

-

整理自互联网，仅供学习交流。



2018-10-30

[公司名称]

[公司地址]

目录

[一、 123 1](#_Toc3539499)

[1.1 1 1](#_Toc3539500)

[1.1.1 3 1](#_Toc3539501)

# MySQL基础

## 数据库概述

数据库（DataBase，DB）：指长期保存在计算机的存储设备上，按照一定规则组织起来，可以被各种用户或应用共享的数据集合。

数据库管理系统（DataBase Management System，DBMS）：指一种操作和管理数据库的大型软件，用于建立、使用和维护数据库，对数据库进行统一管理和控制，以保证数据库的安全性和完整性。用户通过数据库管理系统访问数据库中的数据。

数据库软件应该为**数据库管理系统**，数据库是通过数据库管理系统创建和操作的。

数据库：存储、维护和管理数据的集合。

## 数据库的安装与配置

\* 安装

\* 参照图解

\* 一路下一步

\* 卸载

1.停止mysql服务 net stop mysql 启动mysql服务 net start mysql

2.卸载mysql

3.找到mysql 安装目录下的 my.ini datadir="C:/ProgramData/MySQL/MySQL Server 5.5/Data/"

\* 运行

\* 安装成功了打开cmd --> mysql -uroot -p你的密码

\* 修改mysql root用户密码

1) 停止mysql服务 运行输入services.msc 停止mysql服务

或者 cmd --> net stop mysql

2) 在cmd下 输入 mysqld--skip-grant-tables 启动服务器 光标不动 （不要关闭该窗口）

3) 新打开cmd 输入mysql -u root -p 不需要密码

use mysql;

update user set password=password('abc') WHERE User='root';

4) 关闭两个cmd窗口 在任务管理器结束mysqld 进程

5) 在服务管理页面 重启mysql 服务

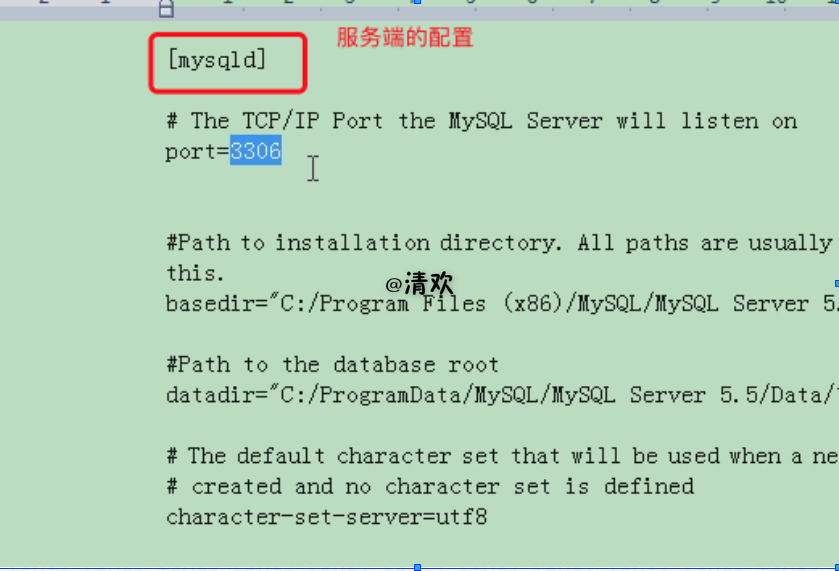
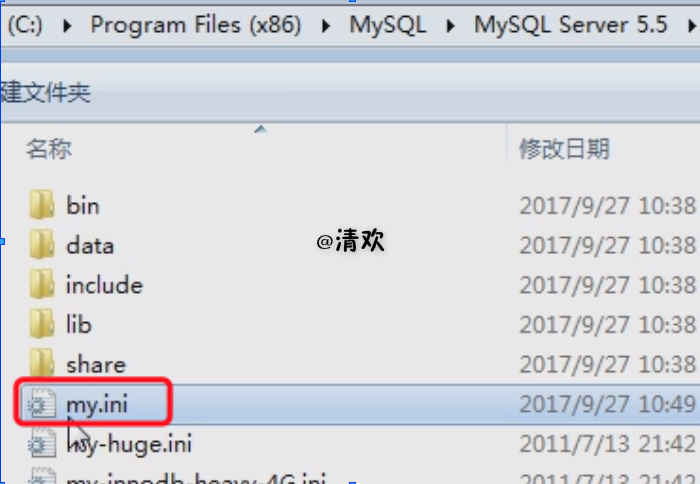
密码修改完成

\* 配置

\* 参照图解

\* 到选择字符集时停

### 配置文件



服务端的端口号，数据库文件的存储位置，使用的编码方式，数据库的存储引擎等都可以在这里配置。

注意：如果更改这个配置文件后，数据库的服务要重启一下才能生效。

## sql概述

SQL：Structure Query Language。（结构化查询语言）

SQL被美国国家标准局（ANSI）确定为关系型数据库语言的美国标准，后来被国际化标准组织（ISO）采纳为关系数据库语言的国际标准。

各数据库厂商都支持ISO的SQL标准。**普通话**

各数据库厂商在标准的基础上做了自己的扩展。**方言**

## Sql的分类

DDL(\*)（Data Definition Language）：数据定义语言，用来定义数据库对象：库、表、列等；

DML(\*\*)（Data Manipulation Language）：数据操作语言，用来定义数据库记录（数据）；

DCL（Data Control Language）：数据控制语言，用来定义访问权限和安全级别；

DQL(\*\*\*\*\*)（Data Query Language）：数据查询语言，用来查询记录（数据）。

\* 注意：sql语句以**;**结尾

### 4.1 DDL:操作数据库、表、列等

使用的关键字：**CREATE、 ALTER、 DROP**

#### \* 4.1.1操作数据库

创建

Create database mydb1;

Create database mydb2 character set gbk;

Create database mydb3 character set gbk COLLATE gbk\_chinese\_ci;

查询

查看当前数据库服务器中的所有数据库

Show databases;

查看前面创建的mydb2数据库的定义信息

Show create database mydb2;

删除前面创建的mydb3数据库

Drop database mydb3;

修改

查看服务器中的数据库，并把mydb2的字符集修改为utf8;

alter database mydb2 character set utf8;

删除

Drop database mydb3;

\* 其他：

查看当前使用的数据库

Select database();

切换数据库

Use mydb2;

#### \* 4.1.2操作数据表

\* 语法：

create table 表名(

字段1 字段类型,

字段2 字段类型,

...

字段n 字段类型

);

\* 常用数据类型：

int：整型

double：浮点型，例如double(5,2)表示最多5位，其中必须有2位小数，即最大值为999.99；

char：固定长度字符串类型； char(10) 'aaa ' 占10位

varchar：可变长度字符串类型； varchar(10) 'aaa' 占3为

text：字符串类型；

blob：字节类型；

date：日期类型，格式为：yyyy-MM-dd；

time：时间类型，格式为：hh:mm:ss

timestamp：时间戳类型 yyyy-MM-dd hh:mm:ss 会自动赋值

datetime:日期时间类型 yyyy-MM-dd hh:mm:ss

当前数据库中的所有表

SHOW TABLES;

查看表的字段信息

DESC employee;

在上面员工表的基本上增加一个image列。

ALTER TABLE employee ADD image blob;

修改job列，使其长度为60。

ALTER TABLE employee MODIFY job varchar(60);

删除image列,一次只能删一列。

ALTER TABLE employee DROP image;

表名改为user。

RENAME TABLE employee TO user;

查看表格的创建细节

SHOW CREATE TABLE user;

修改表的字符集为gbk

ALTER TABLE user CHARACTER SET gbk;

列名name修改为username

ALTER TABLE user CHANGE name username varchar(100);

删除表

DROP TABLE user ;

### DML操作(重要)

查询表中的所有数据

SELECT \* FROM 表名;

DML是对表中的数据进行增、删、改的操作。不要与DDL混淆了。

INSERT 、UPDATE、 DELETE

小知识：

在mysql中，字符串类型和日期类型都要用单引号括起来。

空值：null

#### 插入操作：INSERT:

语法： INSERT INTO 表名（列名1，列名2 ...）VALUES(列值1，列值2...);

注意：列名与列值的类型、个数、顺序要一一对应。

可以把列名当做java中的形参，把列值当做实参。

参不要超出列定义的长度。

如果插入空值，请使用null

插入的日期和字符一样，都使用引号括起来。

练习 ：

create table emp(

id int,

name varchar(100),

gender varchar(10),

birthday date,

salary float(10,2),

entry\_date date,

resume text

);

INSERT INTO emp(id,name,gender,birthday,salary,entry\_date,resume)

VALUES(1,'zhangsan','female','1990-5-10',10000,'2015-5-5-','good girl');

INSERT INTO emp(id,name,gender,birthday,salary,entry\_date,resume)

VALUES(2,'lisi','male','1995-5-10',10000,'2015-5-5','good boy');

INSERT INTO emp(id,name,gender,birthday,salary,entry\_date,resume)

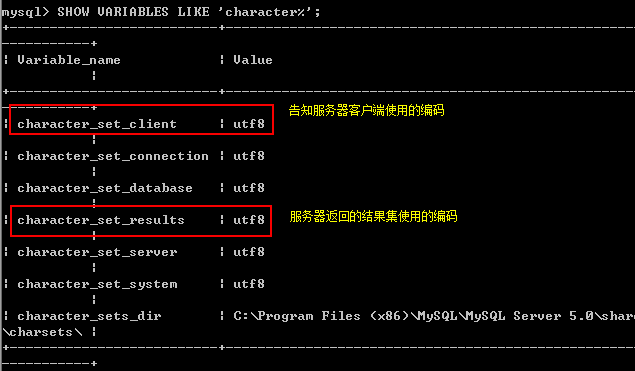
VALUES(3,'你好','male','1995-5-10',10000,'2015-5-5','good boy');



小知识：

查看数据库编码的具体信息

Show variables like ‘character%’;



临时更改客户端和服务器结果集的编码

Set character\_set\_client=gbk;

Set character\_set\_results=gbk;

#### 修改操作 UPDATE: 语法：UPDATE 表名 SET 列名1=列值1，列名2=列值2 。。。 WHERE 列名=值

练习：

将所有员工薪水修改为5000元。

UPDATE emp SET salary=5000

将姓名为’zs’的员工薪水修改为3000元。

UPDATE emp SET salary=3000 WHERE name=’ zhangsan’;

将姓名为’aaa’的员工薪水修改为4000元,job改为ccc。

UPDATE emp SET salary=4000,gender='female' WHERE name='lisi';

将wu的薪水在原有基础上增加1000元。

UPDATE emp SET salary=salary+1000 WHERE gender='male';

#### 4.2.3 删除操作 DELETE:

语法 ： DELETE 表名 【WHERE 列名=值】

练习 ：

删除表中名称为’zs’的记录。

DELETE FROM emp WHERE name=‘zs’;

删除表中所有记录。

DELETE FROM emp;

使用truncate删除表中记录。

TRUNCATE TABLE emp;

DELETE 删除表中的数据，表结构还在;删除后的数据可以找回

TRUNCATE 删除是把表直接DROP掉，然后再创建一个同样的新表。

删除的数据不能找回。执行速度比DELETE快。

### DQL操作

DQL数据查询语言 （重要）

数据库执行DQL语句不会对数据进行改变，而是让数据库发送结果集给客户端。

查询返回的结果集是一张虚拟表。

查询关键字：SELECT

语法： SELECT 列名 FROM表名

【WHERE --> BROUP BY -->HAVING--> ORDER BY】

语法：

SELECT selection\_list /\*要查询的列名称\*/

FROM table\_list /\*要查询的表名称\*/

WHERE condition /\*行条件\*/

GROUP BY grouping\_columns /\*对结果分组\*/

HAVING condition /\*分组后的行条件\*/

ORDER BY sorting\_columns /\*对结果分组\*/

LIMIT offset\_start, row\_count /\*结果限定\*/

创建名：

* 学生表：stu

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **sid** | char(6) | 学生学号 |
| **sname** | varchar(50) | 学生姓名 |
| **age** | int | 学生年龄 |
| **gender** | varchar(50) | 学生性别 |

|  |
| --- |
| CREATE TABLE stu (  sid CHAR(6),  sname VARCHAR(50),  age INT,  gender VARCHAR(50)  ); |
| INSERT INTO stu VALUES('S\_1001', 'liuYi', 35, 'male');  INSERT INTO stu VALUES('S\_1002', 'chenEr', 15, 'female');  INSERT INTO stu VALUES('S\_1003', 'zhangSan', 95, 'male');  INSERT INTO stu VALUES('S\_1004', 'liSi', 65, 'female');  INSERT INTO stu VALUES('S\_1005', 'wangWu', 55, 'male');  INSERT INTO stu VALUES('S\_1006', 'zhaoLiu', 75, 'female');  INSERT INTO stu VALUES('S\_1007', 'sunQi', 25, 'male');  INSERT INTO stu VALUES('S\_1008', 'zhouBa', 45, 'female');  INSERT INTO stu VALUES('S\_1009', 'wuJiu', 85, 'male');  INSERT INTO stu VALUES('S\_1010', 'zhengShi', 5, 'female');  INSERT INTO stu VALUES('S\_1011', 'xxx', NULL, NULL); |

* 雇员表：emp

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **empno** | int | 员工编号 |
| **ename** | varchar(50) | 员工姓名 |
| **job** | varchar(50) | 员工工作 |
| **mgr** | int | 领导编号 |
| **hiredate** | date | 入职日期 |
| **sal** | decimal(7,2) | 月薪 |
| **comm** | decimal(7,2) | 奖金 |
| **deptno** | int | 部分编号 |

|  |
| --- |
| CREATE TABLE emp(  empno INT,  ename VARCHAR(50),  job VARCHAR(50),  mgr INT,  hiredate DATE,  sal DECIMAL(7,2),  comm decimal(7,2),  deptno INT  ) ; |
| INSERT INTO emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20);  INSERT INTO emp values(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30);  INSERT INTO emp values(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30);  INSERT INTO emp values(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20);  INSERT INTO emp values(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30);  INSERT INTO emp values(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30);  INSERT INTO emp values(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10);  INSERT INTO emp values(7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,NULL,20);  INSERT INTO emp values(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10);  INSERT INTO emp values(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30);  INSERT INTO emp values(7876,'ADAMS','CLERK',7788,'1987-05-23',1100,NULL,20);  INSERT INTO emp values(7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30);  INSERT INTO emp values(7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20);  INSERT INTO emp values(7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10); |

* 部分表：dept

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **deptno** | int | 部分编码 |
| **dname** | varchar(50) | 部分名称 |
| **loc** | varchar(50) | 部分所在地点 |

|  |
| --- |
| CREATE TABLE dept(  deptno INT,  dname varchar(14),  loc varchar(13)  ); |
| INSERT INTO dept values(10, 'ACCOUNTING', 'NEW YORK');  INSERT INTO dept values(20, 'RESEARCH', 'DALLAS');  INSERT INTO dept values(30, 'SALES', 'CHICAGO');  INSERT INTO dept values(40, 'OPERATIONS', 'BOSTON'); |

#### 基础查询

##### 查询所有列

**SELECT \* FROM stu;**

##### 查询指定列

**SELECT sid, sname, age FROM stu;**

#### 条件查询

##### 条件查询介绍

条件查询就是在查询时给出WHERE子句，在WHERE子句中可以使用如下运算符及关键字：

* =、!=、<>、<、<=、>、>=；
* BETWEEN…AND；
* IN(set)；
* IS NULL；
* AND；
* OR；
* NOT；

##### 查询性别为女，并且年龄50的记录

**SELECT \* FROM stu**

**WHERE gender='female' AND ge<50;**

##### 查询学号为S\_1001，或者姓名为liSi的记录

**SELECT \* FROM stu**

**WHERE sid ='S\_1001' OR sname='liSi';**

##### 查询学号为S\_1001，S\_1002，S\_1003的记录

**SELECT \* FROM stu**

**WHERE sid IN ('S\_1001','S\_1002','S\_1003');**

##### 查询学号不是S\_1001，S\_1002，S\_1003的记录

**SELECT \* FROM tab\_student**

**WHERE s\_number NOT IN ('S\_1001','S\_1002','S\_1003');**

##### 查询年龄为null的记录

**SELECT \* FROM stu**

**WHERE age IS NULL;**

##### 查询年龄在20到40之间的学生记录

**SELECT \***

**FROM stu**

**WHERE age>=20 AND age<=40;**

或者

**SELECT \***

**FROM stu**

**WHERE age BETWEEN 20 AND 40;**

##### 查询性别非男的学生记录

**SELECT \***

**FROM stu**

**WHERE gender!='male';**

或者

**SELECT \***

**FROM stu**

**WHERE gender<>'male';**

或者

**SELECT \***

**FROM stu**

**WHERE NOT gender='male';**

##### 查询姓名不为null的学生记录

**SELECT \***

**FROM stu**

**WHERE NOT sname IS NULL;**

或者

**SELECT \***

**FROM stu**

**WHERE sname IS NOT NULL;**

#### 3　模糊查询

当想查询姓名中包含a字母的学生时就需要使用模糊查询了。模糊查询需要使用关键字LIKE。

通配符:

\_ 任意一个字母

%：任意0~n个字母

'张%'

###### 3.1　查询姓名由5个字母构成的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_\_\_\_\_';**

模糊查询必须使用LIKE关键字。其中 “\_”匹配任意一个字母，5个“\_”表示5个任意字母。

###### 3.2　查询姓名由5个字母构成，并且第5个字母为“i”的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_\_\_\_i';**

###### 3.3　查询姓名以“z”开头的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE 'z%';**

其中“%”匹配0~n个任何字母。

###### 3.4　查询姓名中第2个字母为“i”的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_i%';**

###### 3.5　查询姓名中包含“a”字母的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '%a%';**

#### 4　字段控制查询

###### 4.1　去除重复记录

去除重复记录（两行或两行以上记录中系列的上的数据都相同），例如emp表中sal字段就存在相同的记录。当只查询emp表的sal字段时，那么会出现重复记录，那么想去除重复记录，需要使用DISTINCT：

**SELECT DISTINCT sal FROM emp;**

###### 4.2　查看雇员的月薪与佣金之和

　　因为sal和comm两列的类型都是数值类型，所以可以做加运算。如果sal或comm中有一个字段不是数值类型，那么会出错。

**SELECT \*,sal+comm FROM emp;**

comm列有很多记录的值为NULL，因为任何东西与NULL相加结果还是NULL，所以结算结果可能会出现NULL。下面使用了把NULL转换成数值0的函数IFNULL：

**SELECT \*,sal+IFNULL(comm,0) FROM emp;**

###### 4.3　给列名添加别名

在上面查询中出现列名为sal+IFNULL(comm,0)，这很不美观，现在我们给这一列给出一个别名，为total：

**SELECT \*, sal+IFNULL(comm,0) AS total FROM emp;**

给列起别名时，是可以省略AS关键字的：

**SELECT \*,sal+IFNULL(comm,0) total FROM emp;**

#### 5　排序

###### 5.1　查询所有学生记录，按年龄升序排序

**SELECT \***

**FROM stu**

**ORDER BY sage ASC;**

或者

**SELECT \***

**FROM stu**

**ORDER BY sage;**

###### 5.2　查询所有学生记录，按年龄降序排序

**SELECT \***

**FROM stu**

**ORDER BY age DESC;**

###### 5.3　查询所有雇员，按月薪降序排序，如果月薪相同时，按编号升序排序

**SELECT \* FROM emp**

**ORDER BY sal DESC,empno ASC;**

#### 6　聚合函数 sum avg max min count

聚合函数是用来做纵向运算的函数：

* COUNT()：统计指定列不为NULL的记录行数；
* MAX()：计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；
* MIN()：计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；
* SUM()：计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为0；
* AVG()：计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为0；

###### 6.1　COUNT

当需要纵向统计时可以使用COUNT()。

* 查询emp表中记录数：

**SELECT COUNT(\*) AS cnt FROM emp;**

* 查询emp表中有佣金的人数：

**SELECT COUNT(comm) cnt FROM emp;**

注意，因为count()函数中给出的是comm列，那么只统计comm列非NULL的行数。

* 查询emp表中月薪大于2500的人数：

**SELECT COUNT(\*) FROM emp**

**WHERE sal > 2500;**

* 统计月薪与佣金之和大于2500元的人数：

**SELECT COUNT(\*) AS cnt FROM emp WHERE sal+IFNULL(comm,0) > 2500;**

* 查询有佣金的人数，以及有领导的人数：

**SELECT COUNT(comm), COUNT(mgr) FROM emp;**

###### 6.2　SUM和AVG

当需要纵向求和时使用sum()函数。

* 查询所有雇员月薪和：

**SELECT SUM(sal) FROM emp;**

* 查询所有雇员月薪和，以及所有雇员佣金和：

**SELECT SUM(sal), SUM(comm) FROM emp;**

* 查询所有雇员月薪+佣金和：

**SELECT SUM(sal+IFNULL(comm,0)) FROM emp;**

* 统计所有员工平均工资：

**SELECT AVG(sal) FROM emp;**

###### 6.3　MAX和MIN

* 查询最高工资和最低工资：

**SELECT MAX(sal), MIN(sal) FROM emp;**

#### 7　分组查询

当需要分组查询时需要使用GROUP BY子句，例如查询每个部门的工资和，这说明要使用部分来分组。

注：凡和聚合函数同时出现的列名，则一定要写在group by 之后

###### 7.1　分组查询

* 查询每个部门的部门编号和每个部门的工资和：

**SELECT deptno, SUM(sal)**

**FROM emp**

**GROUP BY deptno;**

* 查询每个部门的部门编号以及每个部门的人数：

**SELECT deptno,COUNT(\*)**

**FROM emp**

**GROUP BY deptno;**

* 查询每个部门的部门编号以及每个部门工资大于1500的人数：

**SELECT deptno,COUNT(\*)**

**FROM emp**

**WHERE sal>1500**

**GROUP BY deptno;**

###### 7.2　HAVING子句

* 查询工资总和大于9000的部门编号以及工资和：

**SELECT deptno, SUM(sal)**

**FROM emp**

**GROUP BY deptno**

**HAVING SUM(sal) > 9000;**

注：having与where的区别:

1.having是在分组后对数据进行过滤.

where是在分组前对数据进行过滤

2.having后面可以使用分组函数(统计函数)

where后面不可以使用分组函数。

WHERE是对分组前记录的条件，如果某行记录没有满足WHERE子句的条件，那么这行记录不会参加分组；而HAVING是对分组后数据的约束。

#### 8　LIMIT

LIMIT用来限定查询结果的起始行，以及总行数。

###### 8.1　查询5行记录，起始行从0开始

**SELECT \* FROM emp LIMIT 0, 5;**

注意，起始行从0开始，即第一行开始！

###### 8.2　查询10行记录，起始行从3开始

**SELECT \* FROM emp LIMIT 3, 10;**

###### 8.3　分页查询

如果一页记录为10条，希望查看第3页记录应该怎么查呢？

* 第一页记录起始行为0，一共查询10行；
* 第二页记录起始行为10，一共查询10行；
* 第三页记录起始行为20，一共查询10行；

###### 8.3　分页查询

查询语句书写顺序：select – from- where- group by- having- order by-limit

查询语句执行顺序：from - where -group by - having - select - order by-limit

## 数据的完整性

作用：保证用户输入的数据保存到数据库中是正确的。

确保数据的完整性 = 在创建表时给表中添加约束

完整性的分类：

实体完整性:

域完整性:

引用完整性:

### 实体完整性

实体：即表中的一行(一条记录)代表一个实体（entity）

实体完整性的作用：标识每一行数据不重复。

**约束类型： 主键约束（primary key） 唯一约束(unique) 自动增长列(auto\_increment)**

#### 主键约束（primary key）

注：每个表中要有一个主键。

特点：数据唯一，且不能为null

例：

第一种添加方式：

CREATE TABLE student(

id int primary key,

name varchar(50)

);

第二种添加方式：此种方式优势在于，可以创建联合主键

CREATE TABLE student(

id int,

name varchar(50),

primary key(id)

);

CREATE TABLE student(

classid int,

stuid int,

name varchar(50),

primary key(classid，stuid)

);

第三种添加方式：

CREATE TABLE student(

id int,

name varchar(50)

);

ALTER TABLE student ADD PRIMARY KEY (id);

#### 1.2唯一约束(unique)： 特点：数据不能重复。

#### CREATE TABLE student(

Id int primary key,

Name varchar(50) unique

);

#### 1.3自动增长列(auto\_increment)

#### sqlserver数据库 (identity) oracle数据库( sequence)

给主键添加自动增长的数值，列只能是整数类型

CREATE TABLE student(

Id int primary key auto\_increment,

Name varchar(50)

);

INSERT INTO student(name) values(‘tom’);

### 域完整性

域完整性的作用：限制此单元格的数据正确，不对照此列的其它单元格比较

域代表当前单元格

域完整性约束：数据类型 非空约束（not null） 默认值约束(default)

check约束（mysql不支持）check(sex='男' or sex='女')

#### 1.1 数据类型:（数值类型、日期类型、字符串类型）

#### 1.2 非空约束：not null

CREATE TABLE student(

Id int pirmary key,

Name varchar(50) not null,

Sex varchar(10)

);

INSERT INTO student values(1,’tom’,null);

#### 1.3 默认值约束 default

CREATE TABLE student(

Id int pirmary key,

Name varchar(50) not null,

Sex varchar(10) default ‘男’

);

insert into student1 values(1,'tom','女');

insert into student1 values(2,'jerry',default);

### 引用完整性（参照完整性）

外键约束：FOREIGN KEY

例：

CREATE TABLE student(

sid int pirmary key,

name varchar(50) not null,

sex varchar(10) default ‘男’

);

create table score(

id int,

score int,

sid int , -- 外键列的数据类型一定要与主键的类型一致

CONSTRAINT fk\_score\_sid foreign key (sid) references student(id)

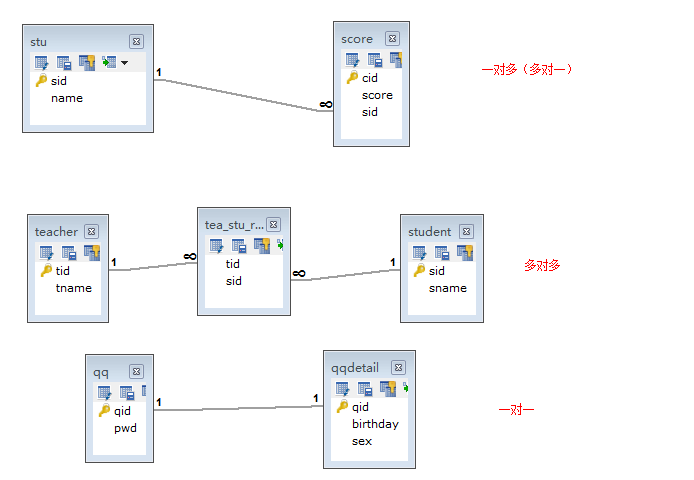
);

第二种添加外键方式。

ALTER TABLE score1 ADD CONSTRAINT fk\_stu\_score FOREIGN KEY(sid) REFERENCES stu(id);

### 4、　表与表之间的关系

* 一对一：例如t\_person表和t\_card表，即人和身份证。这种情况需要找出主从关系，即谁是主表，谁是从表。人可以没有身份证，但身份证必须要有人才行，所以人是主表，而身份证是从表。设计从表可以有两种方案：
* 在t\_card表中添加外键列（相对t\_user表），并且给外键添加唯一约束；
* 给t\_card表的主键添加外键约束（相对t\_user表），即t\_card表的主键也是外键。
* 一对多（多对一）：最为常见的就是一对多！一对多和多对一，这是从哪个角度去看得出来的。t\_user和t\_section的关系，从t\_user来看就是一对多，而从t\_section的角度来看就是多对一！这种情况都是在多方创建外键！
* 多对多：例如t\_stu和t\_teacher表，即一个学生可以有多个老师，而一个老师也可以有多个学生。这种情况通常需要创建中间表来处理多对多关系。例如再创建一张表t\_stu\_tea表，给出两个外键，一个相对t\_stu表的外键，另一个相对t\_teacher表的外键。



## 多表查询（重要）

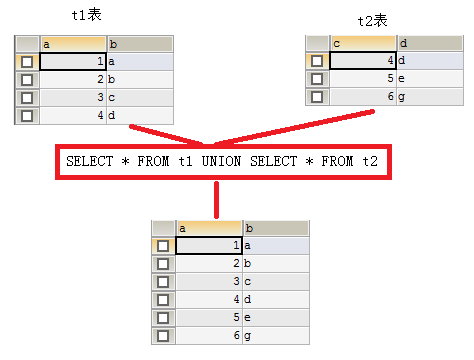
多表查询有如下几种：

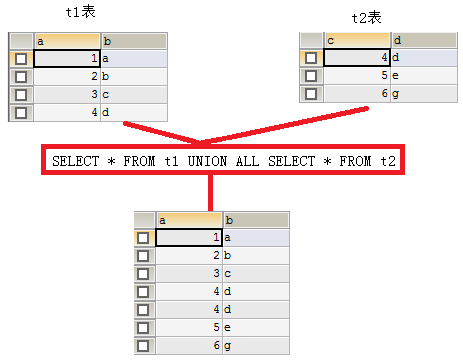
* 合并结果集；UNION 、 UNION ALL
* 连接查询
* 内连接 [INNER] JOIN ON
* 外连接 OUTER JOIN ON
* 左外连接 LEFT [OUTER] JOIN
* 右外连接 RIGHT [OUTER] JOIN
* 全外连接（MySQL不支持）FULL JOIN
* 自然连接 NATURAL JOIN
* 子查询

### 合并结果集

1. 作用：合并结果集就是把两个select语句的查询结果合并到一起！
2. 合并结果集有两种方式：

* UNION：去除重复记录，例如：SELECT \* FROM t1 UNION SELECT \* FROM t2；
* UNION ALL：不去除重复记录，例如：SELECT \* FROM t1 UNION ALL SELECT \* FROM t2。

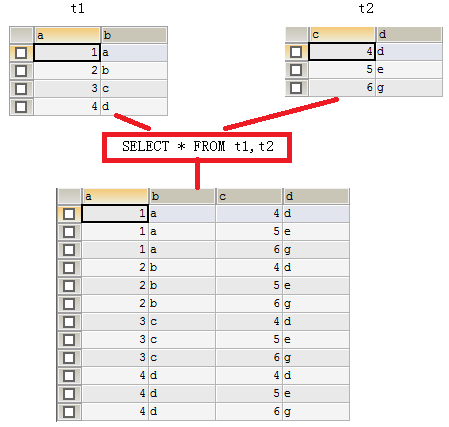




1. 要求：**被合并的两个结果：列数、列类型必须相同。**

### 连接查询 （非常重要）

连接查询就是求出多个表的乘积，例如t1连接t2，那么查询出的结果就是t1\*t2。

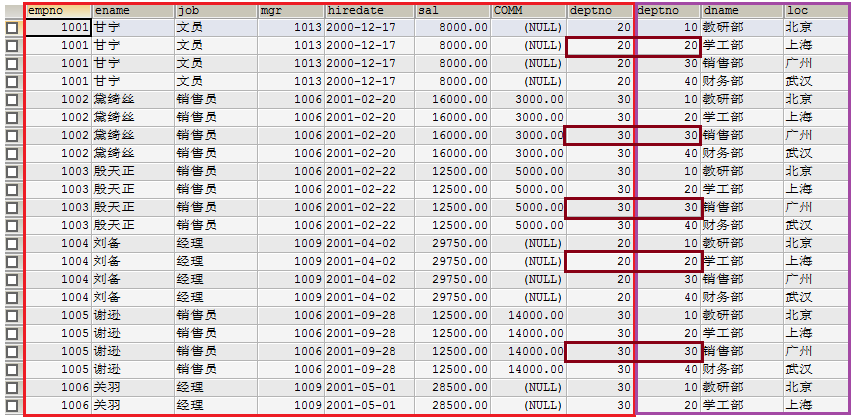


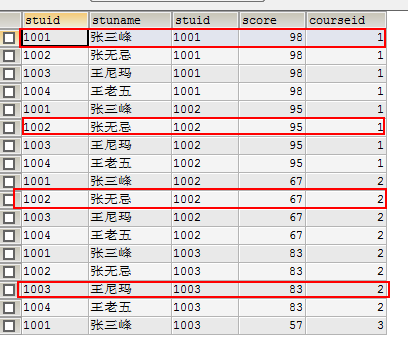
连接查询会产生笛卡尔积，假设集合A={a,b}，集合B={0,1,2}，则两个集合的笛卡尔积为{(a,0),(a,1),(a,2),(b,0),(b,1),(b,2)}。可以扩展到多个集合的情况。

那么多表查询产生这样的结果并不是我们想要的，那么怎么去除重复的，不想要的记录呢，当然是通过条件过滤。通常要查询的多个表之间都存在关联关系，那么就通过关联关系去除笛卡尔积。

你能想像到emp和dept表连接查询的结果么？emp一共14行记录，dept表一共4行记录，那么连接后查询出的结果是56行记录。

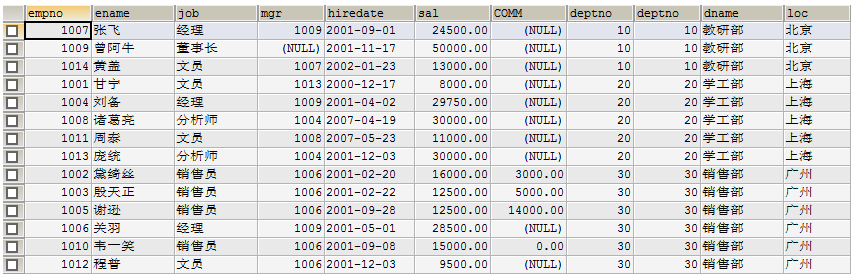
也就你只是想在查询emp表的同时，把每个员工的所在部门信息显示出来，那么就需要使用主外键来去除无用信息了。





**使用主外键关系做为条件来去除无用信息**

|  |
| --- |
| SELECT \* FROM emp,dept WHERE emp.deptno=dept.deptno; |



上面查询结果会把两张表的所有列都查询出来，也许你不需要那么多列，这时就可以指定要查询的列了。

|  |
| --- |
| SELECT emp.ename,emp.sal,emp.comm,dept.dname  FROM emp,dept  WHERE emp.deptno=dept.deptno; |



还可以为表指定别名，然后在引用列时使用别名即可。

|  |
| --- |
| SELECT e.ename,e.sal,e.comm,d.dname  FROM emp AS e,dept AS d  WHERE e.deptno=d.deptno; |

#### 2.1　内连接

上面的连接语句就是内连接，但它不是SQL标准中的查询方式，可以理解为方言！SQL标准的内连接为：

|  |
| --- |
| SELECT \*  FROM emp e  INNER JOIN dept d  ON e.deptno=d.deptno; |

内连接的特点：查询结果必须满足条件。例如我们向emp表中插入一条记录：



　　其中deptno为50，而在dept表中只有10、20、30、40部门，那么上面的查询结果中就不会出现“张三”这条记录，因为它不能满足e.deptno=d.deptno这个条件。

2.2　外连接（左连接、右连接）

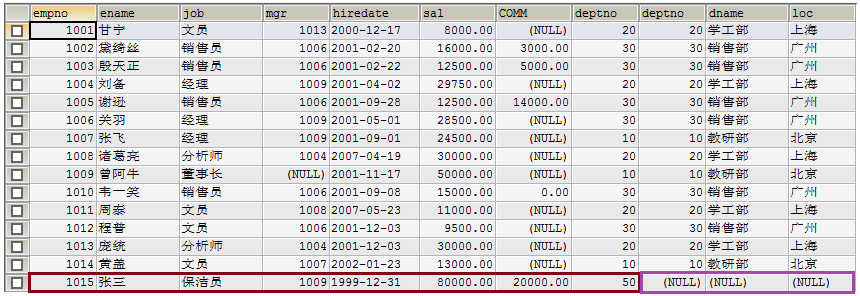
外连接的特点：查询出的结果存在不满足条件的可能。

左连接：

|  |
| --- |
| SELECT \* FROM emp e  LEFT OUTER JOIN dept d  ON e.deptno=d.deptno; |

左连接是先查询出左表（即以左表为主），然后查询右表，右表中满足条件的显示出来，不满足条件的显示NULL。

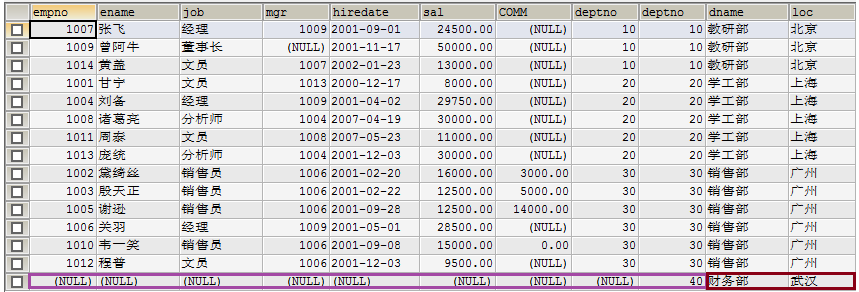
这么说你可能不太明白，我们还是用上面的例子来说明。其中emp表中“张三”这条记录中，部门编号为50，而dept表中不存在部门编号为50的记录，所以“张三”这条记录，不能满足e.deptno=d.deptno这条件。但在左连接中，因为emp表是左表，所以左表中的记录都会查询出来，即“张三”这条记录也会查出，但相应的右表部分显示NULL。



#### 2.3　右连接

右连接就是先把右表中所有记录都查询出来，然后左表满足条件的显示，不满足显示NULL。例如在dept表中的40部门并不存在员工，但在右连接中，如果dept表为右表，那么还是会查出40部门，但相应的员工信息为NULL。

|  |
| --- |
| SELECT \* FROM emp e  RIGHT OUTER JOIN dept d  ON e.deptno=d.deptno; |



**连接查询心得**：

连接不限与两张表，连接查询也可以是三张、四张，甚至N张表的连接查询。通常连接查询不可能需要整个笛卡尔积，而只是需要其中一部分，那么这时就需要使用条件来去除不需要的记录。这个条件大多数情况下都是使用主外键关系去除。

两张表的连接查询一定有一个主外键关系，三张表的连接查询就一定有两个主外键关系，所以在大家不是很熟悉连接查询时，首先要学会去除无用笛卡尔积，那么就是用主外键关系作为条件来处理。如果两张表的查询，那么至少有一个主外键条件，三张表连接至少有两个主外键条件*。*

### 自然连接

大家也都知道，连接查询会产生无用笛卡尔积，我们通常使用主外键关系等式来去除它。而自然连接无需你去给出主外键等式，它会自动找到这一等式：

* 两张连接的表中名称和类型完全一致的列作为条件，例如emp和dept表都存在deptno列，并且类型一致，所以会被自然连接找到！

当然自然连接还有其他的查找条件的方式，但其他方式都可能存在问题！

|  |
| --- |
| SELECT \* FROM emp NATURAL JOIN dept;  SELECT \* FROM emp NATURAL LEFT JOIN dept;  SELECT \* FROM emp NATURAL RIGHT JOIN dept; |

### 子查询（非常重要）

一个select语句中包含另一个完整的select语句。

子查询就是嵌套查询，即SELECT中包含SELECT，如果一条语句中存在两个，或两个以上SELECT，那么就是子查询语句了。

* 子查询出现的位置：
* where后，作为条为被查询的一条件的一部分；
* from后，作表；
* 当子查询出现在where后作为条件时，还可以使用如下关键字：
* any
* all
* 子查询结果集的形式：
* 单行单列（用于条件）
* 单行多列（用于条件）
* 多行单列（用于条件）
* 多行多列（用于表）

练习：

1. **工资高于JONES的员工。**

分析：

查询条件：工资>JONES工资，其中JONES工资需要一条子查询。

第一步：查询JONES的工资

|  |
| --- |
| SELECT sal FROM emp WHERE ename='JONES' |

第二步：查询高于甘宁工资的员工

|  |
| --- |
| SELECT \* FROM emp WHERE sal > (${第一步}) |

结果：

|  |
| --- |
| SELECT \* FROM emp WHERE sal > (SELECT sal FROM emp WHERE ename='JONES') |

**2、查询与SCOTT同一个部门的员工。**

* 子查询作为条件
* 子查询形式为单行单列

**3、工资高于30号部门所有人的员工信息**

分析：

SELECT \* FROM emp WHERE sal>(

SELECT MAX(sal) FROM emp WHERE deptno=30);

查询条件：工资高于30部门所有人工资，其中30部门所有人工资是子查询。高于所有需要使用all关键字。

第一步：查询30部门所有人工资

|  |
| --- |
| SELECT sal FROM emp WHERE deptno=30; |

第二步：查询高于30部门所有人工资的员工信息

|  |
| --- |
| SELECT \* FROM emp WHERE sal > ALL (${第一步}) |

结果：

|  |
| --- |
| SELECT \* FROM emp WHERE sal > ALL (SELECT sal FROM emp WHERE deptno=30) |

* 子查询作为条件
* 子查询形式为多行单列（当子查询结果集形式为多行单列时可以使用ALL或ANY关键字）

**4、查询工作和工资与MARTIN（马丁）完全相同的员工信息**

分析：

查询条件：工作和工资与MARTIN完全相同，这是子查询

第一步：查询出MARTIN的工作和工资

|  |
| --- |
| SELECT job,sal FROM emp WHERE ename='MARTIN' |

第二步：查询出与MARTIN工作和工资相同的人

|  |
| --- |
| SELECT \* FROM emp WHERE (job,sal) IN (${第一步}) |

结果：

|  |
| --- |
| SELECT \* FROM emp WHERE (job,sal) IN (SELECT job,sal FROM emp WHERE ename='MARTIN') |

**5、有2个以上直接下属的员工信息**

SELECT \* FROM emp WHERE empno IN(

SELECT mgr FROM emp GROUP BY mgr HAVING COUNT(mgr)>=2);

* 子查询作为条件
* 子查询形式为单行多列

**5、查询员工编号为7788的员工名称、员工工资、部门名称、部门地址**

分析：（无需子查询）

查询列：员工名称、员工工资、部门名称、部门地址

查询表：emp和dept，分析得出，不需要外连接（外连接的特性：某一行（或某些行）记录上会出现一半有值，一半为NULL值）

条件：员工编号为**7788**

第一步：去除多表，只查一张表，这里去除部门表，只查员工表

|  |
| --- |
| SELECT ename, sal FROM emp e WHERE empno=**7788** |

第二步：让第一步与dept做内连接查询，添加主外键条件去除无用笛卡尔积

|  |
| --- |
| SELECT e.ename, e.sal, d.dname, d.loc  FROM emp e, dept d  WHERE e.deptno=d.deptno AND empno=**7788** |

第二步中的dept表表示所有行所有列的一张完整的表，这里可以把dept替换成所有行，但只有dname和loc列的表，这需要子查询。

第三步：查询dept表中dname和loc两列，因为deptno会被作为条件，用来去除无用笛卡尔积，所以需要查询它。

|  |
| --- |
| SELECT dname,loc,deptno FROM dept; |

第四步：替换第二步中的dept

|  |
| --- |
| SELECT e.ename, e.sal, d.dname, d.loc  FROM emp e, (SELECT dname,loc,deptno FROM dept) d  WHERE e.deptno=d.deptno AND e.empno=**7788** |

* 子查询作为表
* 子查询形式为多行多列

### 自连接:自己连接自己，起别名

求7369员工编号、姓名、经理编号和经理姓名

SELECT e1.empno , e1.ename,e2.mgr,e2.ename

FROM emp e1, emp e2

WHERE e1.mgr = e2.empno AND e1.empno = 7369;

练习：

求各个部门薪水最高的员工所有信息

select e.\* from emp e,

--部门最高工资

(select max(sal) maxsal,deptno from emp

group by deptno) a

where e.deptno = a.deptno

and e.sal =a.maxsal

## MySQL中的函数

## MySQL数据库的备份与恢复

### 生成SQL脚本 导出数据

在控制台使用mysqldump命令可以用来生成指定数据库的脚本文本，但要注意，脚本文本中只包含数据库的内容，而不会存在创建数据库的语句！所以在恢复数据时，还需要自已手动创建一个数据库之后再去恢复数据。

|  |
| --- |
| mysqldump –u用户名 –p密码 数据库名>生成的脚本文件路径 |



现在可以在C盘下找到mydb1.sql文件了！

注意，mysqldump命令是在Windows控制台下执行，无需登录mysql！！！

### 执行SQL脚本 恢复数据

**前提：必须先创建数据库名**

**执行SQL脚本需要登录mysql**，然后进入指定数据库，才可以执行SQL脚本！！！

执行SQL脚本不只是用来恢复数据库，也可以在平时编写SQL脚本，然后使用执行SQL 脚本来操作数据库！大家都知道，在黑屏下编写SQL语句时，就算发现了错误，可能也不能修改了。所以我建议大家使用脚本文件来编写SQL代码，然后执行之！

|  |
| --- |
| SOURCE C:\mydb1.sql |



注意，在执行脚本时需要先行核查当前数据库中的表是否与脚本文件中的语句有冲突！例如在脚本文件中存在create table a的语句，而当前数据库中已经存在了a表，那么就会出错！

还可以通过下面的方式来执行脚本文件：

mysql -uroot -p123 mydb1<c:\mydb1.sql

|  |
| --- |
| mysql –u用户名 –p密码 数据库<要执行脚本文件路径 |



**这种方式无需登录mysql！**

**注意：在CMD下 命令不能加;**

# MySQL锁机制

# HeiMa\_WEB08

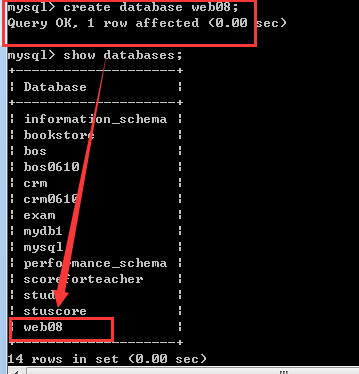
MySQL与JDBC回顾

## 对数据库的操作

### 创建一个库

create database 库名

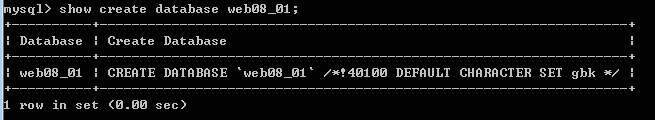
create database 库名 character set 编码



创建带有编码的



查看编码：



### 删除一个库

drop database 库名

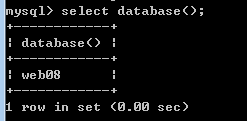


### 使用库

use 库名



### 4.查看当前正在操作的库



## 对数据库表的操作

### 1.创建一张表

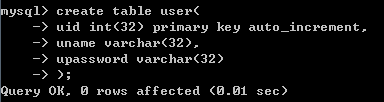
create table 表名(

字段名 类型(长度) [约束],

字段名 类型(长度) [约束],

字段名 类型(长度) [约束]

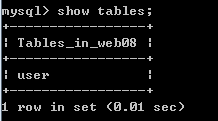
);



### 2.查看数据库表

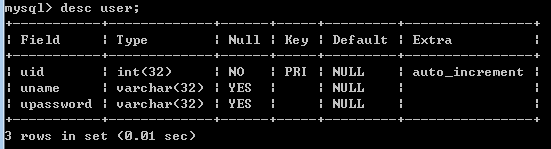
创建完成后，我们可以查看数据库表

show tables;



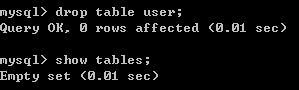
查看表的结构

desc 表名



### 3.删除一张表

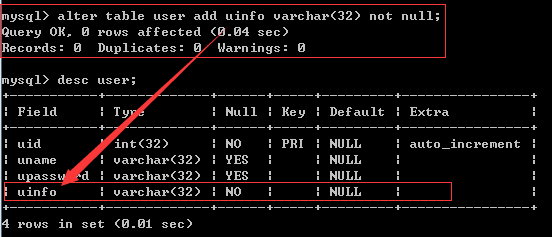
drop table 表名



### 4.修改表

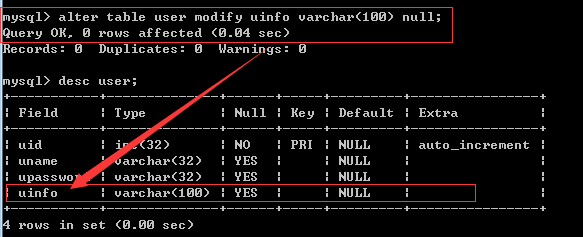
#### 4.1 添加一列

alter table 表名 add 字段名 类型(长度) [约束]



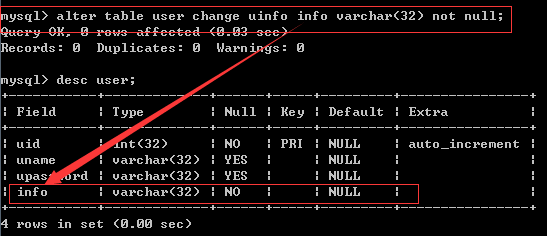
#### 4.2 修改列的类型(长度、约束)

alter table 表名 modify 要修改的字段名 类型(长度) [约束]



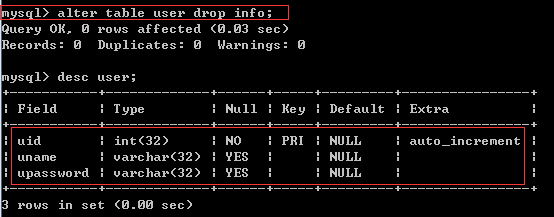
#### 4.3 修改列的列名

alter table 表名 change 旧列名 新列名 类型(长度) [约束]



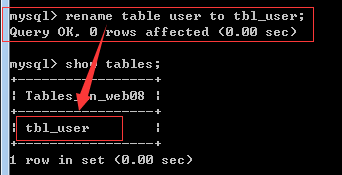
#### 4.4 删除表的列

alter table 表名 drop 列名



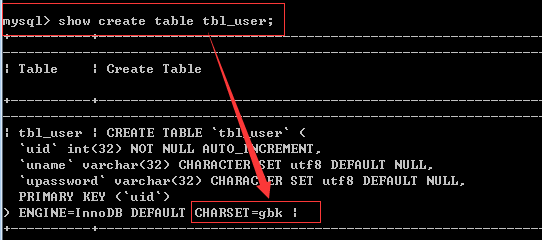
#### 4.5 修改表名

rename table 表名 to 新表名

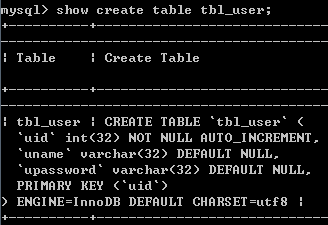


#### 4.6 修改表的字符集

alter table 表名 character set 编码



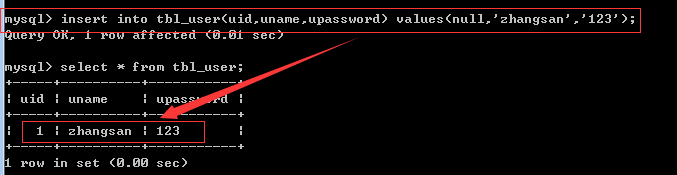
查看当前表的编码



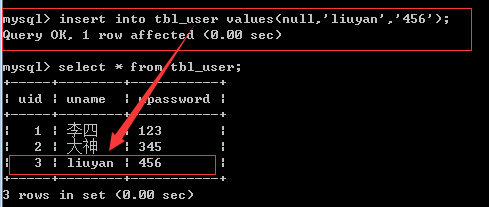
## 对数据库表记录进行操作(修改)

### 1.插入记录

insert into 表名(列名1,列名2,列名3……) values(值1,值2,值3……)



insert into 表名 values(值1,值2,值3……)



#### 1.1 插入数据中文乱码问题解决办法

方式一：【不建议！】

直接修改数据库安装目录里面的my.ini文件的第57行



方式二：

set names gbk;



### 2.修改表记录

#### 2.1 不带条件的

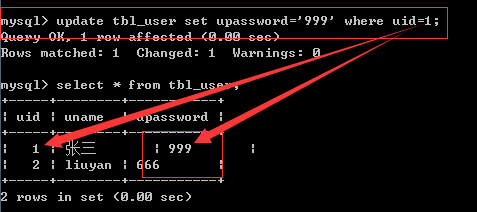
update 表名 set 字段名=值, 字段名=值, 字段名=值……



它会将该列的所有记录都更改

#### 2.2 带条件的

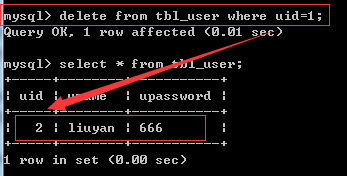
update 表名 set字段名=值, 字段名=值, 字段名=值…… where 条件



### 3.删除表记录

#### 3.1 带条件的

delete from 表名 where 条件



注意，删除后，uid不会重置！

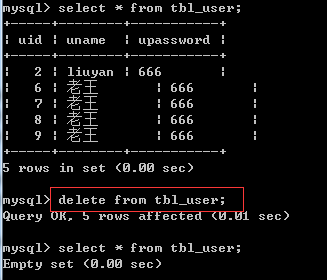
#### 3.2.不带条件的

先准备数据

insert into tbl\_user values(null,’老王’,’666’);

删除操作

delete from 表名;



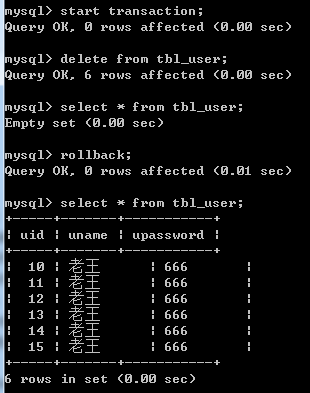
#### 3.3 面试题

说说delete与truncate的区别？

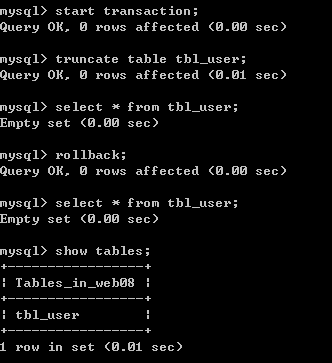
delete删除的时候是一条一条的删除记录，它配合事务，可以将删除的数据找回。

truncate删除，它是将整个表摧毁，然后再创建一张一模一样的表。它删除的数据无法找回。

Delete操作演示：



Truncate操作演示



注意：delete删除，uid不会重置！而使用truncate操作，uid会重置

### 4.查询操作

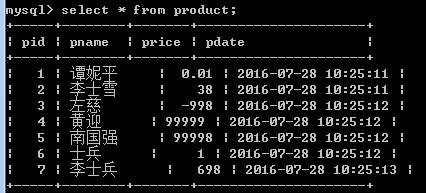
语法：

select [distinct] \*| 列名，列名 from 表名 [where条件]

#### 4.1 简单查询

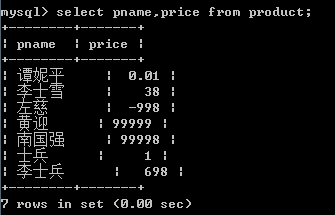
1.查询所有商品

select \* from product；



2. 查询商品名和商品价格

select pname,price from product;



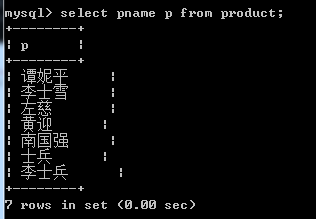
3.查询所有商品信息使用表别名

select \* from product as p;



4.查询商品名，使用列别名

select pname as p from product



5.去掉重复值(按照价格)

select distinct(price) from product;

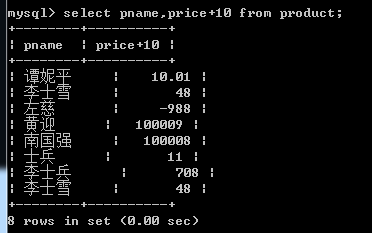
先准备数据：

insert into product values (null,'李士雪',38,null);



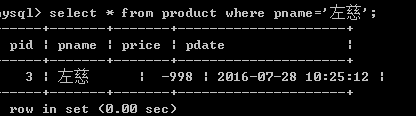
6.将所有的商品的价格+10进行显示

select pname,price+10 from product;

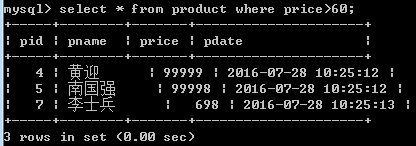


#### 4.2 条件查询

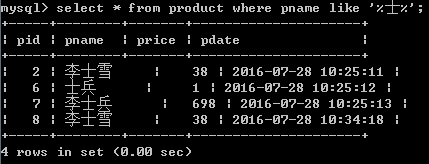
1.查询商品名称为"左慈"的商品信息



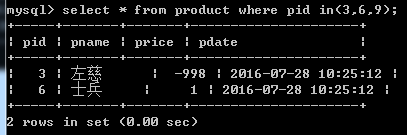
2.查询价格>60元的所有商品信息



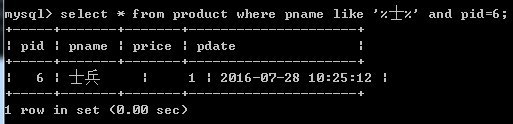
3.查询商品名称含有"士"字的商品信息



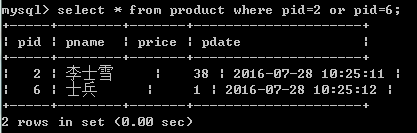
4.查询商品id在(3,6,9)范围内的所有商品信息



5.查询商品名称含有"士"字并且id为6的商品信息

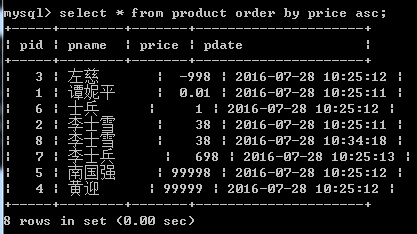


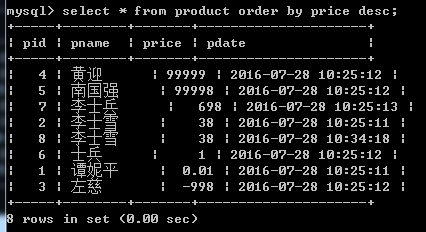
6.查询id为2或者6的商品信息



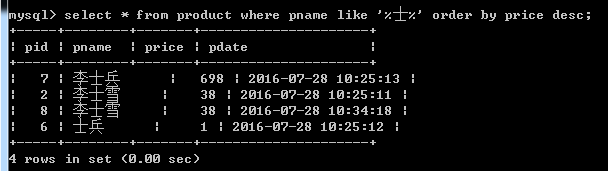
#### 4.3 排序

1.查询所有的商品，按价格进行排序(升序、降序)



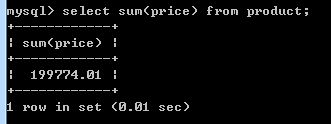


2.查询名称有"士"的商品信息并且按照价格降序排序

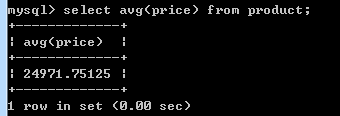


#### 4.4 聚合函数

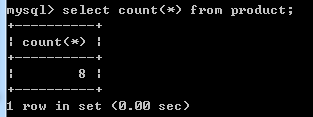
1.获得所有商品的价格的总和



2.获得所有商品的平均价格



3.获得所有商品的个数



#### 4.5 分组操作

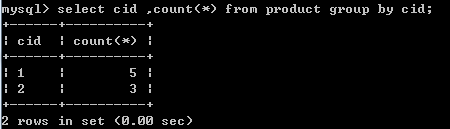
1.添加分类id (alter table product add cid varchar(32);)

2.初始化数据

update product set cid='1';

update product set cid='2' where pid in (5,6,7);

1.根据cid字段分组，分组后统计商品的个数。



2.根据cid分组，分组统计每组商品的平均价格，并且平均价格大于20000元。



#### 4.6 查询总结

select 一般在的后面的内容都是要查询的字段

from 要查询到表

where

group by

having 分组后带有条件只能使用having

order by 它必须放到最后面

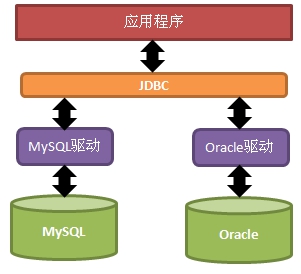
## 回顾JDBC，完成查询

### 1 什么是JDBC

JDBC（Java DataBase Connectivity）就是Java数据库连接，说白了就是用Java语言来操作数据库。原来我们操作数据库是在控制台使用SQL语句来操作数据库，JDBC是用Java语言向数据库发送SQL语句。

### 2 JDBC原理

早期SUN公司的天才们想编写一套可以连接天下所有数据库的API，但是当他们刚刚开始时就发现这是不可完成的任务，因为各个厂商的数据库服务器差异太大了。后来SUN开始与数据库厂商们讨论，最终得出的结论是，由SUN提供一套访问数据库的规范（就是一组接口），并提供连接数据库的协议标准，然后各个数据库厂商会遵循SUN的规范提供一套访问自己公司的数据库服务器的API出现。SUN提供的规范命名为JDBC，而各个厂商提供的，遵循了JDBC规范的，可以访问自己数据库的API被称之为驱动！



JDBC是接口，而JDBC驱动才是接口的实现，没有驱动无法完成数据库连接！每个数据库厂商都有自己的驱动，用来连接自己公司的数据库。

当然还有第三方公司专门为某一数据库提供驱动，这样的驱动往往不是开源免费的！

### 3 JDBC核心类（接口）介绍

JDBC中的核心类有：DriverManager、Connection、Statement，和ResultSet！

DriverManger（驱动管理器）的作用有两个：

* 注册驱动：这可以让JDBC知道要使用的是哪个驱动；
* 获取Connection：如果可以获取到Connection，那么说明已经与数据库连接上了。

Connection对象表示连接，与数据库的通讯都是通过这个对象展开的：

* Connection最为重要的一个方法就是用来获取Statement对象；
* Statement是用来向数据库发送SQL语句的，这样数据库就会执行发送过来的SQL语句
* void executeUpdate(String sql)：执行更新操作（insert、update、delete等）；
* ResultSet executeQuery(String sql)：执行查询操作，数据库在执行查询后会把查询结果，查询结果就是ResultSet；

ResultSet对象表示查询结果集，只有在执行查询操作后才会有结果集的产生。结果集是一个二维的表格，有行有列。操作结果集要学习移动ResultSet内部的“行光标”，以及获取当前行上的每一列上的数据：

* boolean next()：使“行光标”移动到下一行，并返回移动后的行是否存在；
* XXX getXXX(int col)：获取当前行指定列上的值，参数就是列数，列数从1开始，而不是0。

### 4 Hello JDBC

下面开始编写第一个JDBC程序

介绍eclipse的相关知识

单元测试junit

#### 4.1 导入mysql数据库的驱动jar包：

mysql-connector-java-5.1.39-bin.jar；

#### 4.2 注册驱动

**看清楚了，注册驱动就只有一句话：Class.forName(“com.mysql.jdbc.Driver”)，下面的内容都是对这句代码的解释。今后我们的代码中，与注册驱动相关的代码只有这一句。**

DriverManager类的registerDriver()方法的参数是java.sql.Driver，但java.sql.Driver是一个接口，实现类由mysql驱动来提供，mysql驱动中的java.sql.Driver接口的实现类为com.mysql.jdbc.Driver！那么注册驱动的代码如下：

DriverManager.registerDriver(new com.mysql.jdbc.Driver());

上面代码虽然可以注册驱动，但是出现硬编码（代码依赖mysql驱动jar包），如果将来想连接Oracle数据库，那么必须要修改代码的。并且其实这种注册驱动的方式是注册了两次驱动！

JDBC中规定，驱动类在被加载时，需要自己“主动”把自己注册到DriverManger中，下面我们来看看com.mysql.jdbc.Driver类的源代码：

com.mysql.jdbc.Driver.java

|  |
| --- |
| **public** **class** Driver **extends** NonRegisteringDriver **implements** java.sql.Driver {  **static** {  **try** {  java.sql.DriverManager.*registerDriver*(**new** Driver());  } **catch** (SQLException E) {  **throw** **new** RuntimeException("Can't register driver!");  }  }  ……  } |

com.mysql.jdbc.Driver类中的static块会创建本类对象，并注册到DriverManager中。这说明只要去加载com.mysql.jdbc.Driver类，那么就会执行这个static块，从而也就会把com.mysql.jdbc.Driver注册到DriverManager中，所以可以把**注册驱动类**的代码修改为**加载驱动类**。

Class.forName(“com.mysql.jdbc.Driver”);

#### 4.3 获取连接

获取连接需要两步，一是使用DriverManager来注册驱动，二是使用DriverManager来获取Connection对象。

获取连接的也只有一句代码：

DriverManager.getConnection(url,username,password)，

其中username和password是登录数据库的用户名和密码，如果我没说错的话，你的mysql数据库的用户名和密码分别是：root、123。

url查对复杂一点，它是用来找到要连接数据库“网址”，就好比你要浏览器中查找百度时，也需要提供一个url。下面是mysql的url：

jdbc:mysql://localhost:3306/mydb1

JDBC规定url的格式由三部分组成，每个部分中间使用冒号分隔。

* 第一部分是jdbc，这是固定的；
* 第二部分是数据库名称，那么连接mysql数据库，第二部分当然是mysql了；
* 第三部分是由数据库厂商规定的，我们需要了解每个数据库厂商的要求，mysql的第三部分分别由数据库服务器的IP地址（localhost）、端口号（3306），以及DATABASE名称(mydb1)组成。

下面是获取连接的语句：

Connection con = DriverManager.getConnection(“jdbc:mysql://localhost:3306/web08”,”root”,”root”);

还可以在url中提供参数：

jdbc:mysql://localhost:3306/web08**?useUnicode=true&characterEncoding=UTF8**

useUnicode参数指定这个连接数据库的过程中，使用的字节集是Unicode字节集；

characherEncoding参数指定穿上连接数据库的过程中，使用的字节集编码为UTF-8编码。请注意，mysql中指定UTF-8编码是给出的是UTF8，而不是UTF-8。要小心了！

#### 4.4 获取Statement

在得到Connectoin之后，说明已经与数据库连接上了，下面是通过Connection获取Statement对象的代码：

Statement stmt = con.createStatement();

Statement是用来向数据库发送要执行的SQL语句的！

#### 4.5 发送SQL查询语句

String sql = “select \* from user”;

ResultSet rs = stmt.executeQuery(sql);

请注意，执行查询使用的不是executeUpdate()方法，而是executeQuery()方法。executeQuery()方法返回的是ResultSet，ResultSet封装了查询结果，我们称之为结果集。

#### 4.6 读取结果集中的数据

ResultSet就是一张二维的表格，它内部有一个“行光标”，光标默认的位置在“第一行上方”，我们可以调用rs对象的next()方法把“行光标”向下移动一行，当第一次调用next()方法时，“行光标”就到了第一行记录的位置，这时就可以使用ResultSet提供的getXXX(int col)方法来获取指定列的数据了：

rs.next();//光标移动到第一行

rs.getInt(1);//获取第一行第一列的数据

当你使用rs.getInt(1)方法时，你必须可以肯定第1列的数据类型就是int类型，如果你不能肯定，那么最好使用rs.getObject(1)。在ResultSet类中提供了一系列的getXXX()方法，比较常用的方法有：

Object getObject(int col)

String getString(int col)

int getInt(int col)

double getDouble(int col)

#### 4.7 关闭

与IO流一样，使用后的东西都需要关闭！关闭的顺序是先得到的后关闭，后得到的先关闭。

rs.close();

stmt.close();

con.close();

#### 4.8 完成查询操作代码

|  |
| --- |
| **public** **static** Connection getConnection() **throws** Exception {  Class.*forName*("com.mysql.jdbc.Driver");  String url = "jdbc:mysql://localhost:3306/web08";  **return** DriverManager.*getConnection*(url, "root", "root");  } |
| @Test  **public** **void** query() **throws** Exception {  Connection con = *getConnection*();  Statement stmt = con.createStatement();  String sql = "select \* from user";  ResultSet rs = stmt.executeQuery(sql);  **while**(rs.next()) {  String username = rs.getString(1);  String password = rs.getString(2);  System.*out*.println(username + ", " + password);  }  } |

#### 4.9 规范化代码

所谓规范化代码就是无论是否出现异常，都要关闭ResultSet、Statement，以及Connection，如果你还记得IO流的规范化代码，那么下面的代码你就明白什么意思了。

|  |
| --- |
| @Test  **public** **void** query() {  Connection con = **null**;  Statement stmt = **null**;  ResultSet rs = **null**;  **try** {  con = *getConnection*();  stmt = con.createStatement();  String sql = "select \* from user";  rs = stmt.executeQuery(sql);  **while**(rs.next()) {  String username = rs.getString(1);  String password = rs.getString(2);  System.*out*.println(username + ", " + password);  }  } **catch**(Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  **try** {  **if**(rs != **null**) rs.close();  **if**(stmt != **null**) stmt.close();  **if**(con != **null**) con.close();  } **catch**(SQLException e) {}  }  } |

### JDBC对象介绍

#### 1 JDBC中的主要类（接口）

在JDBC中常用的类有：

* DriverManager；
* Connection；
* Statement；
* ResultSet。

#### 2 DriverManager

其实我们今后只需要会用DriverManager的getConnection()方法即可：

1. Class.forName(“com.mysql.jdbc.Driver”);//注册驱动
2. String url = “jdbc:mysql://localhost:3306/web08”;
3. String username = “root”;
4. String password = “root”;
5. Connection con = DriverManager.getConnection(url, username, password);

注意，上面代码可能出现的两种异常：

1. ClassNotFoundException：这个异常是在第1句上出现的，出现这个异常有两个可能：

* 你没有给出mysql的jar包；
* 你把类名称打错了，查看类名是不是com.mysql.jdbc.Driver。

1. SQLException：这个异常出现在第5句，出现这个异常就是三个参数的问题，往往username和password一般不是出错，所以需要认真查看url是否打错。

对于DriverManager.registerDriver()方法了解即可，因为我们今后注册驱动只会Class.forName()，而不会使用这个方法。

#### 3 Connection

Connection最为重要的方法就是获取Statement：

* Statement stmt = con.createStatement();

后面在学习ResultSet方法时，还要学习一下下面的方法：

* Statement stmt = con.createStatement(int,int);

#### 4 Statement

Statement最为重要的方法是：

* int executeUpdate(String sql)：执行更新操作，即执行insert、update、delete语句，其实这个方法也可以执行create table、alter table，以及drop table等语句，但我们很少会使用JDBC来执行这些语句；
* ResultSet executeQuery(String sql)：执行查询操作，执行查询操作会返回ResultSet，即结果集。

* boolean execute()

Statement还有一个boolean execute()方法，这个方法可以用来执行增、删、改、查所有SQL语句。该方法返回的是boolean类型，表示SQL语句是否执行成功。

如果使用execute()方法执行的是更新语句，那么还要调用int getUpdateCount()来获取insert、update、delete语句所影响的行数。

如果使用execute()方法执行的是查询语句，那么还要调用ResultSet getResultSet()来获取select语句的查询结果。

#### 5 ResultSet之滚动结果集（了解）

ResultSet表示结果集，它是一个二维的表格！ResultSet内部维护一个行光标（游标），ResultSet提供了一系列的方法来移动游标：

* void beforeFirst()：把光标放到第一行的前面，这也是光标默认的位置；
* void afterLast()：把光标放到最后一行的后面；
* boolean first()：把光标放到第一行的位置上，返回值表示调控光标是否成功；
* boolean last()：把光标放到最后一行的位置上；
* boolean isBeforeFirst()：当前光标位置是否在第一行前面；
* boolean isAfterLast()：当前光标位置是否在最后一行的后面；
* boolean isFirst()：当前光标位置是否在第一行上；
* boolean isLast()：当前光标位置是否在最后一行上；
* boolean previous()：把光标向上挪一行；
* boolean next()：把光标向下挪一行；
* boolean relative(int row)：相对位移，当row为正数时，表示向下移动row行，为负数时表示向上移动row行；
* boolean absolute(int row)：绝对位移，把光标移动到指定的行上；
* int getRow()：返回当前光标所有行。

上面方法分为两类，一类用来判断游标位置的，另一类是用来移动游标的。如果结果集是不可滚动的，那么只能使用next()方法来移动游标，而beforeFirst()、afterLast()、first()、last()、previous()、relative()方法都不能使用！！！

结果集是否支持滚动，要从Connection类的createStatement()方法说起。也就是说创建的Statement决定了使用Statement创建的ResultSet是否支持滚动。

Statement createStatement(int resultSetType, int resultSetConcurrency)

resultSetType的可选值：

* ResultSet.TYPE\_FORWARD\_ONLY：不滚动结果集；
* ResultSet.TYPE\_SCROLL\_INSENSITIVE：滚动结果集，但结果集数据不会再跟随数据库而变化；
* ResultSet.TYPE\_SCROLL\_SENSITIVE：滚动结果集，但结果集数据不会再跟随数据库而变化；

可以看出，如果想使用滚动的结果集，我们应该选择TYPE\_SCROLL\_INSENSITIVE！其实很少有数据库驱动会支持TYPE\_SCROLL\_SENSITIVE的特性！通常我们也不需要查询到的结果集再受到数据库变化的影响。

resultSetConcurrency的可选值：

* [CONCUR\_READ\_ONLY](#CONCUR_READ_ONLY)：结果集是只读的，不能通过修改结果集而反向影响数据库；
* CONCUR\_UPDATABLE：结果集是可更新的，对结果集的更新可以反向影响数据库。

通常可更新结果集这一“高级特性”我们也是不需要的！

获取滚动结果集的代码如下：

Connection con = …

Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE, [CONCUR\_READ\_ONLY](#CONCUR_READ_ONLY));

String sql = …//查询语句

ResultSet rs = stmt.executeQuery(sql);//这个结果集是可滚动的

#### 6 ResultSet之获取列数据

可以通过next()方法使ResultSet的游标向下移动，当游标移动到你需要的行时，就需要来获取该行的数据了，ResultSet提供了一系列的获取列数据的方法：

* String getString(int columnIndex)：获取指定列的String类型数据；
* int getInt(int columnIndex)：获取指定列的int类型数据；
* double getDouble(int columnIndex)：获取指定列的double类型数据；
* boolean getBoolean(int columnIndex)：获取指定列的boolean类型数据；
* Object getObject(int columnIndex)：获取指定列的Object类型的数据。

上面方法中，参数columnIndex表示列的索引，列索引从1开始，而不是0，这第一点与数组不同。如果你清楚当前列的数据类型，那么可以使用getInt()之类的方法来获取，如果你不清楚列的类型，那么你应该使用getObject()方法来获取。

ResultSet还提供了一套通过列名称来获取列数据的方法：

* String getString(String columnName)：获取名称为columnName的列的String数据；
* int getInt(String columnName)：获取名称为columnName的列的int数据；
* double getDouble(String columnName)：获取名称为columnName的列的double数据；
* boolean getBoolean(String columnName)：获取名称为columnName的列的boolean数据；
* Object getObject(String columnName)：获取名称为columnName的列的Object数据；

### PreparedStatement

#### 1 什么是SQL攻击

在需要用户输入的地方，用户输入的是SQL语句的片段，最终用户输入的SQL片段与我们DAO中写的SQL语句合成一个完整的SQL语句！例如用户在登录时输入的用户名和密码都是为SQL语句的片段！

#### 2 演示SQL攻击

首先我们需要创建一张用户表，用来存储用户的信息。

|  |
| --- |
| **CREATE TABLE user(**  **uid CHAR(32) PRIMARY KEY,**  **username VARCHAR(30) UNIQUE KEY NOT NULL,**  **PASSWORD VARCHAR(30)**  **);**  **INSERT INTO user VALUES('U\_1001', 'zs', 'zs');**  **SELECT \* FROM user;** |

现在用户表中只有一行记录，就是zs。

下面我们写一个login()方法！

|  |
| --- |
| **public** **void** login(String username, String password) {  Connection con = **null**;  Statement stmt = **null**;  ResultSet rs = **null**;  **try** {  con = JdbcUtils.*getConnection*();  stmt = con.createStatement();  String sql = "SELECT \* FROM user WHERE " +  "username='" + username +  "' and password='" + password + "'";  rs = stmt.executeQuery(sql);  **if**(rs.next()) {  System.*out*.println("欢迎" + rs.getString("username"));  } **else** {  System.*out*.println("用户名或密码错误！");  }  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  } **finally** {  JdbcUtils.*close*(con, stmt, rs);  }  } |

下面是调用这个方法的代码：

|  |
| --- |
| login("a' or 'a'='a", "a' or 'a'='a"); |

这行当前会使我们登录成功！因为是输入的用户名和密码是SQL语句片段，最终与我们的login()方法中的SQL语句组合在一起！我们来看看组合在一起的SQL语句：

|  |
| --- |
| SELECT \* FROM tab\_user WHERE username='**a' or 'a'='a**' and password='**a' or 'a'='a**' |

#### 3 防止SQL攻击

过滤用户输入的数据中是否包含非法字符；

* 分步校验！先使用用户名来查询用户，如果查找到了，再比较密码；
* 使用PreparedStatement。

#### 4 PreparedStatement是什么？

PreparedStatement叫预编译声明！

PreparedStatement是Statement的子接口，你可以使用PreparedStatement来替换Statement。

PreparedStatement的好处：

* 防止SQL攻击；
* 提高代码的可读性，以可维护性；

提高效率。

#### 5 PreparedStatement的使用

* 使用Connection的prepareStatement(String sql)：即创建它时就让它与一条SQL模板绑定；
* 调用PreparedStatement的setXXX()系列方法为问号设置值
* 调用executeUpdate()或executeQuery()方法，但要注意，调用没有参数的方法；

|  |
| --- |
| String sql = “select \* from tab\_student where s\_number=?”;  PreparedStatement pstmt = con.prepareStatement(sql);  pstmt.setString(1, “S\_1001”);  ResultSet rs = pstmt.executeQuery();  rs.close();  pstmt.clearParameters();  pstmt.setString(1, “S\_1002”);  rs = pstmt.executeQuery(); |

在使用Connection创建PreparedStatement对象时需要给出一个SQL模板，所谓SQL模板就是有“?”的SQL语句，其中“?”就是参数。

在得到PreparedStatement对象后，调用它的setXXX()方法为“?”赋值，这样就可以得到把模板变成一条完整的SQL语句，然后再调用PreparedStatement对象的executeQuery()方法获取ResultSet对象。

注意PreparedStatement对象独有的executeQuery()方法是没有参数的，而Statement的executeQuery()是需要参数（SQL语句）的。因为在创建PreparedStatement对象时已经让它与一条SQL模板绑定在一起了，所以在调用它的executeQuery()和executeUpdate()方法时就不再需要参数了。

PreparedStatement最大的好处就是在于重复使用同一模板，给予其不同的参数来重复的使用它。这才是真正提高效率的原因。

**所以，建议大家在今后的开发中，无论什么情况，都去需要PreparedStatement，而不是使用Statement。**

## 内连接和外连接

|  |
| --- |
| 在之前，我对MSSQL中的内连接和外连接所得出的数据集不是很清楚。这几天重新温习了一下SQL的书本，现在的思路应该是很清楚了，现在把自己的理解发出来给大家温习下。希望和我一样对SQL的连接语句不太理解的朋友能够有所帮助。（发这么菜的教程，各位大大们别笑话偶了，呵:D ）  有两个表A和表B。 表A结构如下： Aid：int；标识种子，主键，自增ID Aname：varchar  数据情况，即用select \* from A出来的记录情况如下图1所示：   图1:A表数据  表B结构如下： Bid：int；标识种子，主键，自增ID Bnameid：int  数据情况，即用select \* from B出来的记录情况如下图2所示：    图2:B表数据  为了把Bid和Aid加以区分，不让大家有误解，所以把Bid的起始种子设置为100。 有SQL基本知识的人都知道，两个表要做连接，就必须有个连接字段，从上表中的数据可以看出，在A表中的Aid和B表中的Bnameid就是两个连接字段。 下图3说明了连接的所有记录集之间的关系：    图3:连接关系图  现在我们对内连接和外连接一一讲解。 1.内连接：利用内连接可获取两表的公共部分的记录，即图3的记录集C 语句如下：Select \* from A JOIN B ON A.Aid=B.Bnameid 运行结果如下图4所示：    图4:内连接数据  其实select \* from A,B where A.Aid=B.Bnameid与Select \* from A JOIN B ON A.Aid=B.Bnameid的运行结果是一样的。 2.外连接：外连接分为两种，一种是左连接（Left JOIN）和右连接（Right JOIN）  (1)左连接（Left JOIN）：即图3公共部分记录集C＋表A记录集A1。   语句如下：select \* from A Left JOIN B ON A.Aid=B.Bnameid  运行结果如下图5所示：    图5:左连接数据   说明：  在语句中，A在B的左边，并且是Left Join，所以其运算方式为：A左连接B的记录=图3公共部分记录集C＋表A记录集A1  在图3中即记录集C中的存在的Aid为：2 3 6 7 8   图1中即表A所有记录集A中存在的Aid为：1 2 3 4 5 6 7 8 9  表A记录集A1中存在的Aid=(图1中即A表中所有Aid)-(图3中即记录集C中存在的Aid)，最终得出为：1 4 5 9  由此得出图5中A左连接B的记录=图3公共部分记录集C＋表A记录集A1,  最终得出的结果图5中可以看出Bnameid及Bid非NULL的记录都为图3公共部分记录集C中的记录；Bnameid及Bid为NULL的Aid为1 4 5 9的四笔记录就是表A记录集A1中存在的Aid。   (2)右连接（Right JOIN）：即图3公共部分记录集C＋表B记录集B1。  语句如下：select \* from A Right JOIN B ON A.Aid=B.Bnameid  运行结果如下图6所示：    图6:右连接数据   说明：  在语句中，A在B的左边，并且是Right Join，所以其运算方式为：A右连接B的记录=图3公共部分记录集C＋表B记录集B1  在图3中即记录集C中的存在的Aid为：2 3 6 7 8   图2中即表B所有记录集B中存在的Bnameid为：2 3 6 7 8 11  表B记录集B1中存在的Bnameid=(图2中即B表中所有Bnameid)-(图3中即记录集C中存在的Aid)，最终得出为：11  由此得出图6中A右连接B的记录=图3公共部分记录集C＋表B记录集B1,  最终得出的结果图6中可以看出Aid及Aname非NULL的记录都为图3公共部分记录集C中的记录；Aid及Aname为NULL的Aid为11的记录就是表B记录集B1中存在的Bnameid。   总结：  通过上面的运算解说，相信很多人已经想到，上面的情况（包括图3的关系图）说明的都只是A在B的左边的情况， 以下语句B在A的右边的又会出现什么情况呢？？ select \* from B Left JOIN A ON A.Aid=B.Bnameid select \* from B Right JOIN A ON A.Aid=B.Bnameid  其实对图3左右翻转一下就可以得出以下结论： select \* from B Left JOIN A ON A.Aid=B.Bnameid和select \* from A Right JOIN B ON A.Aid=B.Bnameid所得出的记录集是一样的 而 select \* from B Right JOIN A ON A.Aid=B.Bnameid和select \* from A Left JOIN B ON A.Aid=B.Bnameid所得出的记录集也是一样的。 |

你是要弄清楚区别在什么地方还是单纯想要文字说明

文字说明的楼上说了一大堆了，不说了。

弄个例题，直观一点。两个表：

--表stu

id name

1, Jack

2, Tom

3, Kity

4, nono

--表exam

id grade

1, 56

2, 76

11, 89

内连接 （显示两表id匹配的）

select stu.id,exam.id,stu.name, exam.grade from stu inner join exam on stu.id=exam.id

stu.id exam.id name grade

--------------------------------

1 1 Jack 56

2 2 Tom 76

左连接（显示join 左边的表的所有数据，exam只有两条记录，所以stu.id,grade 都用NULL 显示）

select stu.id,exam.id,stu.name, exam.grade from stu left join exam on stu.id=exam.id

1 1 Jack 56

2 2 Tom 76

3 NULL Kity NULL

4 NULL nono NULL

右连接（与作连接相反，显示join右边表的所有数据）

select stu.id,exam.id,stu.name, exam.grade from stu right join exam on stu.id=exam.id

1 1 Jack 56

2 2 Tom 76

NULL 11 NULL 89

## 子查询

[**【SQL】带有ANY或ALL谓词的子查询**](http://www.cnblogs.com/cardgames/articles/1344761.html)

子查询返回单值时可以用比较运算符，而使用ANY或ALL谓词时则必须同时使用比较

运算符，其语义为：  
>ANY 大于子查询结果的某个值  
>ALL 大于子查询结果中的所有值  
<ANY 小于子查询结果中的某个值  
<ALL 小于子查询结果中的所有值  
>=ANY 大于等于子查询结果中的某个值  
>=ALL 大于等于子查询结果中的所有值  
<=ANY 小于等于子查询结果中的某个值  
<=ALL 小于等于子查询结果中的所有值  
=ANY 等于子查询结果中的某个值  
=ALL 等于子查询结果中的所有值(通常没有实际意义)  
!=(或<>)ANY 不等于子查询结果中的某个值  
!=(或<>)ALL 不等于子查询结果中的任何一个值

例：查询其他系中比信息系某一学生年龄小的学生姓名和年龄  
SELECT Sname,Sage  
FROM Student  
WHERE Sage<ANY(SELECT Sage  
 FROM Student   
 WHERE Sdept='IS')   
 AND Sdept<>'IS'  
结果如下：  
Sname Sage  
-------------------------------------  
王敏 18

DBMS执行此查询时，首先处理子查询，找出IS系中所有学生的年龄，构成一个集合

（19，18）。然后处理父查询，找所有不是IS系且年龄小于19或18的学生。  
 本查询可以用集函数来实现，首先用子查询找出IS系中最大年龄（19），  
然后在父查询中查所有非IS系且年龄小于19岁的学生姓名及年龄，SQL语句如下：  
SELECT Sname,Sage  
FROM Student  
WHERE Sage<  
 (SELECT MAX(Sage)  
 FROM Student  
 WHERE Sdept='IS')  
 AND Sdept<>'IS'

例二：  
查询其他系中比信息系所有学生年龄都小的学生姓名及年龄。  
SELECT Sname,Sage  
FROM Student  
WHERE Sage<ALL  
 (  
 SELECT Sage  
 FROM Student  
 WHERE Sdept='IS'  
 )  
 AND Sdept<>'IS'  
查询结果为空表，本查询同样也可以用集函数来实现，SQL语句如下：  
SELECT Sname,Sage  
FROM Student  
WHERE Sage<  
 (SELECT MIN(Sage)  
 FROM Student  
 WHERE Sdept='IS'  
 )  
 AND Sdept<>'IS'  
事实上，用集函数来实现子查询通常比直接用ANY或ALL查询效率要高，ANY与ALL与

集函数的对应关系如下所示

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | = | <>或!= | < | <= | > | >= |
| ANY | IN | - | <MAX | <=MAX | >MIN | >=MIN |
| ALL | -- | NOT IN | <MIN | <=MIN | >MAX | >=MAX |

# MySQL数据库优化专题

## 存储过程

### 什么是存储过程

简单的说，就是一组SQL语句集，功能强大，可以实现一些比较复杂的逻辑功能，类似于JAVA语言中的方法；

ps:存储过程跟触发器有点类似，都是一组SQL集，但是存储过程是主动调用的，且功能比触发器更加强大，触发器是某件事触发后自动调用；

### 存储过程有哪些特性

有输入输出参数，可以声明变量，有if/else,case,while等控制语句，通过编写存储过程，可以实现复杂的逻辑功能；

函数的普遍特性：模块化，封装，代码复用；

速度快，只有首次执行需经过编译和优化步骤，后续被调用可以直接执行，省去以上步骤；

### 创建一个存储过程

|  |
| --- |
| CREATEPROCEDUREuser\_porced()  BEGIN  SELECT  NAME  FROM  users;  END; |

### 调用存储过程

|  |
| --- |
| calluser\_porced(); |

### 传参存储过程

|  |
| --- |
| createPROCEDUREuser\_porcedPa(  inaint(10)  )  BEGIN  select\*fromuserswhereage>a;  END;  calluser\_porcedPa(10); |

### 存储过程优缺点

优点

1.在生产环境下，可以通过直接修改存储过程的方式修改业务逻辑（或bug），而不用重启服务器。但这一点便利被许多人滥用了。有人直接就在正式服务器上修改存储过程，而没有经过完整的测试，后果非常严重。

2.执行速度快。存储过程经过编译之后会比单独一条一条执行要快。但这个效率真是没太大影响。如果是要做大数据量的导入、同步，我们可以用其它手段。

3.减少网络传输。存储过程直接就在数据库服务器上跑，所有的数据访问都在服务器内部进行，不需要传输数据到其它终端。但我们的应付服务器通常与数据库是在同一内网，大数据的访问的瓶颈会是硬盘的速度，而不是网速。

4.能够解决presentation与数据之间的差异，说得文艺青年点就是解决OO模型与二维数据持久化之间的阻抗。领域模型和数据模型的设计可能不是同一个人（一个是SA，另一个是DBA），两者的分歧可能会很大——这不奇怪，一个是以OO的思想来设计，一个是结构化的数据来设计，大家互不妥协——你说为了软件的弹性必须这么设计，他说为了效率必须那样设计，为了抹平鸿沟，就用存储过程来做数据存储的逻辑映射（把属性映射到字段）。好吧，台下已经有同学在叨咕ORM了。

5.方便DBA优化。所有的SQL集中在一个地方，DBA会很高兴。这一点算是ORM的软肋。不过按照CQRS框架的思想，查询是用存储过程还是ORM，还真不是问题——DBA对数据库的优化，ORM一样会受益。况且放在ORM中还能用二级缓存，有些时候效率还会更高。

缺点

1.SQL本身是一种结构化查询语言，加上了一些控制（赋值、循环和异常处理等），但不是OO的，本质上还是过程化的，面对复杂的业务逻辑，过程化的处理会很吃力。这一点算致命伤。

2.不便于调试。基本上没有较好的调试器，很多时候是用print来调试，但用这种方法调试长达数百行的存储过程简直是噩梦。好吧，这一点不算啥，C#/java一样能写出噩梦般的代码。

3.没办法应用缓存。虽然有全局临时表之类的方法可以做缓存，但同样加重了数据库的负担。如果缓存并发严重，经常要加锁，那效率实在堪忧。

4.无法适应数据库的切割（水平或垂直切割）。数据库切割之后，存储过程并不清楚数据存储在哪个数据库中。

5.精通SQL的新手越来越少——不要笑，这是真的，我面试过N多新人，都不知道如何创建全局临时表、不知道having、不知道聚集索引和非聚集索引，更别提游标和提交叉表查询了。好吧，这个缺点算是凑数用的，作为屌丝程序员，我们的口号是：没有不会的，只有不用的。除了少数有语言洁癖的人，我相信精通SQL只是时间问题。

## MySQL如何优化

* 表的设计合理化(符合3NF)
* 添加适当索引(index)[四种:普通索引、主键索引、唯一索引unique、全文索引]
* SQL语句优化
* 分表技术(水平分割、垂直分割)
* 读写[写:update/delete/add]分离
* 存储过程[模块化编程，可以提高速度]
* 对mysql配置优化[配置最大并发数my.ini,调整缓存大小]
* mysql服务器硬件升级
* 定时的去清除不需要的数据,定时进行碎片整理(MyISAM)



**然后：**

**1.对查询进行优化，应尽量避免全表扫描，首先应考虑在where及orderby涉及的列上建立索引。**

**2.应尽量避免在where子句中对字段进行null值判断，否则将导致**[**引擎**](https://www.baidu.com/s?wd=%E5%BC%95%E6%93%8E&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)**放弃使用索引而进行全表扫描，如：**

**selectidfromtwherenumisnull**

**可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：**

**selectidfromtwherenum=0**

**3.应尽量避免在where子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。**

**4.应尽量避免在where子句中使用or来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：**

**selectidfromtwherenum=10ornum=20**

**可以这样查询：**

**selectidfromtwherenum=10**

**unionall**

**selectidfromtwherenum=20**

**5.in和notin也要慎用，否则会导致全表扫描，如：**

**selectidfromtwherenumin(1,2,3)**

**对于连续的数值，能用between就不要用in了：**

**selectidfromtwherenumbetween1and3**

**6.下面的查询也将导致全表扫描：**

**selectidfromtwherenamelike'%abc%'**

**7.应尽量避免在where子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：**

**selectidfromtwherenum/2=100**

**应改为:**

**selectidfromtwherenum=100\*2**

**8.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：**

**selectidfromtwheresubstring(name,1,3)='abc'--name以abc开头的id**

**应改为:**

**selectidfromtwherenamelike'abc%'**

**9.不要在where子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。**

**10.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，**

**否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。**

**11.不要写一些没有意义的查询，如需要生成一个空表结构：**

**selectcol1,col2into#tfromtwhere1=0**

**这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：**

**createtable#t(...)**

**12.很多时候用exists代替in是一个好的选择：**

**selectnumfromawherenumin(selectnumfromb)**

**用下面的语句替换：**

**selectnumfromawhereexists(select1frombwherenum=a.num)**

**13.并不是所有索引对查询都有效，SQL是根据表**[**中数据**](https://www.baidu.com/s?wd=%E4%B8%AD%E6%95%B0%E6%8D%AE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)**来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，**

**如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。**

**14.索引并不是越多越好，索引固然可以提高相应的select的效率，但同时也降低了insert及update的效率，**

**因为insert或update时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。**

**一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。**

**15.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。**

**这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。**

**16.尽可能的使用varchar代替char，因为首先变长字段存储空间小，可以节省存储空间，**

**其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。**

**17.任何地方都不要使用select\*fromt，用具体的字段列表代替“\*”，不要返回用不到的任何字段。**

**18.避免频繁创建和删除临时表，以减少系统表资源的消耗。**

**19.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。**

**20.在新建临时表时，如果一次性插入数据量很大，那么可以使用selectinto代替createtable，避免造成大量log，**

**以提高速度；如果数据量不大，为了缓和系统表的资源，应先createtable，然后insert。**

**21.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先truncatetable，然后droptable，这样可以避免系统表的较长时间锁定。**

**22.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。**

**23.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。**

**24.与临时表一样，游标并不是不可使用。对小型数据集使用FAST\_FORWARD游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。**

在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

25.尽量避免大事务操作，提高系统并发能力。26.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

## 数据库设计

### 什么是数据库范式

为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式。

### 数据库三大范式

第一范式：1NF是对属性的原子性约束，要求属性(列)具有原子性，不可再分解；(只要是关系型数据库都满足1NF)

第二范式：2NF是对记录的惟一性约束，表中的记录是唯一的,就满足2NF,通常我们设计一个主键来实现，主键不能包含业务逻辑。

第三范式：3NF是对字段冗余性的约束，它要求字段没有冗余。没有冗余的数据库设计可以做到。

但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是：在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。

电商系统中，可以用主键id作为订单号吗？不可以，主键不能做具体业务。

反过来订单号可以做为主键。订单号是唯一约束。

企业当中的订单号，不能用uuid，可能会产生幂等性。

用分布式锁，序列、流水号生成算法。

主流：提前把订单号生成好，用一个取一个。快没了继续生成。

订单号前面+字幕+时间戳+用户主键+加密

# SQL优化

如何从一个大项目中，迅速的定位执行速度慢的语句.(定位慢查询)

#### showstatus

使用showstatus使用showstatus查看MySQL服务器状态信息

常用命令

|  |
| --- |
| * mysql数据库启动了多少时间   showstatuslike'uptime'; |
| showstautslike'com\_select'showstautslike'com\_insert'...类推updatedelete（显示数据库的查询，更新，添加，删除的次数） |
| show[session|global]statuslike....如果你不写[session|global]默认是session会话，指取出当前窗口的执行，如果你想看所有(从mysql启动到现在，则应该global) |
| //显示到mysql数据库的连接数  showstatuslike'connections'; |
| //显示慢查询次数  showstatuslike'slow\_queries'; |

#### 慢查询

##### 什么是慢查询

MySQL默认10秒内没有响应SQL结果,则为慢查询，存放到一个日志中

可以去修改MySQL慢查询默认时间

##### 如何修改慢查询

|  |
| --- |
| --查询慢查询时间  showvariableslike'long\_query\_time';  --修改慢查询时间  setlong\_query\_time=1;--但是重启mysql之后，long\_query\_time依然是my.ini中的值 |

##### 如何定位慢查询

##### 初始化测试数据

###### 创建表结构

|  |
| --- |
| /\*部门表\*/  CREATETABLEdept(  deptnoMEDIUMINTUNSIGNEDNOTNULLDEFAULT0,/\*编号\*/  dnameVARCHAR(20)NOTNULLDEFAULT"",/\*名称\*/  locVARCHAR(13)NOTNULLDEFAULT""/\*地点\*/  )ENGINE=MyISAMDEFAULTCHARSET=utf8;  /\*员工表\*/  CREATETABLEemp  (empnoMEDIUMINTUNSIGNEDNOTNULLDEFAULT0,/\*编号\*/  enameVARCHAR(20)NOTNULLDEFAULT"",/\*名字\*/  jobVARCHAR(9)NOTNULLDEFAULT"",/\*工作\*/  mgrMEDIUMINTUNSIGNEDNOTNULLDEFAULT0,/\*上级编号\*/  hiredateDATENOTNULL,/\*入职时间\*/  salDECIMAL(7,2)NOTNULL,/\*薪水\*/  commDECIMAL(7,2)NOTNULL,/\*红利\*/  deptnoMEDIUMINTUNSIGNEDNOTNULLDEFAULT0/\*部门编号\*/  )ENGINE=MyISAMDEFAULTCHARSET=utf8;  /\*薪水\*/  CREATETABLEsalgrade  (  gradeMEDIUMINTUNSIGNEDNOTNULLDEFAULT0,  losalDECIMAL(17,2)NOTNULL,  hisalDECIMAL(17,2)NOTNULL  )ENGINE=MyISAMDEFAULTCHARSET=utf8;  /\*测试数据\*/  INSERTINTOsalgradeVALUES(1,700,1200);  INSERTINTOsalgradeVALUES(2,1201,1400);  INSERTINTOsalgradeVALUES(3,1401,2000);  INSERTINTOsalgradeVALUES(4,2001,3000);  INSERTINTOsalgradeVALUES(5,3001,9999); |

###### 创建函数

|  |
| --- |
| createfunctionrand\_string(nINT)  returnsvarchar(255)#该函数会返回一个字符串  begin  #chars\_str定义一个变量chars\_str,类型是varchar(100),默认值'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';  declarechars\_strvarchar(100)default  'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';  declarereturn\_strvarchar(255)default'';  declareiintdefault0;  whilei<ndo  setreturn\_str=concat(return\_str,substring(chars\_str,floor(1+rand()\*52),1));  seti=i+1;  endwhile;  returnreturn\_str;  end |
| createFUNCTIONrand\_num()  RETURNSint(5)  BEGIN  DECLAREiintdefault0;  seti=floor(10+RAND()\*500);  returni;  END |

###### 创建存储过程

|  |
| --- |
| delimiter$$  createprocedureinsert\_emp(instartint(10),inmax\_numint(10))  begin  declareiintdefault0;  #setautocommit=0把autocommit设置成0  setautocommit=0;  repeat  seti=i+1;  insertintoempvalues((start+i),rand\_string(6),'SALESMAN',0001,curdate(),2000,400,rand\_num());  untili=max\_num  endrepeat;  commit;  end$$  执行存储过程  callinsert\_emp(100001,40000000); |

##### 如何将慢查询定位到日志中

在默认情况下，我们的mysql不会记录慢查询，需要在启动mysql时候，指定记录慢查询才可以

cmd到C:\ProgramFiles(x86)\MySQL\MySQLServer5.5

mysql安装目录

先停用服务中的mysql服务，然后再到安装目录执行以下命令

bin\mysqld.exe--safe-mode--slow-query-log[mysql5.5可以在my.ini指定]（安全模式启动，数据库将操作写入日志，以备恢复）

bin\mysqld.exe–log-slow-queries=d:/abc.log[低版本mysql5.0可以在my.ini指定]

先关闭mysql,再启动,如果启用了慢查询日志，默认把这个文件放在

my.ini文件中记录的位置

#Pathtothedatabaseroot

datadir="C:/ProgramData/MySQL/MySQLServer5.5/Data/"

## 索引

### 什么是索引

索引在任何关系型数据库，mysql，oracle都有

索引主要作用：提高查询效率。二叉树。B+树。全局扫描。

传统方式使用的是全局扫描。加索引后是折半查找。

一般用的是唯一，组合索引。

全文索引。

data目录中，frm文件为数据库结构文件，myd为数据备份文件

注意：

索引并不是加了就会效率高。

索引用来快速地寻找那些具有特定值的记录，所有MySQL索引都以B-树的形式保存。如果没有索引，执行查询时MySQL必须从第一个记录开始扫描整个表的所有记录，直至找到符合要求的记录。表里面的记录数量越多，这个操作的代价就越高。如果作为搜索条件的列上已经创建了索引，MySQL无需扫描任何记录即可迅速得到目标记录所在的位置。如果表有1000个记录，通过索引查找记录至少要比顺序扫描记录快100倍。

### 索引的分类

#### 主键索引

主键是一种唯一性索引，但它必须指定为“PRIMARYKEY”。如果你曾经用过AUTO\_INCREMENT类型的列，你可能已经熟悉主键之类的概念了。主键一般在创建表的时候指定，例如“CREATETABLEtablename([...],PRIMARYKEY(列的列表));”。但是，我们也可以通过修改表的方式加入主键，例如“ALTERTABLEtablenameADDPRIMARYKEY(列的列表);”。每个表只能有一个主键。

##### 创建主键索引

主键是一种唯一性索引，但它必须指定为“PRIMARYKEY”。如果你曾经用过AUTO\_INCREMENT类型的列，你可能已经熟悉主键之类的概念了。主键一般在创建表的时候指定，例如“CREATETABLEtablename([...],PRIMARYKEY(列的列表));”。但是，我们也可以通过修改表的方式加入主键，例如“ALTERTABLEtablenameADDPRIMARYKEY(列的列表);”。每个表只能有一个主键。

当一张表，把某个列设为主键的时候，则该列就是主键索引

createtableaaa

(idintunsignedprimarykeyauto\_increment,

namevarchar(32)notnulldefault'');

这是id列就是主键索引.

createtablebbb(idint,namevarchar(32)notnulldefault'');

如果你创建表时，没有指定主键索引，也可以在创建表后，在添加,指令:

实例:

altertable表名addprimarykey(列名);

删除主键索引

altertablearticlesdropprimarykey;

#### 查询索引

desc表名;不能显示索引名称

showindexfrom表名

showkeysfrom表名

#### 全文索引

##### 创建表结构

|  |
| --- |
| CREATETABLEarticles(  idINTUNSIGNEDAUTO\_INCREMENTNOTNULLPRIMARYKEY,  titleVARCHAR(200),  bodyTEXT,  FULLTEXT(title,body)  )engine=myisamcharsetutf8;  INSERTINTOarticles(title,body)VALUES  ('MySQLTutorial','DBMSstandsforDataBase...'),  ('HowToUseMySQLWell','Afteryouwentthrougha...'),  ('OptimizingMySQL','Inthistutorialwewillshow...'),  ('1001MySQLTricks','1.Neverrunmysqldasroot.2....'),  ('MySQLvs.YourSQL','Inthefollowingdatabasecomparison...'),  ('MySQLSecurity','Whenconfiguredproperly,MySQL...'); |

错误用法:

select\*fromarticleswherebodylike'%mysql%';错误用法索引不会生效

正确用法:

select\*fromarticleswherematch(title,body)against('database')

说明:

1. 在mysql中fulltext索引只针对myisam生效
2. mysql自己提供的fulltext针对英文生效->sphinx(coreseek)技术处理中文
3. 使用方法是match(字段名..)against(‘关键字’)
4. 全文索引：停止词,因为在一个文本中，创建索引是一个无穷大的数，因此，对一些常用词和字符，就不会创建，这些词，称为停止词.比如（a，b，mysql，the）

mysql>selectmatch(title,body)against('database')fromarticles;（输出的是每行和database的匹配度）

#### 唯一索引

这种索引和前面的“普通索引”基本相同，但有一个区别：索引列的所有值都只能出现一次，即必须唯一。唯一性索引可以用以下几种方式创建：

创建索引，例如CREATEUNIQUEINDEX<索引的名字>ONtablename(列的列表)；

修改表，例如ALTERTABLEtablenameADDUNIQUE[索引的名字](列的列表)；

创建表的时候指定索引，例如CREATETABLEtablename([...],UNIQUE[索引的名字](列的列表))；

##### 创建表结构

createtableddd(idintprimarykeyauto\_increment,namevarchar(32)unique);

##### 注意

unique字段可以为NULL,并可以有多NULL,但是如果是具体内容，则不能重复，

但是不能存有重复的空字符串’’

#### 普通索引

普通索引（由关键字KEY或INDEX定义的索引）的唯一任务是加快对数据的访问速度。因此，应该只为那些最经常出现在查询条件（WHEREcolumn=）或排序条件（ORDERBYcolumn）中的数据列创建索引。只要有可能，就应该选择一个数据最整齐、最紧凑的数据列（如一个整数类型的数据列）来创建索引。

createtableccc(

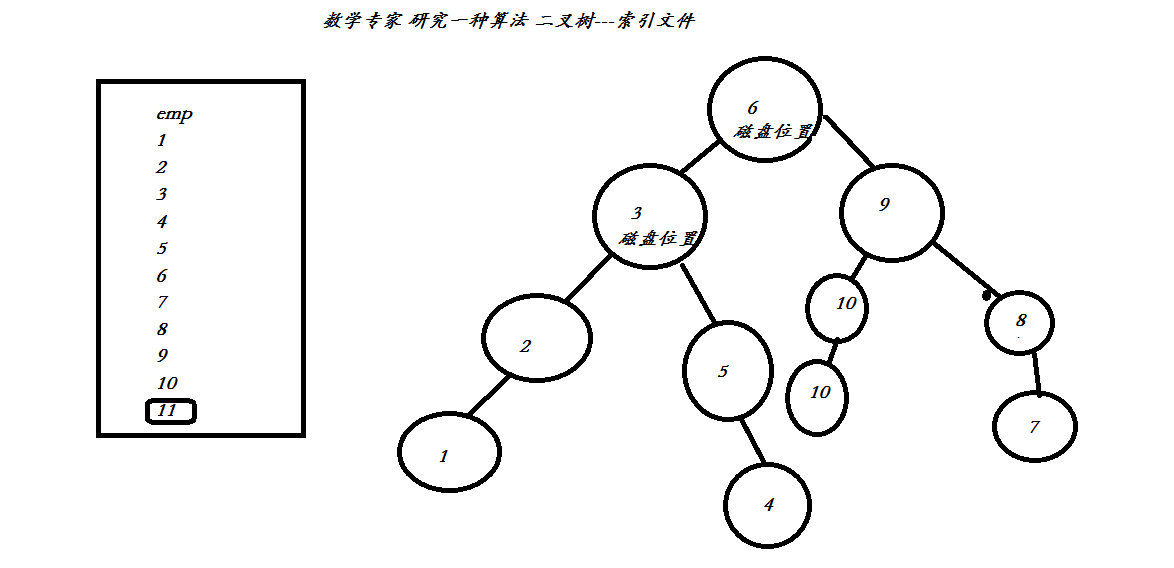
idintunsigned,

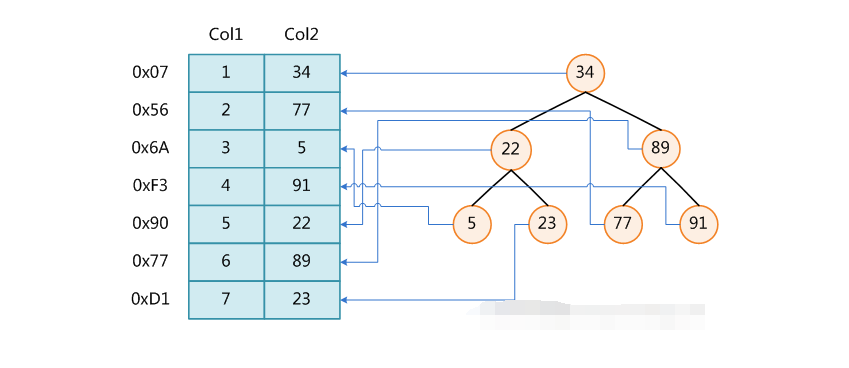
namevarchar(32)

)

createindex索引名on表(列1,列名2);

### 索引的实现原理





**数据库索引**，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。**索引的实现通常使用B树及其变种B+树**。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。

上图展示了一种可能的索引方式。左边是数据表，一共有两列七条记录，最左边的是数据记录的物理地址（注意逻辑上相邻的记录在磁盘上也并不是一定物理相邻的）。为了加快Col2的查找，可以维护一个右边所示的二叉查找树，每个节点分别包含索引键值和一个指向对应数据记录物理地址的指针，这样就可以运用二叉查找在O(log2n)的复杂度内获取到相应数据。

创建索引可以大大提高系统的性能。

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？因为，增加索引也有许多不利的方面。

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。**一般来说，应该在这些列上创建索引：**在经常需要搜索的列上，可以加快搜索的速度；在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

同样，对于有些列不应该创建索引。**一般来说，不应该创建索引的的这些列具有下列特点：**

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为text,image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，**修改性能和检索性能是互相矛盾的**。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

根据数据库的功能，可以在数据库设计器中创建三种索引：**唯一索引、主键索引和聚集索引**。

**唯一索引**

唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在employee表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。**主键索引**数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。**聚集索引**在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

局部性原理与磁盘预读

由于存储介质的特性，磁盘本身存取就比主存慢很多，再加上机械运动耗费，磁盘的存取速度往往是主存的几百分分之一，因此为了提高效率，要尽量减少磁盘I/O。为了达到这个目的，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存。这样做的理论依据是计算机科学中著名的**局部性原理**：**当一个数据被用到时，其附近的数据也通常会马上被使用。程序运行期间所需要的数据通常比较集中。**

由于磁盘顺序读取的效率很高（不需要寻道时间，只需很少的旋转时间），因此对于具有局部性的程序来说，预读可以提高I/O效率。

预读的长度一般为页（page）的整倍数。页是计算机管理存储器的逻辑块，硬件及操作系统往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页（在许多操作系统中，页得大小通常为4k），主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置并向后连续读取一页或几页载入内存中，然后异常返回，程序继续运行。

B-/+Tree索引的性能分析

到这里终于可以分析B-/+Tree索引的性能了。

上文说过一般使用磁盘I/O次数评价索引结构的优劣。先从B-Tree分析，根据B-Tree的定义，可知检索一次最多需要访问h个节点。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入。为了达到这个目的，在实际实现B-Tree还需要使用如下技巧：

每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个node只需一次I/O。

**B-Tree中一次检索最多需要h-1次I/O（根节点常驻内存），渐进复杂度为O(h)=O(logdN)。**一般实际应用中，出度d是非常大的数字，通常超过100，因此h非常小（通常不超过3）。

而红黑树这种结构，h明显要深的多。由于逻辑上很近的节点（父子）物理上可能很远，无法利用局部性，所以红黑树的I/O渐进复杂度也为O(h)，效率明显比B-Tree差很多。

**综上所述，用B-Tree作为索引结构效率是非常高的。**

**应该花时间学习B-树和B+树数据结构**

=============================================================================================================

1）B树

B树中每个节点包含了键值和键值对于的数据对象存放地址指针，所以成功搜索一个对象可以不用到达树的叶节点。

成功搜索包括节点内搜索和沿某一路径的搜索，成功搜索时间取决于关键码所在的层次以及节点内关键码的数量。

在B树中查找给定关键字的方法是：首先把根结点取来，在根结点所包含的关键字K1,…,kj查找给定的关键字（可用顺序查找或二分查找法），若找到等于给定值的关键字，则查找成功；否则，一定可以确定要查的关键字在某个Ki或Ki+1之间，于是取Pi所指的下一层索引节点块继续查找，直到找到，或指针Pi为空时查找失败。

2）B+树

B+树非叶节点中存放的关键码并不指示数据对象的地址指针，非也节点只是索引部分。所有的叶节点在同一层上，包含了全部关键码和相应数据对象的存放地址指针，且叶节点按关键码从小到大顺序链接。如果实际数据对象按加入的顺序存储而不是按关键码次数存储的话，叶节点的索引必须是稠密索引，若实际数据存储按关键码次序存放的话，叶节点索引时稀疏索引。

B+树有2个头指针，一个是树的根节点，一个是最小关键码的叶节点。

所以B+树有两种搜索方法：

一种是按叶节点自己拉起的链表顺序搜索。

一种是从根节点开始搜索，和B树类似，不过如果非叶节点的关键码等于给定值，搜索并不停止，而是继续沿右指针，一直查到叶节点上的关键码。所以无论搜索是否成功，都将走完树的所有层。

B+树中，数据对象的插入和删除仅在叶节点上进行。

这两种处理索引的数据结构的不同之处：  
a，B树中同一键值不会出现多次，并且它有可能出现在叶结点，也有可能出现在非叶结点中。而B+树的键一定会出现在叶结点中，并且有可能在非叶结点中也有可能重复出现，以维持B+树的平衡。  
b，因为B树键位置不定，且在整个树结构中只出现一次，虽然可以节省存储空间，但使得在插入、删除操作复杂度明显增加。B+树相比来说是一种较好的折中。  
c，B树的查询效率与键在树中的位置有关，最大时间复杂度与B+树相同(在叶结点的时候)，最小时间复杂度为1(在根结点的时候)。而B+树的时候复杂度对某建成的树是固定的。可以扫描2的次方。

### 索引的代价

占用磁盘空间

对DML(update、delete、insert)语句的效率影响

增删改会对索引影响，因为索引要重新整理。

|  |  |
| --- | --- |
| 存储引擎 | 允许的索引类型 |
| myisam | btree |
| innodb | btree |
| memory/yeap | Hash,btree |

#### 那些列上适合添加索引

1. 查询作为查询条件字段应该创建索引
2. 唯一性太差的字段不适合单独创建索引,即使频繁

Select\*fromempwheresex=’男’

1. 频繁更新字段，也不要定义索引。
2. 不会出现在where语句的字段不要创建索引

总结:满处一下条件的字段，才应该创建索引

1. 肯定在where条件经常使用
2. 该字段的内容不是唯一的几个值
3. 字段内容不是频繁变化

### 索引的注意事项

#### 创建一张表

新增dept数据

|  |
| --- |
| createPROCEDUREinsert\_dept(instartint(10),inmax\_numint(10))  BEGIN  declareiintDEFAULT0;  setautocommit=0;  REPEAT  seti=i+1;  insertintodeptvalues((start+i),rand\_string(10),rand\_string(8));  UNTILi=max\_num  endREPEAT;  commit;  END  执行  callinsert\_dept(100,10); |

创建主键索引

altertable表名addprimarykey(列名);

创建一个联合索引

altertabledeptaddindexmy\_ind(dname,loc);//dname左边的列,loc就是右边的列

注意:

1.对于创建的多列索引，如果不是使用第一部分，则不会创建索引。

explainselect\*fromdeptwhereloc='aaa'\G

就不会使用到索引

2.模糊查询在like前面有百分号开头会失效。

3.如果条件中有or，即使其中有条件带索引也不会使用。换言之，就是要求使用的所有字段，都必须建立索引,我们建议大家尽量避免使用or关键字

4.如果列类型是字符串，那一定要在条件中将数据使用引号引用起来。否则不使用索引。(添加时,字符串必须’’),也就是，如果列是字符串类型，就一定要用‘’把他包括起来.

5.如果mysql估计使用全表扫描要比使用索引快，则不使用索引。

### 查询所用使用率

showstatuslike‘handler\_read%’;

大家可以注意：  
handler\_read\_key:这个值越高越好，越高表示使用索引查询到的次数。

handler\_read\_rnd\_next:这个值越高，说明查询低效。

使用索引的时候，用like，放在前面索引失效，放在后面生效。





## SQL优化技巧

1. 使用groupby分组查询是，默认分组后，还会排序，可能会降低速度，

在groupby后面增加orderbynull就可以防止排序.

explainselect\*fromempgroupbydeptnoorderbynull;

1. 有些情况下，可以使用连接来替代子查询。因为使用join，MySQL不需要在内存中创建临时表。

select\*fromdept,empwheredept.deptno=emp.deptno;[简单处理方式]

select\*fromdeptleftjoinempondept.deptno=emp.deptno;[左外连接，更ok!]

1. 对查询进行优化，要尽量避免全表扫描，首先应考虑在where及orderby涉及的列上建立索引

应尽量避免在where子句中对字段进行null值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

selectidfromtwherenumisnull

最好不要给数据库留NULL，尽可能的使用NOTNULL填充数据库.

备注、描述、评论之类的可以设置为NULL，其他的，最好不要使用NULL。

不要以为NULL不需要空间，比如：char(100)型，在字段建立时，空间就固定了，不管是否插入值（NULL也包含在内），都是占用100个字符的空间的，如果是varchar这样的变长字段，null不占用空间。

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

selectidfromtwherenum=0

更多mysqlsql语句调优查看**1.对查询进行优化，应尽量避免全表扫描，首先应考虑在where及orderby涉及的列上建立索引。**

**2.应尽量避免在where子句中对字段进行null值判断，否则将导致**[**引擎**](https://www.baidu.com/s?wd=%E5%BC%95%E6%93%8E&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)**放弃使用索引而进行全表扫描，如：**

**selectidfromtwherenumisnull**

**可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：**

**selectidfromtwherenum=0**

**3.应尽量避免在where子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。**

**4.应尽量避免在where子句中使用or来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：**

**selectidfromtwherenum=10ornum=20**

**可以这样查询：**

selectidfromtwherenum=10

unionall

selectidfromtwherenum=20

**5.in和notin也要慎用，否则会导致全表扫描，如：**

**selectidfromtwherenumin(1,2,3)**

**对于连续的数值，能用between就不要用in了：**

**selectidfromtwherenumbetween1and3**

**6.下面的查询也将导致全表扫描：**

**selectidfromtwherenamelike'%abc%'**

**7.应尽量避免在where子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：**

**selectidfromtwherenum/2=100**

**应改为:**

**selectidfromtwherenum=100\*2**

**8.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：**

**selectidfromtwheresubstring(name,1,3)='abc'--name以abc开头的id**

**应改为:**

**selectidfromtwherenamelike'abc%'**

**9.不要在where子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。**

**10.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，**

**否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。**

**11.不要写一些没有意义的查询，如需要生成一个空表结构：**

**selectcol1,col2into#tfromtwhere1=0**

**这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：**

**createtable#t(...)**

**12.很多时候用exists代替in是一个好的选择：**

**selectnumfromawherenumin(selectnumfromb)**

**用下面的语句替换：**

**selectnumfromawhereexists(select1frombwherenum=a.num)**

**13.并不是所有索引对查询都有效，SQL是根据表**[**中数据**](https://www.baidu.com/s?wd=%E4%B8%AD%E6%95%B0%E6%8D%AE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)**来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，**

**如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。**

**14.索引并不是越多越好，索引固然可以提高相应的select的效率，但同时也降低了insert及update的效率，**

**因为insert或update时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。**

**一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。**

**15.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。**

**这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。**

**16.尽可能的使用varchar代替char，因为首先变长字段存储空间小，可以节省存储空间，**

**其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。**

**17.任何地方都不要使用select\*fromt，用具体的字段列表代替“\*”，不要返回用不到的任何字段。**

**18.避免频繁创建和删除临时表，以减少系统表资源的消耗。**

**19.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。**

**20.在新建临时表时，如果一次性插入数据量很大，那么可以使用selectinto代替createtable，避免造成大量log，**

**以提高速度；如果数据量不大，为了缓和系统表的资源，应先createtable，然后insert。**

**21.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先truncatetable，然后droptable，这样可以避免系统表的较长时间锁定。**

**22.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。**

**23.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。**

**24.与临时表一样，游标并不是不可使用。对小型数据集使用FAST\_FORWARD游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。**

**在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。**

**25.尽量避免大事务操作，提高系统并发能力。**

**26.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。**

**实际案例分析：拆分大的DELETE或INSERT语句，批量提交SQL语句**

**如果你需要在一个在线的网站上去执行一个大的DELETE或INSERT查询，你需要非常小心，要避免你的操作让你的整个网站停止相应。因为这两个操作是会锁表的，表一锁住了，别的操作都进不来了。**

**Apache会有很多的子进程或线程。所以，其工作起来相当有效率，而我们的服务器也不希望有太多的子进程，线程和数据库链接，这是极大的占服务器资源的事情，尤其是内存。**

**如果你把你的表锁上一段时间，比如30秒钟，那么对于一个有很高访问量的站点来说，这30秒所积累的访问进程/线程，数据库链接，打开的文件数，可能不仅仅会让你的WEB服务崩溃，还可能会让你的整台服务器马上挂了。**

**所以，如果你有一个大的处理，你一定把其拆分，使用LIMIToracle(rownum),sqlserver(top)条件是一个好的方法。下面是一个mysql示例：**

|  |
| --- |
| **while(1){**  **//每次只做1000条**  **mysql\_query(“deletefromlogswherelog\_date<=’2012-11-01’limit1000”);**  **if(mysql\_affected\_rows()==0){**  **//删除完成，退出！**  **break；**  **}**  **//每次暂停一段时间，释放表让其他进程/线程访问。**  **usleep(50000)**  **}** |

## MySQL数据引擎

使用的存储引擎myisam/innodb/memory

myisam存储:如果表对事务要求不高，同时是以查询和添加为主的，我们考虑使用myisam存储引擎.,比如bbs中的发帖表，回复表.

INNODB存储:对事务要求高，保存的数据都是重要数据，我们建议使用INNODB,比如订单表，账号表.

MyISAM和INNODB的区别

1.事务安全（MyISAM不支持事务，INNODB支持事务）

2.查询和添加速度（MyISAM批量插入速度快）

3.支持全文索引（MyISAM支持全文索引，INNODB不支持全文索引）

4.锁机制（MyISAM时表锁，innodb是行锁）

5.外键MyISAM不支持外键，INNODB支持外键.(在PHP开发中，通常不设置外键，通常是在程序中保证数据的一致)

Memory存储，比如我们数据变化频繁，不需要入库，同时又频繁的查询和修改，我们考虑使用memory,速度极快.（如果mysql重启的话，数据就不存在了）和reids很像。



### Myisam注意事项

如果你的数据库的存储引擎是myisam,请一定记住要定时进行碎片整理

举例说明:

|  |
| --- |
| createtabletest100(idintunsigned,namevarchar(32))engine=myisam;  insertintotest100values(1,’aaaaa’);  insertintotest100values(2,’bbbb’);  insertintotest100values(3,’ccccc’); |

我们应该定义对myisam进行整理

optimizetabletest100;

## 数据库数据备份

### 手动方式

cmd控制台:

在环境变量中配置mysql环境变量

mysqldump–u-账号–密码数据库[表名1表名2..]>文件路径

案例:mysqldump-u-rootroottest>d:\temp.sql

比如:把temp数据库备份到d:\temp.bak

mysqldump-uroot-proottest>f:\temp.bak

如果你希望备份是，数据库的某几张表

mysqldump-uroot-proottestdept>f:\temp.dept.sql

如何使用备份文件恢复我们的数据.

mysql控制台

sourced:\temp.dept.bak

### 自动方式

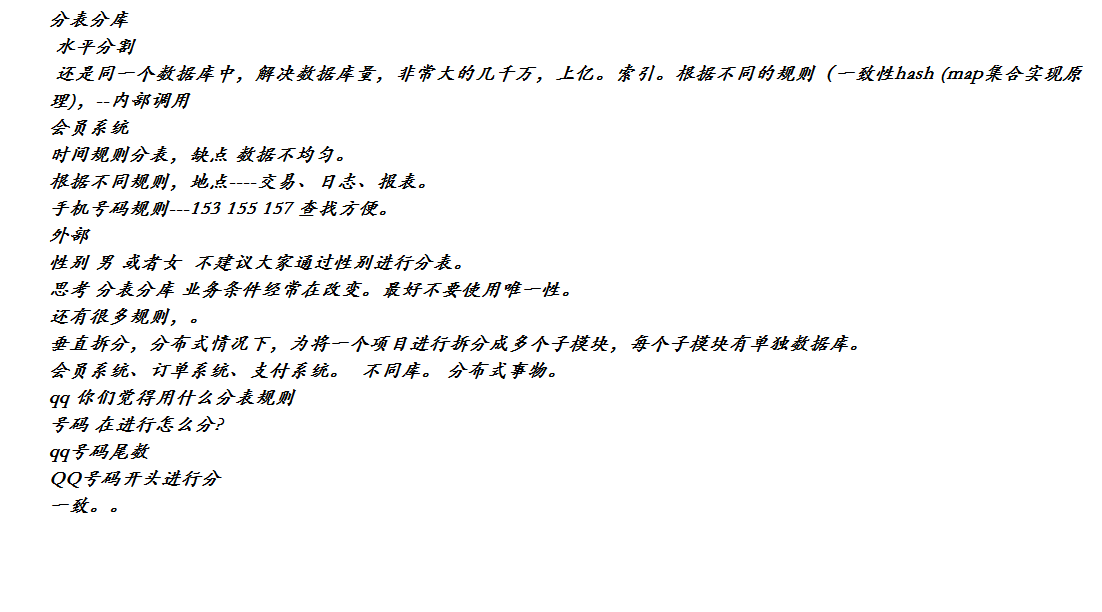
把备份数据库的指令，写入到bat文件,然后通过任务管理器去定时调用bat文件.

mytask.bat内容是:

|  |
| --- |
| @echooff  F:\path\mysqlanzhuang\bin\mysqldump-uroot-proottestdept>f:\temp.dept.sql |

创建执行计划任务执行脚本。

## 分表分库



业务条件经常在改变，最好不要使用一致性。

可能把多个分表分库规则结合在一起。

qq，根据位数，再根据开头。

要根据业务情况，查询条件来灵活运用。

日志系统用时间来分。

### 垂直拆分

垂直拆分就是要把表按模块划分到不同**数据库**表中（当然原则还是不破坏第三范式），这种拆分在大型网站的演变过程中是很常见的。当一个网站还在很小的时候，只有小量的人来开发和维护，各模块和表都在一起，当网站不断丰富和壮大的时候，也会变成多个子系统来支撑，这时就有按模块和功能把表划分出来的需求。其实，相对于垂直切分更进一步的是服务化改造，说得简单就是要把原来强耦合的系统拆分成多个弱耦合的服务，通过服务间的调用来满足业务需求看，因此表拆出来后要通过服务的形式暴露出去，而不是直接调用不同模块的表，淘宝在架构不断演变过程，最重要的一环就是服务化改造，把用户、交易、店铺、宝贝这些核心的概念抽取成独立的服务，也非常有利于进行局部的优化和治理，保障核心模块的稳定性

垂直拆分用于分布式场景。将项目拆分成多个子模块，每个子模块有单独数据库。会员系统，订单系统，支付系统…….都在不同的库中。解耦。要注意分布式事务。

### 水平拆分

上面谈到垂直切分只是把表按模块划分到不同数据库，但没有解决单表大数据量的问题，而水平切分就是要把一个表按照某种规则把数据划分到不同表或数据库里。例如像计费系统，通过按时间来划分表就比较合适，因为系统都是处理某一时间段的数据。而像SaaS应用，通过按用户维度来划分数据比较合适，因为用户与用户之间的隔离的，一般不存在处理多个用户数据的情况，简单的按user\_id范围来水平切分

通俗理解：水平拆分行，行数据拆分到不同表中，垂直拆分列，表数据拆分到不同表中

#### 水平分割案例

思路:在大型电商系统中，每天的会员人数不断的增加。达到一定瓶颈后如何优化查询。

可能大家会想到索引，万一用户量达到上亿级别，如何进行优化呢？

使用水平分割拆分数据库表。

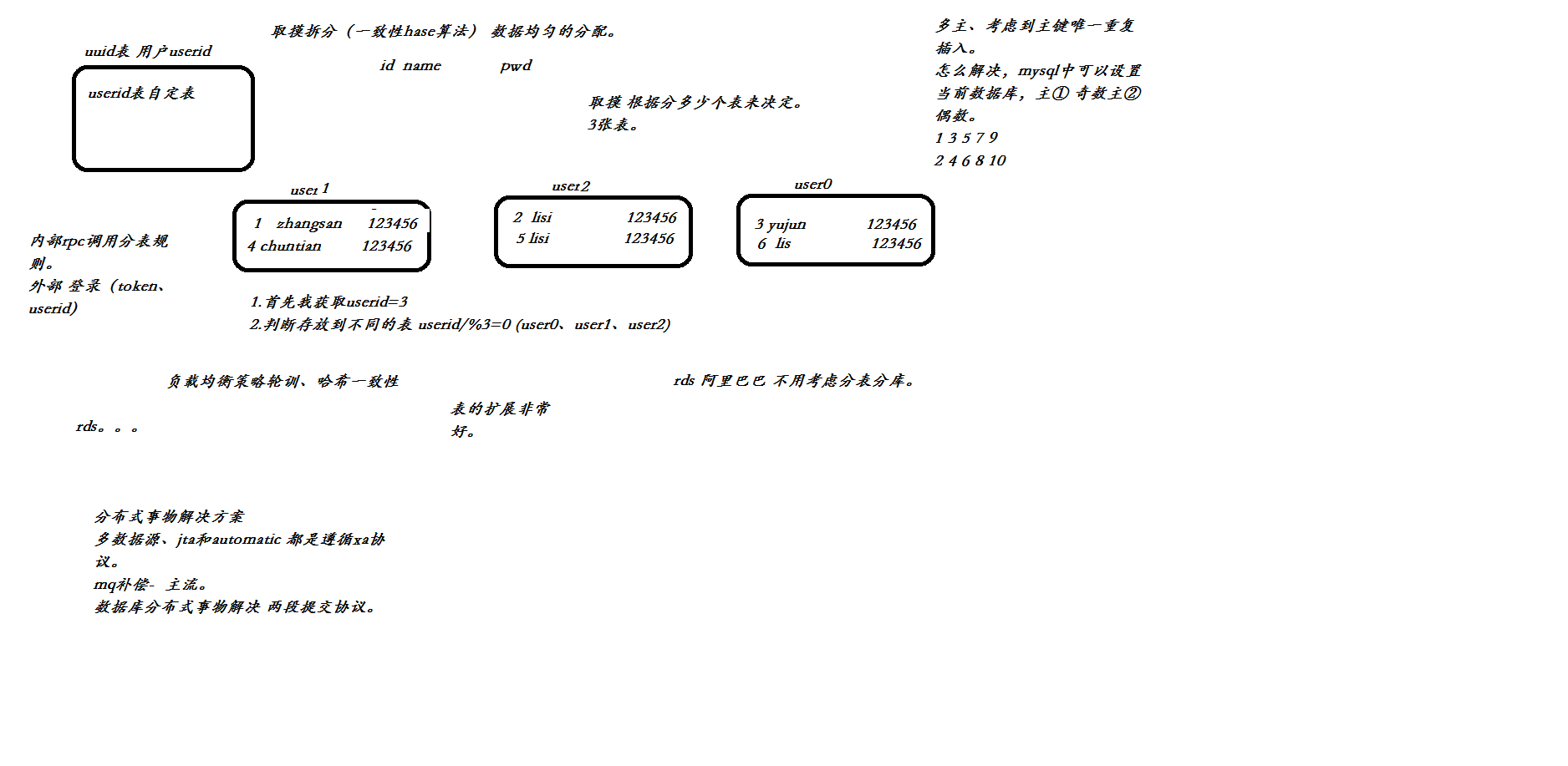
#### 如何使用水平拆分数据库

使用水平分割拆分表，具体根据业务需求，有的按照注册时间、取摸、账号规则、年份等。

首选选用hash一致性，其次是手机号，

#### 使用取摸方式分表

哈希取模算法，数据能够均匀分配。



首先我创建三张表user0/user1/user2,然后我再创建uuid表，该表的作用就是提供自增的id。

|  |
| --- |
| createtableuser0(  idintunsignedprimarykey,  namevarchar(32)notnulldefault'',  pwdvarchar(32)notnulldefault'')  engine=myisamcharsetutf8;  createtableuser1(  idintunsignedprimarykey,  namevarchar(32)notnulldefault'',  pwdvarchar(32)notnulldefault'')  engine=myisamcharsetutf8;  createtableuser2(  idintunsignedprimarykey,  namevarchar(32)notnulldefault'',  pwdvarchar(32)notnulldefault'')  engine=myisamcharsetutf8;  createtableuuid(  idintunsignedprimarykeyauto\_increment)engine=myisamcharsetutf8; |

#### 创建一个demo项目

##### POM文件

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.3.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies> |

##### Service代码

|  |
| --- |
| @Service  publicclassUserService{  @Autowired  privateJdbcTemplatejdbcTemplate;  publicStringregit(Stringname,Stringpwd){  //1.先获取到自定增长ID  StringidInsertSQL="INSERTINTOuuidVALUES(NULL);";  jdbcTemplate.update(idInsertSQL);  LonginsertId=jdbcTemplate.queryForObject("selectlast\_insert\_id()",Long.class);  //2.判断存储表名称  StringtableName="user"+insertId%3;  //3.注册数据  StringinsertUserSql="INSERTINTO"+tableName+"VALUES('"+insertId+"','"+name+"','"+pwd  +"');";  System.*out*.println("insertUserSql:"+insertUserSql);  jdbcTemplate.update(insertUserSql);  return"success";  }  publicStringget(Longid){  StringtableName="user"+id%3;  Stringsql="selectnamefrom"+tableName+"whereid="+id;  System.*out*.println("SQL:"+sql);  Stringname=jdbcTemplate.queryForObject(sql,String.class);  returnname;  }  } |

##### Controller

|  |
| --- |
| @RestController  publicclassUserController{  @Autowired  privateUserServiceuserService;  @RequestMapping("/regit")  publicStringregit(Stringname,Stringpwd){  returnuserService.regit(name,pwd);  }  @RequestMapping("/get")  publicStringget(Longid){  Stringname=userService.get(id);  returnname;  }  } |

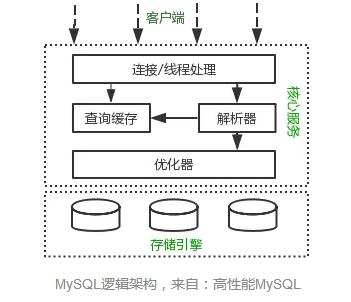
# 学习MySQL优化原理

前言

说起MySQL的查询优化，相信大家收藏了一堆奇技淫巧：不能使用SELECT \*、不使用NULL字段、合理创建索引、为字段选择合适的数据类型..... 你是否真的理解这些优化技巧？是否理解其背后的工作原理？在实际场景下性能真有提升吗？我想未必。因而理解这些优化建议背后的原理就尤为重要，希望本文能让你重新审视这些优化建议，并在实际业务场景下合理的运用。

MySQL逻辑架构

如果能在头脑中构建一幅MySQL各组件之间如何协同工作的架构图，有助于深入理解MySQL服务器。下图展示了MySQL的逻辑架构图。



MySQL逻辑架构整体分为三层，最上层为客户端层，并非MySQL所独有，诸如：连接处理、授权认证、安全等功能均在这一层处理。

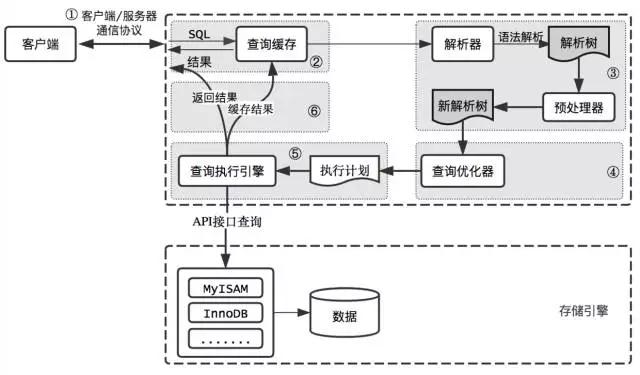
MySQL大多数核心服务均在中间这一层，包括查询解析、分析、优化、缓存、内置函数(比如：时间、数学、加密等函数)。所有的跨存储引擎的功能也在这一层实现：存储过程、触发器、视图等。

最下层为存储引擎，其负责MySQL中的数据存储和提取。和Linux下的文件系统类似，每种存储引擎都有其优势和劣势。中间的服务层通过API与存储引擎通信，这些API接口屏蔽了不同存储引擎间的差异。

MySQL查询过程

我们总是希望MySQL能够获得更高的查询性能，最好的办法是弄清楚MySQL是如何优化和执行查询的。一旦理解了这一点，就会发现：很多的查询优化工作实际上就是遵循一些原则让MySQL的优化器能够按照预想的合理方式运行而已。

当向MySQL发送一个请求的时候，MySQL到底做了些什么呢？



MySQL查询过程

客户端/服务端通信协议

MySQL客户端/服务端通信协议是“半双工”的：在任一时刻，要么是服务器向客户端发送数据，要么是客户端向服务器发送数据，这两个动作不能同时发生。一旦一端开始发送消息，另一端要接收完整个消息才能响应它，所以我们无法也无须将一个消息切成小块独立发送，也没有办法进行流量控制。

客户端用一个单独的数据包将查询请求发送给服务器，所以当查询语句很长的时候，需要设置max\_allowed\_packet参数。但是需要注意的是，如果查询实在是太大，服务端会拒绝接收更多数据并抛出异常。

与之相反的是，服务器响应给用户的数据通常会很多，由多个数据包组成。但是当服务器响应客户端请求时，客户端必须完整的接收整个返回结果，而不能简单的只取前面几条结果，然后让服务器停止发送。因而在实际开发中，尽量保持查询简单且只返回必需的数据，减小通信间数据包的大小和数量是一个非常好的习惯，这也是查询中尽量避免使用SELECT \*以及加上LIMIT限制的原因之一。

查询缓存

在解析一个查询语句前，如果查询缓存是打开的，那么MySQL会检查这个查询语句是否命中查询缓存中的数据。如果当前查询恰好命中查询缓存，在检查一次用户权限后直接返回缓存中的结果。这种情况下，查询不会被解析，也不会生成执行计划，更不会执行。

MySQL将缓存存放在一个引用表（不要理解成table，可以认为是类似于HashMap的数据结构），通过一个哈希值索引，这个哈希值通过查询本身、当前要查询的数据库、客户端协议版本号等一些可能影响结果的信息计算得来。所以两个查询在任何字符上的不同（例如：空格、注释），都会导致缓存不会命中。

如果查询中包含任何用户自定义函数、存储函数、用户变量、临时表、MySQL库中的系统表，其查询结果都不会被缓存。比如函数NOW()或者CURRENT\_DATE()会因为不同的查询时间，返回不同的查询结果，再比如包含CURRENT\_USER或者CONNECION\_ID()的查询语句会因为不同的用户而返回不同的结果，将这样的查询结果缓存起来没有任何的意义。

既然是缓存，就会失效，那查询缓存何时失效呢？MySQL的查询缓存系统会跟踪查询中涉及的每个表，如果这些表（数据或结构）发生变化，那么和这张表相关的所有缓存数据都将失效。正因为如此，在任何的写操作时，MySQL必须将对应表的所有缓存都设置为失效。如果查询缓存非常大或者碎片很多，这个操作就可能带来很大的系统消耗，甚至导致系统僵死一会儿。而且查询缓存对系统的额外消耗也不仅仅在写操作，读操作也不例外：

1. 任何的查询语句在开始之前都必须经过检查，即使这条SQL语句永远不会命中缓存
2. 如果查询结果可以被缓存，那么执行完成后，会将结果存入缓存，也会带来额外的系统消耗

基于此，我们要知道并不是什么情况下查询缓存都会提高系统性能，缓存和失效都会带来额外消耗，只有当缓存带来的资源节约大于其本身消耗的资源时，才会给系统带来性能提升。但要如何评估打开缓存是否能够带来性能提升是一件非常困难的事情，也不在本文讨论的范畴内。如果系统确实存在一些性能问题，可以尝试打开查询缓存，并在数据库设计上做一些优化，比如：

1. 用多个小表代替一个大表，注意不要过度设计
2. 批量插入代替循环单条插入
3. 合理控制缓存空间大小，一般来说其大小设置为几十兆比较合适
4. 可以通过SQL\_CACHE和SQL\_NO\_CACHE来控制某个查询语句是否需要进行缓存

最后的忠告是不要轻易打开查询缓存，特别是写密集型应用。如果你实在是忍不住，可以将query\_cache\_type设置为DEMAND，这时只有加入SQL\_CACHE的查询才会走缓存，其他查询则不会，这样可以非常自由地控制哪些查询需要被缓存。

当然查询缓存系统本身是非常复杂的，这里讨论的也只是很小的一部分，其他更深入的话题，比如：缓存是如何使用内存的？如何控制内存的碎片化？事务对查询缓存有何影响等等，读者可以自行阅读相关资料，这里权当抛砖引玉吧。

语法解析和预处理

MySQL通过关键字将SQL语句进行解析，并生成一颗对应的解析树。这个过程解析器主要通过语法规则来验证和解析。比如SQL中是否使用了错误的关键字或者关键字的顺序是否正确等等。预处理则会根据MySQL规则进一步检查解析树是否合法。比如检查要查询的数据表和数据列是否存在等。

查询优化

经过前面的步骤生成的语法树被认为是合法的了，并且由优化器将其转化成查询计划。多数情况下，一条查询可以有很多种执行方式，最后都返回相应的结果。优化器的作用就是找到这其中最好的执行计划。

MySQL使用基于成本的优化器，它尝试预测一个查询使用某种执行计划时的成本，并选择其中成本最小的一个。在MySQL可以通过查询当前会话的last\_query\_cost的值来得到其计算当前查询的成本。

mysql> select \* from t\_message limit 10;

...省略结果集

mysql> show status like 'last\_query\_cost';

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

| Variable\_name | Value |

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

| Last\_query\_cost | 6391.799000 |

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

示例中的结果表示优化器认为大概需要做6391个数据页的随机查找才能完成上面的查询。这个结果是根据一些列的统计信息计算得来的，这些统计信息包括：每张表或者索引的页面个数、索引的基数、索引和数据行的长度、索引的分布情况等等。

有非常多的原因会导致MySQL选择错误的执行计划，比如统计信息不准确、不会考虑不受其控制的操作成本（用户自定义函数、存储过程）、MySQL认为的最优跟我们想的不一样（我们希望执行时间尽可能短，但MySQL值选择它认为成本小的，但成本小并不意味着执行时间短）等等。

MySQL的查询优化器是一个非常复杂的部件，它使用了非常多的优化策略来生成一个最优的执行计划：

* 重新定义表的关联顺序（多张表关联查询时，并不一定按照SQL中指定的顺序进行，但有一些技巧可以指定关联顺序）
* 优化MIN()和MAX()函数（找某列的最小值，如果该列有索引，只需要查找B+Tree索引最左端，反之则可以找到最大值，具体原理见下文）
* 提前终止查询（比如：使用Limit时，查找到满足数量的结果集后会立即终止查询）
* 优化排序（在老版本MySQL会使用两次传输排序，即先读取行指针和需要排序的字段在内存中对其排序，然后再根据排序结果去读取数据行，而新版本采用的是单次传输排序，也就是一次读取所有的数据行，然后根据给定的列排序。对于I/O密集型应用，效率会高很多）

随着MySQL的不断发展，优化器使用的优化策略也在不断的进化，这里仅仅介绍几个非常常用且容易理解的优化策略，其他的优化策略，大家自行查阅吧。

查询执行引擎

在完成解析和优化阶段以后，MySQL会生成对应的执行计划，查询执行引擎根据执行计划给出的指令逐步执行得出结果。整个执行过程的大部分操作均是通过调用存储引擎实现的接口来完成，这些接口被称为handler API。查询过程中的每一张表由一个handler实例表示。实际上，MySQL在查询优化阶段就为每一张表创建了一个handler实例，优化器可以根据这些实例的接口来获取表的相关信息，包括表的所有列名、索引统计信息等。存储引擎接口提供了非常丰富的功能，但其底层仅有几十个接口，这些接口像搭积木一样完成了一次查询的大部分操作。

返回结果给客户端

查询执行的最后一个阶段就是将结果返回给客户端。即使查询不到数据，MySQL仍然会返回这个查询的相关信息，比如该查询影响到的行数以及执行时间等。

如果查询缓存被打开且这个查询可以被缓存，MySQL也会将结果存放到缓存中。

结果集返回客户端是一个增量且逐步返回的过程。有可能MySQL在生成第一条结果时，就开始向客户端逐步返回结果集了。这样服务端就无须存储太多结果而消耗过多内存，也可以让客户端第一时间获得返回结果。需要注意的是，结果集中的每一行都会以一个满足①中所描述的通信协议的数据包发送，再通过TCP协议进行传输，在传输过程中，可能对MySQL的数据包进行缓存然后批量发送。

回头总结一下MySQL整个查询执行过程，总的来说分为6个步骤：

* 客户端向MySQL服务器发送一条查询请求
* 服务器首先检查查询缓存，如果命中缓存，则立刻返回存储在缓存中的结果。否则进入下一阶段
* 服务器进行SQL解析、预处理、再由优化器生成对应的执行计划
* MySQL根据执行计划，调用存储引擎的API来执行查询
* 将结果返回给客户端，同时缓存查询结果

性能优化建议

看了这么多，你可能会期待给出一些优化手段，是的，下面会从3个不同方面给出一些优化建议。但请等等，还有一句忠告要先送给你：不要听信你看到的关于优化的“绝对真理”，包括本文所讨论的内容，而应该是在实际的业务场景下通过测试来验证你关于执行计划以及响应时间的假设。

1Scheme设计与数据类型优化

选择数据类型只要遵循小而简单的原则就好，越小的数据类型通常会更快，占用更少的磁盘、内存，处理时需要的CPU周期也更少。越简单的数据类型在计算时需要更少的CPU周期，比如，整型就比字符操作代价低，因而会使用整型来存储ip地址，使用DATETIME来存储时间，而不是使用字符串。

这里总结几个可能容易理解错误的技巧：

1. 通常来说把可为NULL的列改为NOT NULL不会对性能提升有多少帮助，只是如果计划在列上创建索引，就应该将该列设置为NOT NULL。
2. 对整数类型指定宽度，比如INT(11)，没有任何卵用。INT使用32位（4个字节）存储空间，那么它的表示范围已经确定，所以INT(1)和INT(20)对于存储和计算是相同的。
3. UNSIGNED表示不允许负值，大致可以使正数的上限提高一倍。比如TINYINT存储范围是-128 ~ 127，而UNSIGNED TINYINT存储的范围却是0 - 255。
4. 通常来讲，没有太大的必要使用DECIMAL数据类型。即使是在需要存储财务数据时，仍然可以使用BIGINT。比如需要精确到万分之一，那么可以将数据乘以一百万然后使用BIGINT存储。这样可以避免浮点数计算不准确和DECIMAL精确计算代价高的问题。
5. TIMESTAMP使用4个字节存储空间，DATETIME使用8个字节存储空间。因而，TIMESTAMP只能表示1970 - 2038年，比DATETIME表示的范围小得多，而且TIMESTAMP的值因时区不同而不同。
6. 大多数情况下没有使用枚举类型的必要，其中一个缺点是枚举的字符串列表是固定的，添加和删除字符串（枚举选项）必须使用ALTER TABLE（如果只只是在列表末尾追加元素，不需要重建表）。
7. schema的列不要太多。原因是存储引擎的API工作时需要在服务器层和存储引擎层之间通过行缓冲格式拷贝数据，然后在服务器层将缓冲内容解码成各个列，这个转换过程的代价是非常高的。如果列太多而实际使用的列又很少的话，有可能会导致CPU占用过高。
8. 大表ALTER TABLE非常耗时，MySQL执行大部分修改表结果操作的方法是用新的结构创建一个张空表，从旧表中查出所有的数据插入新表，然后再删除旧表。尤其当内存不足而表又很大，而且还有很大索引的情况下，耗时更久。当然有一些奇技淫巧可以解决这个问题，有兴趣可自行查阅。

2创建高性能索引

索引是提高MySQL查询性能的一个重要途径，但过多的索引可能会导致过高的磁盘使用率以及过高的内存占用，从而影响应用程序的整体性能。应当尽量避免事后才想起添加索引，因为事后可能需要监控大量的SQL才能定位到问题所在，而且添加索引的时间肯定是远大于初始添加索引所需要的时间，可见索引的添加也是非常有技术含量的。

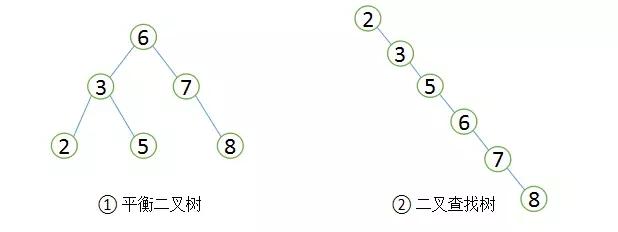
接下来将向你展示一系列创建高性能索引的策略，以及每条策略其背后的工作原理。但在此之前，先了解与索引相关的一些算法和数据结构，将有助于更好的理解后文的内容。

3索引相关的数据结构和算法

通常我们所说的索引是指B-Tree索引，它是目前关系型数据库中查找数据最为常用和有效的索引，大多数存储引擎都支持这种索引。使用B-Tree这个术语，是因为MySQL在CREATE TABLE或其它语句中使用了这个关键字，但实际上不同的存储引擎可能使用不同的数据结构，比如InnoDB就是使用的B+Tree。

B+Tree中的B是指balance，意为平衡。需要注意的是，B+树索引并不能找到一个给定键值的具体行，它找到的只是被查找数据行所在的页，接着数据库会把页读入到内存，再在内存中进行查找，最后得到要查找的数据。

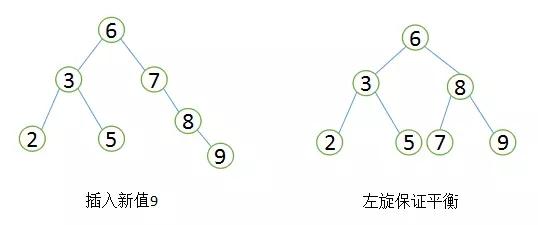
在介绍B+Tree前，先了解一下二叉查找树，它是一种经典的数据结构，其左子树的值总是小于根的值，右子树的值总是大于根的值，如下图①。如果要在这课树中查找值为5的记录，其大致流程：先找到根，其值为6，大于5，所以查找左子树，找到3，而5大于3，接着找3的右子树，总共找了3次。同样的方法，如果查找值为8的记录，也需要查找3次。所以二叉查找树的平均查找次数为(3 + 3 + 3 + 2 + 2 + 1) / 6 = 2.3次，而顺序查找的话，查找值为2的记录，仅需要1次，但查找值为8的记录则需要6次，所以顺序查找的平均查找次数为：(1 + 2 + 3 + 4 + 5 + 6) / 6 = 3.3次，因此大多数情况下二叉查找树的平均查找速度比顺序查找要快。



二叉查找树和平衡二叉树

由于二叉查找树可以任意构造，同样的值，可以构造出如图②的二叉查找树，显然这棵二叉树的查询效率和顺序查找差不多。若想二叉查找数的查询性能最高，需要这棵二叉查找树是平衡的，也即平衡二叉树（AVL树）。

平衡二叉树首先需要符合二叉查找树的定义，其次必须满足任何节点的两个子树的高度差不能大于1。显然图②不满足平衡二叉树的定义，而图①是一课平衡二叉树。平衡二叉树的查找性能是比较高的（性能最好的是最优二叉树），查询性能越好，维护的成本就越大。比如图①的平衡二叉树，当用户需要插入一个新的值9的节点时，就需要做出如下变动。

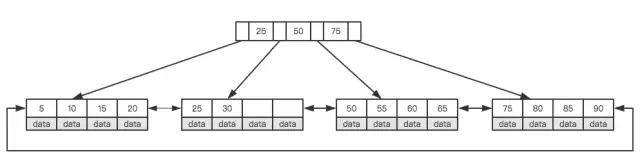


平衡二叉树旋转

通过一次左旋操作就将插入后的树重新变为平衡二叉树是最简单的情况了，实际应用场景中可能需要旋转多次。至此我们可以考虑一个问题，平衡二叉树的查找效率还不错，实现也非常简单，相应的维护成本还能接受，为什么MySQL索引不直接使用平衡二叉树？

随着数据库中数据的增加，索引本身大小随之增加，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘I/O消耗，相对于内存存取，I/O存取的消耗要高几个数量级。可以想象一下一棵几百万节点的二叉树的深度是多少？如果将这么大深度的一颗二叉树放磁盘上，每读取一个节点，需要一次磁盘的I/O读取，整个查找的耗时显然是不能够接受的。那么如何减少查找过程中的I/O存取次数？

一种行之有效的解决方法是减少树的深度，将二叉树变为m叉树（多路搜索树），而B+Tree就是一种多路搜索树。理解B+Tree时，只需要理解其最重要的两个特征即可：第一，所有的关键字（可以理解为数据）都存储在叶子节点（Leaf Page），非叶子节点（Index Page）并不存储真正的数据，所有记录节点都是按键值大小顺序存放在同一层叶子节点上。其次，所有的叶子节点由指针连接。如下图为高度为2的简化了的B+Tree。



简化B+Tree

怎么理解这两个特征？MySQL将每个节点的大小设置为一个页的整数倍（原因下文会介绍），也就是在节点空间大小一定的情况下，每个节点可以存储更多的内结点，这样每个结点能索引的范围更大更精确。所有的叶子节点使用指针链接的好处是可以进行区间访问，比如上图中，如果查找大于20而小于30的记录，只需要找到节点20，就可以遍历指针依次找到25、30。如果没有链接指针的话，就无法进行区间查找。这也是MySQL使用B+Tree作为索引存储结构的重要原因。

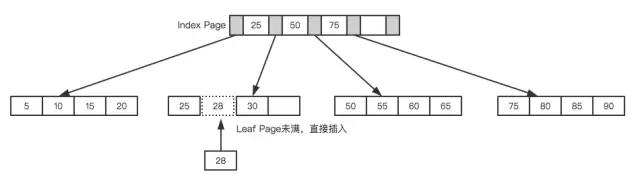
MySQL为何将节点大小设置为页的整数倍，这就需要理解磁盘的存储原理。磁盘本身存取就比主存慢很多，在加上机械运动损耗（特别是普通的机械硬盘），磁盘的存取速度往往是主存的几百万分之一，为了尽量减少磁盘I/O，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存，预读的长度一般为页的整数倍。

“页是计算机管理存储器的逻辑块，硬件及OS往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页（许多OS中，页的大小通常为4K）。主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置并向后连续读取一页或几页载入内存中，然后一起返回，程序继续运行。”

MySQL巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入。为了达到这个目的，每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了读取一个节点只需一次I/O。假设B+Tree的高度为h，一次检索最多需要h-1I/O（根节点常驻内存），复杂度$O(h) = O(log\_{M}N)$。实际应用场景中，M通常较大，常常超过100，因此树的高度一般都比较小，通常不超过3。

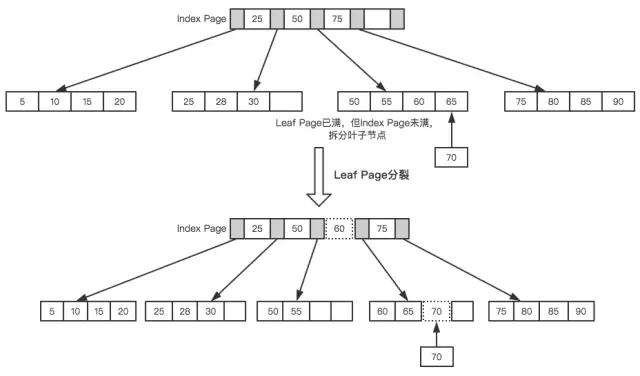
最后简单了解下B+Tree节点的操作，在整体上对索引的维护有一个大概的了解，虽然索引可以大大提高查询效率，但维护索引仍要花费很大的代价，因此合理的创建索引也就尤为重要。

仍以上面的树为例，我们假设每个节点只能存储4个内节点。首先要插入第一个节点28，如下图所示。



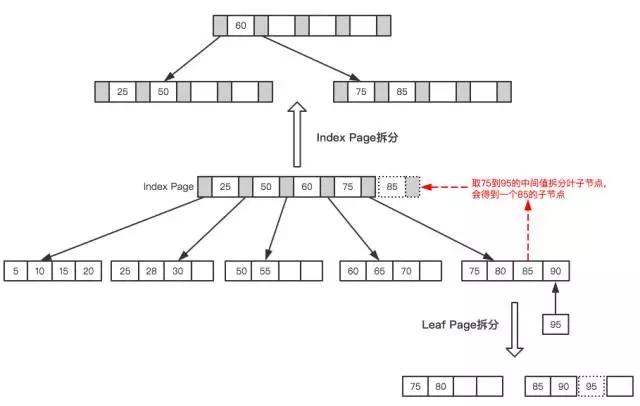
leaf page和index page都没有满

接着插入下一个节点70，在Index Page中查询后得知应该插入到50 - 70之间的叶子节点，但叶子节点已满，这时候就需要进行也分裂的操作，当前的叶子节点起点为50，所以根据中间值来拆分叶子节点，如下图所示。



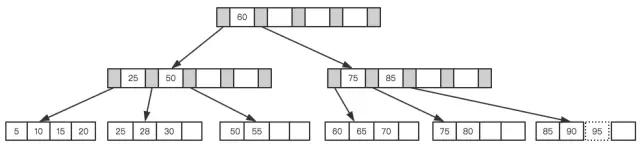
Leaf Page拆分

最后插入一个节点95，这时候Index Page和Leaf Page都满了，就需要做两次拆分，如下图所示。



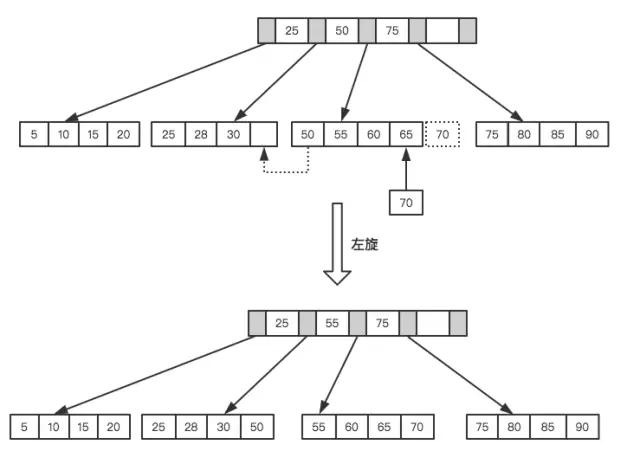
Leaf Page与Index Page拆分

拆分后最终形成了这样一颗树。



最终树

B+Tree为了保持平衡，对于新插入的值需要做大量的拆分页操作，而页的拆分需要I/O操作，为了尽可能的减少页的拆分操作，B+Tree也提供了类似于平衡二叉树的旋转功能。当Leaf Page已满但其左右兄弟节点没有满的情况下，B+Tree并不急于去做拆分操作，而是将记录移到当前所在页的兄弟节点上。通常情况下，左兄弟会被先检查用来做旋转操作。就比如上面第二个示例，当插入70的时候，并不会去做页拆分，而是左旋操作。



左旋操作

通过旋转操作可以最大限度的减少页分裂，从而减少索引维护过程中的磁盘的I/O操作，也提高索引维护效率。需要注意的是，删除节点跟插入节点类似，仍然需要旋转和拆分操作，这里就不再说明。

高性能策略

通过上文，相信你对B+Tree的数据结构已经有了大致的了解，但MySQL中索引是如何组织数据的存储呢？以一个简单的示例来说明，假如有如下数据表：

CREATE TABLE People(

last\_name varchar(50) not null,

first\_name varchar(50) not null,

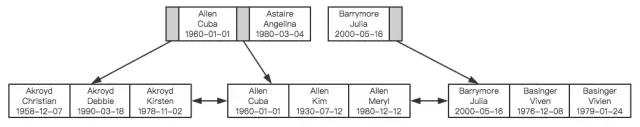
dob date not null,

gender enum(`m`,`f`) not null,

key(last\_name,first\_name,dob)

);

对于表中每一行数据，索引中包含了last\_name、first\_name、dob列的值，下图展示了索引是如何组织数据存储的。



索引如何组织数据存储，来自：高性能MySQL

可以看到，索引首先根据第一个字段来排列顺序，当名字相同时，则根据第三个字段，即出生日期来排序，正是因为这个原因，才有了索引的“最左原则”。

1、MySQL不会使用索引的情况：非独立的列

“独立的列”是指索引列不能是表达式的一部分，也不能是函数的参数。比如：

select \* from where id + 1 = 5

我们很容易看出其等价于 id = 4，但是MySQL无法自动解析这个表达式，使用函数是同样的道理。

2、前缀索引

如果列很长，通常可以索引开始的部分字符，这样可以有效节约索引空间，从而提高索引效率。

3、多列索引和索引顺序

在多数情况下，在多个列上建立独立的索引并不能提高查询性能。理由非常简单，MySQL不知道选择哪个索引的查询效率更好，所以在老版本，比如MySQL5.0之前就会随便选择一个列的索引，而新的版本会采用合并索引的策略。举个简单的例子，在一张电影演员表中，在actor\_id和film\_id两个列上都建立了独立的索引，然后有如下查询：

select film\_id,actor\_id from film\_actor where actor\_id = 1 or film\_id = 1

老版本的MySQL会随机选择一个索引，但新版本做如下的优化：

select film\_id,actor\_id from film\_actor where actor\_id = 1

union all

select film\_id,actor\_id from film\_actor where film\_id = 1 and actor\_id <> 1

* 当出现多个索引做相交操作时（多个AND条件），通常来说一个包含所有相关列的索引要优于多个独立索引。
* 当出现多个索引做联合操作时（多个OR条件），对结果集的合并、排序等操作需要耗费大量的CPU和内存资源，特别是当其中的某些索引的选择性不高，需要返回合并大量数据时，查询成本更高。所以这种情况下还不如走全表扫描。

因此explain时如果发现有索引合并（Extra字段出现Using union），应该好好检查一下查询和表结构是不是已经是最优的，如果查询和表都没有问题，那只能说明索引建的非常糟糕，应当慎重考虑索引是否合适，有可能一个包含所有相关列的多列索引更适合。

前面我们提到过索引如何组织数据存储的，从图中可以看到多列索引时，索引的顺序对于查询是至关重要的，很明显应该把选择性更高的字段放到索引的前面，这样通过第一个字段就可以过滤掉大多数不符合条件的数据。

<section style="margin: 10px 0px; padding: 15px 20px 15px 45px; max-width: 100%; box-sizing: border-box; font-size: 14px; line-height: 22.39px; outline: 0px; border-width: 0px; border-style: initial; border-color: currentcolor; vertical-align: baseline; background-image: url(" http:="" mmbiz.qpic.cn="" mmbiz\_jpg="" tibrg3aoijttt5wd7pstdp8xn9fcaqn0hzm4ung7awpvy0vhxe5stzfr97tfcd3orepfelzkiawqpkjmvgnbnenq="" 0?wx\_fmt="jpeg&quot;);" background-color:="" rgb(241,="" 241,="" 241);="" background-position:="" 1%="" 5px;="" background-repeat:="" no-repeat;="" word-wrap:="" break-word="" !important;"="">

索引选择性是指不重复的索引值和数据表的总记录数的比值，选择性越高查询效率越高，因为选择性越高的索引可以让MySQL在查询时过滤掉更多的行。唯一索引的选择性是1，这时最好的索引选择性，性能也是最好的。

理解索引选择性的概念后，就不难确定哪个字段的选择性较高了，查一下就知道了，比如：

SELECT \* FROM payment where staff\_id = 2 and customer\_id = 584

是应该创建(staff\_id,customer\_id)的索引还是应该颠倒一下顺序？执行下面的查询，哪个字段的选择性更接近1就把哪个字段索引前面就好。

select count(distinct staff\_id)/count(\*) as staff\_id\_selectivity,

count(distinct customer\_id)/count(\*) as customer\_id\_selectivity,

count(\*) from payment

多数情况下使用这个原则没有任何问题，但仍然注意你的数据中是否存在一些特殊情况。举个简单的例子，比如要查询某个用户组下有过交易的用户信息：

select user\_id from trade where user\_group\_id = 1 and trade\_amount > 0

MySQL为这个查询选择了索引(user\_group\_id,trade\_amount)，如果不考虑特殊情况，这看起来没有任何问题，但实际情况是这张表的大多数数据都是从老系统中迁移过来的，由于新老系统的数据不兼容，所以就给老系统迁移过来的数据赋予了一个默认的用户组。这种情况下，通过索引扫描的行数跟全表扫描基本没什么区别，索引也就起不到任何作用。

推广开来说，经验法则和推论在多数情况下是有用的，可以指导我们开发和设计，但实际情况往往会更复杂，实际业务场景下的某些特殊情况可能会摧毁你的整个设计。

4、避免多个范围条件

实际开发中，我们会经常使用多个范围条件，比如想查询某个时间段内登录过的用户：

select user.\* from user where login\_time > '2017-04-01' and age between 18 and 30;

这个查询有一个问题：它有两个范围条件，login\_time列和age列，MySQL可以使用login\_time列的索引或者age列的索引，但无法同时使用它们。

5、覆盖索引

如果一个索引包含或者说覆盖所有需要查询的字段的值，那么就没有必要再回表查询，这就称为覆盖索引。覆盖索引是非常有用的工具，可以极大的提高性能，因为查询只需要扫描索引会带来许多好处：

* 索引条目远小于数据行大小，如果只读取索引，极大减少数据访问量
* 索引是有按照列值顺序存储的，对于I/O密集型的范围查询要比随机从磁盘读取每一行数据的IO要少的多

6、使用索引扫描来排序

MySQL有两种方式可以生产有序的结果集，其一是对结果集进行排序的操作，其二是按照索引顺序扫描得出的结果自然是有序的。如果explain的结果中type列的值为index表示使用了索引扫描来做排序。

扫描索引本身很快，因为只需要从一条索引记录移动到相邻的下一条记录。但如果索引本身不能覆盖所有需要查询的列，那么就不得不每扫描一条索引记录就回表查询一次对应的行。这个读取操作基本上是随机I/O，因此按照索引顺序读取数据的速度通常要比顺序地全表扫描要慢。

在设计索引时，如果一个索引既能够满足排序，又满足查询，是最好的。

只有当索引的列顺序和ORDER BY子句的顺序完全一致，并且所有列的排序方向也一样时，才能够使用索引来对结果做排序。如果查询需要关联多张表，则只有ORDER BY子句引用的字段全部为第一张表时，才能使用索引做排序。ORDER BY子句和查询的限制是一样的，都要满足最左前缀的要求（有一种情况例外，就是最左的列被指定为常数，下面是一个简单的示例），其它情况下都需要执行排序操作，而无法利用索引排序。

// 最左列为常数，索引：(date,staff\_id,customer\_id)

select staff\_id,customer\_id from demo where date = '2015-06-01' order by staff\_id,customer\_id

7、冗余和重复索引

冗余索引是指在相同的列上按照相同的顺序创建的相同类型的索引，应当尽量避免这种索引，发现后立即删除。比如有一个索引(A,B)，再创建索引(A)就是冗余索引。冗余索引经常发生在为表添加新索引时，比如有人新建了索引(A,B)，但这个索引不是扩展已有的索引(A)。

大多数情况下都应该尽量扩展已有的索引而不是创建新索引。但有极少情况下出现性能方面的考虑需要冗余索引，比如扩展已有索引而导致其变得过大，从而影响到其他使用该索引的查询。

8、删除长期未使用的索引

定期删除一些长时间未使用过的索引是一个非常好的习惯。

关于索引这个话题打算就此打住，最后要说一句，索引并不总是最好的工具，只有当索引帮助提高查询速度带来的好处大于其带来的额外工作时，索引才是有效的。对于非常小的表，简单的全表扫描更高效。对于中到大型的表，索引就非常有效。对于超大型的表，建立和维护索引的代价随之增长，这时候其他技术也许更有效，比如分区表。最后的最后，explain后再提测是一种美德。

特定类型查询优化

1.优化COUNT()查询

COUNT()可能是被大家误解最多的函数了，它有两种不同的作用，其一是统计某个列值的数量，其二是统计行数。统计列值时，要求列值是非空的，它不会统计NULL。如果确认括号中的表达式不可能为空时，实际上就是在统计行数。最简单的就是当使用COUNT(\*)时，并不是我们所想象的那样扩展成所有的列，实际上，它会忽略所有的列而直接统计所有的行数。

我们最常见的误解也就在这儿，在括号内指定了一列却希望统计结果是行数，而且还常常误以为前者的性能会更好。但实际并非这样，如果要统计行数，直接使用COUNT(\*)，意义清晰，且性能更好。

有时候某些业务场景并不需要完全精确的COUNT值，可以用近似值来代替，EXPLAIN出来的行数就是一个不错的近似值，而且执行EXPLAIN并不需要真正地去执行查询，所以成本非常低。通常来说，执行COUNT()都需要扫描大量的行才能获取到精确的数据，因此很难优化，MySQL层面还能做得也就只有覆盖索引了。如果不还能解决问题，只有从架构层面解决了，比如添加汇总表，或者使用redis这样的外部缓存系统。

2.优化关联查询

在大数据场景下，表与表之间通过一个冗余字段来关联，要比直接使用JOIN有更好的性能。如果确实需要使用关联查询的情况下，需要特别注意的是：

1. 确保ON和USING字句中的列上有索引。在创建索引的时候就要考虑到关联的顺序。当表A和表B用列c关联的时候，如果优化器关联的顺序是A、B，那么就不需要在A表的对应列上创建索引。没有用到的索引会带来额外的负担，一般来说，除非有其他理由，只需要在关联顺序中的第二张表的相应列上创建索引（具体原因下文分析）。
2. 确保任何的GROUP BY和ORDER BY中的表达式只涉及到一个表中的列，这样MySQL才有可能使用索引来优化。

要理解优化关联查询的第一个技巧，就需要理解MySQL是如何执行关联查询的。当前MySQL关联执行的策略非常简单，它对任何的关联都执行嵌套循环关联操作，即先在一个表中循环取出单条数据，然后在嵌套循环到下一个表中寻找匹配的行，依次下去，直到找到所有表中匹配的行为为止。然后根据各个表匹配的行，返回查询中需要的各个列。

太抽象了？以上面的示例来说明，比如有这样的一个查询：

SELECT A.xx,B.yy

FROM A INNER JOIN B USING(c)

WHERE A.xx IN (5,6)

假设MySQL按照查询中的关联顺序A、B来进行关联操作，那么可以用下面的伪代码表示MySQL如何完成这个查询：

outer\_iterator = SELECT A.xx,A.c FROM A WHERE A.xx IN (5,6);

outer\_row = outer\_iterator.next;

while(outer\_row) {

inner\_iterator = SELECT B.yy FROM B WHERE B.c = outer\_row.c;

inner\_row = inner\_iterator.next;

while(inner\_row) {

output[inner\_row.yy,outer\_row.xx];

inner\_row = inner\_iterator.next;

}

outer\_row = outer\_iterator.next;

}

可以看到，最外层的查询是根据A.xx列来查询的，A.c上如果有索引的话，整个关联查询也不会使用。再看内层的查询，很明显B.c上如果有索引的话，能够加速查询，因此只需要在关联顺序中的第二张表的相应列上创建索引即可。

3.优化LIMIT分页

当需要分页操作时，通常会使用LIMIT加上偏移量的办法实现，同时加上合适的ORDER BY字句。如果有对应的索引，通常效率会不错，否则，MySQL需要做大量的文件排序操作。

一个常见的问题是当偏移量非常大的时候，比如：LIMIT 10000 20这样的查询，MySQL需要查询10020条记录然后只返回20条记录，前面的10000条都将被抛弃，这样的代价非常高。

优化这种查询一个最简单的办法就是尽可能的使用覆盖索引扫描，而不是查询所有的列。然后根据需要做一次关联查询再返回所有的列。对于偏移量很大时，这样做的效率会提升非常大。考虑下面的查询：

SELECT film\_id,description FROM film ORDER BY title LIMIT 50,5;

如果这张表非常大，那么这个查询最好改成下面的样子：

SELECT film.film\_id,film.description

FROM film INNER JOIN (

SELECT film\_id FROM film ORDER BY title LIMIT 50,5

) AS tmp USING(film\_id);

这里的延迟关联将大大提升查询效率，让MySQL扫描尽可能少的页面，获取需要访问的记录后在根据关联列回原表查询所需要的列。

有时候如果可以使用书签记录上次取数据的位置，那么下次就可以直接从该书签记录的位置开始扫描，这样就可以避免使用OFFSET，比如下面的查询：

SELECT id FROM t LIMIT 10000, 10;

改为：

SELECT id FROM t WHERE id > 10000 LIMIT 10;

其它优化的办法还包括使用预先计算的汇总表，或者关联到一个冗余表，冗余表中只包含主键列和需要做排序的列。

4.优化UNION

MySQL处理UNION的策略是先创建临时表，然后再把各个查询结果插入到临时表中，最后再来做查询。因此很多优化策略在UNION查询中都没有办法很好的时候。经常需要手动将WHERE、LIMIT、ORDER BY等字句“下推”到各个子查询中，以便优化器可以充分利用这些条件先优化。

除非确实需要服务器去重，否则就一定要使用UNION ALL，如果没有ALL关键字，MySQL会给临时表加上DISTINCT选项，这会导致整个临时表的数据做唯一性检查，这样做的代价非常高。当然即使使用ALL关键字，MySQL总是将结果放入临时表，然后再读出，再返回给客户端。虽然很多时候没有这个必要，比如有时候可以直接把每个子查询的结果返回给客户端。

结语

理解查询是如何执行以及时间都消耗在哪些地方，再加上一些优化过程的知识，可以帮助大家更好的理解MySQL，理解常见优化技巧背后的原理。希望本文中的原理、示例能够帮助大家更好的将理论和实践联系起来，更多的将理论知识运用到实践中。

其他也没啥说的了，给大家留两个思考题吧，可以在脑袋里想想答案，这也是大家经常挂在嘴边的，但很少有人会思考为什么？

1. 有非常多的程序员在分享时都会抛出这样一个观点：尽可能不要使用存储过程，存储过程非常不容易维护，也会增加使用成本，应该把业务逻辑放到客户端。既然客户端都能干这些事，那为什么还要存储过程？
2. JOIN本身也挺方便的，直接查询就好了，为什么还需要视图呢？

参考资料

姜承尧 著；MySQL技术内幕-InnoDB存储引擎；机械工业出版社，2013

Baron Scbwartz 等著；宁海元 周振兴等译；高性能MySQL（第三版）; 电子工业出版社， 2013

由 B-/B+树看 MySQL索引结构

<https://segmentfault.com/a/1190000004690721>

# 优化2

          硬件配置优化

CPU选择：多核的CPU，主频高的CPU

内存：更大的内存

磁盘选择：更快的转速、RAID、阵列卡，

网络环境选择：尽量部署在局域网、SCI、光缆、千兆网、双网线提供冗余、0.0.0.0多端口绑定监听

II         操作系统级优化

使用64位的操作系统，更好的使用大内存。

设置noatime,nodiratime

[zhangxy@dowload\_server1 ~]$ cat /etc/fstab

LABEL=/         /          ext3    defaults,noatime,nodiratime       1 1

/dev/sda5      /data       xfs     defaults,noatime,nodiratime       1 2

优化内核参数

net.ipv4.tcp\_keepalive\_time=7200

net.ipv4.tcp\_max\_syn\_backlog=1024

net.ipv4.tcp\_syncookies=1

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_tw\_recycle = 1

net.ipv4.neigh.default.gc\_thresh3 = 2048

net.ipv4.neigh.default.gc\_thresh2 = 1024

net.ipv4.neigh.default.gc\_thresh1 = 256

net.ipv4.conf.default.rp\_filter = 1

net.ipv4.conf.default.forwarding = 1

net.ipv4.conf.default.proxy\_arp = 0

net.ipv4.tcp\_syncookies = 1

net.core.netdev\_max\_backlog = 2048

net.core.dev\_weight = 64

net.ipv4.tcp\_rmem = 4096 87380 16777216

net.ipv4.tcp\_wmem = 4096 65536 16777216

net.ipv4.tcp\_rfc1337 = 1

net.ipv4.tcp\_sack = 0

net.ipv4.tcp\_fin\_timeout = 20

net.ipv4.tcp\_keepalive\_probes = 5

net.ipv4.tcp\_max\_orphans = 32768

net.core.optmem\_max = 20480

net.core.rmem\_default = 16777216

net.core.rmem\_max = 16777216

net.core.wmem\_default = 16777216

net.core.wmem\_max = 16777216

net.core.somaxconn = 500

net.ipv4.tcp\_orphan\_retries = 1

net.ipv4.tcp\_max\_tw\_buckets = 18000

net.ipv4.ip\_forward = 0

net.ipv4.conf.default.proxy\_arp = 0

net.ipv4.conf.all.rp\_filter = 1

kernel.sysrq = 1

net.ipv4.conf.default.send\_redirects = 1

net.ipv4.conf.all.send\_redirects = 0

net.ipv4.ip\_local\_port\_range = 5000    65000

kernel.shmmax = 167108864

vm.swappiness=0

加大文件描述符限制

Vim /etc/security/limits.conf

加上

\*        soft    nofile  65535

\*        hard    nofile  65535

文件系统选择 xfs

/dev/sda5      /data       xfs     defaults,noatime,nodiratime        1 2

III      Mysql设计优化  
III.1存储引擎的选择

Myisam：数据库并发不大，读多写少,而且都能很好的用到索引，sql语句比较简单的应用，TB数据仓库

Innodb：并发访问大，写操作比较多，有外键、事务等需求的应用，系统内存较大。

III.2命名规则

多数开发语言命名规则：比如MyAdress

多数开源思想命名规则：my\_address

避免随便命名

III.3字段类型选择

字段类型的选择的一般原则：

根据需求选择合适的字段类型，在满足需求的情况下字段类型尽可能小。

只分配满足需求的最小字符数，不要太慷慨。

原因：更小的字段类型更小的字符数占用更少的内存，占用更少的磁盘空间，占用更少的磁盘IO，以及占用更少的带宽。

III.3.1  整型：

见如下图:

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节 | 最小值 | 最大值 |
|  |  | (带符号的/无符号的) | (带符号的/无符号的) |
| TINYINT | 1 | -128 | 127 |
|  |  | 0 | 255 |
| SMALLINT | 2 | -32768 | 32767 |
|  |  | 0 | 65535 |
| MEDIUMINT | 3 | -8388608 | 8388607 |
|  |  | 0 | 16777215 |
| INT | 4 | -2147483648 | 2147483647 |
|  |  | 0 | 4294967295 |
| BIGINT | 8 | -9223372036854775808 | 9223372036854775807 |
|  |  | 0 | 18446744073709551615 |

根据满足需求的最小整数为选择原则，能用INT的就不要用BIGINT。

用无符号INT存储IP，而非CHAR(15)。

III.3.2  浮点型：

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 字节 | 精度类型 | 使用场景 |
| FLOAT(M,D) | 4 | 单精度 | 精度要求不高，数值比较小 |
| DOUBLE(M,D)（REAL） | 8 | 双精度 | 精度要求不高，数值比较大 |
| DECIMAL(M,D)(NUMERIC） | M+2 | 自定义精度 | 精度要求很高的场景 |

III.3.3  时间类型

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 取值范围 | 存储空间 | 零值表示法 |
| DATE | 1000-01-01~9999-12-31 | 3字节 | 0000-00-00 |
| TIME | -838:59:59~838:59:59 | 3字节 | 00:00:00 |
| DATETIME | 1000-01-01  00:00:00~9999-12-31  23:59:59 | 8字节 | 0000-00-00 00:00:00 |
| TIMESTAMP | 19700101000000~2037年的某个时刻 | 4字节 | 00000000000000 |
| YEAR | YEAR(4):1901~2155 YEAR(2)：1970~2069 | 1字节 | 0000 |

III.3.4  字符类型

|  |  |  |
| --- | --- | --- |
| 类型 | 最大长度 | 占用存储空间 |
| CHAR[(M)] | M字节 | M字节 |
| VARCHAR[(M)] | M字节 | M+1字节 |
| TINYBLOD，TINYTEXT | 2^8-1字节 | L+1字节 |
| BLOB，TEXT | 2^16-1字节 | L+2 |
| MEDIUMBLOB，MEDIUMTEXT | 2^24-1字节 | L+3 |
| LONGBLOB，LONGTEXT | 2^32-1字节 | L+4 |
| ENUM('value1','value2',...) | 65535个成员 | 1或2字节 |
| SET('value1','value2',...) | 64个成员 | 1,2,3,4或8字节 |

注：L表示可变长度的意思

对于varchar和char的选择要根据引擎和具体情况的不同来选择，主要依据如下原则:

1.   如果列数据项的大小一致或者相差不大，则使用char。

2.   如果列数据项的大小差异相当大，则使用varchar。

3.   对于MyISAM表，尽量使用Char，对于那些经常需要修改而容易形成碎片的myisam和isam数据表就更是如此，它的缺点就是占用磁盘空间。

4.   对于InnoDB表，因为它的数据行内部存储格式对固定长度的数据行和可变长度的数据行不加区分（所有数据行共用一个表头部分，这个标头部分存放着指向各有关数据列的指针），所以使用char类型不见得会比使用varchar类型好。事实上，因为char类型通常要比varchar类型占用更多的空 间，所以从减少空间占用量和减少磁盘i/o的角度，使用varchar类型反而更有利。

5.         表中只要存在一个varchar类型的字段，那么所有的char字段都会自动变成varchar类型，因此建议定长和变长的数据分开。

III.4编码选择

单字节 latin1

多字节 utf8(汉字占3个字节，英文字母占用一个字节)

如果含有中文字符的话最好都统一采用utf8类型，避免乱码的情况发生。

III.5主键选择原则

注：这里说的主键设计主要是针对INNODB引擎

1.   能唯一的表示行。

2.   显式的定义一个数值类型自增字段的主键，这个字段可以仅用于做主键，不做其他用途。

3.   MySQL主键应该是单列的，以便提高连接和筛选操作的效率。

4.   主键字段类型尽可能小，能用SMALLINT就不用INT，能用INT就不用BIGINT。

5.   尽量保证不对主键字段进行更新修改，防止主键字段发生变化，引发数据存储碎片，降低IO性能。

6.   MySQL主键不应包含动态变化的数据，如时间戳、创建时间列、修改时间列等。

7.   MySQL主键应当有计算机自动生成。

8.   主键字段放在数据表的第一顺序。

推荐采用数值类型做主键并采用auto\_increment属性让其自动增长。

III.6其他需要注意的地方

NULL OR NOT NULL

尽可能设置每个字段为NOT NULL，除非有特殊的需求，原因如下：

1.   使用含有NULL列做索引的话会占用更多的磁盘空间，因为索引NULL列需要而外的空间来保存。

2.   进行比较的时候，程序会更复杂。

3.   含有NULL的列比较特殊，SQL难优化，如果是一个组合索引，那么这个NULL 类型的字段会极大影响整个索引的效率。

索引

索引的缺点：极大地加速了查询，减少扫描和锁定的数据行数。

索引的缺点：占用磁盘空间，减慢了数据更新速度，增加了磁盘IO。

添加索引有如下原则：

1.   选择唯一性索引。

2.   为经常需要排序、分组和联合操作的字段建立索引。

3.   为常作为查询条件的字段建立索引。

4.   限制索引的数据，索引不是越多越好。

5.   尽量使用数据量少的索引，对于大字段可以考虑前缀索引。

6.   删除不再使用或者很少使用的索引。

7.   结合核心SQL优先考虑覆盖索引。

8.   忌用字符串做主键。

反范式设计

适当的使用冗余的反范式设计，以空间换时间有的时候会很高效。

IV     Mysql软件优化

开启mysql复制，实现读写分离、负载均衡，将读的负载分摊到多个从服务器上，提高服务器的处理能力。

使用推荐的GA版本，提升性能

利用分区新功能进行大数据的数据拆分

V        Mysql配置优化

注意：全局参数一经设置，随服务器启动预占用资源。

key\_buffer\_size参数

mysql索引缓冲，如果是采用myisam的话要重点设置这个参数，根据（key\_reads/key\_read\_requests）判断

innodb\_buffer\_pool\_size参数

INNODB 数据、索引、日志缓冲最重要的引擎参数，根据（hit riatos和FILE I/O）判断

wait\_time\_out参数

线程连接的超时时间，尽量不要设置很大，推荐10s

max\_connections参数

服务器允许的最大连接数，尽量不要设置太大，因为设置太大的话容易导致内存溢出，需要通过如下公式来确定：

SET @k\_bytes = 1024;

SET @m\_bytes = @k\_bytes \* 1024;

SET @g\_bytes = @m\_bytes \* 1024;

SELECT

      (

 @@key\_buffer\_size + @@query\_cache\_size + @@tmp\_table\_size+

      @@innodb\_buffer\_pool\_size + @@innodb\_additional\_mem\_pool\_size+

      @@innodb\_log\_buffer\_size+

      @@max\_connections \*

 ( @@read\_buffer\_size + @@read\_rnd\_buffer\_size + @@sort\_buffer\_size+

         @@join\_buffer\_size + @@binlog\_cache\_size + @@thread\_stack

        ) )

/ @g\_bytes AS MAX\_MEMORY\_USED\_GB;

thread\_concurrency参数

线程并发利用数量，(cpu+disk)\*2,根据(os中显示的请求队列和tickets)判断

sort\_buffer\_size参数

获得更快的--ORDER BY,GROUP BY,SELECT DISTINCT,UNION DISTINCT

read\_rnd\_buffer\_size参数

当根据键进行分类操作时获得更快的--ORDER BY

join\_buffer\_size参数

join连接使用全表扫描连接的缓冲大小，根据select\_full\_join判断

read\_buffer\_size参数

全表扫描时为查询预留的缓冲大小，根据select\_scan判断

tmp\_table\_size参数

临时内存表的设置，如果超过设置就会转化成磁盘表，根据参数(created\_tmp\_disk\_tables)判断

innodb\_log\_file\_size参数(默认5M)

记录INNODB引擎的redo log文件，设置较大的值意味着较长的恢复时间。

innodb\_flush\_method参数(默认fdatasync)

Linux系统可以使用O\_DIRECT处理数据文件，避免OS级别的cache，O\_DIRECT模式提高数据文件和日志文件的IO提交性能

innodb\_flush\_log\_at\_trx\_commit(默认1)

1.   0表示每秒进行一次log写入cache，并flush log到磁盘。

2.   1表示在每次事务提交后执行log写入cache，并flush log到磁盘。

3.   2表示在每次事务提交后，执行log数据写入到cache，每秒执行一次flush log到磁盘。

VI     Mysql语句级优化

1.   性能查的读语句，在innodb中统计行数,建议另外弄一张统计表，采用myisam，定期做统计.一般的对统计的数据不会要求太精准的情况下适用。

2.   尽量不要在数据库中做运算。

3.   避免负向查询和%前缀模糊查询。

4.   不在索引列做运算或者使用函数。

5.   不要在生产环境程序中使用select \* from 的形式查询数据。只查询需要使用的列。

6.   查询尽可能使用limit减少返回的行数，减少数据传输时间和带宽浪费。

7.   where子句尽可能对查询列使用函数，因为对查询列使用函数用不到索引。

8.   避免隐式类型转换，例如字符型一定要用’’，数字型一定不要使用’’。

9.   所有的SQL关键词用大写，养成良好的习惯，避免SQL语句重复编译造成系统资源的浪费。

10. 联表查询的时候，记得把小结果集放在前面，遵循小结果集驱动大结果集的原则。

11. 开启慢查询，定期用explain优化慢查询中的SQL语句。

# FAQ

### 查看mysql版本的六种方法

:selected

:checked

1. [root@localhost ~]# mysql -V

2. [root@localhost ~]# mysql --help | grep Distrib

在mysql下有四种

3. 登陆时会显示

4. mysql> status;

5. mysql> select version();

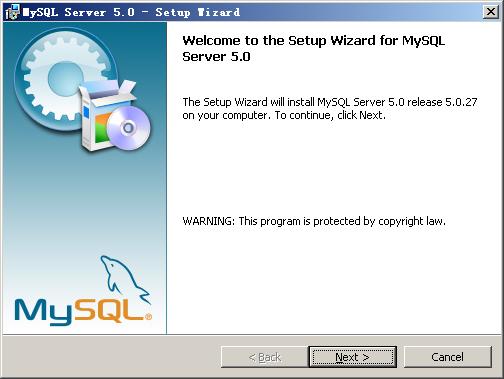
6. mysql> show variables like "%version%";

windows下cmd操作的全部贴图（cmd没有grep命令，直接mysql --help）

### MySQL安装图解

#### MYSQL的安装

１、打开下载的mysql安装文件mysql-5.0.27-win32.zip，双击解压缩，运行“setup.exe”。

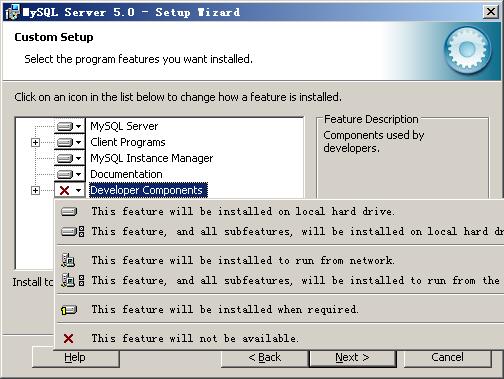


２、选择安装类型，有“Typical（默认）”、“Complete（完全）”、“Custom（用户自定义）”三个选项，选择“Custom”，按“next”键继续。

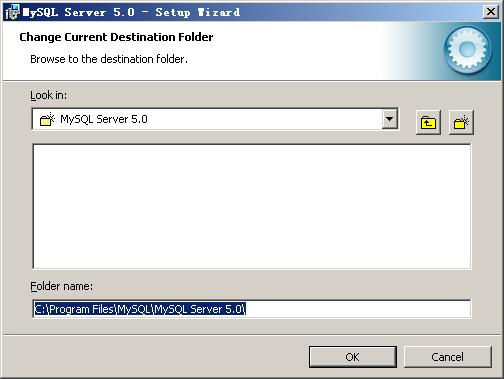


３、在“Developer Components（开发者部分）”上左键单击，选择“This feature,

and all subfeatures, will be installed on local hard drive.”，即“此部分，及下属子部分内容，全部安装在本地硬盘上”。在上面的“MySQL Server（mysql服务器）”、“Client Programs（mysql客户端程序）”、“Documentation（文档）”也如此操作，以保证安装所有文件。点选“Change...”，手动指定安装目录。



４、填上安装目录，我的是“F:\Server\MySQL\MySQL Server 5.0”，也建议不要放在与操作系统同一分区，这样可以防止系统备份还原的时候，数据被清空。按“OK”继续。



确认一下先前的设置，如果有误，按“Back”返回重做。按“Install”开始安装。



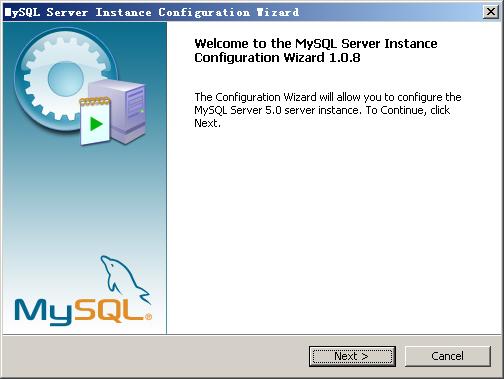
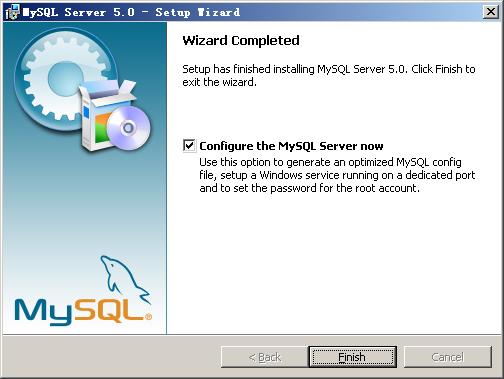
５、正在安装中，请稍候，直到出现下面的界面



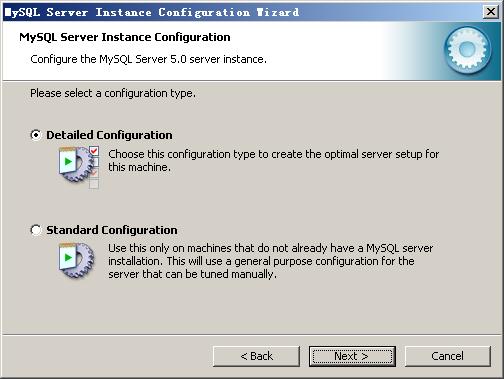
这里是询问你是否要注册一个mysql.com的账号，或是使用已有的账号登陆mysql.com，一般不需要了，点选“Skip Sign-Up”，按“Next”略过此步骤。继续则完成MYSQL的安装。

#### 二、MYSQL的配置

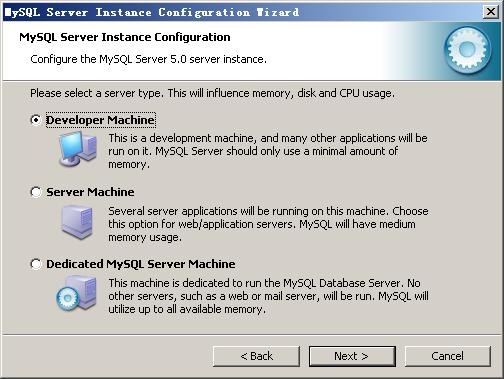
１、安装完成了，出现如下界面将进入mysql配置向导。



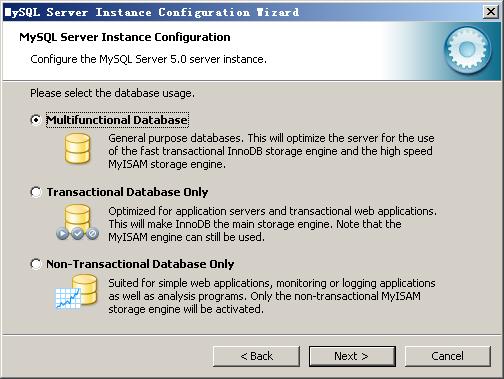
２、选择配置方式，“Detailed Configuration（手动精确配置）”、“Standard Configuration（标准配置）”，我们选择“Detailed Configuration”，方便熟悉配置过程。



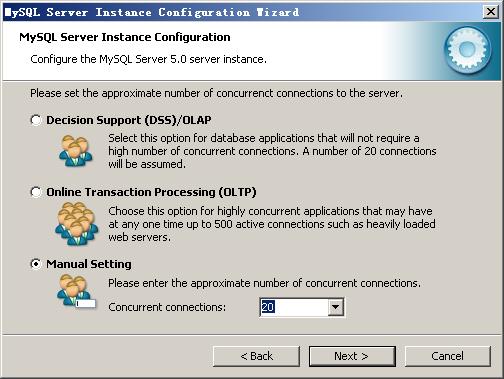
３、选择服务器类型，“Developer Machine（开发测试类，mysql占用很少资源）”、“Server Machine（服务器类型，mysql占用较多资源）”、“Dedicated MySQL Server Machine（专门的数据库服务器，mysql占用所有可用资源）”



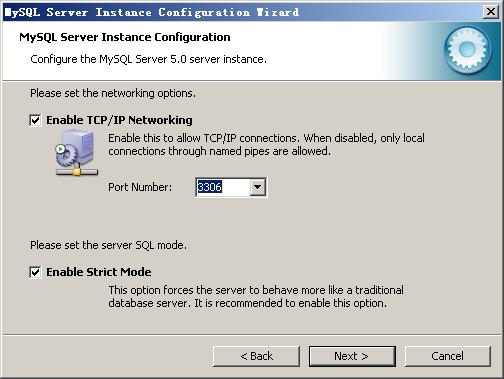
４、选择mysql数据库的大致用途，“Multifunctional Database（通用多功能型，好）”、“Transactional Database Only（服务器类型，专注于事务处理，一般）”、“Non-Transactional Database Only（非事务处理型，较简单，主要做一些监控、记数用，对MyISAM数据类型的支持仅限于non-transactional），按“Next”继续。



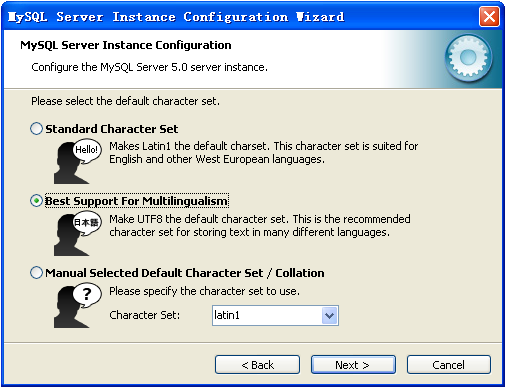
５、选择网站并发连接数，同时连接的数目，“Decision Support(DSS)/OLAP（20个左右）”、“Online Transaction Processing(OLTP)（500个左右）”、“Manual Setting（手动设置，自己输一个数）”。



６、是否启用TCP/IP连接，设定端口，如果不启用，就只能在自己的机器上访问mysql数据库了，在这个页面上，您还可以选择“启用标准模式”（Enable Strict Mode），这样MySQL就不会允许细小的语法错误。如果是新手，建议您取消标准模式以减少麻烦。但熟悉MySQL以后，尽量使用标准模式，因为它可以降低有害数据进入数据库的可能性。按“Next”继续



７、就是对mysql默认数据库语言编码进行设置（重要），一般选UTF-8，按 “Next”继续。



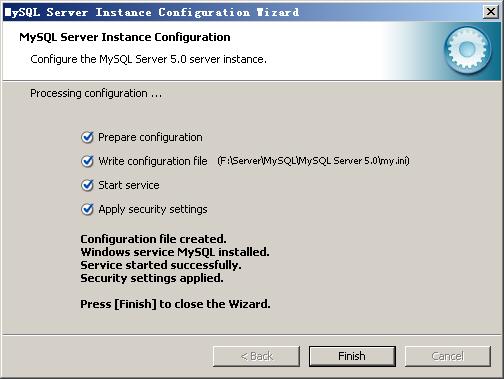
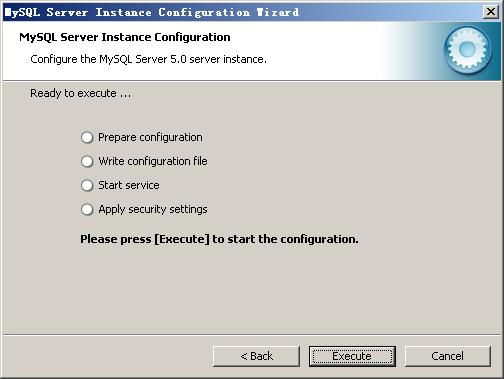
８、选择是否将mysql安装为windows服务，还可以指定Service Name（服务标识名称），是否将mysql的bin目录加入到Windows PATH（加入后，就可以直接使用bin下的文件，而不用指出目录名，比如连接，“mysql.exe -uusername -ppassword;”就可以了，不用指出mysql.exe的完整地址，很方便），我这里全部打上了勾，Service Name不变。按“Next”继续。



９、询问是否要修改默认root用户（超级管理）的密码。“Enable root access from remote machines（是否允许root用户在其它的机器上登陆，如果要安全，就不要勾上，如果要方便，就勾上它）”。最后“Create An Anonymous Account（新建一个匿名用户，匿名用户可以连接数据库，不能操作数据，包括查询）”，一般就不用勾了，设置完毕，按“Next”继续。



１０、确认设置无误，按“Execute”使设置生效，即完成MYSQL的安装和配置。



**注意：设置完毕，按“Finish”后有一个比较常见的错误，就是不能“Start service”，一般出现在以前有安装mysql的服务器上，解决的办法，先保证以前安装的mysql服务器彻底卸载掉了；不行的话，检查是否按上面一步所说，之前的密码是否有修改，照上面的操作；如果依然不行，将mysql安装目录下的data文件夹备份，然后删除，在安装完成后，将安装生成的 data文件夹删除，备份的data文件夹移回来，再重启mysql服务就可以了，这种情况下，可能需要将数据库检查一下，然后修复一次，防止数据出错。**

### linux下查看Mysql默认编码、修改默认编码

https://blog.csdn.net/qy20115549/article/details/53931330

#### 登陆mysql

首先，使用命令登陆mysql  
如下，截图。

#### 查看编码

接着，使用命令查看数据库的编码方式。

|  |
| --- |
| show variables like 'character%'; |

修改my.cnf文件

下面是退出数据库，关闭数据库，修改配置文件。

关闭数据库命令如下：

service mysqld stop

• 1

下面修改配置文件：

如下图。

vi /etc/my.cnf

• 1

加入如下代码：

[client]

default\_character\_set=utf8

[mysqld]

collation\_server = utf8\_general\_ci

character\_ 5

重启mysql

service mysqld restart

### 重置root密码

方法一:

在my.ini的[mysqld]字段加入：

skip-grant-tables

重启mysql服务，这时的mysql不需要密码即可登录数据库

然后进入mysql

mysql>use mysql;

mysql>更新 user set password=password('新密码') WHERE User='root';

mysql>flush privileges;

运行之后最后去掉my.ini中的skip-grant-tables，重启mysqld即可。

方法二:

不使用修改my.ini重启服务的方法，通过非服务方式加skip-grant-tables运行mysql来修改mysql密码

停止mysql服务

打开命令行窗口，在bin目录下使用mysqld-nt.exe启动，即在命令行窗口执行: mysqld-nt --skip-grant-tables

然后另外打开一个命令行窗口，登录mysql，此时无需输入mysql密码即可进入。

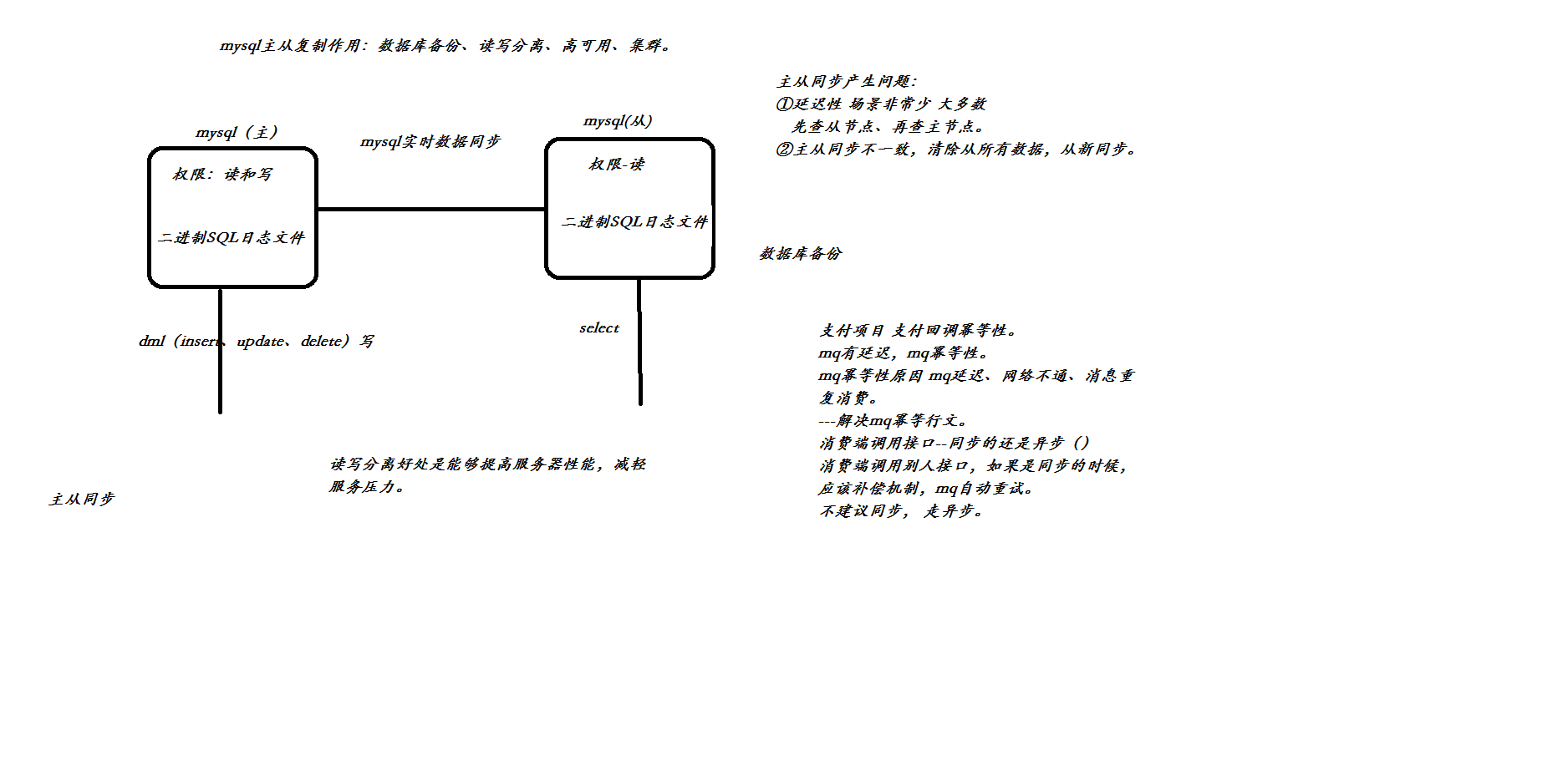
按以上方法修改好密码后,关闭命令行运行mysql的那个窗口，此时即关闭了mysql，如果发现mysql仍在运行

的话可以结束掉对应进程来关闭。

启动mysql服务

# 主从复制





## 海量数据的存储问题

如今随着互联网的发展，数据的量级也是撑指数的增长，从GB到TB到PB。对数据的各种操作也是愈加的困难，传统的关系性数据库已经无法满足快速查询与插入数据的需求。这个时候NoSQL的出现暂时解决了这一危机。它通过降低数据的安全性，减少对事务的支持，减少对复杂查询的支持，来获取性能上的提升。

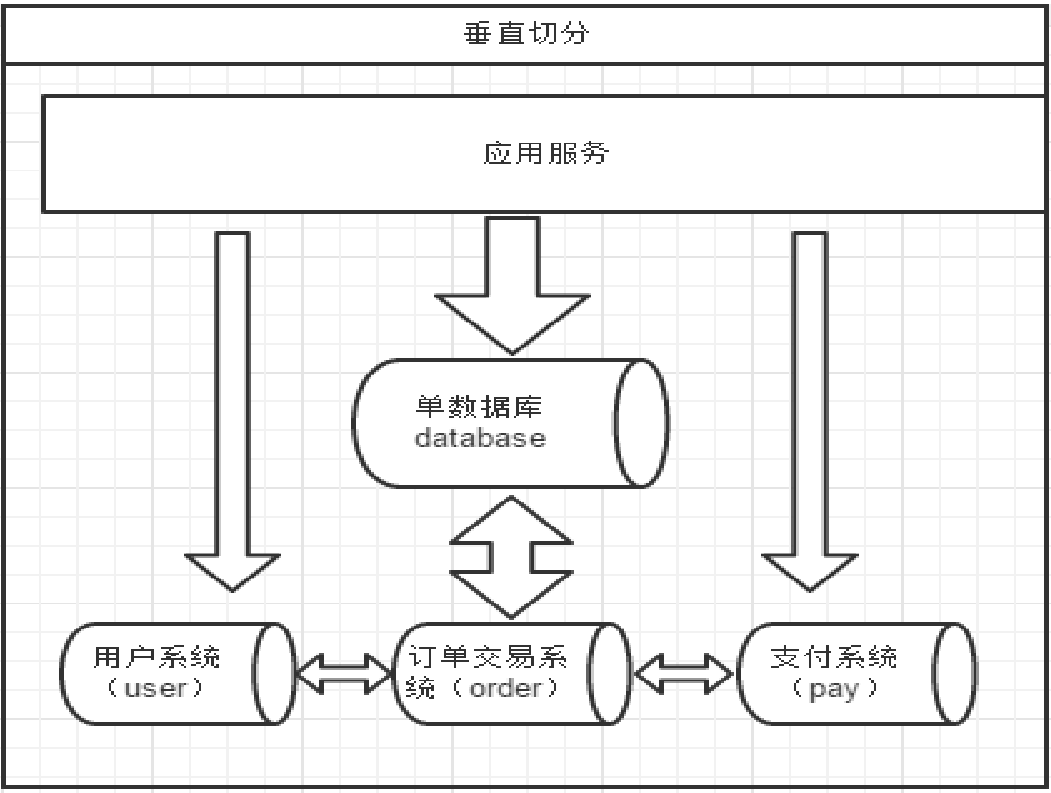
但是，在有些场合NoSQL一些折衷是无法满足使用场景的，就比如有些使用场景是绝对要有事务与安全指标的。这个时候NoSQL肯定是无法满足的，所以还是需要使用关系性数据库。如果使用关系型数据库解决海量存储的问题呢？此时就需要做数据库集群，为了提高查询性能将一个数据库的数据分散到不同的数据库中存储。

### 什么是数据库分片

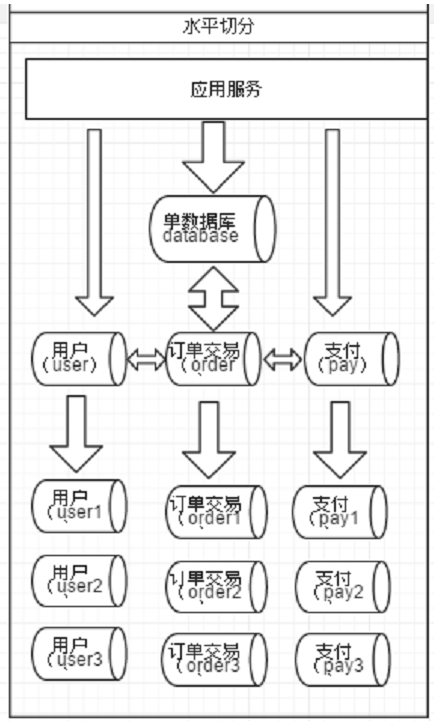
简单来说，就是指通过某种特定的条件，将我们存放在同一个数据库中的数据分散存放到多个数据库（主机）上面，以达到分散单台设备负载的效果。

数据的切分（Sharding）根据其切分规则的类型，可以分为两种切分模式。

1. 一种是按照不同的表（或者Schema）来切分到不同的数据库（主机）之上，这种切可以称之为数据的垂直（纵向）切分



1. 另外一种则是根据表中的数据的逻辑关系，将同一个表中的数据按照某种条件拆分到多台数据库（主机）上面，这种切分称之为数据的水平（横向）切分。

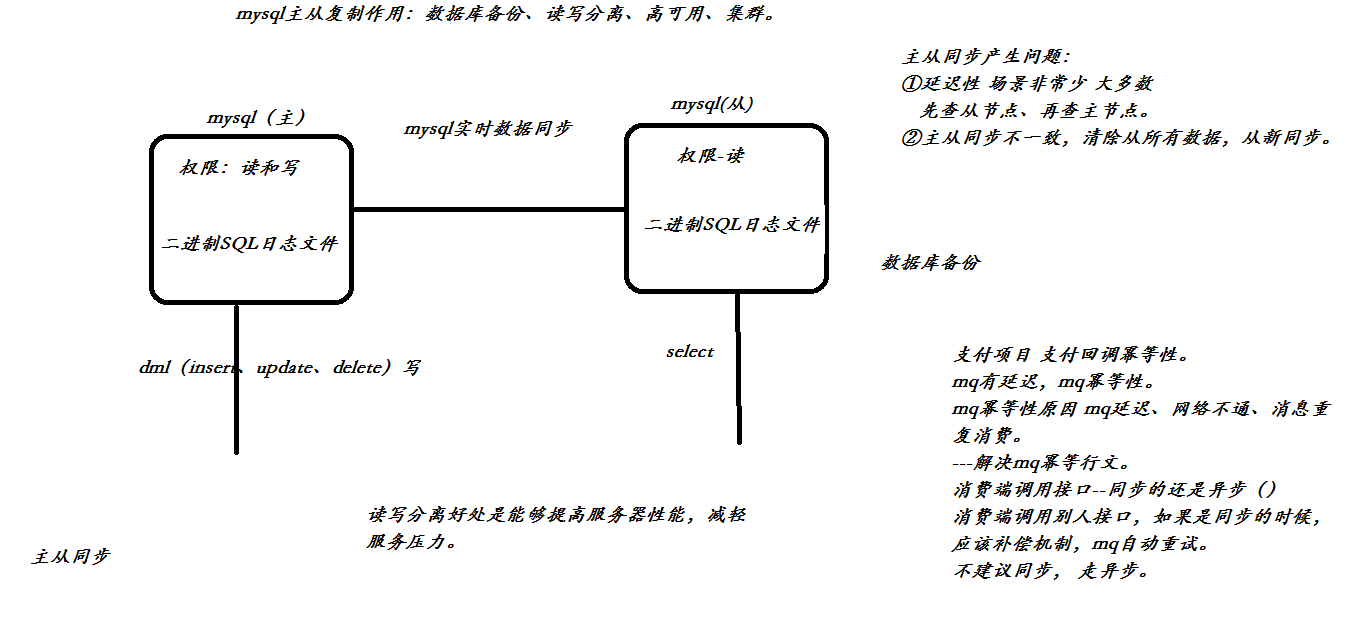


### 如何实现数据库分片

当数据库分片后，数据由一个数据库分散到多个数据库中。此时系统要查询时需要切换不同的数据库进行查询，那么系统如何知道要查询的数据在哪个数据库中？当添加一条记录时要向哪个数据库中插入呢？这些问题处理起来都是非常的麻烦。

这种情况下可以使用一个数据库中间件mycat来解决相关的问题。接下来了解一下什么是mycat。

## 概念



影响MySQL-A数据库的操作，在数据库执行后，都会写入本地的日志系统A中。

假设，实时的将变化了的日志系统中的数据库事件操作，在MYSQL-A的3306端口，通过网络发给MYSQL-B。

MYSQL-B收到后，写入本地日志系统B，然后一条条的将数据库事件在数据库中完成。

那么，MYSQL-A的变化，MYSQL-B也会变化，这样就是所谓的MYSQL的复制，即MYSQLreplication。

在上面的模型中，MYSQL-A就是主服务器，即master，MYSQL-B就是从服务器，即slave。

日志系统A，其实它是MYSQL的日志类型中的二进制日志，也就是专门用来保存修改数据库表的所有动作，即binlog。【注意MYSQL会在执行语句之后，释放锁之前，写入二进制日志，确保事务安全】

日志系统B，并不是二进制日志，由于它是从MYSQL-A的二进制日志复制过来的，并不是自己的数据库变化产生的，有点接力的感觉，称为中继日志，即relaylog。

可以发现，通过上面的机制，可以保证MYSQL-A和MYSQL-B的数据库数据一致，但是时间上肯定有延迟，即MYSQL-B的数据是滞后的。

【即便不考虑什么网络的因素，MYSQL-A的数据库操作是可以并发的执行的，但是MYSQL-B只能从relaylog中读一条，执行下。因此MYSQL-A的写操作很频繁，MYSQL-B很可能跟不上。】

## 解决问题

数据如何不被丢失

备份

读写分离

数据库负载均衡

高可用

## 环境搭建

1. 准备环境

两台windows操作系统ip分别为:172.27.185.1(主)、172.27.185.2(从)

1. 连接到主服务(172.27.185.1)服务器上，给从节点分配账号权限。

GRANTREPLICATIONSLAVEON\*.\*TO'root'@'172.27.185.2'IDENTIFIEDBY'root';

1. 在主服务my.ini文件新增

|  |
| --- |
| server-id=200  log-bin=mysql-bin  relay-log=relay-bin  relay-log-index=relay-bin-index |

重启mysql服务

1. 在从服务my.ini文件新增

|  |
| --- |
| server-id=210  replicate-do-db=itmayiedu#需要同步数据库 |

重启mysql服务

1. 从服务同步主数据库

|  |
| --- |
| stopslave;  change  mastertomaster\_host='172.27.185.1',master\_user='root',master\_password='root';  startslave;  showslavestatus; |

## 注意事项

1. 一定要在同一个局域网中
2. 使用360WiFi创建局域网
3. 最好把防火墙全部关闭掉

## 什么是读写分离

在数据库集群架构中，让主库负责处理事务性查询，而从库只负责处理select查询，让两者分工明确达到提高数据库整体读写性能。当然，主数据库另外一个功能就是负责将事务性查询导致的数据变更同步到从库中，也就是写操作。

## 读写分离的好处

1）分摊服务器压力，提高机器的系统处理效率

读写分离适用于读远比写的场景，如果有一台服务器，当select很多时，update和delete会被这些select访问中的数据堵塞，等待select结束，并发性能并不高，而主从只负责各自的写和读，极大程度的缓解X锁和S锁争用；

假如我们有1主3从，不考虑上述1中提到的从库单方面设置，假设现在1分钟内有10条写入，150条读取。那么，1主3从相当于共计40条写入，而读取总数没变，因此平均下来每台服务器承担了10条写入和50条读取（主库不承担读取操作）。因此，虽然写入没变，但是读取大大分摊了，提高了系统性能。另外，当读取被分摊后，又间接提高了写入的性能。所以，总体性能提高了，说白了就是拿机器和带宽换性能；

2）增加冗余，提高服务可用性，当一台数据库服务器宕机后可以调整另外一台从库以最快速度恢复服务

## 主从复制原理

依赖于二进制日志，binary-log.

二进制日志中记录引起数据库发生改变的语句

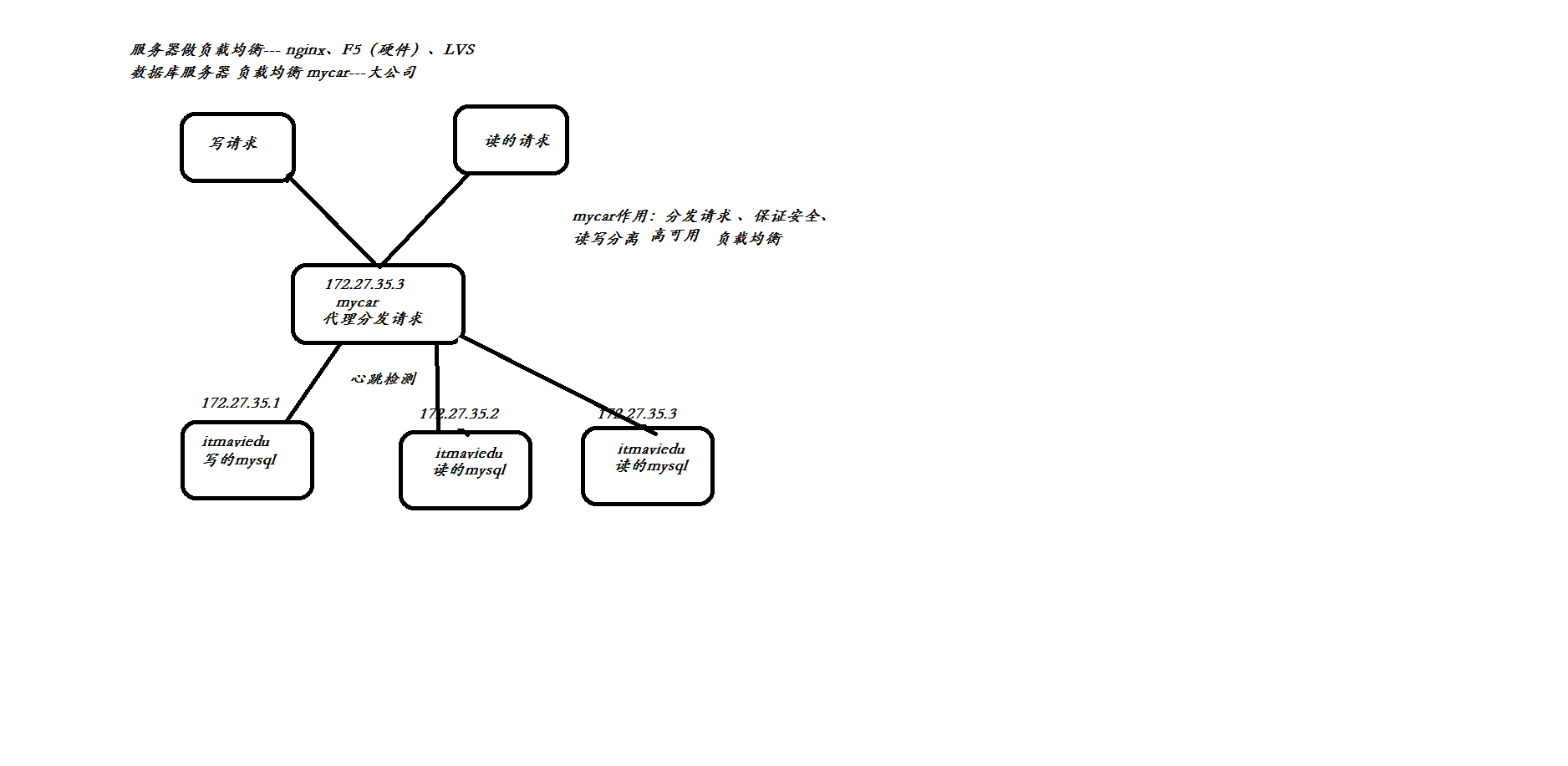
Insert、delete、update、createtable

## Scale-up与Scale-out区别

ScaleOut是指Application可以在水平方向上扩展。一般对数据中心的应用而言，Scaleout指的是当添加更多的机器时，应用仍然可以很好的利用这些机器的资源来提升自己的效率从而达到很好的扩展性。

ScaleUp是指Application可以在垂直方向上扩展。一般对单台机器而言，ScaleUp值得是当某个计算节点（机器）添加更多的CPUCores，存储设备，使用更大的内存时，应用可以很充分的利用这些资源来提升自己的效率从而达到很好的扩展性。

## MyCat



### 什么是Mycat

是一个开源的分布式数据库系统，但是因为数据库一般都有自己的数据库引擎，而Mycat并没有属于自己的独有数据库引擎，所有严格意义上说并不能算是一个完整的数据库系统，只能说是一个在应用和数据库之间起桥梁作用的中间件。

在Mycat中间件出现之前，MySQL主从复制集群，如果要实现读写分离，一般是在程序段实现，这样就带来了一个问题，即数据段和程序的耦合度太高，如果数据库的地址发生了改变，那么我的程序也要进行相应的修改，如果数据库不小心挂掉了，则同时也意味着程序的不可用，而对于很多应用来说，并不能接受；

引入Mycat中间件能很好地对程序和数据库进行解耦，这样，程序只需关注数据库中间件的地址，而无需知晓底层数据库是如何提供服务的，大量的通用数据聚合、事务、数据源切换等工作都由中间件来处理；

Mycat中间件的原理是对数据进行分片处理，从原有的一个库，被切分为多个分片数据库，所有的分片数据库集群构成完成的数据库存储，有点类似磁盘阵列中的RAID0.

### Mycat安装

#### 创建表结构

|  |
| --- |
| CREATEDATABASEIFNOTEXISTS`weibo\_simple`;  **--------------------------------------**  **--Tablestructurefor`t\_users`用户表**  **--------------------------------------**  **DROPTABLEIFEXISTS`t\_users`**;  **CREATETABLE`t\_users`**(  **`user\_id`varchar**(**64**)**NOTNULLCOMMENT'注册用户ID'**,  **`user\_email`varchar**(**64**)**NOTNULLCOMMENT'注册用户邮箱'**,  **`user\_password`varchar**(**64**)**NOTNULLCOMMENT'注册用户密码'**,  **`user\_nikename`varchar**(**64**)**NOTNULLCOMMENT'注册用户昵称'**,  **`user\_creatime`**datetime**NOTNULLCOMMENT'注册时间'**,  **`user\_status`**tinyint(**1**)**NOTNULLCOMMENT'验证状态1：已验证0：未验证'**,  **`user\_deleteflag`**tinyint(**1**)**NOTNULLCOMMENT'删除标记1：已删除0：未删除'**,  PRIMARY**KEY**(**`user\_id`**)  )**ENGINE**=**InnoDBDEFAULTCHARSET**=utf8;  **---------------------------------------**  **--Tablestructurefor`t\_message`微博表**  **---------------------------------------**  **DROPTABLEIFEXISTS`t\_message`**;  **CREATETABLE`t\_message`**(  **`messages\_id`varchar**(**64**)**NOTNULLCOMMENT'微博ID'**,  **`user\_id`varchar**(**64**)**NOTNULLCOMMENT'发表用户'**,  **`messages\_info`varchar**(**255**)**DEFAULTNULLCOMMENT'微博内容'**,  **`messages\_time`**datetime**DEFAULTNULLCOMMENT'发布时间'**,  **`messages\_commentnum`int**(**12**)**DEFAULTNULLCOMMENT'评论次数'**,  **`message\_deleteflag`**tinyint(**1**)**NOTNULLCOMMENT'删除标记1：已删除0：未删除'**,  **`message\_viewnum`int**(**12**)**DEFAULTNULLCOMMENT'被浏览量'**,  PRIMARY**KEY**(**`messages\_id`**),  **KEY`user\_id`**(**`user\_id`**),  **CONSTRAINT`t\_message\_ibfk\_1`**FOREIGN**KEY**(**`user\_id`**)**REFERENCES`t\_users`**(**`user\_id`**)  )**ENGINE**=**InnoDBDEFAULTCHARSET**=utf8; |

#### 配置server.xml

|  |
| --- |
| <!--添加user-->  <username="mycat">  <propertyname="password">mycat</property>  <propertyname="schemas">mycat</property>  </user>    <!--添加user-->  <username="mycat\_red">  <propertyname="password">mycat\_red</property>  <propertyname="schemas">mycat</property>  <propertyname="readOnly">true</property>  </user> |

#### 配置schema.xml

|  |
| --- |
| <?xmlversion="1.0"?>  <!DOCTYPEmycat:schemaSYSTEM"schema.dtd">  <mycat:schemaxmlns:mycat="http://org.opencloudb/">  <!--与server.xml中user的schemas名一致-->  <schemaname="mycat"checkSQLschema="true"sqlMaxLimit="100">  <tablename="t\_users"primaryKey="user\_id"dataNode="dn1"rule="rule1"/>  <tablename="t\_message"type="global"primaryKey="messages\_id"dataNode="dn1"/>  </schema>  <dataNodename="dn1"dataHost="jdbchost"database="weibo\_simple  "/>  <dataHostname="jdbchost"maxCon="1000"minCon="10"balance="1"  writeType="0"dbType="mysql"dbDriver="native"switchType="1"  slaveThreshold="100">  <heartbeat>selectuser()</heartbeat>  <writeHosthost="hostMaster"url="172.27.185.1:3306"user="root"password="root">  </writeHost>  <writeHosthost="hostSlave"url="172.27.185.2:3306"user="root"password="root"/>  </dataHost>  </mycat:schema> |

#### 配置rule.xml文件

|  |
| --- |
| <?xmlversion="1.0"encoding="UTF-8"?>  <!----LicensedundertheApacheLicense,Version2.0(the"License");  -youmaynotusethisfileexceptincompliancewiththeLicense.-You  mayobtainacopyoftheLicenseat--http://www.apache.org/licenses/LICENSE-2.0  --Unlessrequiredbyapplicablelaworagreedtoinwriting,software-  distributedundertheLicenseisdistributedonan"ASIS"BASIS,-WITHOUT  WARRANTIESORCONDITIONSOFANYKIND,eitherexpressorimplied.-Seethe  Licenseforthespecificlanguagegoverningpermissionsand-limitations  undertheLicense.-->  <!DOCTYPEmycat:ruleSYSTEM"rule.dtd">  <mycat:rulexmlns:mycat="http://org.opencloudb/">  <tableRulename="rule1">  <rule>  <columns>user\_id</columns>  <algorithm>func1</algorithm>  </rule>  </tableRule>  <functionname="func1"class="org.opencloudb.route.function.AutoPartitionByLong">  <propertyname="mapFile">autopartition-long.txt</property>  </function>  </mycat:rule> |

#### 为了更好地定位错误，修改log4j.xml

<levelvalue="debug"/>

双击startup\_nowrap.bat开始启动

## 常见问题

SHOWMASTERSTATUS如果为，则在my.ini文件中添加一行

log-bin=mysql-bin

给账号分配权限

grantallprivilegeson\*.\*to'root'@'172.27.185.1'identifiedby'root';

# Mycat

## Mycat介绍

### 什么是Mycat？

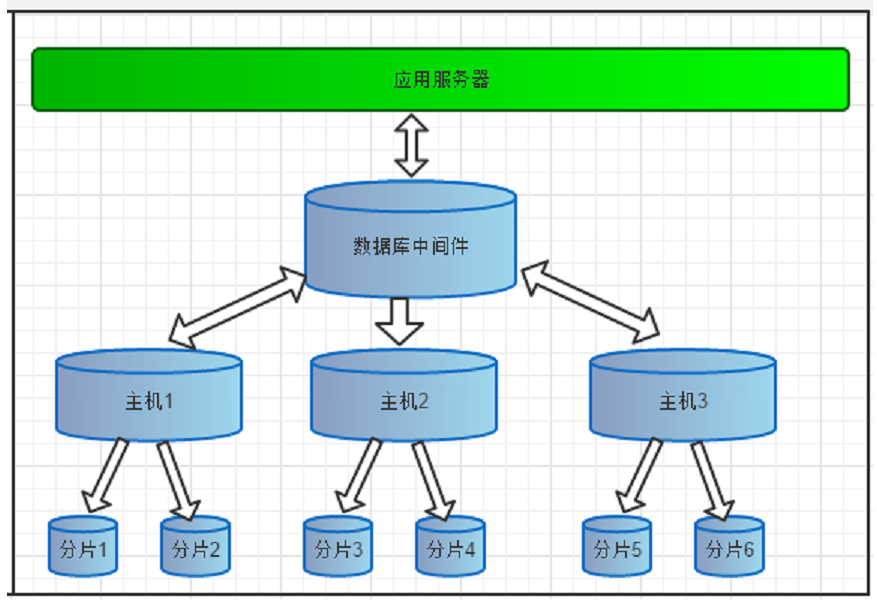
Mycat背后是阿里曾经开源的知名产品——Cobar。Cobar的核心功能和优势是MySQL数据库分片，此产品曾经广为流传，据说最早的发起者对Mysql很精通，后来从阿里跳槽了，阿里随后开源的Cobar，并维持到2013年年初，然后，就没有然后了。

Cobar的思路和实现路径的确不错。基于Java开发的，实现了MySQL公开的二进制传输协议，巧妙地将自己伪装成一个MySQLServer，目前市面上绝大多数MySQL客户端工具和应用都能兼容。比自己实现一个新的数据库协议要明智的多，因为生态环境在哪里摆着。

Mycat是基于cobar演变而来，对cobar的代码进行了彻底的重构，使用NIO重构了网络模块，并且优化了Buffer内核，增强了聚合，Join等基本特性，同时兼容绝大多数数据库成为通用的数据库中间件。

简单的说，MyCAT就是：

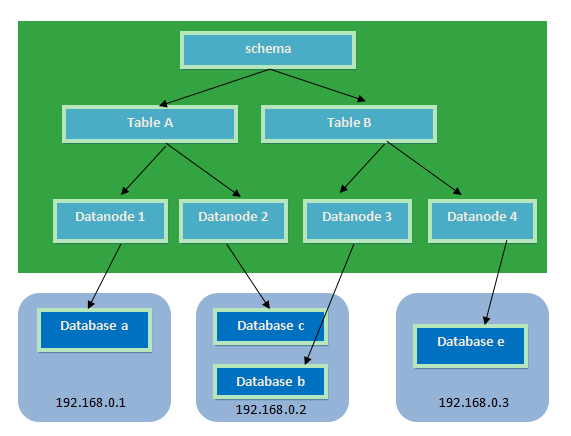
一个新颖的数据库中间件产品支持mysql集群，或者mariadbcluster，**提供高可用性数据分片集群。你可以像使用mysql一样使用mycat。对于开发人员来说根本感觉不到mycat的存在。**



### Mycat支持的数据库



### Mycat的分片策略



### 概念说明

#### 逻辑库(schema)：

前面一节讲了数据库中间件，通常对实际应用来说，并不需要知道中间件的存在，业务开发人员只需要知道数据库的概念，所以数据库中间件可以被看做是一个或多个数据库集群构成的逻辑库。

#### 逻辑表（table）：

既然有逻辑库，那么就会有逻辑表，分布式数据库中，对应用来说，读写数据的表就是逻辑表。逻辑表，可以是数据切分后，分布在一个或多个分片库中，也可以不做数据切分，不分片，只有一个表构成。

分片表：是指那些原有的很大数据的表，需要切分到多个数据库的表，这样，每个分片都有一部分数据，所有分片构成了完整的数据。总而言之就是需要进行分片的表。

非分片表：一个数据库中并不是所有的表都很大，某些表是可以不用进行切分的，非分片是相对分片表来说的，就是那些不需要进行数据切分的表。

#### 分片节点(dataNode)

数据切分后，一个大表被分到不同的分片数据库上面，每个表分片所在的数据库就是分片节点（dataNode）。

#### 节点主机(dataHost)

数据切分后，每个分片节点（dataNode）不一定都会独占一台机器，同一机器上面可以有多个分片数据库，这样一个或多个分片节点（dataNode）所在的机器就是节点主机（dataHost）,为了规避单节点主机并发数限制，尽量将读写压力高的分片节点（dataNode）均衡的放在不同的节点主机（dataHost）。

#### 分片规则(rule)

前面讲了数据切分，一个大表被分成若干个分片表，就需要一定的规则，这样按照某种业务规则把数据分到某个分片的规则就是分片规则，数据切分选择合适的分片规则非常重要，将极大的避免后续数据处理的难度。

## Mycat的下载及安装

### 安装环境

1. jdk：要求jdk必须是1.7及以上版本
2. Mysql：推荐mysql是5.5以上版本
3. Mycat：

Mycat的官方网站：

<http://www.mycat.org.cn/>

下载地址：

<https://github.com/MyCATApache/Mycat-download>

### 安装步骤

Mycat有windows、linux多种版本。本教程为linux安装步骤，windows基本相同。

第一步：下载Mycat-server-xxxx-linux.tar.gz

第二步：将压缩包解压缩。建议将mycat放到/usr/local/mycat目录下。

第三步：进入mycat目录，启动mycat

./mycatstart

停止：

./mycatstop

mycat支持的命令{console|start|stop|restart|status|dump}

Mycat的默认端口号为：8066

## Mycat的分片

### 需求

把商品表分片存储到三个数据节点上。

### 安装环境分析

两台mysql数据库服务器：

Host1：192.168.25.134

Host2：192.168.25.166

host1环境

操作系统版本:centos6.4

数据库版本:mysql-5.6

mycat版本：1.4release

数据库名:db1、db3

mysql节点2环境

操作系统版本:centos6.4

数据库版本:mysql-5.6

mycat版本：1.4release

数据库名:db2

MyCat安装到节点1上（需要安装jdk）

### 配置schema.xml

#### Schema.xml介绍

Schema.xml作为MyCat中重要的配置文件之一，管理着MyCat的逻辑库、表、分片规则、DataNode以及DataSource。弄懂这些配置，是正确使用MyCat的前提。这里就一层层对该文件进行解析。

schema标签用于定义MyCat实例中的逻辑库

Table标签定义了MyCat中的逻辑表

dataNode标签定义了MyCat中的数据节点，也就是我们通常说所的数据分片。

dataHost标签在mycat逻辑库中也是作为最底层的标签存在，直接定义了具体的数据库实例、读写分离配置和心跳语句。

|  |
| --- |
| 注意：若是LINUX版本的MYSQL，则需要设置为Mysql大小写不敏感，否则可能会发生表找不到的问题。  在MySQL的配置文件中/etc/my.cnf[mysqld]中增加一行  　　lower\_case\_table\_names=1 |

#### Schema.xml配置

|  |
| --- |
| <?xmlversion="1.0"?>  <!DOCTYPEmycat:schemaSYSTEM"schema.dtd">  <mycat:schemaxmlns:mycat="http://org.opencloudb/">  <schemaname="e3mall"checkSQLschema="false"sqlMaxLimit="100">  <!--autoshardingbyid(long)-->  <tablename="tb\_item"dataNode="dn1,dn2,dn3"rule="auto-sharding-long"/>  </schema>  <dataNodename="dn1"dataHost="localhost1"database="db1"/>  <dataNodename="dn2"dataHost="localhost2"database="db2"/>  <dataNodename="dn3"dataHost="localhost1"database="db3"/>  <dataHostname="localhost1"maxCon="1000"minCon="10"balance="0"  writeType="0"dbType="mysql"dbDriver="native"switchType="1"slaveThreshold="100">  <heartbeat>selectuser()</heartbeat>  <!--canhavemultiwritehosts-->  <writeHosthost="hostM1"url="192.168.25.134:3306"user="root"  password="root">  <!--canhavemultireadhosts-->  </writeHost>  </dataHost>  <dataHostname="localhost2"maxCon="1000"minCon="10"balance="0"  writeType="0"dbType="mysql"dbDriver="native"switchType="1"slaveThreshold="100">  <heartbeat>selectuser()</heartbeat>  <!--canhavemultiwritehosts-->  <writeHosthost="hostM1"url="192.168.25.166:3306"user="root"  password="root">  <!--canhavemultireadhosts-->  </writeHost>  </dataHost>  </mycat:schema> |

### 配置server.xml

#### Server.xml介绍

server.xml几乎保存了所有mycat需要的系统配置信息。最常用的是在此配置用户名、密码及权限。

#### Server.xml配置

|  |
| --- |
| <username="test">  <propertyname="password">test</property>  <propertyname="schemas">e3mall</property>  <propertyname="readOnly">false</property>  </user> |

### 配置rule.xml

rule.xml里面就定义了我们对表进行拆分所涉及到的规则定义。我们可以灵活的对表使用不同的分片算法，或者对表使用相同的算法但具体的参数不同。这个文件里面主要有tableRule和function这两个标签。在具体使用过程中可以按照需求添加tableRule

和function。

此配置文件可以不用修改，使用默认即可。

### 测试分片

#### 创建表

配置完毕后，重新启动mycat。使用mysql客户端连接mycat，创建表。

|  |
| --- |
| ------------------------------  --Tablestructurefortb\_item  ------------------------------  DROPTABLEIFEXISTS`tb\_item`;  CREATETABLE`tb\_item`(  `id`bigint(20)NOTNULLCOMMENT'商品id，同时也是商品编号',  `title`varchar(100)NOTNULLCOMMENT'商品标题',  `sell\_point`varchar(500)DEFAULTNULLCOMMENT'商品卖点',  `price`bigint(20)NOTNULLCOMMENT'商品价格，单位为：分',  `num`int(10)NOTNULLCOMMENT'库存数量',  `barcode`varchar(30)DEFAULTNULLCOMMENT'商品条形码',  `image`varchar(500)DEFAULTNULLCOMMENT'商品图片',  `cid`bigint(10)NOTNULLCOMMENT'所属类目，叶子类目',  `status`tinyint(4)NOTNULLDEFAULT'1'COMMENT'商品状态，1-正常，2-下架，3-删除',  `created`datetimeNOTNULLCOMMENT'创建时间',  `updated`datetimeNOTNULLCOMMENT'更新时间',  PRIMARYKEY(`id`),  KEY`cid`(`cid`),  KEY`status`(`status`),  KEY`updated`(`updated`)  )ENGINE=InnoDBDEFAULTCHARSET=utf8COMMENT='商品表'; |

#### 插入数据

将此文件中的数据插入到数据库：



#### 分片测试

由于配置的分片规则为“auto-sharding-long”，所以mycat会根据此规则自动分片。

每个datanode中保存一定数量的数据。根据id进行分片

经测试id范围为：

Datanode1：1~5000000

Datanode2：5000000~10000000

Datanode3：10000001~15000000

当15000000以上的id插入时报错：

[Err]1064-can'tfindanyvaliddatanode:TB\_ITEM->ID->15000001

此时需要添加节点了。

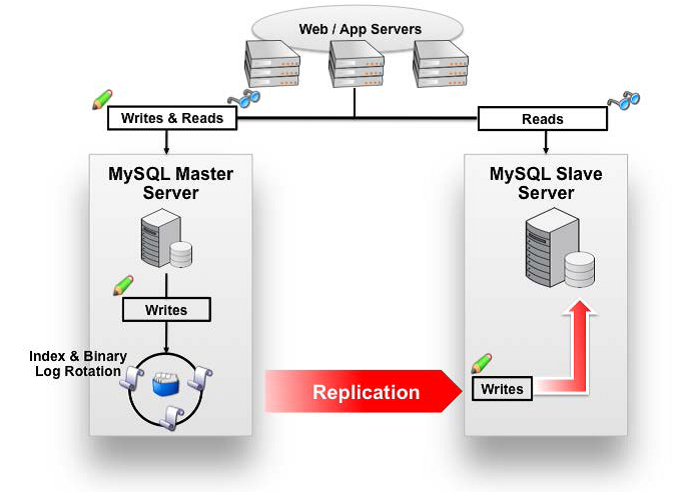
## Mycat读写分离

数据库读写分离对于大型系统或者访问量很高的互联网应用来说，是必不可少的一个重要功能。对于MySQL来说，标准的读写分离是主从模式，一个写节点Master后面跟着多个读节点，读节点的数量取决于系统的压力，通常是1-3个读节点的配置



Mycat读写分离和自动切换机制，需要mysql的主从复制机制配合。

### Mysql的主从复制



主从配置需要注意的地方

1、主DBserver和从DBserver数据库的版本一致

2、主DBserver和从DBserver数据库数据名称一致

3、主DBserver开启二进制日志,主DBserver和从DBserver的server\_id都必须唯一

### Mysql主服务器配置

第一步：修改my.conf文件：

在[mysqld]段下添加：

|  |
| --- |
| binlog-do-db=db1  binlog-ignore-db=mysql  #启用二进制日志  log-bin=mysql-bin  #服务器唯一ID，一般取IP最后一段  server-id=134 |

第二步：重启mysql服务

servicemysqldrestart

第三步：建立帐户并授权slave

mysql>GRANTFILEON\*.\*TO'backup'@'%'IDENTIFIEDBY'123456';

mysql>GRANTREPLICATIONSLAVE,REPLICATIONCLIENTON\*.\*to'backup'@'%'identifiedby'123456';

#一般不用root帐号，“%”表示所有客户端都可能连，只要帐号，密码正确，此处可用具体客户端IP代替，如192.168.145.226，加强安全。

刷新权限

mysql>FLUSHPRIVILEGES;

查看mysql现在有哪些用户

mysql>selectuser,hostfrommysql.user;

第四步：查询master的状态

mysql>showmasterstatus;

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

|File|Position|Binlog\_Do\_DB|Binlog\_Ignore\_DB|Executed\_Gtid\_Set|

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

|mysql-bin.000001|120|db1|mysql||

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

1rowinset

### Mysql从服务器配置

第一步：修改my.conf文件

[mysqld]

server-id=166

第二步：配置从服务器

mysql>changemastertomaster\_host='192.168.25.134',master\_port=3306,master\_user='backup',master\_password='123456',master\_log\_file='mysql-bin.000001',master\_log\_pos=120

注意语句中间不要断开，master\_port为mysql服务器端口号(无引号)，master\_user为执行同步操作的数据库账户，“120”无单引号(此处的120就是showmasterstatus中看到的position的值，这里的mysql-bin.000001就是file对应的值)。

第二步：启动从服务器复制功能

Mysql>startslave;

第三步：检查从服务器复制功能状态：

mysql>showslavestatus

……………………(省略部分)

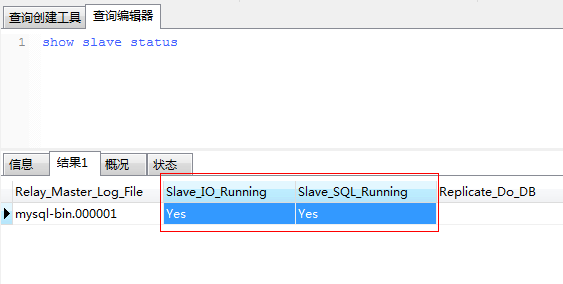
Slave\_IO\_Running:Yes//此状态必须YES

Slave\_SQL\_Running:Yes//此状态必须YES

……………………(省略部分)

注：Slave\_IO及Slave\_SQL进程必须正常运行，即YES状态，否则都是错误的状态(如：其中一个NO均属错误)。

|  |
| --- |
| 错误处理：  如果出现此错误：  Fatalerror:TheslaveI/OthreadstopsbecausemasterandslavehaveequalMySQLserverUUIDs;theseUUIDsmustbedifferentforreplicationtowork.  因为是mysql是克隆的系统所以mysql的uuid是一样的，所以需要修改。 |
| 解决方法：  删除/var/lib/mysql/auto.cnf文件，重新启动服务。 |



以上操作过程，从服务器配置完成。

### Mycat配置

Mycat1.4支持MySQL主从复制状态绑定的读写分离机制，让读更加安全可靠，配置如下：

|  |
| --- |
| <dataNodename=*"dn1"*dataHost=*"localhost1"*database=*"db1"*/>  <dataNodename=*"dn2"*dataHost=*"localhost1"*database=*"db2"*/>  <dataNodename=*"dn3"*dataHost=*"localhost1"*database=*"db3"*/>  <dataHostname=*"localhost1"*maxCon=*"1000"*minCon=*"10"***balance=*"1"***  **writeType=*"0"***dbType=*"mysql"*dbDriver=*"native"***switchType=*"2"*slaveThreshold=*"100"***>  <heartbeat>**showslavestatus**</heartbeat>  <writeHosthost=*"hostM"*url=*"192.168.25.134:3306"*user=*"root"*  password=*"root"*>  <readHosthost=*"hostS"*url=*"192.168.25.166:3306"*user=*"root"*  password=*"root"*/>  </writeHost>  </dataHost> |

1. **设置balance="1"与writeType="0"**

Balance参数设置：

1.balance=“0”,所有读操作都发送到当前可用的writeHost上。

2.balance=“1”，所有读操作都随机的发送到readHost。

3.balance=“2”，所有读操作都随机的在writeHost、readhost上分发

WriteType参数设置：

1.writeType=“0”,所有写操作都发送到可用的writeHost上。

2.writeType=“1”，所有写操作都随机的发送到readHost。

3.writeType=“2”，所有写操作都随机的在writeHost、readhost分上发。

“readHost是从属于writeHost的，即意味着它从那个writeHost获取同步数据，因此，当它所属的writeHost宕机了，则它也不会再参与到读写分离中来，即“不工作了”，这是因为此时，它的数据已经“不可靠”了。基于这个考虑，目前mycat1.3和1.4版本中，若想支持MySQL一主一从的标准配置，并且在主节点宕机的情况下，从节点还能读取数据，则需要在Mycat里配置为两个writeHost并设置banlance=1。”

1. **设置switchType="2"与slaveThreshold="100"**

**switchType目前有三种选择：**

**-1：表示不自动切换**

**1：默认值，自动切换**

**2：基于MySQL主从同步的状态决定是否切换**

“Mycat心跳检查语句配置为showslavestatus，dataHost上定义两个新属性：switchType="2"与slaveThreshold="100"，此时意味着开启MySQL主从复制状态绑定的读写分离与切换机制。Mycat心跳机制通过检测showslavestatus中的"Seconds\_Behind\_Master","Slave\_IO\_Running","Slave\_SQL\_Running"三个字段来确定当前主从同步的状态以及Seconds\_Behind\_Master主从复制时延。“

## 附：Centos6.5下安装mysql

第一步：查看mysql是否安装。

rpm-qa|grepmysql

第二步：如果mysql的版本不是想要的版本。需要把mysql卸载。

yumremovemysqlmysql-servermysql-libsmysql-common

rm-rf/var/lib/mysql

rm/etc/my.cnf

第三步：安装mysql。需要使用yum命令安装。在安装mysql之前需要安装mysql的下载源。需要从oracle的官方网站下载。

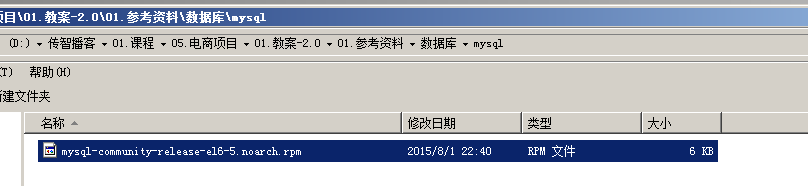
1. 下载mysql的源包。

我们是centos6.4对应的rpm包为：mysql-community-release-el6-5.noarch.rpm

1. 安装mysql下载源：

yumlocalinstallmysql-community-release-el6-5.noarch.rpm

()此附件可保存



1. 在线安装mysql：

yuminstallmysql-community-server

第四步：启动mysql

servicemysqldstart

第五步：需要给root用户设置密码。

/usr/bin/mysqladmin-urootpassword'new-password'　　//为root账号设置密码

第六步：远程连接授权。

GRANTALLPRIVILEGESON\*.\*TO'myuser'@'%'IDENTIFIEDBY'mypassword'WITHGRANTOPTION;

注意：'myuser'、'mypassword'需要替换成实际的用户名和密码。