据稳定有序，其插入与修改拥有较稳定的对数时间复杂度。B+ 树元素自底向上插入。

##### B+树的定义

###### B+树是应文件系统所需而出的一种B-树的变型树。一棵m阶的B+树和m阶的B-树的差异在于：

1.有n棵子树的结点中含有n个关键字，每个关键字不保存数据，只用来索引，所有数据都保存在叶子节点。

2.所有的叶子结点中包含了全部关键字的信息，及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。

3.所有的非终端结点可以看成是索引部分，结点中仅含其子树（根结点）中的最大（或最小）关键字。   
通常在B+树上有两个头指针，一个指向根结点，一个指向关键字最小的叶子结点。

###### B+树是B-树的变体，也是一种多路搜索树：

1.其定义基本与B-树同，除了：

2.非叶子结点的子树指针与关键字个数相同；

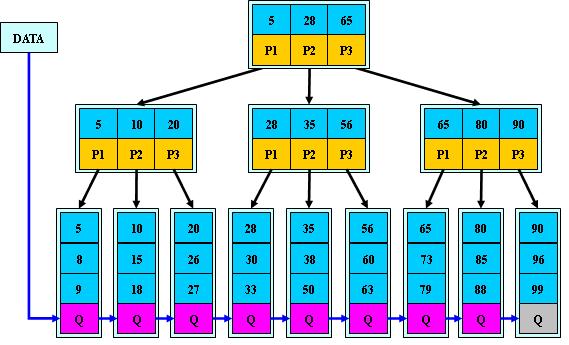
3.非叶子结点的子树指针P[i]，指向关键字值属于[K[i], K[i+1])的子树

（B-树是开区间）；

5.为所有叶子结点增加一个链指针；

6.所有关键字都在叶子结点出现；

###### 如：（M=3）

   
B+的搜索与B-树也基本相同，区别是B+树只有达到叶子结点才命中（B-树可以在

非叶子结点命中），其性能也等价于在关键字全集做一次二分查找；

##### B+的特性：

1.所有关键字都出现在叶子结点的链表中（稠密索引），且链表中的关键字恰好

是有序的；

2.不可能在非叶子结点命中；

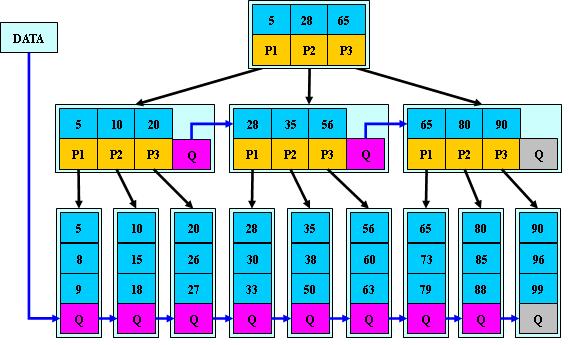
3.非叶子结点相当于是叶子结点的索引（稀疏索引），叶子结点相当于是存储

（关键字）数据的数据层；

4.更适合文件索引系统；

#### B\*树：

##### 是B+树的变体，在B+树的非根和非叶子结点再增加指向兄弟的指针；



##### B\*树定义了非叶子结点关键字个数至少为(2/3)\*M，即块的最低使用率为2/3

（代替B+树的1/2）；

B+树的分裂：当一个结点满时，分配一个新的结点，并将原结点中1/2的数据

复制到新结点，最后在父结点中增加新结点的指针；B+树的分裂只影响原结点和父

结点，而不会影响兄弟结点，所以它不需要指向兄弟的指针；

B\*树的分裂：当一个结点满时，如果它的下一个兄弟结点未满，那么将一部分

数据移到兄弟结点中，再在原结点插入关键字，最后修改父结点中兄弟结点的关键字

（因为兄弟结点的关键字范围改变了）；如果兄弟也满了，则在原结点与兄弟结点之

间增加新结点，并各复制1/3的数据到新结点，最后在父结点增加新结点的指针；

所以，B\*树分配新结点的概率比B+树要低，空间使用率更高；

#### 小结：

##### B-树：

多路搜索树，每个结点存储M/2到M个关键字，非叶子结点存储指向关键

字范围的子结点；

所有关键字在整颗树中出现，且只出现一次，非叶子结点可以命中；

##### B+树：

在B-树基础上，为叶子结点增加链表指针，所有关键字都在叶子结点

中出现，非叶子结点作为叶子结点的索引；B+树总是到叶子结点才命中；

##### B\*树：

在B+树基础上，为非叶子结点也增加链表指针，将结点的最低利用率

从1/2提高到2/3；

## 聚簇索引与二级索引

### 一.索引简介

众所周知，索引是关系型数据库中给数据库表中一列或多列的值排序后的存储结构，SQL的主流索引结构有B+树以及Hash结构，聚集索引以及非聚集索引用的是B+树索引。这篇文章会总结SQL Server以及MySQL的InnoDB和MyISAM两种SQL的索引。

SQL Sever索引类型有：唯一索引，主键索引，聚集索引，非聚集索引。  
  
MySQL 索引类型有：唯一索引，主键（聚集）索引，非聚集索引，全文索引。

### 二.聚集索引

聚集（clustered）索引，也叫聚簇索引。

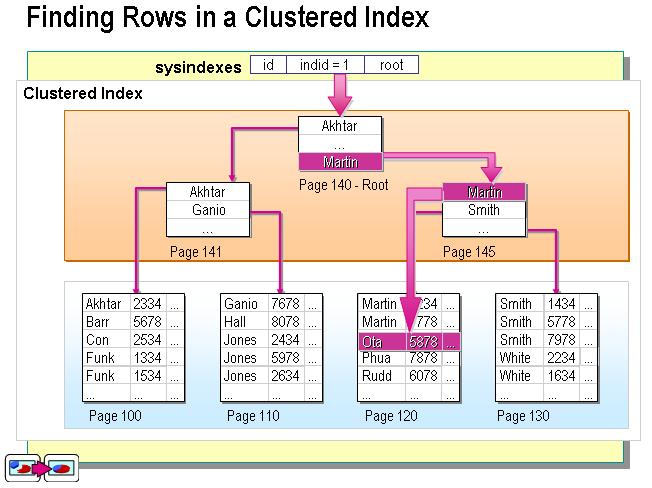
定义：数据行的物理顺序与列值（一般是主键的那一列）的逻辑顺序相同，一个表中只能拥有一个聚集索引。

单单从定义来看是不是显得有点抽象，打个比方，一个表就像是我们以前用的新华字典，聚集索引就像是拼音目录，而每个字存放的页码就是我们的数据物理地址，我们如果要查询一个“哇”字，我们只需要查询“哇”字对应在新华字典拼音目录对应的页码，就可以查询到对应的“哇”字所在的位置，而拼音目录对应的A-Z的字顺序，和新华字典实际存储的字的顺序A-Z也是一样的，如果我们中文新出了一个字，拼音开头第一个是B，那么他插入的时候也要按照拼音目录顺序插入到A字的后面，现在用一个简单的示意图来大概说明一下在数据库中的样子：

| **地址** | **id** | **username** | **score** |
| --- | --- | --- | --- |
| 0x01 | 1 | 小明 | 90 |
| 0x02 | 2 | 小红 | 80 |
| 0x03 | 3 | 小华 | 92 |
| .. | .. | .. | .. |
| 0xff | 256 | 小英 | 70 |

注：第一列的地址表示该行数据在磁盘中的物理地址，后面三列才是我们SQL里面用的表里的列，其中id是主键，建立了聚集索引。

结合上面的表格就可以理解这句话了吧：数据行的物理顺序与列值的**顺序相同**，如果我们查询id比较靠后的数据，那么这行数据的地址在磁盘中的物理地址也会比较靠后。而且由于物理排列方式与聚集索引的顺序相同，所以也就只能建立一个聚集索引了。

  
  
  
**聚集索引实际存放的示意图**

从上图可以看出聚集索引的好处了，索引的叶子节点就是对应的数据节点（MySQL的MyISAM除外，此存储引擎的聚集索引和非聚集索引只多了个唯一约束，其他没什么区别），可以直接获取到对应的全部列的数据，而非聚集索引在索引没有覆盖到对应的列的时候需要进行二次查询，后面会详细讲。因此在查询方面，聚集索引的速度往往会更占优势。

#### 创建聚集索引

如果不创建索引，系统会自动创建一个隐含列作为表的聚集索引。

1.创建表的时候指定主键（注意：SQL Sever默认主键为聚集索引，也可以指定为非聚集索引，而MySQL里主键就是聚集索引）

create table t1(

id int primary key,

name nvarchar(255)

)

2.创建表后添加聚集索引

**SQL Server**

create clustered index clustered\_index on table\_name(colum\_name)

**MySQL**

alter table table\_name add primary key(colum\_name)

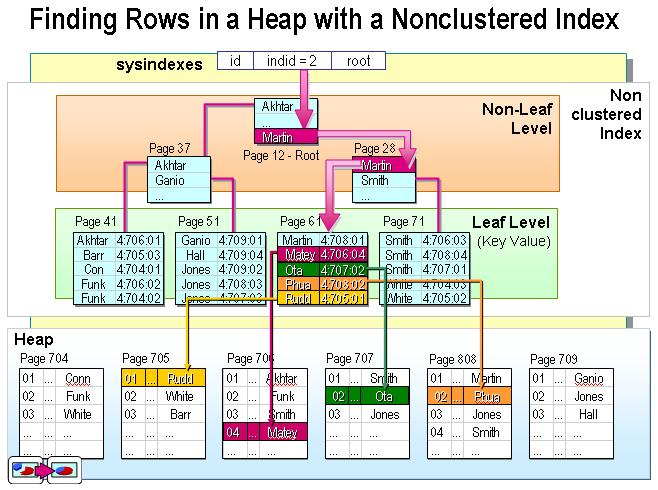
值得注意的是，最好还是在创建表的时候添加聚集索引，由于聚集索引的物理顺序上的特殊性，因此如果再在上面创建索引的时候会根据索引列的排序移动全部数据行上面的顺序，会非常地耗费时间以及性能。

### 三.非聚集索引

非聚集（unclustered）索引。

定义：该索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同，一个表中可以拥有多个非聚集索引。

其实按照定义，除了聚集索引以外的索引都是非聚集索引，只是人们想细分一下非聚集索引，分成普通索引，唯一索引，全文索引。如果非要把非聚集索引类比成现实生活中的东西，那么非聚集索引就像新华字典的偏旁字典，他结构顺序与实际存放顺序不一定一致。

  
  
  
**非聚集索引实际存放的示意图**

#### 聚合索引的二次查询问题

非聚集索引叶节点仍然是索引节点，只是有一个指针指向对应的数据块，此如果使用非聚集索引查询，而查询列中包含了其他该索引没有覆盖的列，那么他还要进行第二次的查询，查询节点上对应的数据行的数据。

如有以下表t1：

| **id** | **username** | **score** |
| --- | --- | --- |
| 1 | 小明 | 90 |
| 2 | 小红 | 80 |
| 3 | 小华 | 92 |
| .. | .. | .. |
| 256 | 小英 | 70 |

以及聚集索引clustered index(id), 非聚集索引index(username)。

使用以下语句进行查询，不需要进行二次查询，直接就可以从非聚集索引的节点里面就可以获取到查询列的数据。

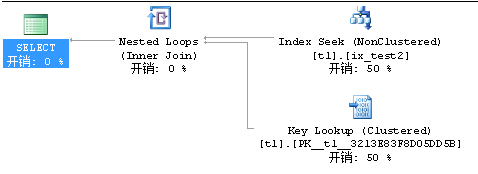
select id, username from t1 where username = '小明'

select username from t1 where username = '小明'

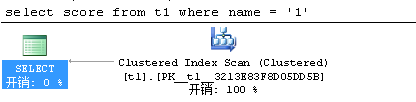
但是使用以下语句进行查询，就需要二次的查询去获取原数据行的score：

select username, score from t1 where username = '小明'

在SQL Server里面查询效率如下所示，Index Seek就是索引所花费的时间，Key Lookup就是二次查询所花费的时间。可以看的出二次查询所花费的查询开销占比很大，达到50%。



在SQL Server里面会对查询自动优化，选择适合的索引，因此如果在数据量不大的情况下，SQL Server很有可能不会使用非聚集索引进行查询，而是使用聚集索引进行查询，即便需要扫描整个聚集索引，效率也比使用非聚集索引效率要高。



本人试过在含有30w行表上建立非聚集索引，查询非聚集索引覆盖以外的列就会变成聚集索引的全索引扫描（index scan）查询来避免二次查询，而在另外一张200w行表才会用到非聚集索引seek对应的列再进行kek lookup，有关于SQL Server的有Index seek，index scan, table scan，key LookUp这几个概念，可以查看这个[blog](http://www.cnblogs.com/xwdreamer/archive/2012/07/06/2579504.html)，描写比较详细。

但在MySQL里面就算表里数据量少且查询了非键列，也不会使用聚集索引去全索引扫描，但如果强制使用聚集索引去查询，性能反而比非聚集索引查询要差，这就是两种SQL的不同之处。

还有一点要注意的是非聚集索引其实叶子节点除了会存储索引覆盖列的数据，也会存放聚集索引所覆盖的列数据。

#### 如何解决非聚集索引的二次查询问题

##### 复合索引（覆盖索引）

建立两列以上的索引，即可查询复合索引里的列的数据而不需要进行回表二次查询，如index(col1, col2)，执行下面的语句

select col1, col2 from t1 where col1 = '213';

要注意使用复合索引需要满足最左侧索引的原则，也就是查询的时候如果where条件里面没有最左边的一到多列，索引就不会起作用。

在SQL Server中还有include的用法，可以把非聚集索引里包含的列包含进来，而不一定需要建立复合索引。

### 四.总结与使用心得

1. 使用聚集索引的查询效率要比非聚集索引的效率要高，但是如果需要频繁去改变聚集索引的值，写入性能并不高，因为需要移动对应数据的物理位置。
2. 非聚集索引在查询的时候可以的话就避免二次查询，这样性能会大幅提升。
3. 不是所有的表都适合建立索引，只有数据量大表才适合建立索引，且建立在选择性高的列上面性能会更好。



## 存储引擎

**什么是存储引擎？**

MySQL中的数据用各种不同的技术存储在文件(或者内存)中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过选择不同的技术，你能够获得额外的速度或者功能，从而改善你的应用的整体功能。

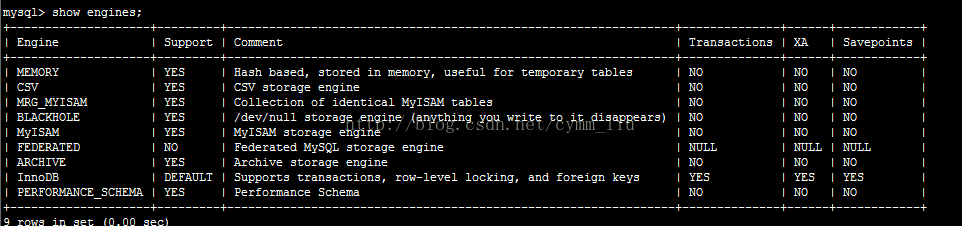
例如，如果你在研究大量的临时数据，你也许需要使用内存MySQL存储引擎。内存存储引擎能够在内存中存储所有的表格数据。又或者，你也许需要一个支持事务处理的数据库(以确保事务处理不成功时数据的回退能力)。

**这些不同的技术以及配套的相关功能在 MySQL中被称作存储引擎(也称作表类型)。** MySQL默认配置了许多不同的存储引擎，可以预先设置或者在MySQL服务器中启用。你可以选择适用于服务器、数据库和表格的存储引擎，以便在选择如何存储你的信息、如何检索这些信息以及你需要你的数据结合什么性能和功能的时候为你提供最大的灵活性。

关系数据库表是用于存储和组织信息的数据结构，可以将表理解为由行和列组成的表格，类似于Excel的电子表格的形式。有的表简单，有的表复杂，有的表根本不用来存储任何长期的数据，有的表读取时非常快，但是插入数据时去很差；而我们在实际开发过程中，就可能需要各种各样的表，不同的表，就意味着存储不同类型的数据，数据的处理上也会存在着差异，那么。对于MySQL来说，它提供了很多种类型的存储引擎（或者说不通的表类型），我们可以根据对数据处理的需求，选择不同的存储引擎，从而最大限度的利用MySQL强大的功能。

在mysql客户端中，使用以下命令可以查看MySQL支持的引擎。

show engines;



**MyISAM**

它不支持事务，也不支持外键，尤其是访问速度快，对事务完整性没有要求或者以SELECT、INSERT为主的应用基本都可以使用这个引擎来创建表。  
每个MyISAM在磁盘上存储成3个文件，其中文件名和表名都相同，但是扩展名分别为：

* .frm(存储表定义)
* MYD(MYData，存储数据)
* MYI(MYIndex，存储索引)

　　数据文件和索引文件可以放置在不同的目录，平均分配IO，获取更快的速度。要指定数据文件和索引文件的路径，需要在创建表的时候通过DATA DIRECTORY和INDEX DIRECTORY语句指定，文件路径需要使用绝对路径。  
　　每个MyISAM表都有一个标志，服务器或myisamchk程序在检查MyISAM数据表时会对这个标志进行设置。MyISAM表还有一个标志用来表明该数据表在上次使用后是不是被正常的关闭了。如果服务器以为当机或崩溃，这个标志可以用来判断数据表是否需要检查和修复。如果想让这种检查自动进行，可以在启动服务器时使用--myisam-recover现象。这会让服务器在每次打开一个MyISAM数据表是自动检查数据表的标志并进行必要的修复处理。MyISAM类型的表可能会损坏，可以使用CHECK TABLE语句来检查MyISAM表的健康，并用REPAIR TABLE语句修复一个损坏到MyISAM表。  
　　MyISAM的表还支持3种不同的存储格式：

* 静态(固定长度)表
* 动态表
* 压缩表

　　其中静态表是默认的存储格式。静态表中的字段都是非变长字段，这样每个记录都是固定长度的，这种存储方式的优点是存储非常迅速，容易缓存，出现故障容易恢复；缺点是占用的空间通常比动态表多。静态表在数据存储时会根据列定义的宽度定义补足空格，但是在访问的时候并不会得到这些空格，这些空格在返回给应用之前已经去掉。同时需要注意：在某些情况下可能需要返回字段后的空格，而使用这种格式时后面到空格会被自动处理掉。  
　　动态表包含变长字段，记录不是固定长度的，这样存储的优点是占用空间较少，但是频繁到更新删除记录会产生碎片，需要定期执行OPTIMIZE TABLE语句或myisamchk -r命令来改善性能，并且出现故障的时候恢复相对比较困难。  
　　压缩表由myisamchk工具创建，占据非常小的空间，因为每条记录都是被单独压缩的，所以只有非常小的访问开支。

**InnoDB**

InnoDB是一个健壮的事务型存储引擎，这种存储引擎已经被很多互联网公司使用，为用户操作非常大的数据存储提供了一个强大的解决方案。我的电脑上安装的MySQL 5.6.13版，InnoDB就是作为默认的存储引擎。InnoDB还引入了行级锁定和外键约束，在以下场合下，使用InnoDB是最理想的选择：

1.更新密集的表。InnoDB存储引擎特别适合处理多重并发的更新请求。  
2.事务。InnoDB存储引擎是支持事务的标准MySQL存储引擎。  
3.自动灾难恢复。与其它存储引擎不同，InnoDB表能够自动从灾难中恢复。  
4.外键约束。MySQL支持外键的存储引擎只有InnoDB。  
5.支持自动增加列AUTO\_INCREMENT属性。

一般来说，如果需要事务支持，并且有较高的并发读取频率，InnoDB是不错的选择。

**MEMORY**

使用MySQL Memory存储引擎的出发点是速度。为得到最快的响应时间，采用的逻辑存储介质是系统内存。虽然在内存中存储表数据确实会提供很高的性能，但当mysqld守护进程崩溃时，所有的Memory数据都会丢失。获得速度的同时也带来了一些缺陷。它要求存储在Memory数据表里的数据使用的是长度不变的格式，这意味着不能使用BLOB和TEXT这样的长度可变的数据类型，VARCHAR是一种长度可变的类型，但因为它在MySQL内部当做长度固定不变的CHAR类型，所以可以使用。

一般在以下几种情况下使用Memory存储引擎：

1.目标数据较小，而且被非常频繁地访问。在内存中存放数据，所以会造成内存的使用，可以通过参数max\_heap\_table\_size控制Memory表的大小，设置此参数，就可以限制Memory表的最大大小。

2.如果数据是临时的，而且要求必须立即可用，那么就可以存放在内存表中。

3.存储在Memory表中的数据如果突然丢失，不会对应用服务产生实质的负面影响。

Memory同时支持散列索引和B树索引。B树索引的优于散列索引的是，可以使用部分查询和通配查询，也可以使用<、>和>=等操作符方便数据挖掘。散列索引进行“相等比较”非常快，但是对“范围比较”的速度就慢多了，因此散列索引值适合使用在=和<>的操作符中，不适合在<或>操作符中，也同样不适合用在order by子句中。

可以在表创建时利用USING子句指定要使用的版本。例如：

复制代码代码如下:

create table users  
(  
    id smallint unsigned not null auto\_increment,  
    username varchar(15) not null,  
    pwd varchar(15) not null,  
    index using hash (username),  
    primary key (id)  
)engine=memory;

上述代码创建了一个表，在username字段上使用了HASH散列索引。下面的代码就创建一个表，使用BTREE索引。

复制代码代码如下:

create table users  
(  
    id smallint unsigned not null auto\_increment,  
    username varchar(15) not null,  
    pwd varchar(15) not null,  
    index using btree (username),  
    primary key (id)  
)engine=memory;

**MERGE**

MERGE存储引擎是一组MyISAM表的组合，这些MyISAM表结构必须完全相同，尽管其使用不如其它引擎突出，但是在某些情况下非常有用。说白了，Merge表就是几个相同MyISAM表的聚合器；Merge表中并没有数据，对Merge类型的表可以进行查询、更新、删除操作，这些操作实际上是对内部的MyISAM表进行操作。Merge存储引擎的使用场景。

对于服务器日志这种信息，一般常用的存储策略是将数据分成很多表，每个名称与特定的时间端相关。例如：可以用12个相同的表来存储服务器日志数据，每个表用对应各个月份的名字来命名。当有必要基于所有12个日志表的数据来生成报表，这意味着需要编写并更新多表查询，以反映这些表中的信息。与其编写这些可能出现错误的查询，不如将这些表合并起来使用一条查询，之后再删除Merge表，而不影响原来的数据，删除Merge表只是删除Merge表的定义，对内部的表没有任何影响。

**ARCHIVE**

Archive是归档的意思，在归档之后很多的高级功能就不再支持了，仅仅支持最基本的插入和查询两种功能。在MySQL 5.5版以前，Archive是不支持索引，但是在MySQL 5.5以后的版本中就开始支持索引了。Archive拥有很好的压缩机制，它使用zlib压缩库，在记录被请求时会实时压缩，所以它经常被用来当做仓库使用。

**存储引擎的一些问题**

1.如何查看服务器有哪些存储引擎可以使用？  
为确定你的MySQL服务器可以用哪些存储引擎，执行如下命令：

复制代码代码如下:

show engines;

这个命令就能搞定了。

2.如何选择合适的存储引擎？  
（1）选择标准可以分为：  
（2）是否需要支持事务；  
（3）是否需要使用热备；  
（4）崩溃恢复：能否接受崩溃；  
（5）是否需要外键支持；  
然后按照标准，选择对应的存储引擎即可。

**总结**

这篇文章总结了几种比较常用的存储引擎，对于实际的工作，需要根据具体的情况而定，结合实际的项目实例进行应用，才是最好的学习方法。

# 3.MySQL锁原理

## 表锁

## 意向锁

## 行锁

## 间隙锁

## next-key lock