# Mysql 1

## 1.数据库的基础

### （1） 什么是数据库

数据库：database，存储数据的仓库。

数据库：高效存储和处理数据的介质(介质主要是两种：磁盘和内存)。

### （2）数据库分类

数据库基于存储介质的不同，可分为两类：

关系型数据库（磁盘）（SQL）和非关系型数据库（内存）(NoSqQL)。

### （3） 不同的数据库阵营中的产品有哪些

关系型数据库：大型：Oracle，DB2

中型：SQL-SERVER，Mysql等

小型：accdss等

非关系型数据库：memcached，mongodb，redis（同步到磁盘）

### （4）两种数据阵营的区别

关系型数据库：安全（保存磁盘基本不丢失），符合人的思维习惯，容易理解；比较浪费空间（二维数据表）

非关系型数据库：效率高，不安全（断电丢失）

### 1.1关系型数据库

#### 1.1.1什么是关系型数据库

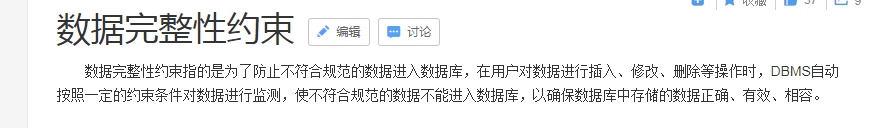
关系型数据库：是一种建立在关系模型（数学模式）上的数据库

关系模型：是一种所谓建立在关系上的模型，关系模型就是指二维表格模型,因而一个关系型数据库就是由二维表及其之间的联系组成的一个数据组织。

关系模型包含三个方面：

数据结构：数据存储的问题，二维表（行和列）

操作指令集合：所有SQL语句（DDL,DML【DQL】和DCL）

完整性约束：表内数据约束（字段与字段），表与表之间约束(外键) 

#### 1.1.2关系型数据库的设计

关系型数据库：从需要存储的数据需求中分析，如果是一类数据（实体）应该设计成一张二维表，表示由表头（字段名：用来规定数据的名字）和数据部分组成（实际存储的数据单元）

二维表：行和列

|  |  |  |
| --- | --- | --- |
| 表头 | 字段名1 | 字段名2 |
| 数据单元 | 数据1 | 数据2 |

案例分析：分析一个教学系统，讲师负责教学，教学生，在教师教学生

①找出系统中所存在的实体：讲师表，学生表，班级表

②找出实体中应该存在的数据信息

a)讲师：姓名，性别，年龄，工资

b)学生：姓名，性别，学号，学科

c)班级：班级名字，教师编号

注：关系型数据库：维护实体内部，实体与实体之间的联系

\*实体内部联系：每个学生都有姓名，性别，学号，学科信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓名 | 性别 | 学号 | 学科 | 年龄 |
| A | 男 | Ictast0001 | PHP | 20 |
| B | 女 | Ictast0002 | PHP |  |
| C |  | Ictast0003 | UI |  |

第二行的所有字段，都是在描述陈明这个学生（内部联系）；第二列只能放性别（内部约束）

关系型数据库的特点之一：如果表中对应的某个字段没有值（数据）但是系统依然要分配空间（关系型数据库比较浪费空间）

\*实体与实体之间的联系：每个学生肯定属于某个班级，每个班级一定有多个学生（一对多）

学生表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 姓名 | 性别 | 学号 | 学科 | 年龄 |
| A | 男 | Ictast0001 | PHP | 20 |
| B | 女 | Ictast0002 | PHP |  |
| C |  | Ictast0003 | UI |  |

班级表

|  |  |
| --- | --- |
| 班级名称 | 教室编号 |
| PHP0810 | B205 |
| PHP0810 | A203 |
| UI0810 | B330 |

如何将学生表和班级表关联起来？

解决方案：在学生表中增加一个班级字段俩指定班级（必须能唯一指定一个班级信息）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 性别 | 学号 | 学科 | 年龄 | 班级名称 |
| A | 男 | Ictast0001 | PHP | 20 | PHP0810 |
| B | 女 | Ictast0002 | PHP |  | PHP0810 |
| C |  | Ictast0003 | UI |  | UI0810 |

学生实体与班级实体的关联关系：实体与实体之间的关联关系

#### 1.2关键字说明

（1）数据库：database

（2）数据库系统：DBS(Database System)是一种虚拟系统，将多种内容关联起来的称呼

DBS = DBNS + DB

DBMS:Database Mangement System数据库管理系统，专门管理数据库

DBA:Database Adminstrator数据库管理员

（3）行/记录：row/record本质是一个东西：都是指表中的一行（一条记录）

行：是从结构角度出发

记录：是从数据结构出发

（4）列/字段：column/field本质是同一个东西

#### 1.3SQL

（1）SQL:Structured Query Language结构化查询语言（数据以查询为主：99%是在查询操作）

（2）SQL是关系型数据库的操作指令，SQL是一种约束，但不强制：不同的数据库产品（如Oracle，mysql）内部可能会有一些细微的区别。

（3）SQL分为三个部分

DDL:Data Defination Lanuage，数据定义语言，用来维护存储数据的结构（数据库，表）代表指令：create，drop，alter等

DML: Data Manipulation Language 数据操作语言，用来对数据进行操作（数据表中的内容），代表指令：insert，delete，update等；其中DML内部有单独进行了一次分类：DQL(Data Query Language:数据查询语言，如select)

DCL:Data Control Language，数据控制语言，主要负责权限管理（用户），代表指令：grant，revoke等。

#### 1.4 Mysql数据库

（1）Mysql数据库是一种c/s结构的软件：客户端/服务器，若想访问服务器必须通过客户端（服务器一种运行，客户端在需要的时候运行）

（2）交互方式

1.客户端连接认证：连接服务器，认证身份：mysql.exe -hPup

2.发送SQL指令

3.服务器接收SQL指令，处理SQL指令，返回操作结果。

4.客户端接收结果：显示结果

5.断开连接（释放资源：服务器并发限制）

#### 1.5 Mysql服务器对象

（1）没有办法完全了解服务器内部的内容：只能粗略的去分析数据库服务器的内部结构。

（2）将mysql服务器内部对象分成四层:

系统（DBMS）->数据库（DB）->数据表（Table）->字段（field）

## 2.SQL基本操作（CRUD）

将SQL的基本操作对象进行分类，分为三类：库操作，表操作（字段）数据操作

### 2.1库操作：

对数据库的增删改查

#### 2.1.1新增数据库

（1）Create database数据库名字平[库选项]

（2）库选项：用来约束数据库，分为两个选项

 字符集设定：charset/character set 具体字符集（数据存储的编码格式）

                常用字符集：CBK和UTF8

校对集设定:  collate具体校对集（数据比较的规则）

（3）数据库名是英文名

Create database mydatabasecharset utf8；

（4）数据库名是中文名

Set names gbk；

Create database 中国charset utf8；

#### 2.1.2查看数据库

（1）查看所有数据库show databases;

（2）查看指定部分数据库：模糊查询

**show databases like‘pattern’     -- pattern是匹配模式**

%：表示匹配多个字符

\_:  表示单个字符

（3）查看数据库的创建语句：

show create database 数据库名

#### 2.1.3更新数据库

（1）数据库名字不可以修改

（2）数据库修改仅限库选项：字符集和校对集（校对集依赖字符集）

（3）Alter database数据库名字[库选项];

Charset/characterset [=]字符集

Collate校对集

#### 2.1.4删除数据库

（1）Drop database数据库名字

（2）在对应的数据库存储的文件夹内：数据库名字对应的文件夹也被删除（级联删除：里面的数据表全部被删除）

## 3表操作

### 3.1新增数据表

Create table[if not exists]（

字段名字 数据类型， --字段名（就是列表名字），数据类型（int整型char定长字符varchar不定长字符）

字段名字 数据类型   --最后一行不需要逗号

）[表选项]；

（1）If not exists:如果表名不存在，那么就创建，否则不执行创建代码（检查功能）

（2）表选项：控制表的表现

字符集：charset/character set具体字符集(保证表中数据储存的字符集)

校对集：collate具体校对集；

储存引擎: engine具体的存储引擎（innodb和myisam）-- innodb：默认存储引擎

（3）没有选择数据库时错误提示

任何一个表的设计都必须指定数据库

方案1：显式地指定表的所属

语法：create table【if not exists】**数据库名字.** 表名字（

字段名字 数据类型

字段名字 数据类型

）【表选项】；

方案2：隐式地指定表的所属数据库：先进入到某个数据库环境，然后这样创建的表自动归属到某个指定的数据库。

**Use 数据库名字**

Create table 【if not exists】表名字（

字段名 数据类型

字段名 数据类型

）【表选项】；

进入数据库环境：use数据库名字

当创建数据表的SQL指令执行之后，到底发生了什么？

（1）指定数据库下已经存在对应的表

（2）在数据库对应的文件夹下，会产生对应表的结构文件（跟存储引擎有关系）

### 3.2查看数据表

（1）查看所有表：show tables;

（2）查看部分表：show table like‘pattern’

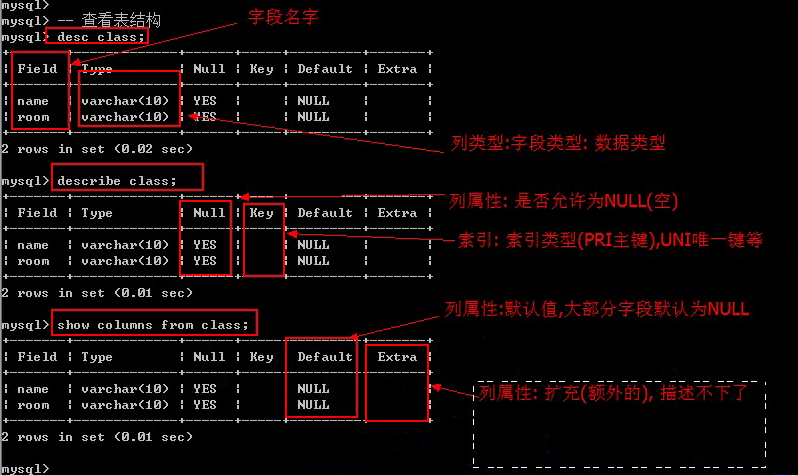
（3）查看表的创建语句：show create table表名;

show create table表名\g --\g = ;

show create table表名\G --\G = 将表格旋转90°纵向查看

（4）查看表结构：

查看表中字段信息：Desc/describe/show columns from表名;



Field---字段名

Type---数据类型

Null---列属性：是否允许为Null(空)

Key---索引：索引类型（PRI主键，UNI唯一键等）

Default---列属性：大部分字段默认为Null（空） -- 意思是：默认值

Extra---列属性：扩充（额外的），描述不下了

### 3.3修改数据表

表本身存在，还包含字段：表的修改分为两个部分--修改表本身和修改字段

#### 3.3.1修改表本身

（1）修改表名：rename  table老表名 to新表名

修改表选项：字符集，校对集和存储引擎

Alter table表选项[=]值；

#### 3.3.2修改字段

字段操作包括：新增，修改，重名，删除

##### （1）新增字段：

alter table表名add [column]数据类型[列属性][位置];

位置：字段名可以存放在表中的任意位置

First：第一个位置

After 字段名：在该字段名之后，默认是最后一个字段之后

##### （2）修改字段：

alter table表名modify字段名 数据类型[位置][属性]

##### （3）重名字段：

alter table表名change旧字段 新字段名 数据类型[位置][属性]

##### （4）删除字段：

alter table表名drop字段名;

#### 3.4删除数据表

Drop table表名1，表名2........;  --可以一次删除多张表

当删除数据表的指令执行后发生了什么？

（1）在表空间中，没有了指定的表（数据也没有了）

（2）在数据库对应的文件夹下，表对应的文件（与引擎有关）也会被删除。

## 4. 数据操作

### 4.1新增数据

方案1：给全表字段插入数据，不需要指定字段列表，需要数据的值出现顺序必须与表中设计的字段出现的顺序一致。凡是非数值数据，都需要使用引号（建议单引号）包裹。可以一次性插入多条记录

Insert into表名values（值列表）[值列表];

方案2：给部分字段插入数据，需要选定字段列表：字段列表出现的顺序与字段的顺序无关。但是值列表的顺序必须与选定的字段顺序一致。

Insert into表名（字段列表）values（值列表）[，值列表];

### 4.2查看数据

Select \*/字段列表from表名[where条件];

### 4.3更新数据

Update表名set字段=值[where条件];  --建议都有where

注：更新不一定成功，有影响的才成功。

### 4.4删除数据

Delete from表名[where条件];

## 5. 中文数据问题

中文数据问题本质是字符集问题。

（1）查看服务器的所有字符集：show character set;

（2）查看服务器默认对我处理的字符集：

Show variables like‘character\_set%’ ;

Character\_set\_client：服务器默认的客户端字符集

Character\_set\_connection：连接层字符集

Character\_setdatabase：当前所在数据库的字符集

Character\_set\_results：服务器默认的给外部数据的字符集

表中出现乱码分析1：

问题根源：客户端数据只能是GBK，而服务器认为是UTF8。

解决方案：改变服务器，默认的接收字符集为GBK。

set character\_set\_client = gbk;

表中出现乱码分析2：

查看数据，中文为乱码？

问题根源：数据来源是服务器，解析数据的是客户端（客户端只识别gbk），但是服务器给的数据是utf8

解决方案：修改服务器给客户端的数据字符集为GBK。

set character\_set\_results = gbk;

中文显示正常！

（3）设置服务器对客户端字符集的认识可以使用快捷方式：

set names字符集

 注：修改Character\_set\_client，Character\_set\_results和Character\_set\_connection

（4）connection连接层：是字符集转变的中间者，如果统一了效率更高，不统一也没问题。

## 6. 校对集

校对集：数据比较的方式

（1）校对集有三种格式：

bin：binary，二进制比较，取出二进制位，一位一位比较，区分大小写。

cs：case sensitive 大小写敏感，区分大小写。

ci：case insensitive 大小写不敏感，不区分大小写。

（2）查看数据库所支持的校对集：Show collation;

（3）校对集应用：只有当数据产生比较的时候，校对集才会生效。

第一步：创建不同校对集表

校对集：utf8\_bin

create table my\_collate\_bin(

name char(1)

)charset utf8 collate utf8\_bin;

校对集：utf8\_general\_ci

Create table my\_collate\_ci(

name char(1)

)charset utf8 collate utf8\_general\_ci;

第二步：插入数据

insert into my\_collate\_bin values('a'),('A'),('B'),('b');

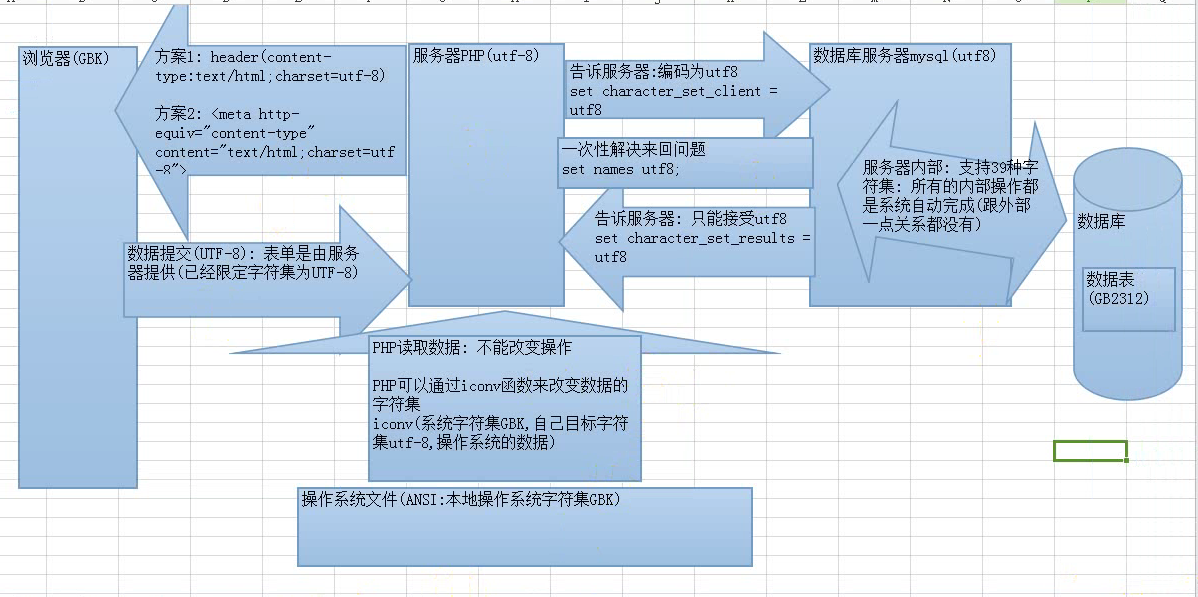
insert into my\_collate\_ci values('a'),('A'),('B'),('b');

第三步：查看数据

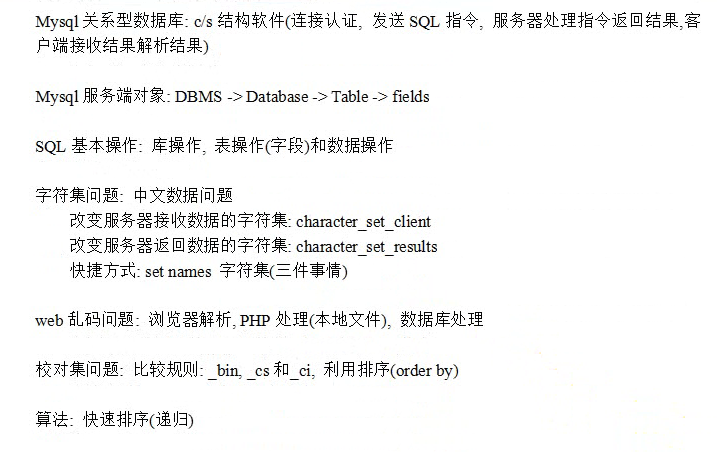
select \* from my\_collate\_bin order by name;

select \* from my\_collate\_ci order by name;

## 7.乱码问题



## 8.总结（回顾）



# Mysql 2

## 1. 数据类型（列类型）

（1）数据类型：对数据进行统一的分类，从系统的角度出发时为了能使用统一的方式进行管理，更好地利用有限的空间。

（2）SQL中将数据类型分成了三大类：数值类型，字符串类型和时间日期类型



### 1.1数值型

数值型数据：都是数据。

系统将数值型分为整数型和小数型。

#### 1.1.1整数型

在SQL中要考虑如何节省磁盘空间，所以系统将整形细分为5类。

 Tinyint 迷你整形，使用1个字节（8位）存储，表示状态最多为256种。-- 位 bit（最小的单位） -- 字节 byte

 Smallint 小整形，使用2 group by个字节（16位）存储，表示状态最多为65536（2的16）种。-- 8位=1字节（8bit =1byte）

 Mediuint 中整形，使用3个字节（24）存储，表示状态最多为16777216种。

  Int 标志整形，使用4个字节存储，表示状态最多为4294967296种。

 Bigint 大整形，使用8个字节存储

（1）使用相应数据类型时，存储的值需在规定的范围内。

（2）SQL中的数值类型全部都是默认有符号：分正负

（3）每个字段数据类型之后括号中的数字代表的是显示宽度。

（4）显示宽度的意义：当数据显示宽度不够的时候，会自动让数据变成对应的显示宽度。通常需要搭配一个前导0来增加宽度，不改变值大小。Zerofill（零填充）会导致无符号数。

（5）零填充的意义（显示宽度）：保证数据格式

#### 1.1.2 小数型

小数型：带有小数点或者范围超出整形的数值类型

SQL中将小数型分为：浮点型和定点型

 浮点型：小数点浮动，精度有限，而且会丢失精度。

Float：单精度，占用4个字节存储数据，精度范围大概7位左右。

Double：双精度，占用8个字节存储数据，精度范围大概为15为左右。

创建浮点数表：（1）直接float（表示没有小数部分）

（2）float(M,D) M代表总长度，D代表小数部分长度，整形长度  为（M-D）

（3）插入的数据可以是小数，也可以是科学计数法。

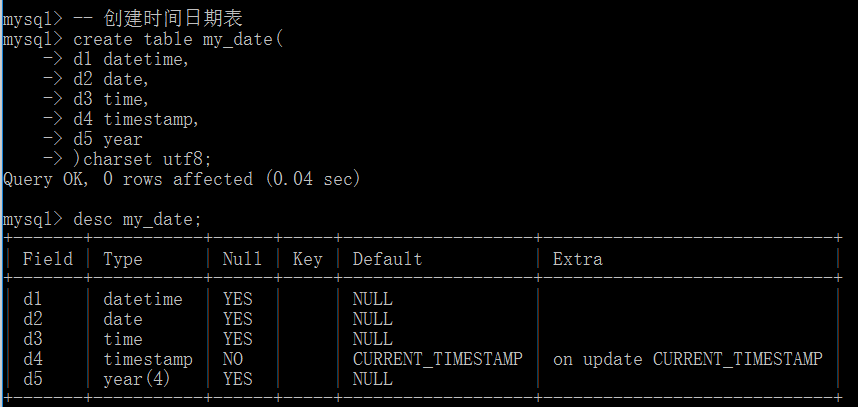
（4）整数部分 不能超出长度，小数部分可以。（4舍5入）

 定点型：小数点固定，精度固定，不会丢失精度。

绝对保证整数部分和不会被四舍五入（不会丢失精度），小数部分有可能（理论小数部分也不会丢失精度）。

### 1.2时间日期类型





### 1.3 字符串类型

在SQL中，将字符串类型分成了6类：char，vachar，text，blob，enum和set。

#### 1.3.1 定长字符串（char）

（1）在磁盘（二维表）中定义结构时，就确定了最终数据的存储长度。

（2）Char(L)

L代表存储长度，单位为字符，最大长度值为255。

Char[4]在utf8环境下，需要4\*3=12个字节。

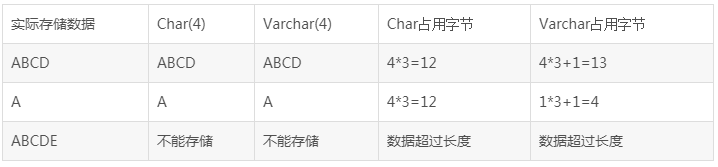
#### 1.3.2变长字符串varchar

（1）在分配空间时，按照最大的空间分配。根据具体的数据来确定用多少空间。

（2）Varchar(L)：L表示字符串长度，理论长度65536个字符，但是会多出1到2个字节来确定存储的实际长度。

（3）Varchar(10)：若存了10个汉字，utf8环境下占用空间：10\*3+1=31

定长与变长的存储实际空间(UTF8)



如何选择定长或者变长字符串？

定长的磁盘空间比较浪费，但是效率高。如果数据长度基本一样，就使用定长，如身份证，手机号码，电话号码等。

变长的磁盘空间比较节省，但是效率低。如果不同数据长度有变化，如姓名，地址。

#### 1.3.3 文本字符串

如果数据量非常大（超过255）就会使用文本字符串。文本字符串根据储存的数据格式进行分类：text和blob

Text：储存文字（二进制数据实际上都是只存数据的储存路径）

Blob：储存二进制数据（通常不用）

#### 1.3.4 枚举字符串

（1）枚举（enum）：事先将所有可能出现的结果都设计好，实际上储存的数据必须是规定好的数据中的一个。

（2）枚举的定义：enum（可能出现的元素列表）;

（3）枚举作用

作用1：规范数据格式，数据只能是规定的数据中的一个。

作用2：节省储存空间（枚举别名：单选框）。在mysql中，系统自动转换数据格式。

（4）枚举原理：枚举在进行数据规范的时候（定义的时候），系统自动建立一个数字与枚举元素的对应关系（关系放到日志中）。

在进行数据插入的时候，系统自动将字符转成对应的数字存储。

在进行数据提取的时候，系统自动将数值转换成对应的字符串显示。

（5）枚举存储的是数值（1,2.....），故可直接插入数值。

#### 1.3.5 集合字符串

（1）集合跟枚举相似，实际存储的是数值，而不是字符串（集合是多选）

（2）集合的使用方式

定义：set（元素列表）

使用：可以使用元素列表中的元素，使用逗号分隔。

（3） 集合中每一个元素都是对应一个二进制位，被选中为1，没有则为0.

## 2. Mysql记录长度

（1）Mysql规定：任何一条记录最长不能超过65535个字节（varchar永远达不到理论值）。

（2）Utf8下的varchar的实际顶配：21844（21844\*3+2=65534）字符

（3）Gbk下的varchar的实际顶配：32766（32766\*2+2=65534）字符

（4）Mysql记录中，如果有任何一个字段允许为空，那么系统会自动从整个记录中保留一个字节来存储NULL（若想释放Null所占用的字节，必须保证所有的字符都不允许为空）。

（5）Mysql中text文本字符串，不占用记录长度（额外存储），但是text文本字符串也是属于记录的一部分，一定需要占用记录中的部分长度（10个字符串）。

## 3. 列属性

（1）真正约束自动的是数据类型，但是数据类型的约束很单一，需要有一些额外的约束来保证数据的合法性。

（2）列属性：NULL/NOT NULL，default，primary，unique key，auto\_increment，comment。

### 3.1 空属性

两个值NULL（默认的）和NOT NULL（不为空）。

虽然默认为空，但是实际开发中，尽可能让数据不为空。

### 3.2列描述

comment

没有实际含义，专门用来描述字段，会根据表创建语句保存。用来给程序员（数据库管理员）来进行了解的。

### 3.3默认值

某一种数据经常性出现某个具体的值，可以在一开始就指定。在需要的真是数据时，用户可以选择性的使用默认值。

# Mysql 3

## 1. 字段属性

主键，唯一键和自增长。

### 主键

（primary key）

一张表中只有一个字段可以使用主键，用来唯一约束该字段里面的数据，使其不能重复。

一张表只能有最多一个主键

#### 1.1.1增加主键

SQL操作中有多种方式可以给表增加主键：大体分为三种

方案1：在创建表的时候，直接在字段之后跟primary key关键字（主键本身不允许为空）【例如：身份证】

代码：

-- 增加主键

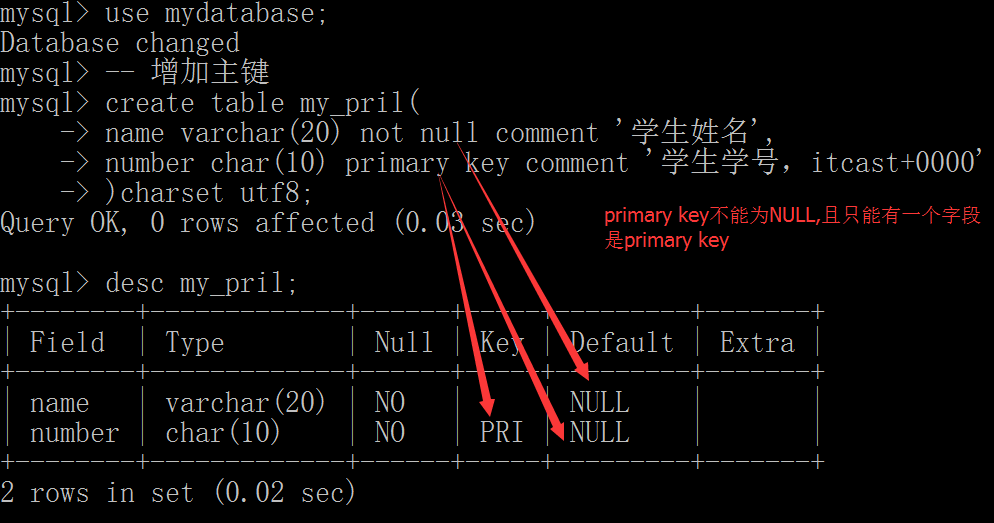
**create table my\_pril(**

**name varchar(20) not null comment '学生姓名',**

**number char(10) primary key comment '学生学号，itcast+0000'**

**)charset utf8;**

效果

：

优点：非常直接；缺点：只能使用一个字段作为主键

方案2：在创建表的时候，在所有字段之后，使用primary key（主键列表）来创建主键（如果有多个字段作为主键，可以为复合主键）

代码：

##### -- 复合主键

create table my\_pri2(

number char(10) comment'学号：itcast+0000',

course char(8)  comment'课程代码：3901+0000',

score tinyint unsigned default 60 comment'成绩',

-- 增加主键

primary key(number,course)

)charset utf8;

效果：

方案3：当表已经创建好了之后，再次额外追加主键。可以通过修改字段属性，也可以直接追加：alter table 表名 add primary key（字段列表）;

代码：

##### -- 追加主键

-- 创建没有主键的表

create table my\_pri3(

course char(8) not null comment'课程编号：3901+0000',

name varchar(10) not null comment'课程名字'

)charset utf8;

-- 查看表

desc my\_pri3;

-- 追加主键

alter table my\_pri3 add primary key(course);

-- 查看表

desc my\_pri3;

效果：

#### 1.1.2 主键约束

主键对应的字段中的数据不允许重复。一旦重复，则数据插入失败。

代码：

-- 查看表字段

desc my\_pri2;

-- 向my\_pri2表插入数据

insert into my\_pri2 values

('itcast0001','68010001'80),

('itcast0001','68010002',90),

('itcast0001','68010002',80);

效果：

#### 1.1.3 主键更新&删除主键

没有办法更新主键，主键必须先删除，才能增加。

Alter table 表名 drop primary key;

1.1.3 主键分类

在实际创建表的过程中，很少使用真实的业务数据作为主键字段（业务主键，如学号，课程号）。大部分的时候使用逻辑性的字段（字段没有业务含义，值是什么都没有关系），将这种字段主键称为逻辑主键。

代码：

-- 使用逻辑型字段作为主键

create table my\_student(

Id int primary key auto\_increment comment'逻辑主键：自增长',

Number char(10) not null comment'学号',

name varchar(10) not null

)charset utf8;

-- 查看表

desc my\_student;

效果：

### 1.2 自动增长

（1）当相应字段为默认值或Null，自增长会自动被系统触发。系统会从当前字段中已有的最大值再进行+1操作，得到一个新的不同的字段。

（2）自增长通常是跟主键搭配.

（3）自增长特点：auto\_increment

#### 1.2.1 新增自增长

1. 任何一个字段要做自增长，本身必须是一个索引(key)。

-- 自增长

create table my\_auto(

id int auto\_increment comment'自动增长',

name varchar(10) not null

)charset utf8;

2.自增长字段必须是数字（整形）。

-- 自增长

create table my\_auto(

id varchar(1) auto\_increment comment'自动增长',

name varchar(10) not null

)charset utf8;

3.一张表最多只能有一个自增长。

#### 1.2.2 自增长的使用

（1）自增长的第一个元素是1

（2）自增长每次自增1

-- 三种方式触发自增长

insert into my\_auto (name) values('邓丽君');

insert into my\_auto values(null,'baby');

insert into my\_auto values(default,'汪涵');

-- 查看表数据

select \* from my\_auto;

#### 1.2.3 修改自增长

（1）自增长涉及到字段改变，必须先删除自增长，后增加（一张表只能有一个自增长）。修改当前自增长已经存在的值（修改的值必须比当前自增长最大值还要大）。Alter table 表名 auto\_increment = 值;

（2）思考：为什么自增长是从1开始且每次自增1？

所有系统的实现（如字符集，校对集）都是有系统内部的变量进行控制的。

查看自增长对应的变量：show variables like ‘auto\_increment%’;

#### 1.2.4 删除自增长

自增长是字段的一个属性：可以通过modify来进行修改（保证字段没有auto\_increment即可）Alter table 表名 modify 字段 类型;

alter table my\_auto modify id int primary key; -- 错误：理论上主键是单次存在的。

alter table my\_auto modify id int; -- 有主键的时候，不能再添加主键。

-- 查看表

desc my\_auto;

#### 1.2.5 唯一键（unique key）

（1）一张表往往需要多个字段具有唯一性（数据不能重复），但是一张表中只能有一个主键。唯一键可解决表中多个字段需要唯一性约束的问题。

（2）唯一键的本质和主键差不多，唯一键默认允许自动为空，而且多个为空（空字段不参与唯一性比较）。

（3）增加唯一键

方案1：在创建表的时候，字段后直接跟unique/unique key。

-- 唯一键

create table my\_unique1(

number char(10) unique comment'学号：唯一允许为空',

name varchar(20) not null

)charset utf8;

方案2：在所有的字段后增加unique key（字段列表）; 复合唯一键

-- 复合唯一键

create table my\_unique2(

number char(10) not null comment'学号：唯一允许为空',

name varchar(20) not null,

unique key(number)

)charset utf8;

方案3：在创建表后增加唯一键。

create table my\_unique3(

id int primary key auto\_increment,

number char(10) not null,

name varchar(20) not null

)charset utf8;

alter table my\_unique3 add unique key(number);

（4）唯一键的约束

唯一键与主键本质相同，唯一区别是唯一键默认允许为空，而且是多个空。

  （5）更新唯一键&删除唯一键

更新唯一键：先删除后新增（唯一键可以有多个，可以不删除）

删除唯一键：

Alter table 表名 drop index 索引名字; -- 唯一键默认使用字段名作为索引名字。

### 1.3 索引

（1）几乎所有的索引值都是建立在字段之上

（2）索引：系统根据某种算法，将已有的数据（未来可能新增的数据），单独建立一个文件，文件能实现快速的数据匹配数据，并且能快速的找到对应表中的记录。

（3）索引的意义

1）提升查询数据的效率

2）约束数据的有效性

（4）增加索引的前提条件：索引本身会产生索引文件（有时候可能比数据文件还大），会非常耗费磁盘空间。

（5）如果某个字段需要作为查询的条件经常使用，那么可以使用索引（一定会想办法增加）；

 如果某个字段需要进行数据的有效性约束，也可以使用索引（主键，唯一键）。

（6）mysql中提供了多种索引

1）主键索引：primary key

2）唯一索引：unique key

3）全文索引：fulltext index

4）普通索引：index6

全文索引：针对文字内部的关键字进行索引

全文索引的最大问题在于如何确定关键字

英文很容易：英文单词与单词之间有空格

中文很难：没有空格，而且中文可以随意组合

## 2关系

将实体与实体的关系（反应到最终数据库表的设计上来）：将关系分成三种：一对一，一对多和多对多。

### 2.1 一对一

（1）一张表的一条记录只能和另外一张表的记录 进行对应，反之亦然。

（2）学生表：姓名，性别，年龄，身高，体重，婚姻状况，籍贯，家庭住址，紧急联系人

（3）表设计成以上这种形式是符合要求的。

（4）但是姓名，年龄，身高，体重属于常用数据；婚姻，籍贯，住址和联系人属于不常用数据。如果每次查询都是查询所有数据，不常用的数据就会影响效率。

（5）解决方案：将常用和不常用的信息分离存储，分成两张表。

（6）不常用的信息表需保证不常用信息与常用信息一定能够对手，找到一个具有唯一性的字段来共同连接两张表

（7）一个常用表中的一条记录，永远只能在一张不常用表中匹配一条记录；反过来，一张不常用表中的一条记录在常用表中只能匹配一天记录也只能匹配一条记录，即一对一关系。

### 2.2 一对多

表A中有一条记录可以对应表B中的多条记录；但是表B中的一条记录只能对应表A中的一条记录。

母亲与孩子的关系：母亲，孩子两个实体

一个妈妈可在孩子表中找到多条记录；但是一个孩子只能找到一个妈妈的记录（典型一对多的关系）。

以上设计解决了实体设计表问题，但是没有解决关系问题。孩子找不出妈，妈也找不到孩子。

解决方案：在某一张表中增加一个字段，使其能找到另外一张表的记录。应该在孩子表中增加一个字段指向妈妈表。因为孩子表的记录只能匹配到一条妈妈表的记录。

### 2.3 多对多

表A一条记录对应表B中的多条记录；同样，表B中的一条记录对应表A中的多条记录。

教师教学：老师和学生

以上设计方案：实现了实体的设计，但是没有维护实体的关系。

一个老师教过多个学生；一个学生有多个老师。

解决方案：增加一张新表

中间关系表：老师月学生的关系

中间表与老师表形成了一对多的关系。而中间表示多表，维护了能够唯一找到一表的关系。

学生照老师：找出学生id -> 中间表寻找匹配记录（多条）-> 老师表匹配（一条）

老师找学生：找出老师id -> 中间表寻找匹配记录（多条）-> 学生表匹配（一条）

## 3. 范式

范式是一种离散化的数学知识，是为了解决一种数据存储与优化的问题。保存数据的存储之后，凡是能够通过关系寻找出来的数据，坚决不再重复存储。终极目标是减少数据冗余。

凡是是一种分层结构的规范，分为六层。每一层都比上一层更加严格，若要满足下一层范式，前提是满足上一层范式。

六层范式：1NF,2NF,3NF.....4NF。1NF是最低层，要求最低。6NF是最高层，最严格。

Mysql属于关系型数据库，有空间浪费。故气致力于节省空间，与范式不谋而合。在设计数据库的时候，会利用范式指导设计。

但是数据库不但是解决空间问题，还要保持效率问题。范式值解决空间问题，所有数据库的设计不能完全按照范式要求实现。一般情况下，只有前三种范式需要满足。

范式在数据库的设计中有指导意义，但是不是强制规范。

### 3.1 INF

第一范式：在设计储存数据的时候，如果存储的表中在取出来使用前还需要额外的处理（字段拆分），则该表的设计不满足第一范式。第一范式要求字段的数据具有原子性（不可再分）。

如果需求是将数据查出来后，要求显示一个老师从什么时候开始上课，到什么结课。需要将代课时间进行拆分，故不符合1NF，数据不具有原子性，可以再拆分。

解决方案：将代课时间拆分成两个字段就可解决问题。

### 3.2 2NF

第二范式：在数据表设计过程中，如果有复合主键（多字段主键），且表中有字段并不是由整个主键来确定，而是依赖主键中的某个字段（主键部分）。存在字段依赖主键的部分问题

以上表中：因为讲师没有办法作为主键，需要结合班级才可以作为主键（复合主键）。代课时间，开始和结束都依赖主键（讲师和班级）；性别不依赖班级，只依赖讲师；教师只依赖班级，依赖讲师。故不符合第二范式。

解决方案1：可以将性别和讲师单独成表，班级与教师单独成表。

解决方案2：取消复合主键，使用逻辑主键。

ID  =  讲师 + 班级

### 3.3 3NF

第三范式：理论上讲，一张表中的所有字段都应该直接依赖主键（逻辑主键除外）。人员工艺表设计中存在一个字段，并不直接依赖主键，而是通过某个非主键字段依赖，从而实现依赖主键。这种依赖非主键字段的依赖关系称之为传递依赖，第三范式就是要解决传递依赖。

讲师代课表

以上设计方案中，性别依赖讲师存在，讲师依赖主键。教师依赖班级，班级依赖主键。性别和班级都存在传递依赖。

解决方案：将存在传递依赖的字段，以及依赖字段本身单独取出来形成一个单独的表。

讲师代课表

讲师表                     班级表

讲师表：ID等价于讲师    班级表：ID等价于班级

讲师表和班级表的真正意义上的主键分别是讲师和班级，id只是逻辑主键，为的是解决教师和班级可能出现同名的问题，故不存在传递依赖。

### 3.4 逆规范化：

磁盘利用率与效率的对抗。

## 4. 数据高级操作

 基本语法

Insert into 表名 [(字段列表)]  values(值列表);

### 4.1 主键冲突

当主键冲突时，可以选择性进行处理：更新和替换

（1）主键冲突：更新操作

Insert into 表名 [(字段列表)]  values(值列表) on duplicate key update 字段=新值;

（2）主键冲突：替换

Replace intto 表名 values(值列表);

### 4.2 蠕虫复制

（1）从已有的数据中获取数据，然后将数据进行新增操作（数据成倍增加）

表创建高级操作：从已有表创建新表（复制表结构）。

create table my\_copy like my\_student;

（2）蠕虫复制：先查出数据，然后将查出的数据新增一遍。

Insert into 表名 [(字段列表)] select 字段列表/\* from 数据表名;

（3）蠕虫复制意义：

1.从已有的表拷贝数据到新表

2.可以迅速让表中的数据膨胀到一定的数量级：测试表的压力以及效率

### 4.3 更新数据

基本语法：update 表名 set 字段 = 值[where 条件];

高级新增语法：

update 表名 set 字段 = 值[where 条件] [limit 更新数量];

### 4.4 删除数据

(1)Delete from table 表名 [where 条件] [limit 删除数量];

(2)数据的删除是不会改变表的结构，只能删除表后重建表。

Truncate 表名;  -- 先删除表，后新增表

### 4.5 查询数据

（1）基本语法：

Select \*/字段列表 from 表名[where 条件];

（2）完整语法：

Select[select选项] 字段列表[字段别名]/\* from 数据源 [where 条件子句][group by子句][having 子句][order by子句][limit 子句];

（3）select选项：对查出的结果的处理方式

All：默认的，保留所有的结果

Distinct:去重，将查出结果重复部分去掉(所有字段相同,才叫重复)

（4）字段别名： 字段 [as] 别名

定义：当数据进行查询出来的时候，有时候名字并不一定就满足需求（多表查询的时候，会有同名字段）需要对字段名进行重命名：别名

select Id,

Number as 学号,

name as 姓名

from my\_student;

4.6 数据源

定义：数据来源，关系型数据库的来源都是数据表。本质上只要保证数据类似二维表，最终都可以作为数据源。

数据源分为多种：单表数据源，多表数据源，子查询语句

（1）单表数据源：select \* from 表名;

（2）多表数据源：select \* from 表名1,表名2.....;



笛卡尔积没有意义，应尽量避免。

（3）[子查询](#_3.子查询)：数据的来源是一条查询语句（查询语句的结果是二维表）

Select \* from（select 语句）as 表名;

多表、不是表都可以（保证输出是表）

### 4.7 where子句

（1）用来判断数据，筛选数据

（2）Where子句返回结果：0或者1,0代表false，1代表true

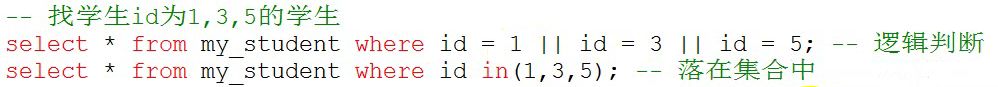
（3）判断条件：

比较运算符：>，<，>=，<=，!=，=（更接近于比较，而不是赋值），<>，like（模糊），between and（区间），in/not in（多个数据）

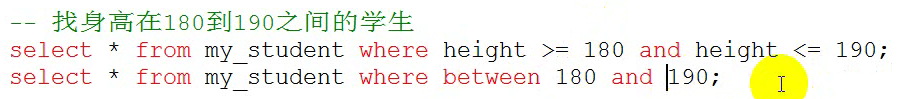
逻辑运算符：&&（and），||（or），！（not）

（4）where原理：where是唯一一个直接从磁盘获取数据的时候就开始判断的条件。从磁盘取出一条记录，开始进行where判断，判断的结果如果成立就保存到内存；如果失败就直接放弃。





（5） Between本身是闭区间：左边的值必须小于或者等于右边的值。



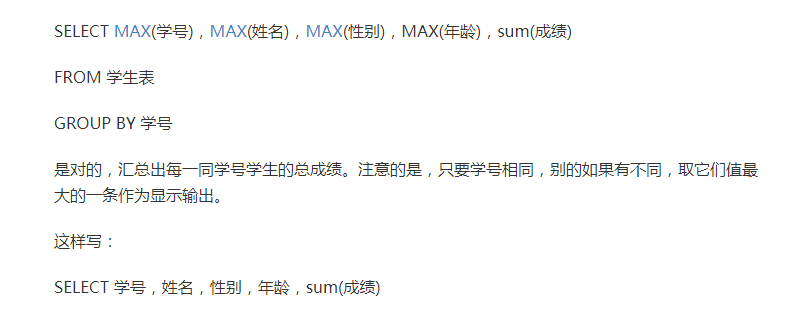
随机取数：rand（）\* 20 +20

20~40随机取数

### 4.8 group by

（1）Group by是分组的意思，根据某个字段进行分组（相同的放在一组，不同的分到不同的组）。

（2）基本语法：(select \* from 表名)group by 字段名;



（3）分组的意义是为了统计数据（按组统计：按分组字段进行数据统计）

（4）Sql提供了一系列统计函数(聚合函数)

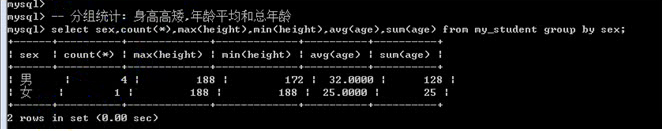
1）count()：统计分组后的记录数（每组分别有多少记录）

2）Max()：统计每组中的最大值

3）Min()：统计最小值

4）Avg()：统计平均值

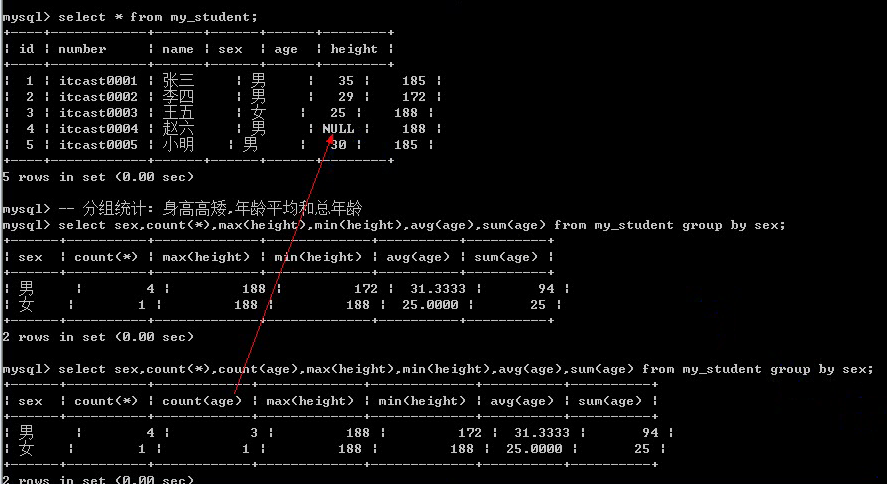
5）Sum()：统计和



细节：

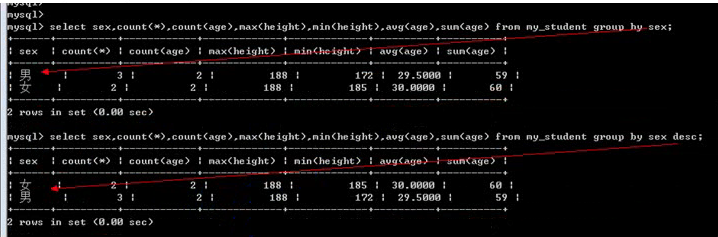
（5）Count函数两种参数：① \*代表统计记录

② 字段名代码统计对应的字段（不统计Null）



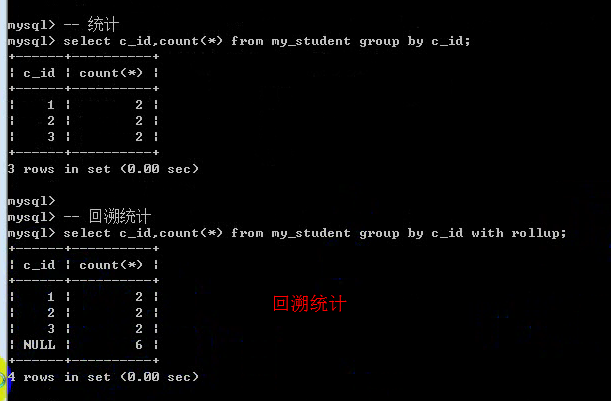
（6）分组会自动排序：根据分组字段，默认升序

Group by 字段 [asc|desc];  -- 对分组的结果进行排序



（7）多字段分组：先根据一个字段分组，然后对分组后的结果再次按照其他字段进行分组。

（8） Group\_concat(字段)：对分组的结果中的某个字段进行字符串连接（保留该组所有的某个字段）。

（9）

### 4.9 having子句

定义：与where子句一样：进行条件判断的。

（1）Where是针对磁盘数据进行判断，有效数据进入内存后，进行分组操作，分组的结果需要having来处理。

（2）Having能做where能做的几乎所有事情，但是where却不能做having能做的很多事情。

1. 分组统计的结果或者说统计函数都只能在having中使用
2. Having能够使用字段别名，而where不能。 Where是从磁盘取数据，而别名是在字段进入内存后才会产生的。

### 4.10 Order by子句

（1）Orderby：排序，根据某个字段进行升序或降序排序，依赖校对集。

（2）使用基本语法：order by 字段名[asc/desc]

-- asc默认的，升序

（3）排序可以进行多字段排序：先根据某个字段进行排序，然后按照排序号的内部，再根据某个数据进行排序。

### 4.11 limit子句

（1）Limit子句是一种限制结果的语句：限制数量

（2）Limit两种使用方式：

1）方案1：只用来限制长度（数据量）：limit数据量

Select \* from my\_student limit 2;

2）限制起始位置，限制数量：limit起始位置，长度

Select \* from my\_student 0,2 -- 从0开始，寻找两个

limit方案2主要是用来实现数据的分页，为用户节省时间，提高服务器的响应效率，减少资源的浪费。

对于用户来讲，可以点击分页按钮：1,2,3,4

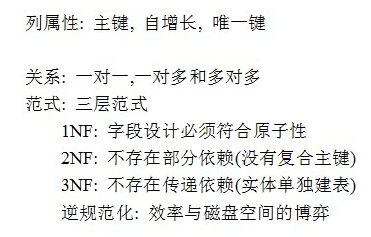
对服务器来讲：根据用户选择的页码来获取不同的数据：

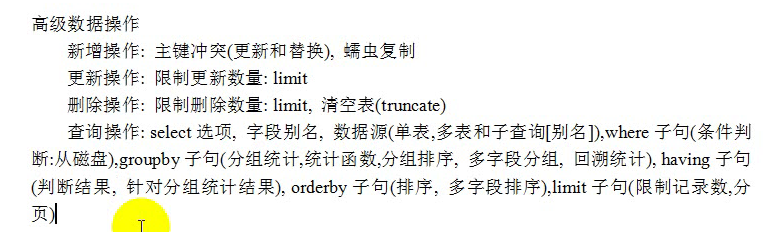
limit  offset，length

Length：每页显示的数据量，基本不变

Offset = （页码-1）\*每页显示量

## 5.回顾





# Mysql 4

## 1. 连接查询

（1）连接查询：将多张表（大于2张）进行记录的连接（按照某个指定的条件进行数据拼接）。最终结果是记录数有可能变化，字段数一定会增加（至少两张表合并）。

（2）连接查询的意义：在用户查看数据的时候，需要显示的数据来自多张表。

（3）连接查询（join）使用方式：

左表 join 右表

左表：在join关键字左边的表

右表：在join关键字右边的表

### 1.1 连接查询分类

Sql将连接查询分成四类：**内连接，外连接，自然连接和交叉连接**

#### 1.1.1交叉连接

（1）交叉连接：cross join，从一张表中循环取出每一条记录，每天记录都去另外一张表进行匹配。匹配一定保留（没有条件匹配），而连接本身字段就会增加，最终形成的结果叫：笛卡尔积。

（2）基本语法：左表 cross join 右表 == from 左表,右表

（3）笛卡尔积没有意义，应尽量避免。交叉连接存在的价值是保证连接结构的完整性。

#### 1.1.2 内连接

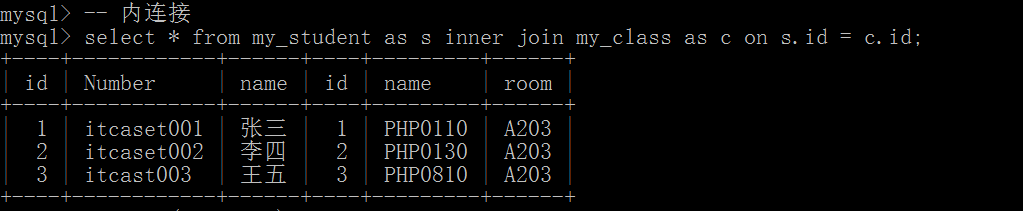
（1）内连接（[inner]  join）:从左表中取出每一条记录，去右表中与所有的记录进行匹配，匹配必须是某个条件在左表与右表中相同，最终才会保留结果，否则不保留。

（2）基本语法：左表 [inner]  join 右表 on 左表**.**字段=右表**.**字段;

On表示连接条件，条件字段就是代表相同的业务含义

-- 内连接

select \* from my\_student as s inner join my\_class as c on s.id = c.id;



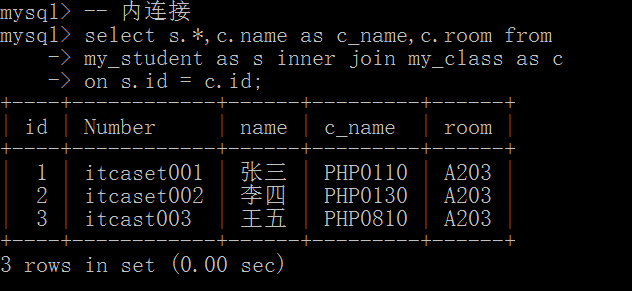
（3）字段别名以及表别名的使用：在查询数据的时候，不同表有同名字段，这个时候需要加上表名才能区分，而表名太长，通常使用别名。

-- 内连接+别名

select **s.\*,c.name as c\_name,c.room** from

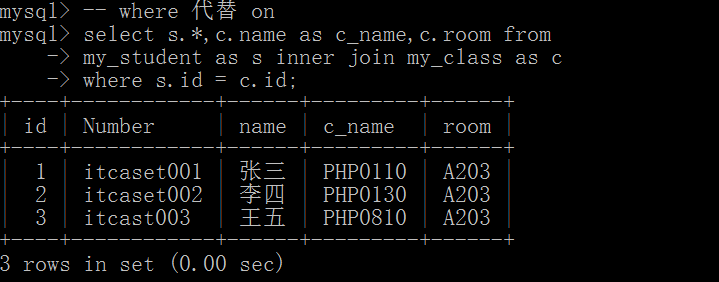
my\_student as s inner join my\_class as c

on s.id = c.id;



（4）内连接可以没有连接条件：没有on之后的内容，这个时候系统会保留所有结果（笛卡尔积 ）

（5）内连接可以使用where代替on关键字（**where没有on效率高**）。



#### 1.1.3 外连接

外连接：outer join，以某张表为主，取出其所有记录，然后每条与另外一张表进行连接。

不会管等不能匹配上条件，最终都会保留：能匹配，正确保留；不能匹配，其他表的字段都置空NULL

外连接分为两种：是以某张表为主：有主表

Left join：左外链接（左连接），以左边为主表；

Right join：右外连接（右连接），以右表为主表

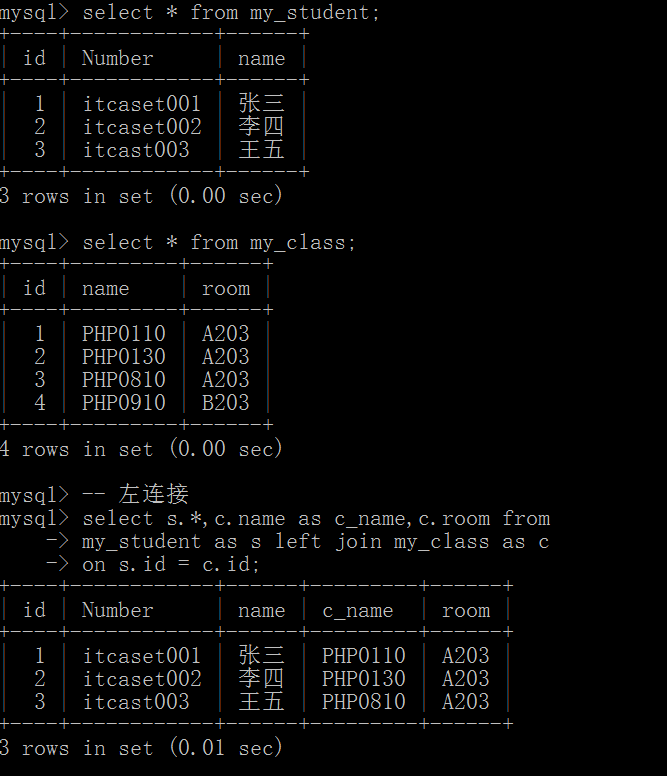
基本语法：左表 left/right join 右表 on 左表**.**字段 = 右表**.**字段；

**1）左连接**

**select s.\*,c.name as c\_name,c.room from**

**my\_student as s left join my\_class as c  --  my\_student为主表**

**on s.id = c.id;**



**（2）右连接**

**select s.\*,c.name as c\_name,c.room from**

**my\_student as s right join my\_class as c  -- my\_class 为主表**

**on s.id = c.id;**

虽然左连接和右连接有主表差异，但是显示的结果：左表的数据在左表，右表的数据在右边，左连接和右连接可以互换。

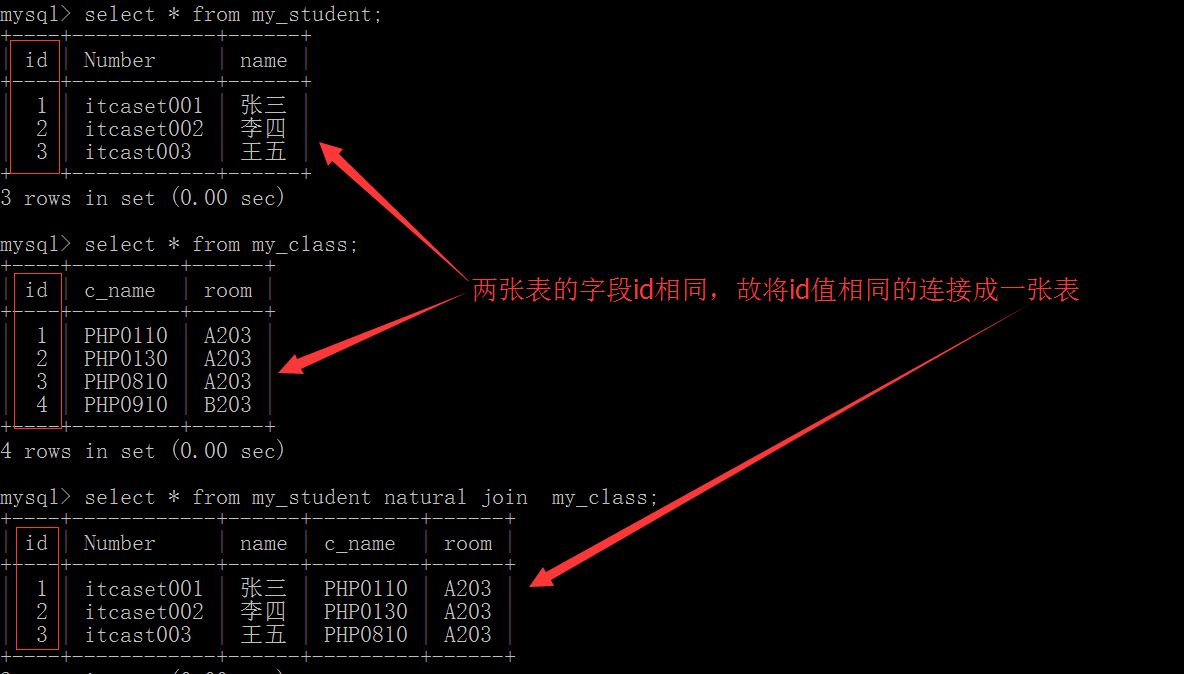
#### 1.1.4 自然连接

（natural join）

（1）自然连接，就是自动匹配连接条件：系统以字段名字作为匹配条件（同名字段作为条件，多个同名字段都作为条件）。

（2）自然连接分为**自然内连接**和**自然外连接**。

自然内连接： 左表 natural join 右表



自然连接自动使用同名字段作为连接条件，连接之后会合并同名字段。

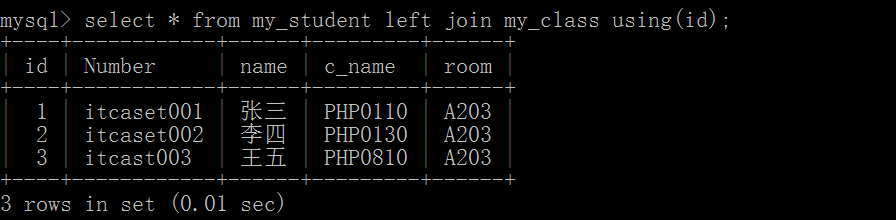
（3）自然外连接：**左表 natural left/right join 右表**

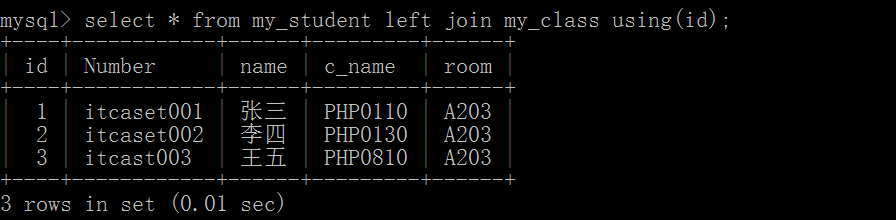
（4）其实内连接和外连接都可以模拟自然连接，使用同名字段，合并字段。

左表 left/right/inner join 右表 using（字段名）; -- 使用同名字段作为连接条件，自动合并条件。

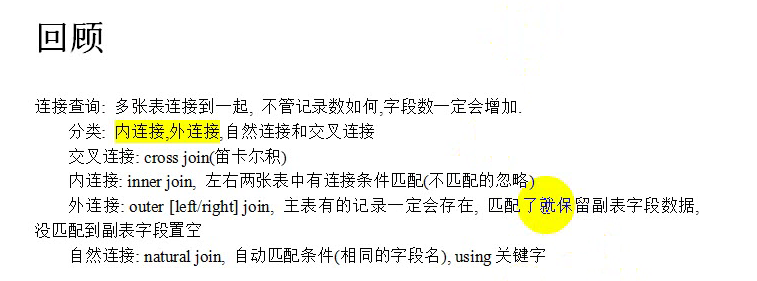
（5）外连接模拟自然连接：using

select \* from my\_student left join my\_class using(id);





## 2.回顾



# Mysql 5

# ****1.外键****

（1）外键（foreign key）：外面的键（键不在自己表中），如果一张表中有一个字段（非主键）指向另外一张表的主键，那么将该字段称为外键。

## ****1.1增加外键****

外键可以在创建表的时候或者创建表之后增加（但是要考虑数据问题）。

一张表可以有多个外键

**1）创建表的时候增加外键**：在所有的表字段语句之后，使用foreign key（外键字段） references外部表（主键字段）

--创建外键

create table my\_foreign1(

id int primary key auto\_increment,

name varchar(20) not null comment'学生姓名',

c\_id int comment '班级id',

--增加外键

foreign key(c\_id) references my\_class(id)

)charset utf8;



**2）在新增表之后增加外键**，修改表结构。

**Alter table表名 add [constraint外键名字] foreign key（外键字段）references   父表（主键字段）**

**[constraint外键名字]：可以自己命名**

--增加外键

alter table my\_foreign2 add

--指定外键名

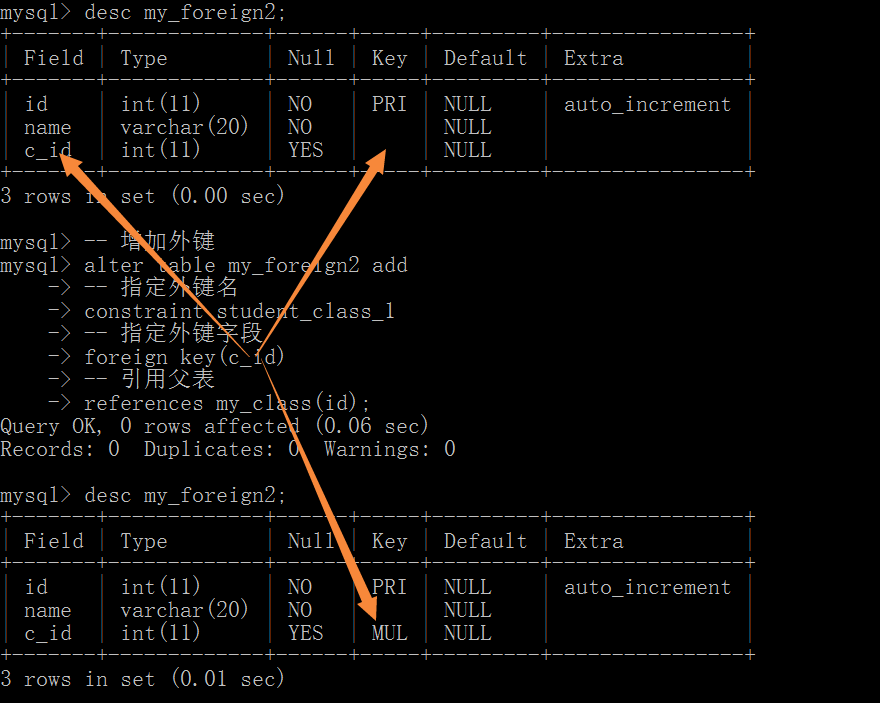
constraint student\_class\_1

--指定外键字段

foreign key(c\_id)

--引用父表

references my\_class(id);



## ****1.2修改外键&删除外键****

外键不可修改，只能先删除后新增。

删除外键的语法：

**Alter table表名 drop foreign key  外键名;**

--查看表结构

desc my\_foreign1;

--删除外键

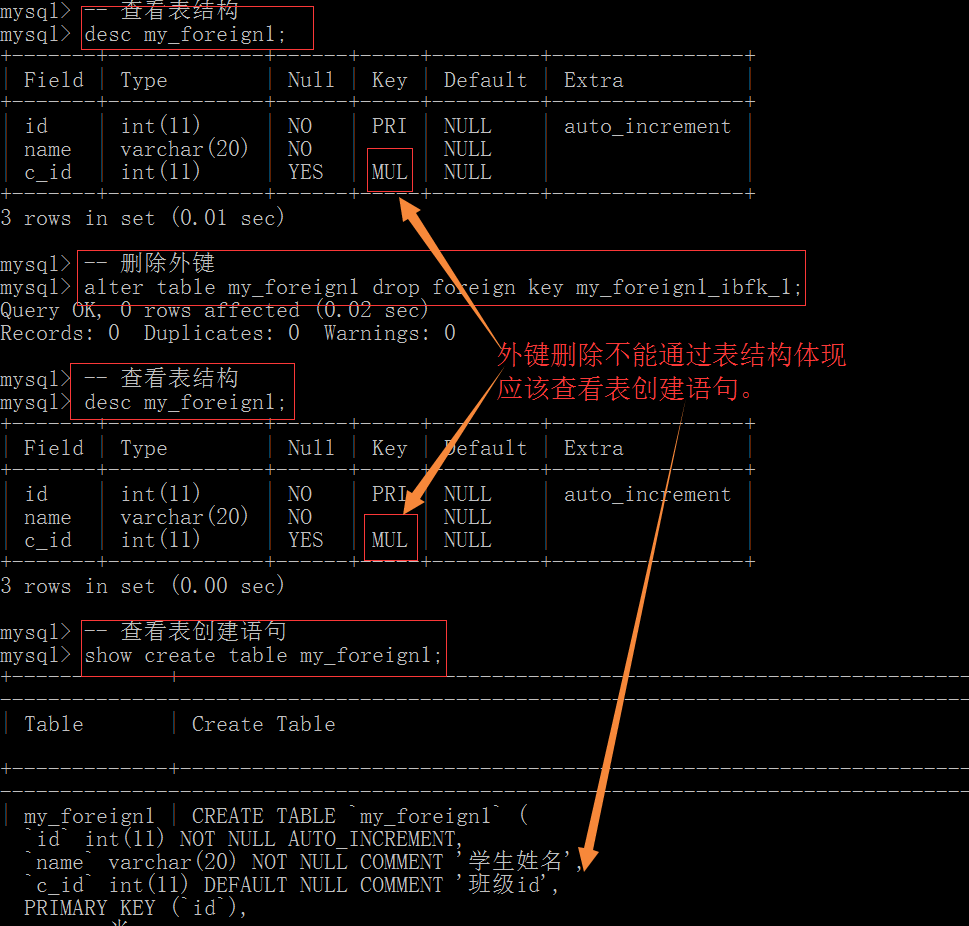
alter table my\_foreign1 drop foreign key my\_foreign1\_ibfk\_1;

--查看表结构

desc my\_foreign1;

--查看表创建语句

show create table my\_foreign1;

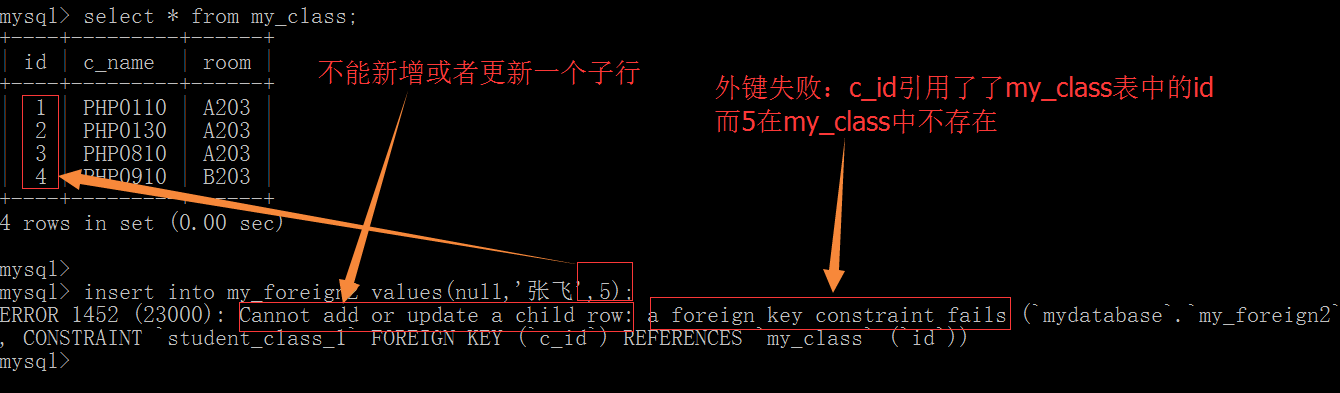


## 1.3外键作用

外键默认作用：（1）一个对父表约束（2）一个对字表约束（外键字段所在的表）

（1）对字表的约束：子表数据进行写操作（增和改）的时候，如果对应的外键字段在父表中找不到对应的匹配，那么操作会失败（约束子表数据操作）。

insert into my\_foreign2 values(null,'张飞',5);



（2）对父表的约束：父表数据进行写操作（删除和修改），如果对应的主键在子表中已经被数据引用，那就不允许操作。

## 1.4外键条件

（1）外键要存在，首先必须保证表的存储引擎是innodb（默认存储引擎）

如果不是innodb存储引擎，那么外键可以创建成功，但是没有约束效果。

（2）外键字段的字段类型（列类型）与父表的主键类型完全一致。

（3）一张表中外键名字不能重复。

（4）增加外键的字段（数据已经存在），必须保证数据

## 1.5外键约束

（1）所谓外键约束：就是指外键的作用

（2）之前所讲的外键作用：是默认的作用，其实可以通过对外键的需求，进行定制。

（3）外键约束有三种约束模式，都是针对父表的约束。

1. district：严格模式（默认的），父表不能删除或者更新一个已经被字表数据 引用的记录

2）cascade：级联模式，父表的操作对字表关联的数据也跟着被删除。

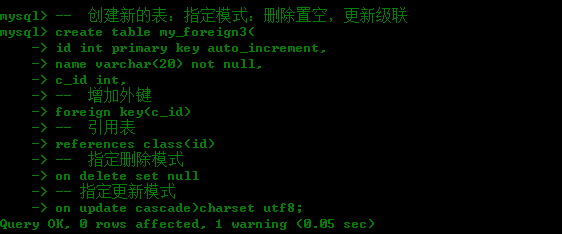
3）setnull：置空模式，父表的操作之后，字表对应的数据（外键字段）被置空。

通常的一个合理做法（约束模式）：删除的时候字表置空，更新的时候子表级联操作。

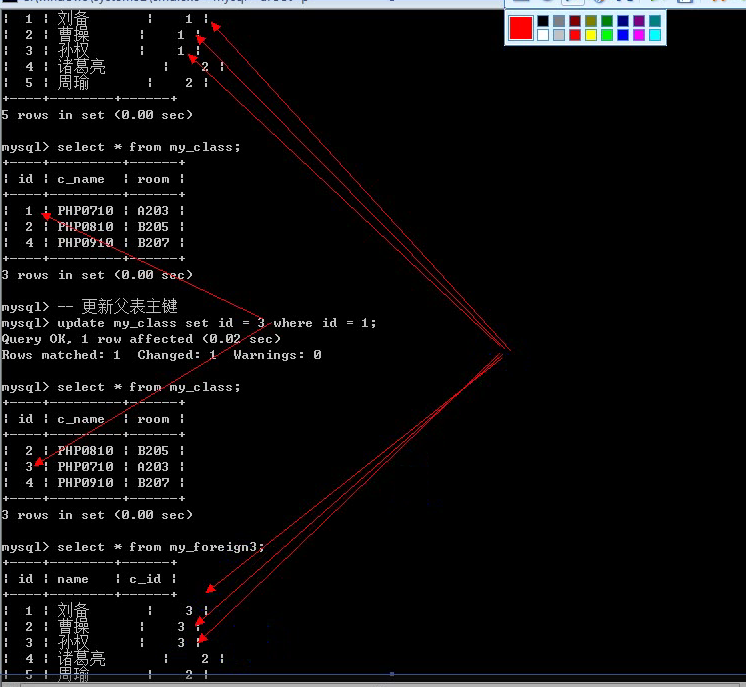
指定模式的语法：

Foreign key(外键字段) references父表（主键字段）on delete模式update 模式

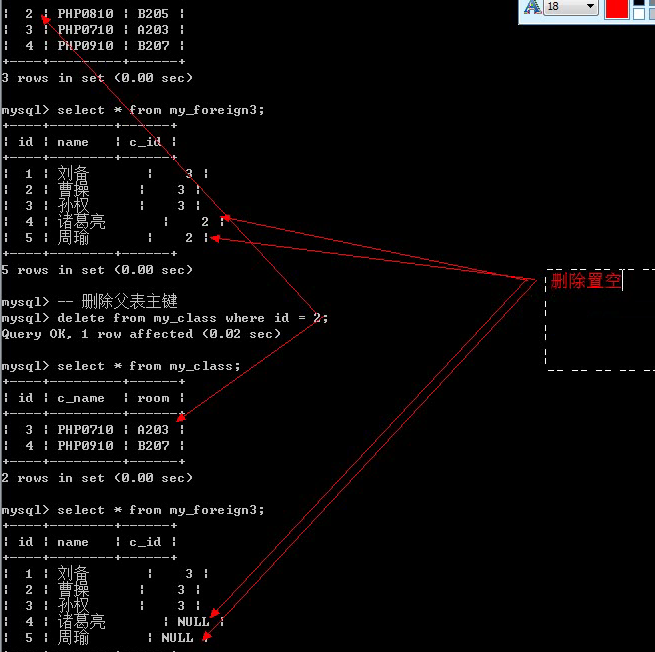
Foreign key(外键字段) references父表（主键字段）on delete set null on update cascade;



级联更新：



删除置空：



删除置空的前提条件：外键字段允许为空（如果不满足条件，外键无法创建）

# ****2联合查询****

（1）联合查询：将多次查询（多条select语句），在记录上进行拼接（字段不会增加）。

## ****2.1基本语法****

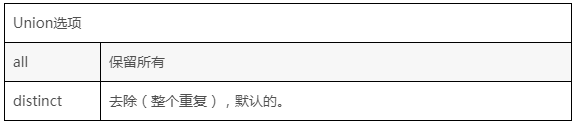
（1）多条select语句构成，每一条select语句获取的字段数必须严格一致（但是字段类型无关）

**Select语句1**

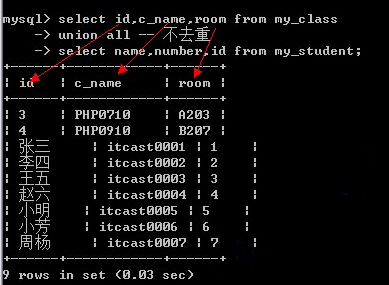
**Union[union选项]**

**Select语句2 .........**

**Union选项：与select选项一样有两个**



联合查询只要求字段一样，跟数据类型无关。



## ****2.2意义****

联合查询的意义分为两种：

**（1）查询同一张表，但是需求不同：**如查询学生信息，男生身高升序，女生身高降序。

**（2）多表查询**：多张表的结构时完全一样的，保存的数据（结构）也是一样的。

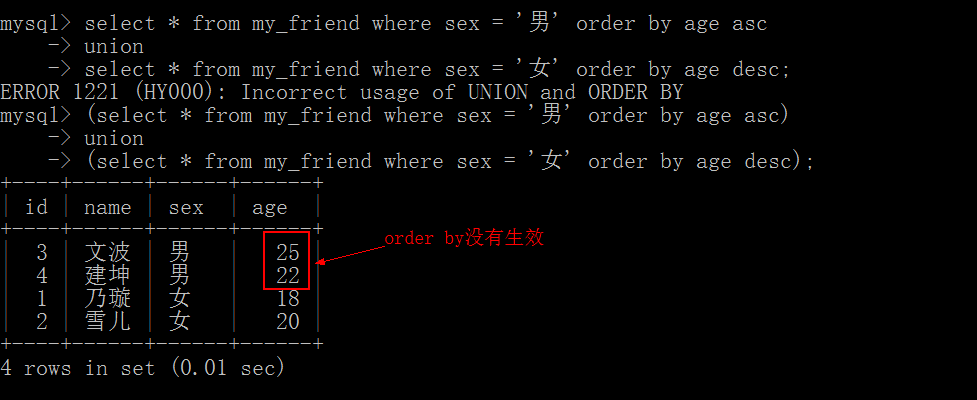
## 2.3 order by使用

（1）在联合查询中：order by不能直接使用，需要对查询语句使用括号才行。

(select \* from my\_friend where sex = '男' order by age asc)

union

(select \* from my\_friend where sex = '女' order by age desc);

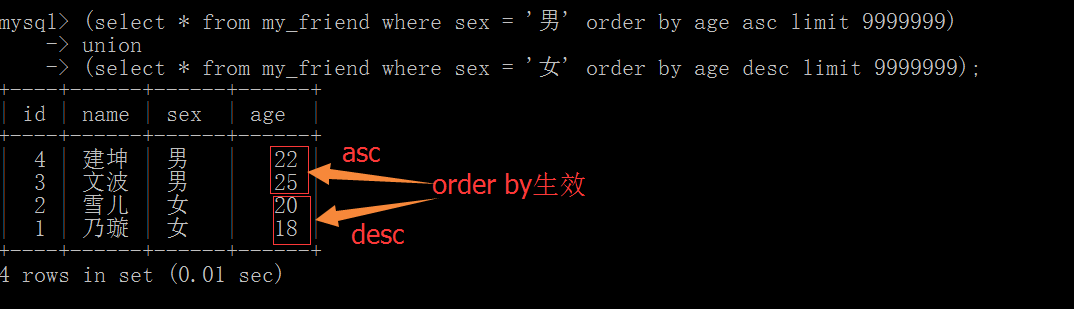


（2）若要order by生效，必须搭配limit。Limit使用限定的最大数即可。

(select \* from my\_friend where sex = '男' order by age asc limit 9999999)

union

(select \* from my\_friend where sex = '女' order by age desc limit 9999999);



# ****3.子查询****

（1）子查询：sub query，查询时在某个查询结果之上进行（一条select语句内包含另外一条select语句）。

## ****3.1子查询分类****

（1）**按位置分类**：子查询（select语句）在外部查询（select语句）中出现的位置。

|  |  |
| --- | --- |
| From子查询 | 子查询跟在from之后 |
| Where子查询 | 子查询出现在where条件中 |
| Exists子查询 | 子查询出现在exists里面 |

（2）**按结果分类**：根据子查询得到的数据进行分类（理论上讲任何一个查询得到的结果都可以理解为二维表）。

|  |  |
| --- | --- |
| 标量子查询 | 子查询得到的结果是一行一列。（在where之后） |
| 列子查询 | 子查询得到的结果是一列多行。（在where之后） |
| 行子查询 | 子查询得到的结果是一行多列。（在where之后） |
| 表子查询 | 子查询得到的结果是多行多列。（在from之后） |

## 3.2标量子查询

--一行一列

需求：知道班级名字为PHP0810后，获取该表所有学生。

分析：

（1）确定数据源：获取符合条件的所有学生

Select \* from my\_student wherec\_id = ?;

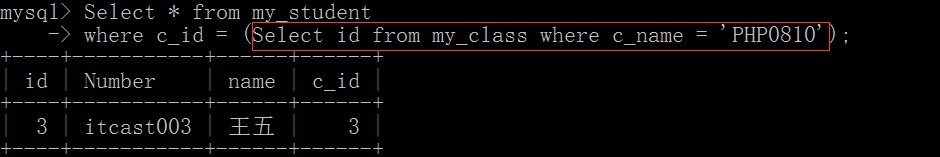
（2）获取PHP0810班级的id：可以通过班级名字确定

Select id from my\_class where c\_name =‘PHP0810’; ----🡪替换掉上边的问号

**代码：**

**Select \* from my\_student**

**where c\_id = (Select id from my\_class where c\_name = 'PHP0810');**



## ****3.3列子查询****

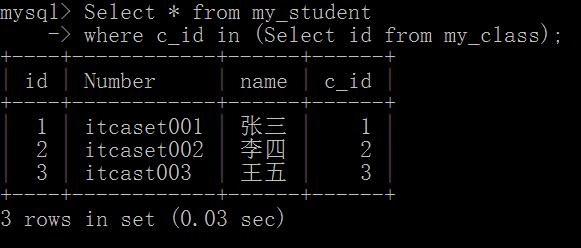
需求：查询所有在读班级的学生（班级表中存在的班级）

（1）确定数据源：学生

Select \* from my\_student where **c\_id in(?)**;

（2）确定有效班级的id：所有班级的id

Select id from my\_class;



（3）列子查询返回的结果比较：一列多行，需要使用in作为条件匹配。其中在mysql中还有几个类似的条件：all，some，any

1）=any等价于in（其中一个满足即可）。

2）Any与some是一样的。

3）=all即为全部。

陈述语句代码：

Select \* from my\_studentwhere c\_id=any(Select id from my\_class);

Select \* from my\_studentwhere c\_id=some(Select id from my\_class);

Select \* from my\_studentwhere c\_id=all(Select id from my\_class);

否定语句代码：

Select \* from my\_studentwhere c\_id!=any(Select id from my\_class); --  不等于其中任意结果，所有结果（NULL除外）

Select \* from my\_studentwhere c\_id!=some(Select id from my\_class);--  所有结果（NULL除外）

Select \* from my\_studentwhere c\_id !=all(Select id from my\_class); -- my\_class表中不存在的id，包括null。

## 3.4行子查询

行子查询：返回的结果是多行多列（一行多列）

需求:要求查询整个学生中，年龄最大且身高是最高的学生。

（1）确定数据源：

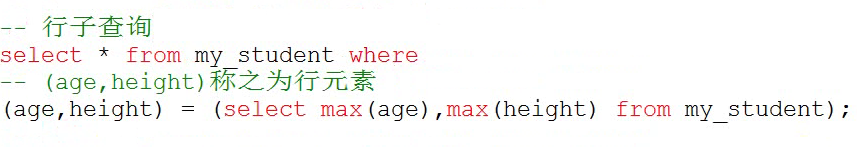
Select \* from my\_student where age =? And heigh = ?;

（2）确定最大的年龄和最高的身高

Select max(age) from my\_student;

Select max(height) from my\_student;

（3）行子查询：需要构造行元素，行元素由多个字段构成。



## ****3.5表子查询****

表子查询：子查询返回的结果是多行多列的二维表，子查询返回的结果是当做二维表来使用的。

需求：找出每一个班最高的一个学生。

（1）确定数据源

select \* from my\_studnet oder by height desc;

（2）从每个班选出第一个学生

Select \* from my\_student  group by c\_id;

代码：

Select \* from (select \* from my\_student order by height desc) as student group by c\_id;

先执行括号里面的，括号里面执行order by查询，查询完之后生成一个虚拟表**.**

## ****6 exists子查询****

Exists：是否存在的意思，exists子查询就是用来判断某些条件是否满足（跨表），exists是接在where之后，exists返回的结果只有0和1。

需求：查询所有学生：前提条件是班级存在

1. 确定数据源

Select \* from my\_student where ?;

1. 确定条件是否满足

Exists(Select \* from class); -- 是否存在

# ****4.视图（view）****

视图是一种有结构（有行有列）但是没有结果（结构中不真实存放数据）的虚拟表，虚拟表的结构来源不是自己定义，而是从对应的基表中产生（视图的数据来源）。

## ****4.1创建视图****

（1）基本语法：

**Create view视图名字 as select语句** -- select语句可以是普通查询，连接查询，联合查询或者子查询。

（2）创建单表视图：基表只有一个

创建多表视图：基表有多个（至少两个）

**代码：**

--视图：单表

create view my\_v1 as

select \* from my\_student;

create view my\_v2 as

select \* from my\_class;

--视图：多表

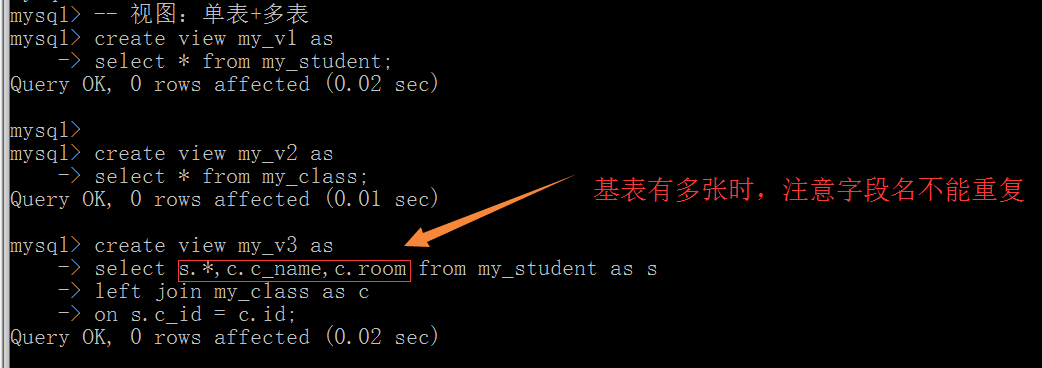
create view my\_v3 as

select s.\*,c.c\_name,c.room from my\_student as s

left join my\_class as c

on s.c\_id = c.id;

效果：



## ****4.2查看视图****

（1）视图是一张虚拟表，表的所有查看方式都适用于视图：

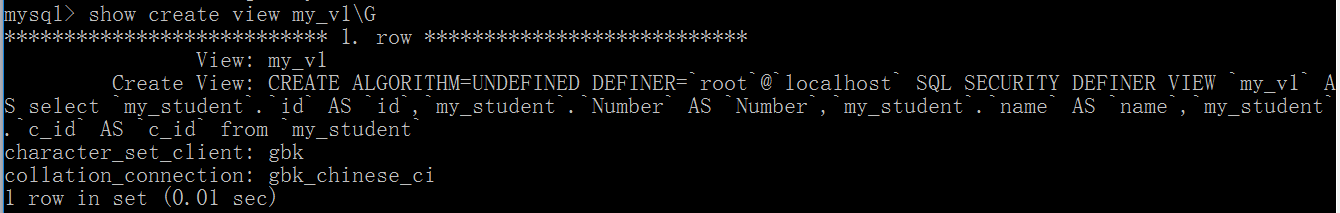
show tables;

Desc视图名;

Show create table视图名;

（2）视图比表还是有一个关键字的区别：view

查看视图的创建语句时可以使用Show create view视图名;



（3）视图一旦创建：系统会在视图对应的数据库文件夹下创建一个对应的结构文件：frm文件。

## ****4.3使用视图****

使用视图主要是为了查询：将视图当做表一样查询即可

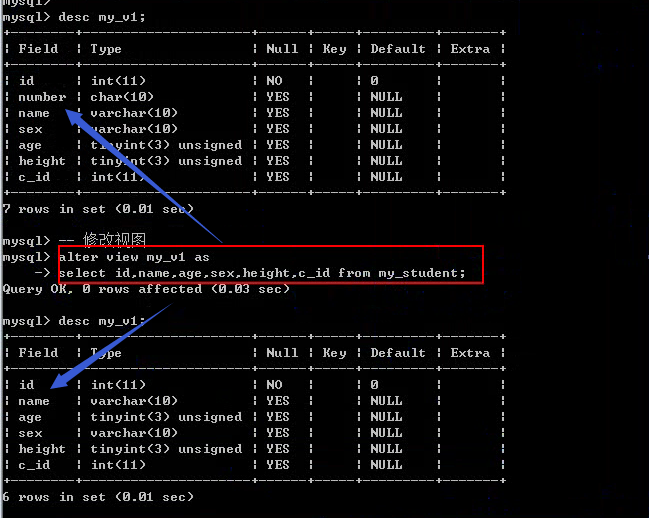
视图的执行其实本质就是执行封装的select语句。

## ****4.4修改视图****

视图本身不可修改，但是视图来源来源是可以修改的。

修改视图：修改试图本身的来源语句（select语句）

**Alter view视图名 as 新的select语句;**



## ****4.5删除视图****

**Drop view视图名;**

## 4.6视图的意义

（1）视图可以接受sql语句，将一条复杂的查询语句是在视图进行保存，以后可以直接对视图进行操作。

（2）数据安全：视图操作主要是针对查询的，如果对视图结构进行处理（删除），不会影响基表数据（相对安全）。

（3）视图往往是在大项目中使用，而且是多系统使用。可以对外提供有用的数据，但是隐藏关键（或无用）的数据，保证数据安全。

（4）视图可以对外提供友好型：不同的视图提供不同的数据，对外专门设计。

（5）视图可以更好（容易）的进行权限控制。

## 4.7视图数据操作

视图的确可以进行数据写操作但是有很多限制

### 4.7.1新增数据

数据新增就是直接对视图进行数据新增。

（1）多表视图不能新增数据。

（2）可以向单表视图插入数据：但是视图中的字段必须包含”基表中所有不能为空（或没有默认值）的字段”。

（3）单表视图是可以向基表插入数据的。