目录

[图形界面操作 4](#_Toc14120243)

[逻辑删除 5](#_Toc14120244)

[E:\程序人生\SQL\video\MySQL 6](#_Toc14120245)

[数据库基础 6](#_Toc14120246)

[关系型数据库 6](#_Toc14120247)

[关键字说明 6](#_Toc14120248)

[SQL 7](#_Toc14120249)

[Mysql数据库 7](#_Toc14120250)

[SQL基本操作 7](#_Toc14120251)

[SQL编辑器 8](#_Toc14120252)

[库操作 8](#_Toc14120253)

[查看数据库 9](#_Toc14120254)

[更新数据库 10](#_Toc14120255)

[删除数据库 10](#_Toc14120256)

[表操作 10](#_Toc14120257)

[新增数据表 10](#_Toc14120258)

[查看数据表 11](#_Toc14120259)

[修改数据表 11](#_Toc14120260)

[修改表本身 11](#_Toc14120261)

[修改字段 12](#_Toc14120262)

[删除数据表 12](#_Toc14120263)

[数据操作 12](#_Toc14120264)

[新增数据 12](#_Toc14120265)

[查看数据 13](#_Toc14120266)

[更新数据 13](#_Toc14120267)

[删除数据 13](#_Toc14120268)

[中文数据问题 13](#_Toc14120269)

[校对集问题 16](#_Toc14120270)

[Web乱码问题 19](#_Toc14120271)

[数据类型（列类型） 19](#_Toc14120272)

[列属性 20](#_Toc14120273)

[空属性 21](#_Toc14120274)

[列描述 21](#_Toc14120275)

[默认值 22](#_Toc14120276)

[主键 23](#_Toc14120277)

[增加主键 23](#_Toc14120278)

[主键约束 25](#_Toc14120279)

[更新主键&删除主键 25](#_Toc14120280)

[主键分类 25](#_Toc14120281)

[自动增长 25](#_Toc14120282)

[新增自增长 25](#_Toc14120283)

[自增长的使用 26](#_Toc14120284)

[修改自增长 27](#_Toc14120285)

[删除自增长 28](#_Toc14120286)

[唯一键 28](#_Toc14120287)

[增加唯一键 28](#_Toc14120288)

[唯一键约束 29](#_Toc14120289)

[更新唯一键&删除唯一键 29](#_Toc14120290)

[索引 30](#_Toc14120291)

[关系 31](#_Toc14120292)

[一对一 31](#_Toc14120293)

[一对多（多对一） 31](#_Toc14120294)

[多对多 32](#_Toc14120295)

[数据高级操作 33](#_Toc14120296)

[主键冲突 33](#_Toc14120297)

[蠕虫复制 33](#_Toc14120298)

[更新数据 34](#_Toc14120299)

[删除数据 34](#_Toc14120300)

[查询数据 34](#_Toc14120301)

[Select选项 34](#_Toc14120302)

[字段别名 35](#_Toc14120303)

[数据源 35](#_Toc14120304)

[where子句 37](#_Toc14120305)

[Group by子句 37](#_Toc14120306)

[Having子句 39](#_Toc14120307)

[Order by子句 40](#_Toc14120308)

[Limit子句 40](#_Toc14120309)

[连接查询 41](#_Toc14120310)

[函数应用 41](#_Toc14120311)

[MYSQL高级 42](#_Toc14120312)

[MyISAM和InnoDB对比 42](#_Toc14120313)

[索引 42](#_Toc14120314)

E:\程序人生\Python\video\python\第4章 数据库\第1节 MySQL

简介

数据库：

功能：存储数据

特点：优化了读写，方便存储，快速查找

分布式存储：多地保存，数据更安全

数据库系统解决的问题：持久化存储、优化读写、保证数据的有效性

当前使用的数据库，主要分为两类：

文档型，如sqlite,就是一个文件，通过对文件的复制完成数据库的复制

服务型，如mysql,数据存储在一个物理文件中，但是需要使用终端以tcp/ip协议连接，进行数据库的读写操作

关系型数据库：

Oracle/Mysql/SqlServer使用了相同的设计模型（E-R设计模型），它们的区别是核心代码的实现方式不一样，即在硬盘中对数据的存储方式和读取方式不一样，但是数据结构的设计都是基于E-R结构设计

Mysql数据库是当前应用非常广泛的一款关系型数据库，免费的

官方网站：[www.mysql.com](http://www.mysql.com)

数据库排名网站：<http://db-engines.com/en/ranking>

主要操作：

数据库的操作：创建、删除

表的操作：创建、修改、删除

数据的操作：增加、修改、删除、查询，简称crud(create、read、update、delete)

E-R模型

E表示entry, 实体

R表示relationship, 关系

一个实体转换转换为数据库中的一个表

关系描述两个实体之间的对应规则，包括：

一对一

一对多

多对多

关系转换为数据库表中的一个列，在关系型数据库中一行就是一个对象

三范式

第一范式（1NF）:列不可拆分

第二范式（2NF）:唯一标识

第三范式（3NF）:引用主键

说明：后一个范式，都是在前一个范式的基础上建立的

数据完整性

一个数据库就是一个完整的业务单元，可以包含多张表，数据被存储在表中

在表中为了更加准确的存储数据，保证数据的正确有效，可以在创建表的时候，为表添加一些强制性的验证，包括数据字段的类型、约束

字段类型

在mysql中包含的数据类型很多，常用类型为：

数字：int, decimal

字符串：char, varchar, text

日期：datetime

布尔：bit

约束

主键primary key

非空not null

唯一 unique

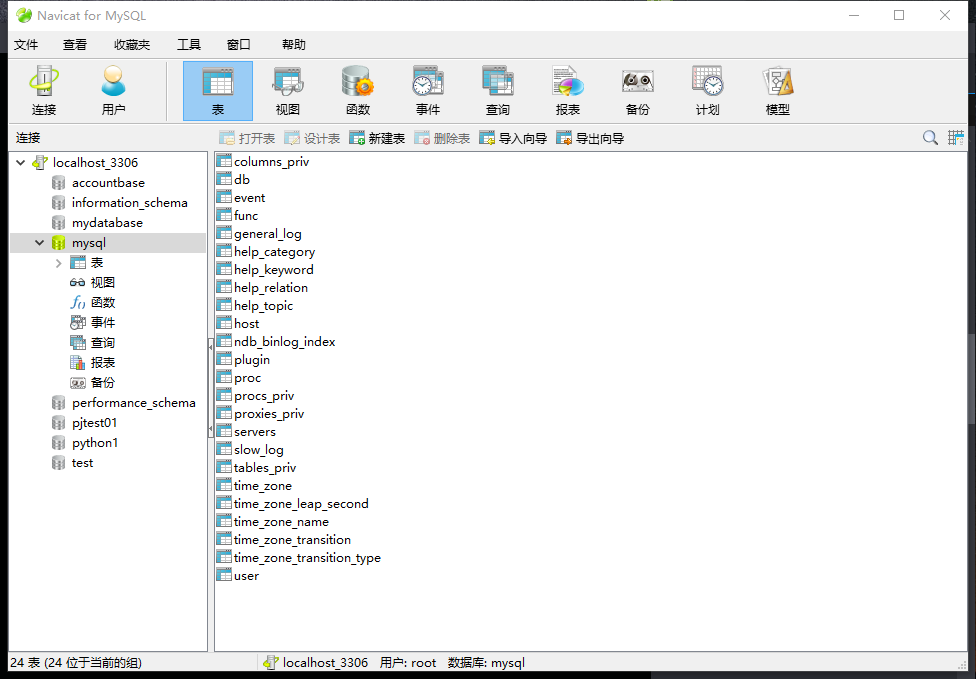
默认 default

外键 foreign key

## 图形界面操作

Navicat for MySQL软件简介

Navicat for MySQL 是一款强大的 MySQL 数据库管理和开发工具，它为专业开发者提供了一套强大的足够尖端的工具，但对于新用户仍然易于学习。Navicat For Mysql中文网站：http://www.formysql.com/  
Navicat for MySQL 基于Windows平台，为 MySQL 量身订作，提供类似于 MySQL 的用管理界面工具。此解决方案的出现，将解放 PHP、J2EE 等程序员以及数据库设计者、管理者的大脑，降低开发成本，为用户带来更高的开发效率。  
Navicat for MySQL 使用了极好的图形用户界面（GUI），可以用一种安全和更为容易的方式快速和容易地创建、组织、存取和共享信息。 用户可完全控制 MySQL 数据库和显示不同的管理资料，包括一个多功能的图形化管理用户和访问权限的管理工具，  
方便将数据从一个数据库转移到另一个数据库中（Local to Remote、Remote to Remote、Remote to Local），进行档案备份。  
Navicat for MySQL 支援 Unicode，以及本地或远程 MySQL 服务器多连线，用户可浏览数据库、建立和删除数据库、编辑数据、建立或执行 SQL queries、管理用户权限（安全设定）、将数据库备份/复原、汇入/汇出数据（支援 CSV, TXT, DBF 和 XML 档案种类）等。



## 逻辑删除

对于重要数据，并不希望物理删除，一旦删除，数据无法找回

一般对于重要数据，会设置一个isDelete的列，类型为bit,表示逻辑删除

对于大量增长的非重要数据，可以进行物理删除

数据的重要性，要根据实际开发决定

# E:\程序人生\SQL\video\MySQL

# 数据库基础

1. 什么是数据库？

数据库：database,存储数据的仓库

数据库：高效的存储和处理数据的介质（介质主要是2种：磁盘和内存）

1. 数据库的分类？

数据库基于存储介质的不同，分为2类：关系型数据库(SQL)和非关系型数据库(NoDSQL：not only SQL)

1. 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：access等

非关系型数据库：mongodb,redis(同步到磁盘)

1. 两种数据库阵营的区别？

关系型数据库：安全（保存磁盘基本不可能丢失），容易理解，比较浪费空间（二维表）

非关系型数据库：效率高（运行在内存），不安全（断电丢失）

## 关系型数据库

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（数学模型）上的数据库。

## 关键字说明

数据库：database

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=DBMS + DB

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

DBA:Database Administrator,数据库管理员

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，记录是从数据角度出发

列/字段：column/field

## SQL

SQL:Structured Query Language,结构化查询语言（数据以查询为主：99%是在进行查询操作）

SQL分为三个部分

DDL:Data Definition Language,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

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内部又单独进行了一个分类：DQL(Data Query Language:数据查询语言，如select)

DCL：Data control Language，数据控制语言，主要是负责权限管理（用户）

代表指令：grant，revoke等

SQL是关系型数据库的操作指令，SQL是一种约束，但不强制（类似W3C）:不同的数据库产品（如Oracle,mysql）可能内部会有一些细微的差别

## Mysql数据库

Mysql数据库是一种c/s结构的软件：客户端/服务端，若想访问服务器必须通过客户端（服务器一直运行，客户端在需要使用的时候运行）

交互方式：

1. 客户端连接认证：连接服务器，认证身份：mysql.exe -hpup（-hlocalhost -p3306 -uroot -p）
2. 发送SQL指令
3. 服务器接受SQL指令：处理SQL指令：返回操作结果
4. 客户端接受结果：显示结果
5. 断开连接（释放资源：服务器并发限制）(exit/quit/\q)

## SQL基本操作

基本操作：CRUD

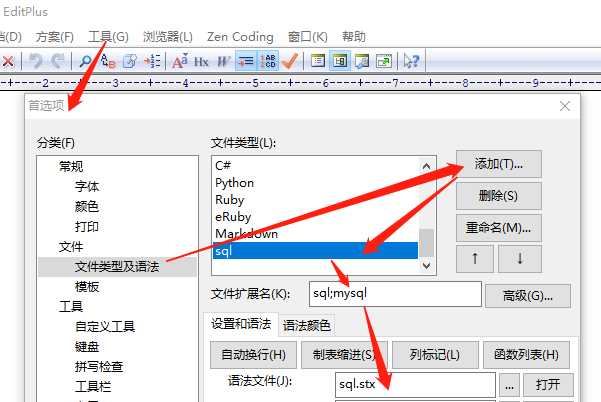
将SQL的基本操作根据操作对象进行分类，分为3类：库操作，表操作（字段），数据操作

### SQL编辑器

可使用EditPlus

配置sql语法模板（有颜色区分，高亮显示等）

1. 下载sql.stx
2. 将sql.stx放到EditPlus.exe所在目录
3. 配置



### 库操作

对数据库的增删改查

新增数据库

语法：create database 数据库名字 [库选项];

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：GBK和UTF8

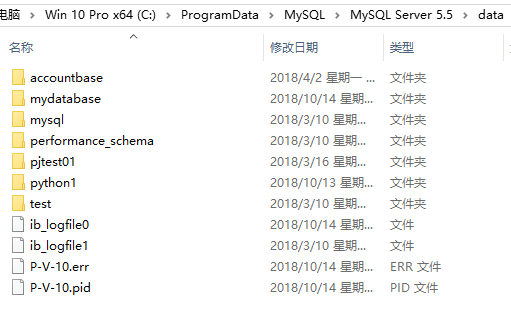
校对集设定：collate 具体校对集（数据比较的规则）

-- 创建数据库

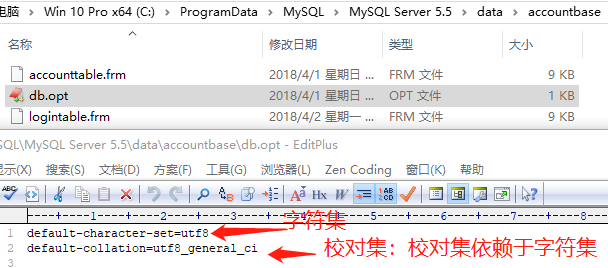
create database mydatabase charset utf8;

当创建数据库的SQL语句执行之后，发生了什么？

1. 在数据库系统中，增加了对应的数据库信息
2. 会在保存数据的文件夹下：Data目录，创建一个对应数据库名字的文件夹



1. 每个数据库下都有一个opt文件：保存了库选项



## 查看数据库

1. 查看所有数据库：show databases;
2. 查看指定部分的数据库：模糊查询

Show databases like ‘pattern’; -- pattern是匹配模式

%：表示匹配多个字符

\_：表示匹配单个字符

-- 查看以abc\_开始的数据库：\_需要被转义

Show databases like ‘information\\_%’;

1. 查看数据库的创建语句：show create database 数据库名字；

查看结果会与创建时的sql语句不太一样，因为数据库在执行SQL语句之前会优化SQL:系统保存的结果是优化的结果

## 更新数据库

数据库名字不可修改。

数据库的修改仅限库选项：字符集和校对集（校对集依赖字符集）

Alter database 数据库名字 [库选项];

Charset/character set [=] 字符集

Collate [=] 校对集

## 删除数据库

Drop database 数据库名字;

注意：数据库删除不可逆，删除要慎重，应该先进行备份后操作。

## 表操作

表与字段是密不可分的。

### 新增数据表

Create table [if not exists] 表名(

字段名字 数据类型，

字段名字 数据类型 --最后一行不需要逗号

)[表选项];

表选项:控制表的表现

字符集:charset/character set 具体字符集;

校对集:collate 具体校对集;

存储引擎:engine 具体的存储引擎(innodb和myisam)

方案1：显式的指定表所属的数据库

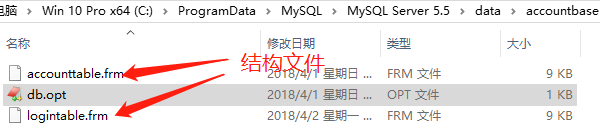
Create table 数据库名.表名();

方案2：隐式的指定表所属数据库：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.

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）



## 查看数据表

数据库可以查看的方式，表都可以查看.

1. 查看所有表：show tables;
2. 查看部分表：show tables like ‘pattern’;
3. 查看表创建语句

Show create table 表名\g --\g等价于;

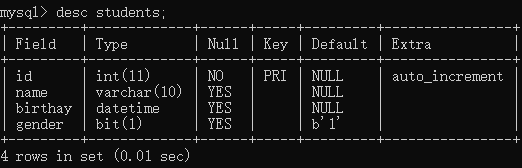
Show create table 表名\G --\G将查到的结构旋转90度变为纵向

1. 查看表结构：查看表中的字段信息

Desc 表名;

Describe 表名;

Show columns from 表名;



## 修改数据表

表的修改分为两个部分：修改表本身和修改字段

### 修改表本身

表本身可以修改：表名和表选项

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 表名 表选项 [=] 值;

### 修改字段

字段操作很多：新增，修改，重命名，删除

新增字段

alter table 表名 add [column] 字段名 字段类型 [列属性][位置];

位置：字段名可以插入到表中的任何位置

First:第一个位置

After:在哪个字段之后：after 字段名；默认的是在最后一个字段之后

修改字段：通常是修改属性或者数据类型

Alter table 表名 modify 字段名 数据类型 [属性][位置];

重命名字段

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

删除字段

Alter table 表名 drop 字段名;

小心：如果表中已经存在数据，那么删除字段会清空该字段的所有数据(不可逆)

### 删除数据表

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

注意：删除有危险，操作需谨慎(不可逆)

## 数据操作

### 新增数据

有两种方案

方案1、给全表字段插入数据，不需要指定字段列表：要求数据的值出现的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（建议是单引号）包裹

Insert into 表名 values(值列表)[,(值列表)]; -- 可以一次性插入多条数据

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的顺序一致

Insert into 表名(字段列表) values(值列表)[,(值列表)];

### 查看数据

Select \* from 表名 [where条件];

Select 字段列表 from 表名 [where条件];

### 更新数据

Update 表名 set 字段 = 值 [where条件]; -- 建议都有where,否则会更新全部

### 删除数据

Delete from 表名 [where条件];

## 中文数据问题

中文数据问题本质是字符集问题。

计算机只识别二进制，人类更多的是识别符号：需要有个二进制与字符的对应关系（字符集）

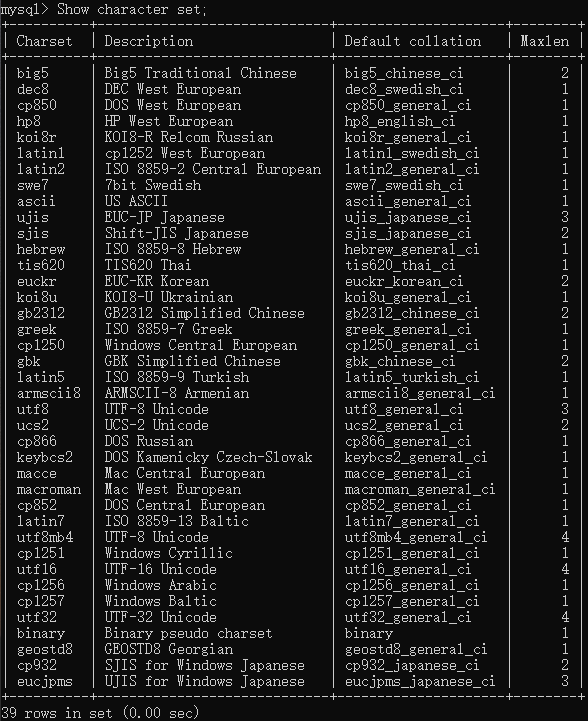


报错：服务器没有识别对应的四个字节：服务器认为数据是UTF8,一个汉字有三个字节：读取三个字节转换成汉子（失败），剩余的再读三个字节（不够）：最终失败。

所有的数据库服务器认为（表现）的一些特性都是通过服务器端的变量来保存：系统先读取自己的变量，看看应该怎样表现。

-- 查看服务器识别哪些字符集

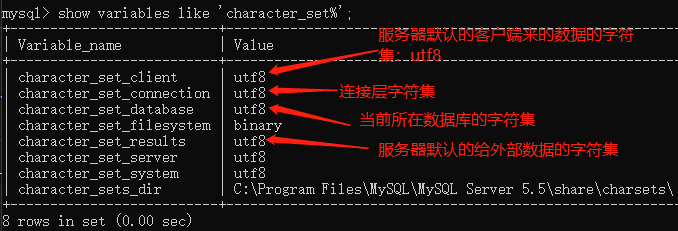
Show character set;



基本上：服务器是万能，什么字符集都支持

-- 既然服务器识别这么多：总有一种是服务器默认的跟客户端打交道的字符集

Show variables like ‘character\_set%’;



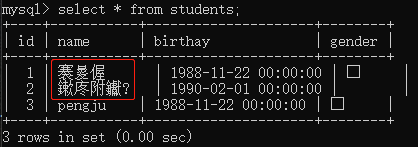
问题根源：客户端（cmd.exe）数据只能是GBK,而服务器认为是UTF8:矛盾产生

解决方案：

改变服务器，默认的接受字符为GBK;

Set character\_set\_client = gbk;

插入中文后查看数据，显示乱码：

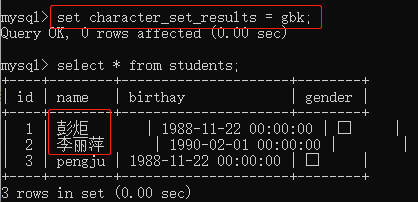


原因：数据来源是服务器，解析数据是客户端（客户端只识别GBK:只会两个字节一个汉字）：但是事实服务器给的数据却是UTF8,三个字节一个汉字：导致乱码

解决方案：

修改服务器给客户端的数据字符集为GBK;

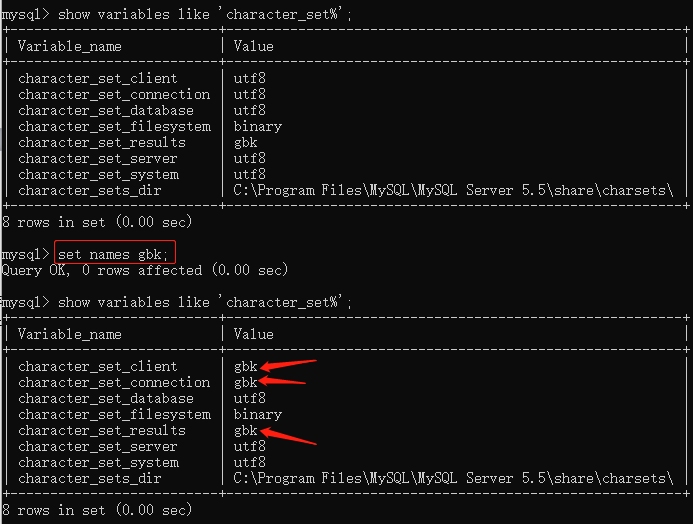
Set character\_set\_results = gbk;



Set 变量 = 值; 修改只是会话级别（当前客户端，当次连接有效：关闭失效）

设置服务器对客户端的字符集的认识：可以使用快捷方式：set names 字符集；

Set names gbk; ==>定价于 character\_set\_client,character\_set\_results,character\_set\_connection



connection连接层：是字符集转变的中间者，如果统一了效率更高，不同意也没问题。

## 校对集问题

校对集：数据比较的方式

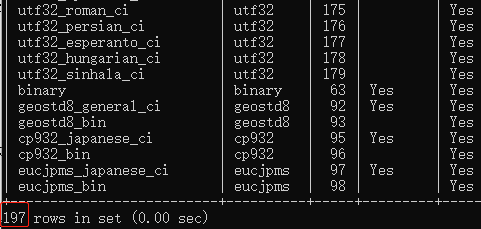
校对集有三种格式：

\_bin:binary,二进制比较，取出二进制位，一位一位的比较，区分大小写

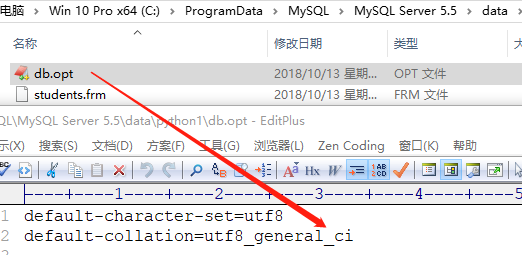
\_cs:case sensitive,大小写敏感，区分大小写

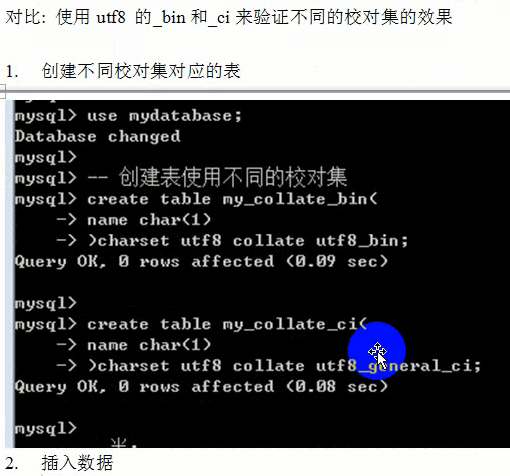
\_ci:case insensitice,大小写不敏感，不区分大小写

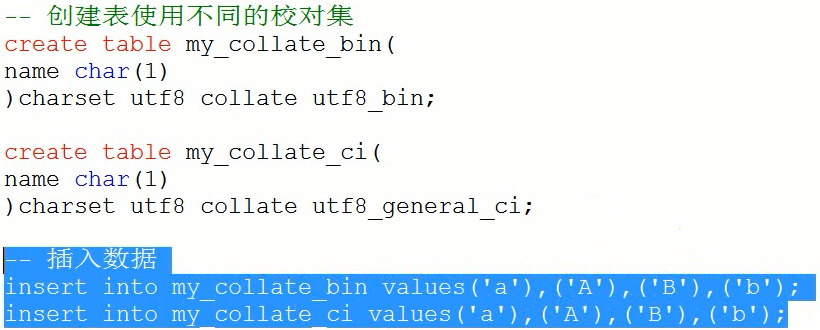
查看数据库所支持的校对集：show collation;

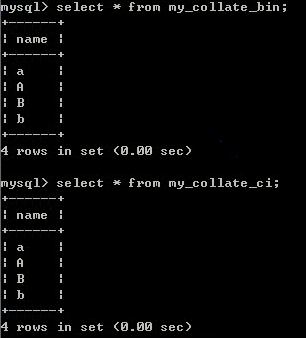


校对集应用：只有当数据产生比较的时候，校对集才会生效。

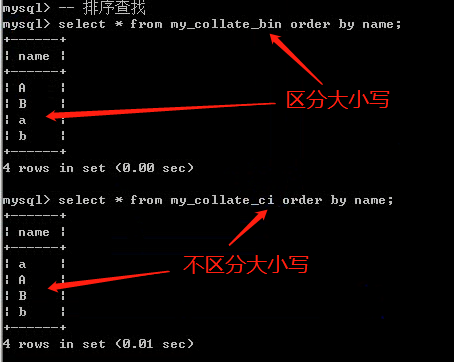








1. 比较：根据某个字段进行排序：order by 字段名 [asc|desc];asc升序，desc降序，默认是升序



校对集：必须在没有数据之前声明好，如果有了数据，再对校对集进行修改：那么修改无效

### Web乱码问题

动态网站由三部分构成：浏览器，web服务器，数据库服务器，三个部分都有自己的字符集（中文），数据需要在三个部分之间来回传递：很容易产生乱码。

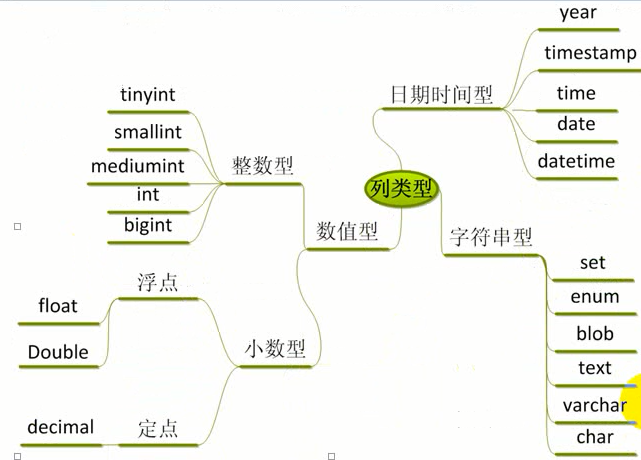
如何解决乱码问题：统一编码（三码合一）

但是事实上不可能：浏览器是用户管理（根本不可能控制）

但是必须要解决这些问题：主要靠php来做

## 数据类型（列类型）

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

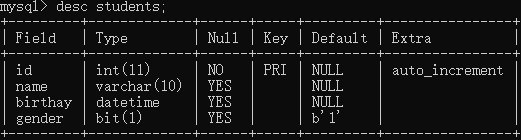




## 列属性

列属性：真正约束字段的是数据类型，但是数据类型的约束很单一，需要有一些额外的约束，来更加保证数据的合法性。

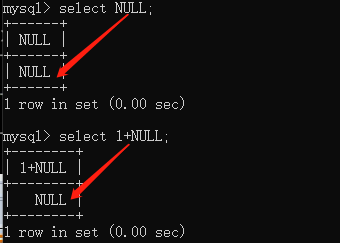
列属性有很多：NULL/Not NULL, default, primary key, unique key,auto\_increment,comment

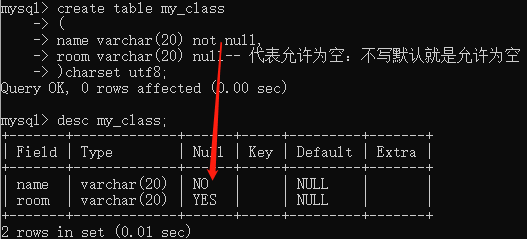


### 空属性

两个值：NULL(默认的)和Not NULL(不为空)

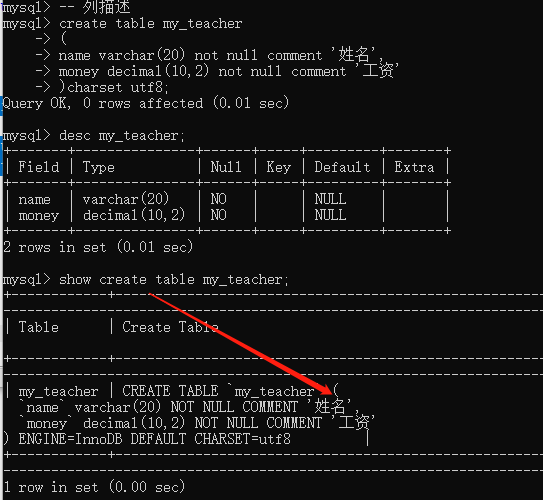
虽然数据库对字段默认基本都是为NULl,但是在实际开发中，尽可能的要保证数据不应该为空：空数据没有意义；空数据没办法参与运算





### 列描述

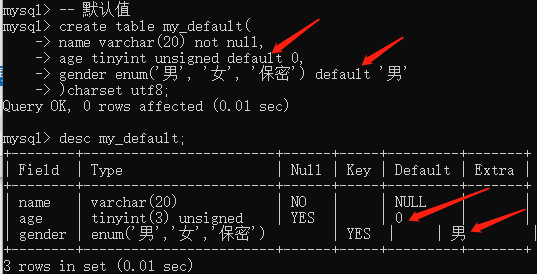
列描述：comment,描述，没有实际含义：是专门用来描述字段，会根据表创建语句保存：用来给程序猿（数据库管理员）来进行了解的。



### 默认值

默认值：某一种数据会经常性的出现某个具体的值，可以在一开始就指定好：在需要真实数据的时候，用户可以选择性的使用默认值。

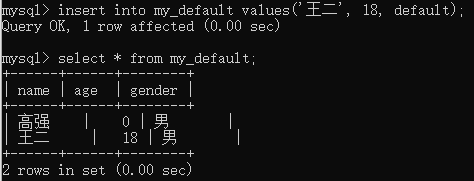
默认值关键字：default



默认值的生效：使用，在数据进行插入的时候，不给该字段赋值



要想使用默认值，可以不一定去指定列表，故意不使用字段列表：可以使用default关键字代替值



### 主键

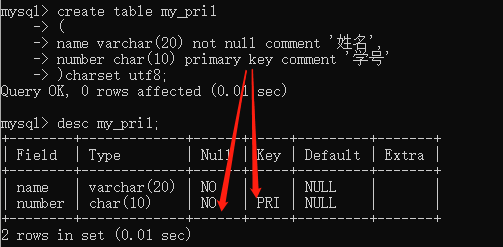
主键：primary key,主要的键，一张表只能有一个字段可以使用对应的键，用来唯一的约束该字段里面的数据，不能重复：这种称之为主键

一张表最多只有一个主键。

### 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

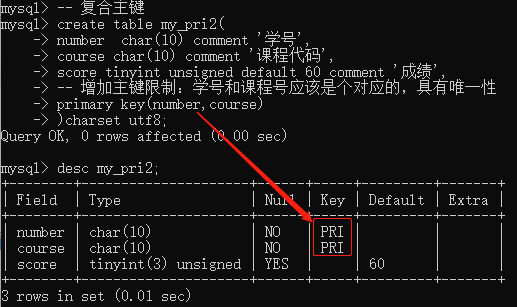
方案1：在创建表的时候，直接在字段后面，跟primary key关键字（主键本身不允许为空，所以定义主键后默认也不为空）



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，在所有的字段之后，使用primary key(主键字段列表)来创建主键（如果有多个字段作为主键，可以是复合主键）



方案3：当表已经创建好之后，额外追加主键：可以通过修改表字段属性，也可以直接追加

alter table 表名 add primary key(字段列表);

或

Alter table 表名 modify 字段名 字段属性 primary key;

前提：表中字段对应的数据本身是独立的（不重复），如果数据有重复，主键会追加失败。

### 主键约束

主键对应的字段中的数据不允许重复：一旦重复，数据操作失败（增和改）

### 更新主键&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; (主键是惟一的，因此删除主键不需要带字段名)

### 主键分类

在实际创建表的过程中，很少使用真实业务数据作为主键字段（业务主键，如学号，课程号）；

大部分的时候是使用逻辑性的字段（字段没有业务含义，值是什么都没有关系），将这种字段主键称之为逻辑主键。

### 自动增长

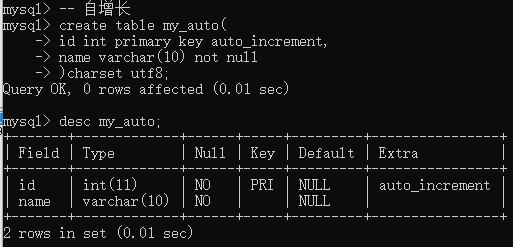
自增长：当对应的字段，不给值，或者说给默认值，或者给NULL的时候，会自动的被系统触发，系统会从当前字段中已有的最大值进行+1操作，得到一个新的在不同的字段。

自增长通常是跟主键搭配。

### 新增自增长

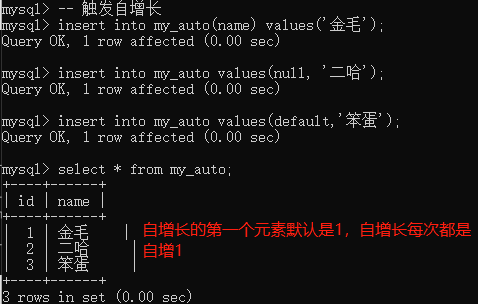
自增长特点：auto\_increment

1. 任何一个字段要做自增长必须前提是本身是一个索引（key 一栏有值）
2. 自增长字段必须是数字（整形）
3. 一张表最多只能有一个自增长

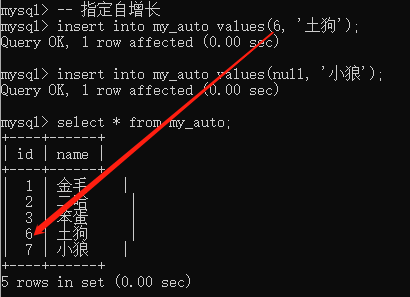


### 自增长的使用

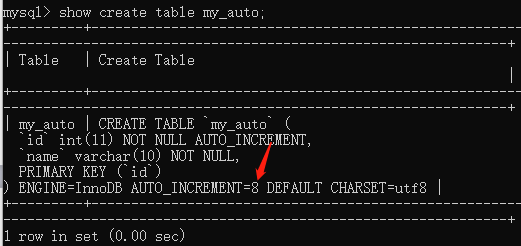
当自增长被给定的值为NULL,或默认值的时候会触发自动增长



自增长如果对应的字段输入了值，那么自增长失效：但是下一次还是能够正确的自增长（从最大值+1）



如何确定下一次自增长？可以通过查看表创建语句看到

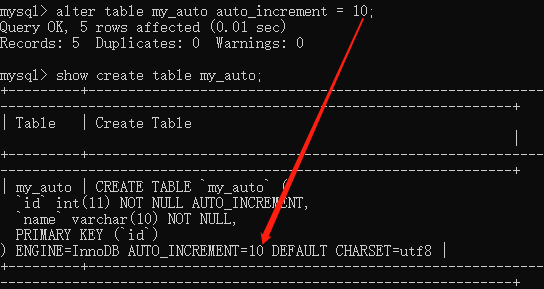


### 修改自增长

自增长如果涉及到字段改变：必须先删除自增长，再增加（一张表只能有一个自增长）

修改当前自增长已经存在的值：修改只能比当前已有的自增长的最大值大，不能小（小不生效）

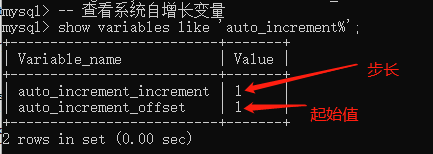
Alter table 表名 auto\_increment = 值;



思考：为什么自增长是从1开始？为什么每次都是自增1呢？

所有系统的变量（如 字符集，校对集）都是由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%’;



可以修改变量实现不同的效果：修改是针对整个数据库修改，而不是单张表：（修改是会话级）

Set auto\_increment\_increment = 5; -- 一次自增5

### 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; (类型后面不要加主键，主键理论上单独存在的)

## 唯一键

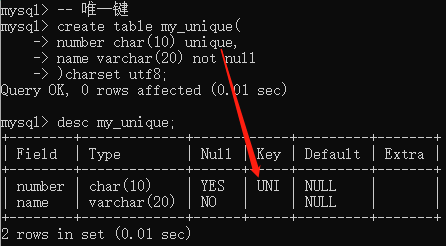
一张表往往有很多字段需要具有唯一性，数据不能重复：但是一张表中只能有一个主键：唯一键（unique key）就可以解决表中有多个字段需要唯一性约束的问题。

唯一键的本质与主键差不多：唯一键默认的允许自动为空，而且可以多个为空（空字段不参与唯一性比较）

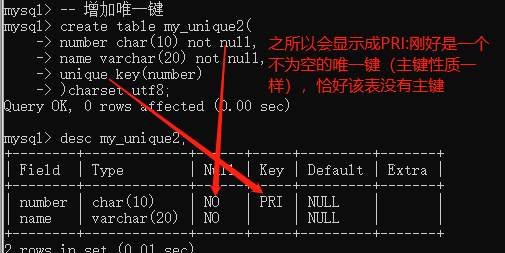
### 增加唯一键

基本与主键差不多：三种方案

方案1：在创建表的时候，字段之后直接跟unique/ unique key



方案2：在所有的字段之后增加unique key(字段列表); -- 支持复合唯一键



方案3：在创建表之后增加主键

Alter table 表名 add unique key (字段名);

### 唯一键约束

唯一键与主键本质相同：唯一区别就是唯一键允许默认为空，而且是多个为空

如果唯一键也不允许为空：那么与主键的约束作用是一致的。

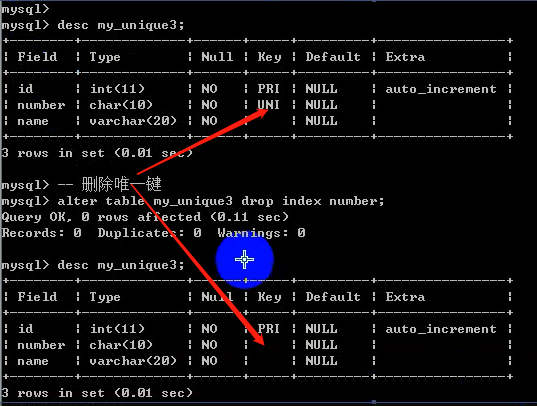
### 更新唯一键&删除唯一键

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





## 索引

几乎所有的索引都是建立在字段之上。

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：文件能够实现快速的匹配数据，并且能够快速的找到对应表中的记录。

索引的意义：

1. 提升查询数据的效率
2. 约束数据的有效性（唯一性等）

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

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，也可能使用索引（主键、唯一键）

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：index (不具有唯一性，但经常需要作为查询条件)

全文索引：针对文章内部的关键字进行索引

全文索引最大的问题：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，而且中文可以各种随意组合（分词：sphinx）

## 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： 人与身份证

一对多（多对一）： 人与银行卡

多对多： 朋友与朋友

所有的关系都是表与表之间的关系。

### 一对一

一张表的一条记录一定只能与另外一张表的一条记录进行对应；反之亦然。

### 一对多（多对一）

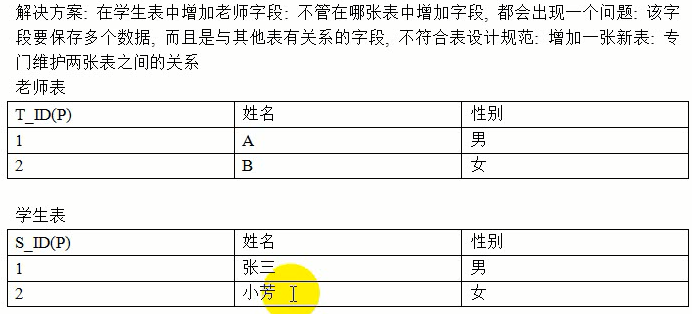
一张表中有一条记录可以对应另一张表中的多条记录；但是反过来，另外一张表的一条记录只能对应第一张表的一条记录。

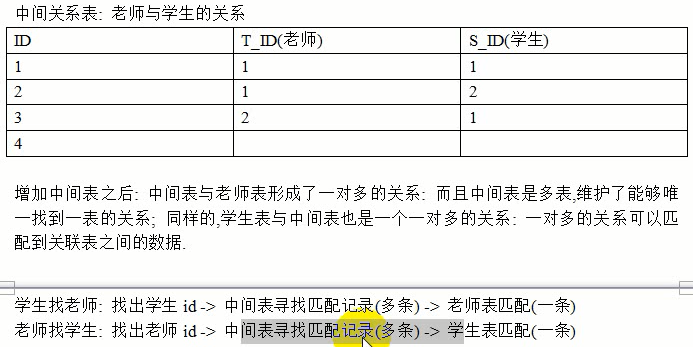


### 多对多

一张表（A）中的一条记录能够对应另外一张表（B）中的多条记录；同时B表中的一条记录也能对应A表中的多条记录。







## 数据高级操作

### 主键冲突

基本语法

Insert into 表名 [字段列表] values (值列表);

在数据插入的时候，假设主键对应的值已经存在：插入一定会失败！

当主键存在冲突的时候（duplicate key）,可以选择性的进行处理：更新和替换

主键冲突：更新操作

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

主键冲突：替换(干掉之前的再插入新的)

Replace into 表名[字段列表：包含主键] values(值列表);

### 蠕虫复制

从已有的数据中去获取数据，然后将数据又进行新增操作：数据成倍增加。

创建表高级操作：从已有表(可以是其它表也可以是自身表)创建新表（只会复制表结构，不会复制数据）

Create table 表名 like 数据库.表名;

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;

蠕虫复制的意义：

1. 从已有表拷贝数据到新表
2. 可以迅速地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;

### 更新数据

基本语法

Update 表名 set 字段 = 值 [where条件];

高级语法

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

### 删除数据

与更新类似：可以通过limit 来限制数量

Delete from 表名 [where 条件][limit数量];

删除：如果表中存在主键自增长，那么当数据删除之后，自增长不会还原

思路：数据的删除是不会改变表结构，只能删除表后重建表

Truncate 表名; -- 先删除该表，再创建该表（重置自增长）

### 查询数据

基本语法：

Select 字段列表/\* from 表名 [where条件];

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];

### Select选项

Select选项：select对查出来的结果的处理方式

All:默认的，保留所有结果

Distinct:去重，查出来的结果，将重复去除（所有字段都相同）

### 字段别名

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，可能会有同名字段），需要对字段名进行重命名：别名

语法

字段名 [as] 别名;

Eg：select id,number as 学号,name as 姓名 from my\_student;

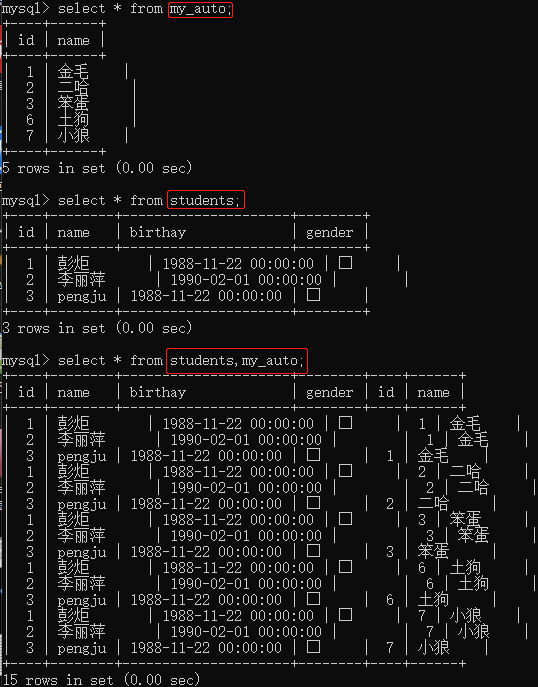
### 数据源

数据源：数据的来源，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，查询语句

单表数据源：select \* from 表名;

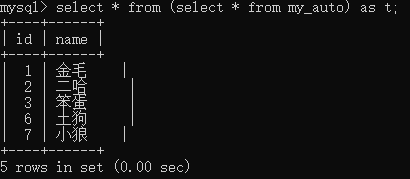
多表数据源：select \* from 表名1,表名2;



从一张表中取出一条记录，去另外一张表中匹配所有的记录，而且全部保留：（记录数和字段数），将这种结果称为：笛卡尔积（交叉连接）：笛卡尔积没什么卵用，应该尽量避免。

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;



### where子句

用来判断数据，筛选数据

Where子句返回结果:0 或者 1, 0代表false,1代表true.

(sql中没有bool类型，0在sql中意义比较特殊，因此sql中枚举类型,字符串位置都是从1开始)

判断条件：

比较运算符：>,<,>=,<=,!=,<>,=,like,between and,in/not in

逻辑云算符：&&(and), ||(or), !

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保存到内存；如果失败直接放弃

条件查询1：找出id为1,3,5的学生

Select \* from my\_student where id = 1 || id = 3 || id = 5; -- 逻辑判断

Select \* from my\_student where id in (1,3,5); -- 落在集合中

条件查询2：找身高在180到190之间的学生

Select \* from my\_student where height >= 180 && height <= 190;

Select \* from my\_student where height >= 180 and height <= 190;

Select \* from my\_student where height between 180 and 190;

Between 本身是闭区间；between 左边的值必须小于或者等于右边的值。

Select \* from table where 1; -- 所有条件都满足

### Group by子句

Group by:分组的意思，根据某个字段进行分组（相同的放一组，不同的分到不同的组）

基本语法：group by 字段名;



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）

SQL提供了一系列统计函数

Count():统计分组后的记录数：每一组有多少记录

Max(): 统计每组中最大的值

Min(): 统计最小值

Avg(): 统计平均值

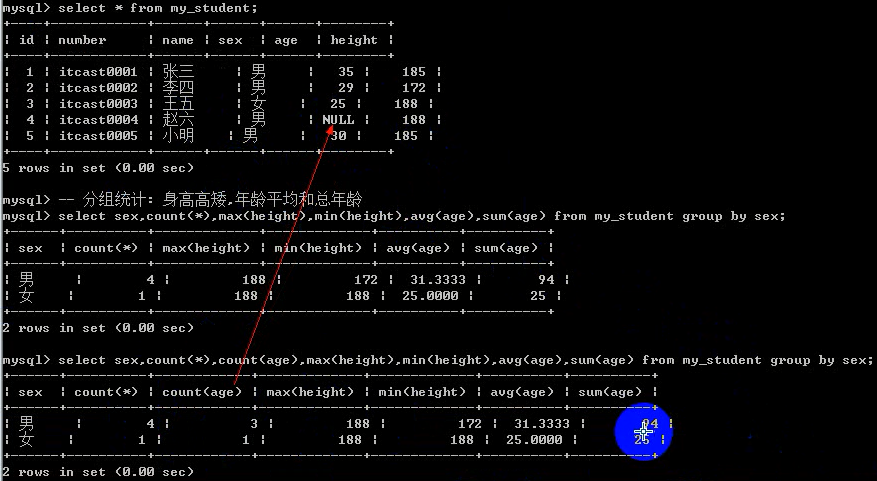
Sum(): 统计和

分组统计1：身高高矮，平均年龄 和 总年龄

Select sex,count(\*),max(height),min(height),avg(age),sum(age) from my\_student group by sex;

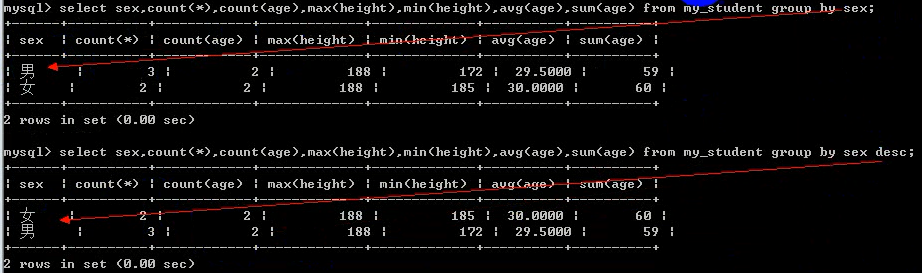


count函数：里面可以使用两种参数：\*代表统计记录，字段名代表统计对应的字段(NULL 不统计)



分组会自动排序：根据分组字段：默认升序

Group by 字段 [asc|desc]; -- 是对分组的结果合并之后的整个结果进行排序



多字段分组：先根据一个字段进行分组，然后对分组后的结果再次按照其它字段进行分组

Select 字段1,字段2, count(\*) from 表名 group by 字段1,字段2;



有一个函数：可以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）：

Group\_concat(字段);



回溯统计：

### Having子句

Having 子句：与where子句一样：进行条件判断的

Where是针对磁盘数据进行判断：进入到内存之后，会进行分组操作：分组结果就需要having来处理。

Having几乎能做where能做的所有事情，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只有having能够使用.

-- 求出所有班级人数大于等于2的学生人数

Select c\_id,count(\*) from my\_student group by c\_id having count(\*) >= 2;

2、having 能够使用字段别名：where不能：where是从磁盘取数据，而名字只可能是字段名：别名是在字段进入到内存后才会产生.

Select c\_id,count(\*) as total from my\_student group by c\_id having total >= 2;

Select name as 名字,number 学号 from student; -- pass

Select name as 名字,number 学号 from student having 名字 like ‘张%’; -- pass

Select name as 名字,number 学号 from student having 名字 like ‘张%’; -- fail

### Order by子句

Order by: 排序，根据某个字段进行升序或者降序排序，依赖校对集。

基本语法

Order by 字段名 [asc|desc]; -- asc是升序（默认的），desc是降序

排序可以进行多字段排序：先根据某个字段进行排序，然后排序好的内部，再按照某个数据进行再次排序：

Order by 字段名1, 字段名2 desc; -- 按字段名1升序，字段名2降序

### Limit子句

Limit子句是一种限制结果的语句：限制数量。

Limit有两种使用方式

方案1：只用来限制长度（数据量）：limit 数据量;

方案2：限制起始位置，限制数量：limit 起始位置（偏移量），长度;

Select \* from my\_student limit 0,2; --记录数是从0开始编号

方案2主要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

对于服务器来讲：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: offset=（页码 - 1）\* 每页显示量

## 连接查询

连接查询：将多张表进行记录的连接（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2张表合并）

## 函数应用

Update my\_student set age=floor(rand() \* 20 + 20), height = floor(rand() \* 20 + 170);

Rand():取得一个0到1之间的随机数

Floor():向下取整

## MYSQL高级

### MyISAM和InnoDB对比

|  |  |  |
| --- | --- | --- |
| 对比项 | MyISAM | InnoDB |
| 主外键 | 不支持 | 支持 |
| 事务 | 不支持 | 支持 |
| 行表锁 | 表锁，即使操作一条记录也会锁住整个表，不适合高并发的操作 | 行锁，操作时只锁某一行，不对其它行有影响，适合高并发操作 |
| 缓存 | 只缓存索引，不缓存真实数据 | 不仅缓存索引还缓存真实数据，对内存要求较高，而且内存大小对性能有决定性的影响 |
| 表空间 | 小 | 大 |
| 关注点 | 性能 | 事务 |
| 默认安装 | Y | Y |

### 索引

没建索引数据是杂乱无章的，建立索引之后，索引表会对索引字段进行排序，会很快定位到数据

对频繁使用的字段建立索引，可以查找的更快

SQL性能下降原因：

1. SQL语句写的烂
2. 索引失效
3. 关联查询太多join(设计缺陷或不得已的需求)

### 什么是索引？

### 索引的优劣势

优势：

类似大学图书馆建书目索引，提高数据检索的效率，降低数据库的IO成本；

通过索引对数据进行排序，降低数据排序的成本，降低CPU的消耗。

劣势：

索引也是一张表，该表保存了主键与索引字段，并指向实体表的记录，所以索引列也是要占空间的；

虽然索引大大提高了查询速度，但却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE,因为更新表时，mysql不仅要保存数据，还要保存一下索引文件每次更新添加了索引列的字段，都会调整因为更新所带来的键值变化后的索引信息。

### 哪些情况适合建立索引哪些情况不适合建立索引？

**适合建索引**

1. 主键自动建立唯一索引
2. 频繁作为查询条件的字段应该创建索引
3. 查询中与其它表关联的字段，外键关系建立索引
4. 频繁更新的字段不适合创建索引（因为每次更新不单单是更新了记录还会更新索引）
5. Where条件里用不到的字段不创建索引
6. 单键/组合索引的选择问题，who?(在高并发下倾向创建组合索引)
7. 查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度
8. 查询中统计或者分组字段

**不适合建索引**

1. 表记录太少（eg:数据量在百万以下基本不用创建索引）
2. 经常增删改的表（虽提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT,UPDATE,DELETE时，mysql不仅要保存数据，还要保存一下索引文件）
3. 数据重复且分布平均的字段（如果某个数据列包含许多重复的内容，为它建立索引就没有太大的实际效果）

### MySQL常见瓶颈

**CPU：** CPU在饱和的时候一般发生在数据装入内存或从磁盘上读取数据的时候

**IO:** 磁盘I/O瓶颈发生在装入数据远大于内存容量的时候

**服务器硬件的性能瓶颈：**top, free, iostat, vmstat来查看系统的性能状态