

MySQL 数据库

认识什么是库？

京东仓库：



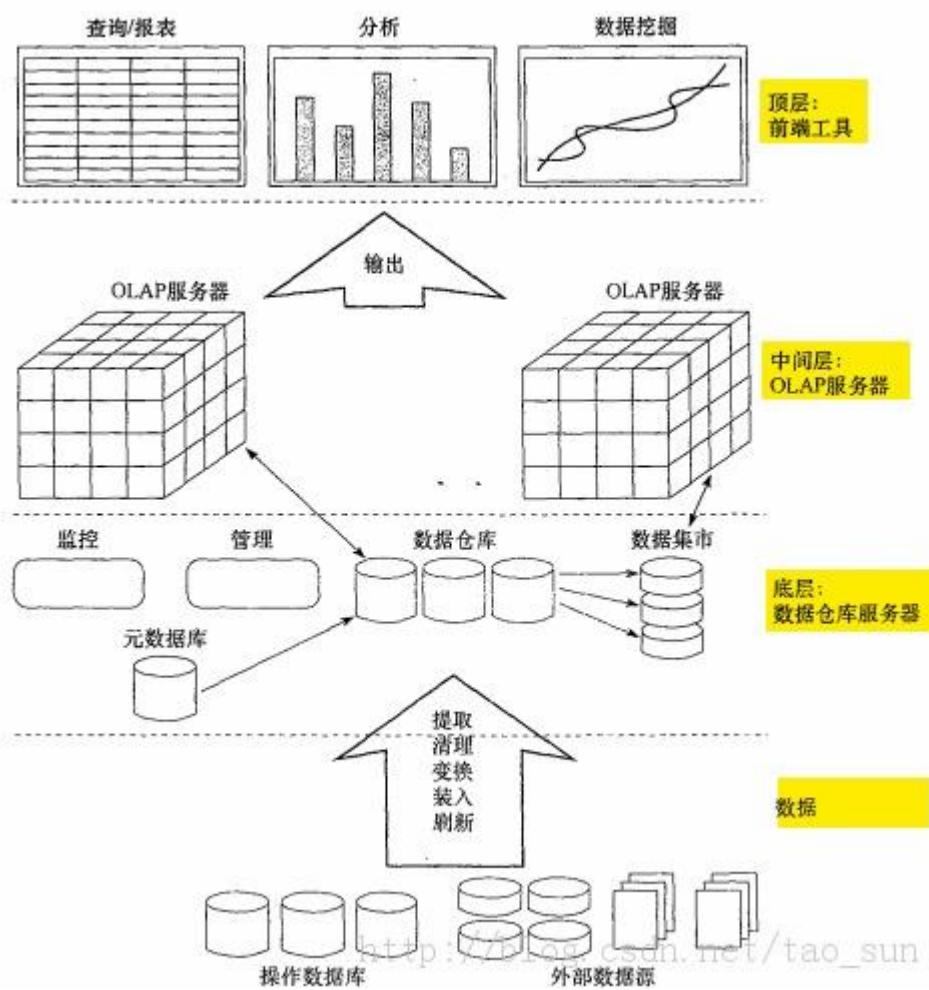
弹药库：



金库：



数据仓库：



1. 数据库概念介绍

数据库(Database)是按照一定的结构在电脑上保存和管理数据的软件。

数据库：存储数据的仓库。



玩家数量超过 2 亿！



在没有计算机之前，我们通过人工来管理数据，随着数据量的不断增长，通过人工来管理数据效率极其的低下，有了计算机之后，人们开始使用计算机来管理数据，并开发了专门用来管理、组织数据的软件，数据库。

1.1 数据库分类

1.1.1 关系数据库

关系数据库，是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。

数据库的另外一种区分方式：基于存储介质
存储介质分为两种：磁盘和内存

关系型数据库：存储在磁盘中
非关系型数据库：存储在内存中

1.1.2 非关系数据库

非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。
必须强调的是，数据的持久存储，尤其是海量数据的持久存储，还是需要关系数据库这员老将

由于关系型数据库本身天然的多样性，以及出现的时间较短，因此，不像关系型数据库，有几种数据库能够一统江山，关系型数据库的非常多，并且大部分都是开源的，这里列出一些：

Redis,Tokyo

Cabinet,Cassandra,Voldemort,**MongoDB**,Dynomite,Hbase,CouchDB,Hypertable,Riak,Tin,Flare,Lightcloud,KiokuDB,Scalaris,Kai,ThruDB

2. 关系型数据库

2.1 基本概念

关系数据库，是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。现实世界中的各种实体以及实体之间的各种联系均用关系模型来表示。关系模型是由埃德加·科德于 1970 年首先提出的，并配合“科德十二定律”。现如今虽然对此模型有一些批评意见，但它还是数据存储的传统标准。关系模型由**关系数据结构、关系操作集合、关系完整性约束**三部分组成。

关系数据结构：指的数据以什么方式来存储，是一种二维表的形式存储
本质：二维表

编号	姓名	性别	年龄	电话	家庭住址
1	张三丰	男	12	138 3838 1438	花果山 1 号坑
2	黑山老妖	女	10	138 3838 1438	花果山 2 号坑
3	东方不败	男	10	138 3838 1438	花果山 3 号坑

关系操作集合：如何来关联和管理对应的存储数据，SQL 指令

获取张三的年纪：已知条件为姓名

```
Select 年龄 from 二维表 where 姓名 = 张三;
```

关系完整性约束：数据内部有对应的关联关系，以及数据与数据之间也有对应的关联关系

编号	姓名	性别	年龄	电话	家庭住址
1	张三丰	男	12	138 3838 1438	花果山 1 号坑
2	黑山老妖	女	18	138 3838 1438	花果山 2 号坑
3	东方不败	男	10	138 3838 1438	花果山 3 号坑

表内约束：对应的具体列只能放对应的数据（不能乱放）

表间约束：自然界各实体都是有着对应的关联关系（外键）

2.2 典型关系型数据库

Oracle、DB2、Microsoft SQL Server、Microsoft Access、MySQL、SQLite

小型关系型数据库：Microsoft Access、SQLite

中型关系型数据库：SQL Server、MySQL

大型关系型数据库：Oracle、DB2

MySQL 当前跟 Oracle 是一个公司的：隶属于 Oracle

2.3 库、表、字段、记录

先来搞明白三者之间的关系

```

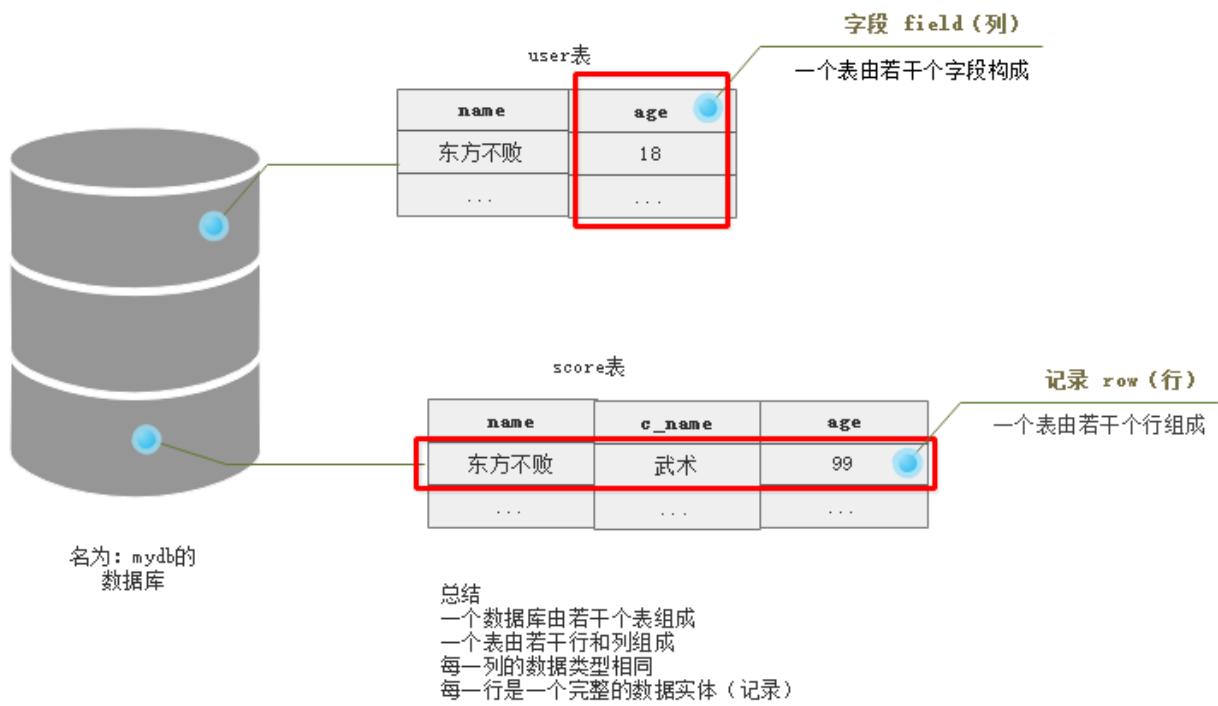
1
2
3 想了解同学们的信息:
4
5 班级学生信息表
6
7
8
9
10
11
12
13
14
15
16
17
18 表:
19 把所有的同学的信息放到一起, 就是一个学生信息表
20 表用来存储同一类数据, 由字段(列)和记录(行)来构成的
21
22 字段:

```

name	age	sex	address
王二小	18	男	5#250
张三丰	19	男	5#438
凤姐	17	女	2#202

宠物表

name	age	sex	address
小蛤	1	母	狗窝



数据库软件中保存的数据是以表格的形式管理的,
比如学生信息:

编号	姓名	性别	年龄	电话	家庭住址
1	张三丰	男	12	138 3838 1438	花果山 1 号坑
2	黑山老妖	女	10	138 3838 1438	花果山 2 号坑
3	东方不败	男	10	138 3838 1438	花果山 3 号坑

在数据库中保存以上数据需要以下三步:

1. 创建数据库

2. 在数据库中创建学生表
3. 向学生表中插入三条记录数据

3. MySQL

3.1 基本介绍

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL 是最好的 RDBMS (Relational Database Management System, 关系数据库管理系统) 应用软件。

AB 公司被 Sun 公司收购→Sun 公司又被 Oracle 公司收购了，所以现在 MySQL 和 Oracle 现在同属于一家公司。

- 1) MySQL 是一种开源免费的数据库产品
- 2) MySQL 对 PHP 的支持是最好（wamp 或者 lamp）

MySQL 中用到的操作指令就是 SQL 指令

3.2 wampserver 中的 MySQL





名称	修改日期	类型	大小
bin	2017/5/10 18:35	文件夹	
data	2017/7/31 11:08	文件夹	
lib	2017/5/10 18:35	文件夹	
share	2017/5/10 18:36	文件夹	
COPYING	2016/7/12 13:55	文件	18 KB
my.ini	2017/5/10 18:36	配置设置	7 KB
my-default.ini	2016/7/12 14:41	配置设置	2 KB
README	2016/7/12 13:55	文件	3 KB
wampserver.conf	2015/12/11 12:55	CONF 文件	1 KB

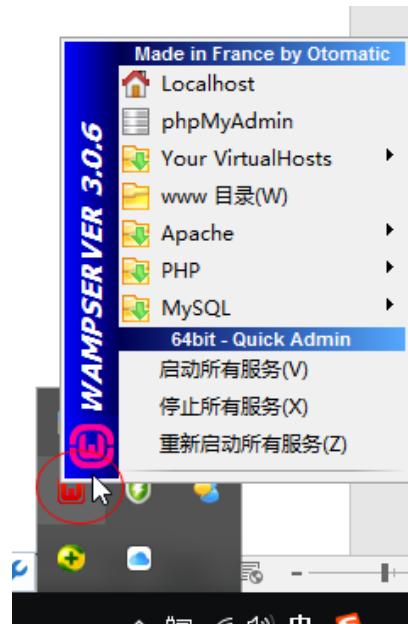
3.2.1 MySQL 启动、关闭、重启

MySQL 是一种 C/S 结构：客户端和服务端，服务端对应的软件：mysqld.exe
Mysql 服务端架构有以下几层构成：

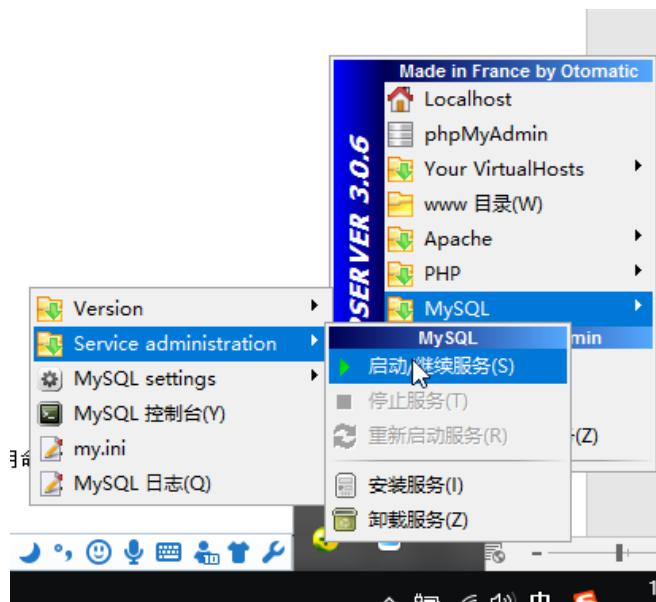
- 1、数据库管理系统（最外层）：DBMS，专门管理服务器端的所有内容
- 2、数据库（第二层）：DB，专门用于存储数据的仓库（可以有很多个）
- 3、二维数据表（第三层）：Table，专门用于存储具体实体的数据
- 4、字段（第四层）：Field，具体存储某种类型的数据（实际存储单元）

3.2.2 通过 wamp 控制台启动 MySQL

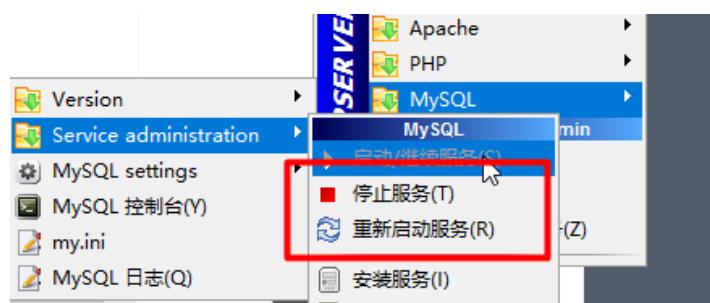
- 左键单击 wamp server 图标，弹出如下菜单



- 选择 MySQL → Service administration → 启动继续服务



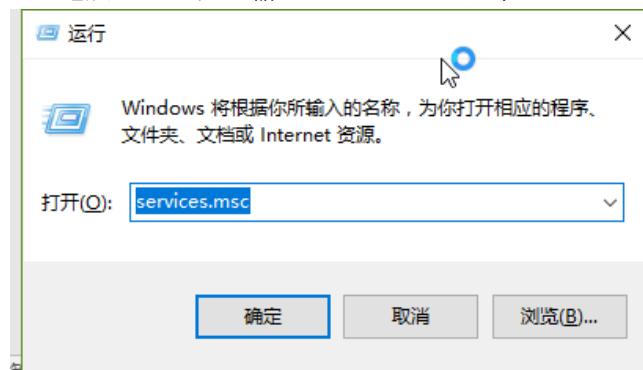
启动成功后，状态如下：



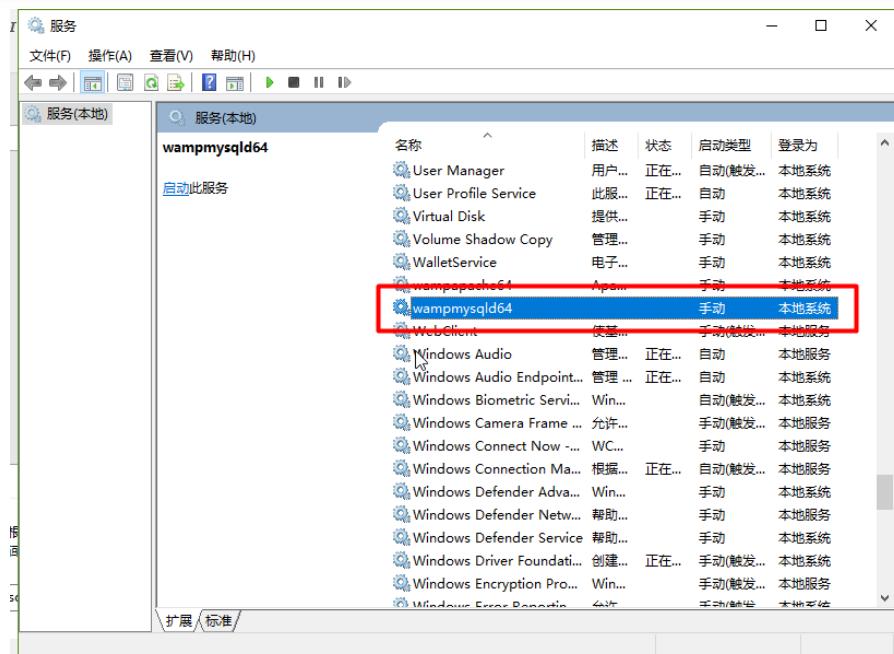
3.2.3 通过“服务”启动 MySQL

通过 Windows 下打开 cmd 命令窗口，然后使用命令进行管理

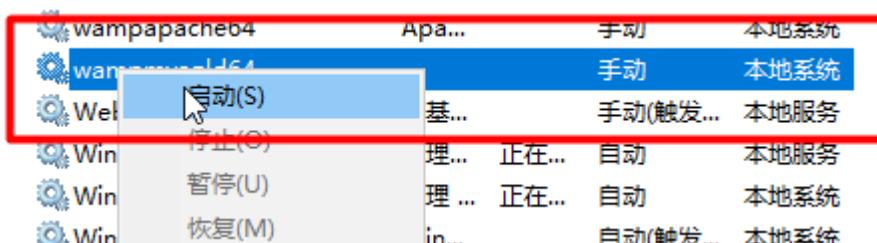
- 电脑 win+r 键，输入 services.msc 回车



打开系统服务管理器，并找到 wampmysqld64 (MySQL) 服务



- 右键→启动,启动服务即可



3.3 操作 MySQL 的方式

操作 MySQL 数据库有两种方式:

- 在命令行中直接输入 SQL 语句 (下次课会学习)
- 使用图形界面管理数据库, 如: phpMyAdmin、Navicat for MySQL 等。

以下学习如何使用 phpMyAdmin 图形界面来管理数据。

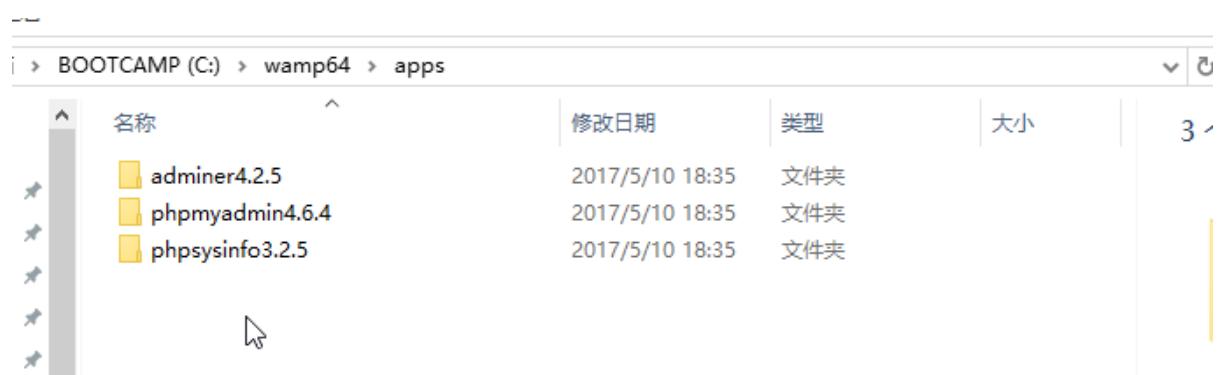
4. 使用 phpMyAdmin

4.1 基本介绍



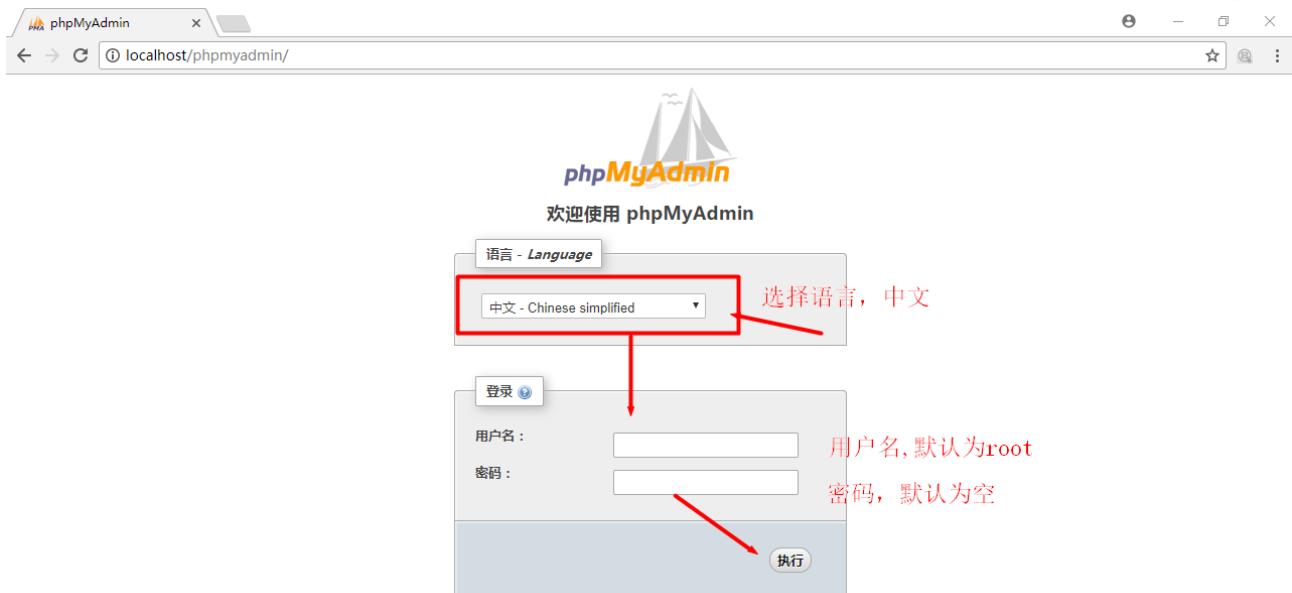
phpMyAdmin，简称 PMA 是一个以 PHP 为基础，以 Web-Base 方式架构在网站主机上的 MySQL 的数据库管理工具，让管理者可用 Web 接口管理 MySQL 数据库。

phpMyAdmin 在安装 wampserver 的时候已经集成安装了，具体位置在
C:\{安装路径}\wamp64\apps



4.2 开始使用 phpMyAdmin

注意：因为 phpMyAdmin 是使用 php 开发，需要 php 运行环境，所以使用之前必须开启 apache 服务



输入用户名为: root 密码: 默认为空 (不用写), 点击执行即可登录成功

4.3 创建数据库

如果要保存数据（学生信息表）到我们的数据库中

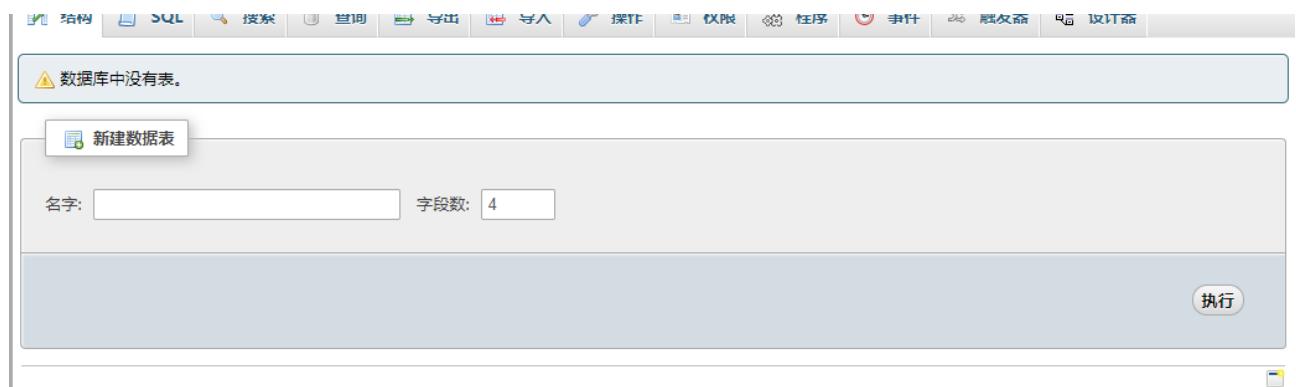
- 1、创建一个数据库，名称为： mydb
- 2、在数据库中创建一张数据表，名称为： student
- 3、在表中创建字段信息（姓名、性别、年龄...）

4、插入一条一条的记录（行）

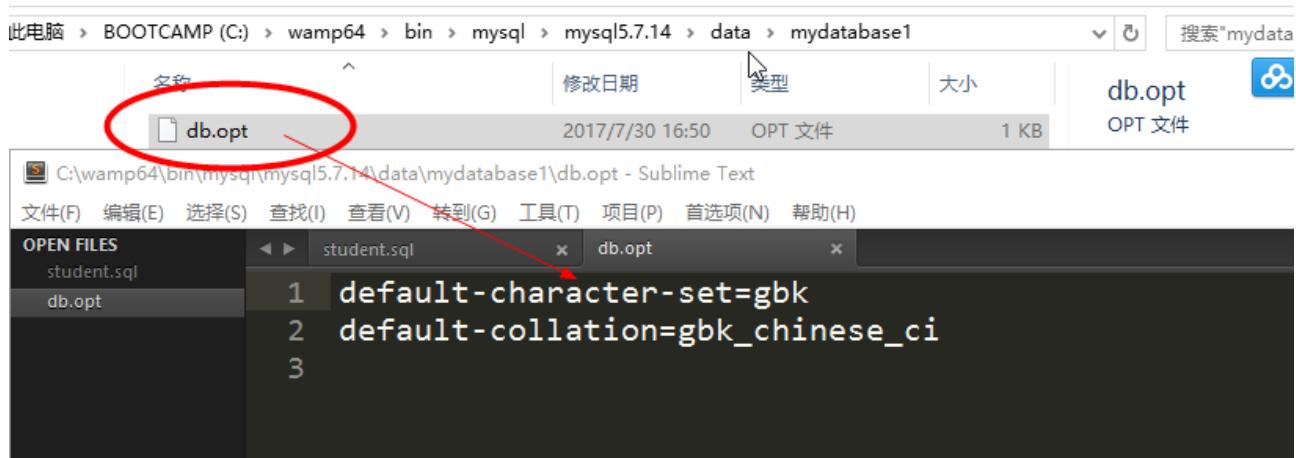


数据库名: mydatabase1
数据库字符集: utf8_general_ci

创建成功后显示如下：



创建完成后，会在 mysql 下的 data 目录生成数据库相关的文件



```
1 default-character-set=gbk
2 default-collation=gbk_chinese_ci
3
```

练习 1：创建一个名为 czxy 的数据库，设置编码为 uft8_general_ci

字符集

练习 2：创建一个名为 mydb 的数据库，设置编码为 gbk_chinese_ci

字符集

4.4 创建数据表

编号	姓名	性别	年龄	电话	家庭住址
1	张三丰	男	12	138 3838 1438	花果山 1 号坑
2	黑山老妖	女	10	138 3838 1438	花果山 2 号坑
3	东方不败	男	10	138 3838 1438	花果山 3 号坑

第一步：点击选择数据库

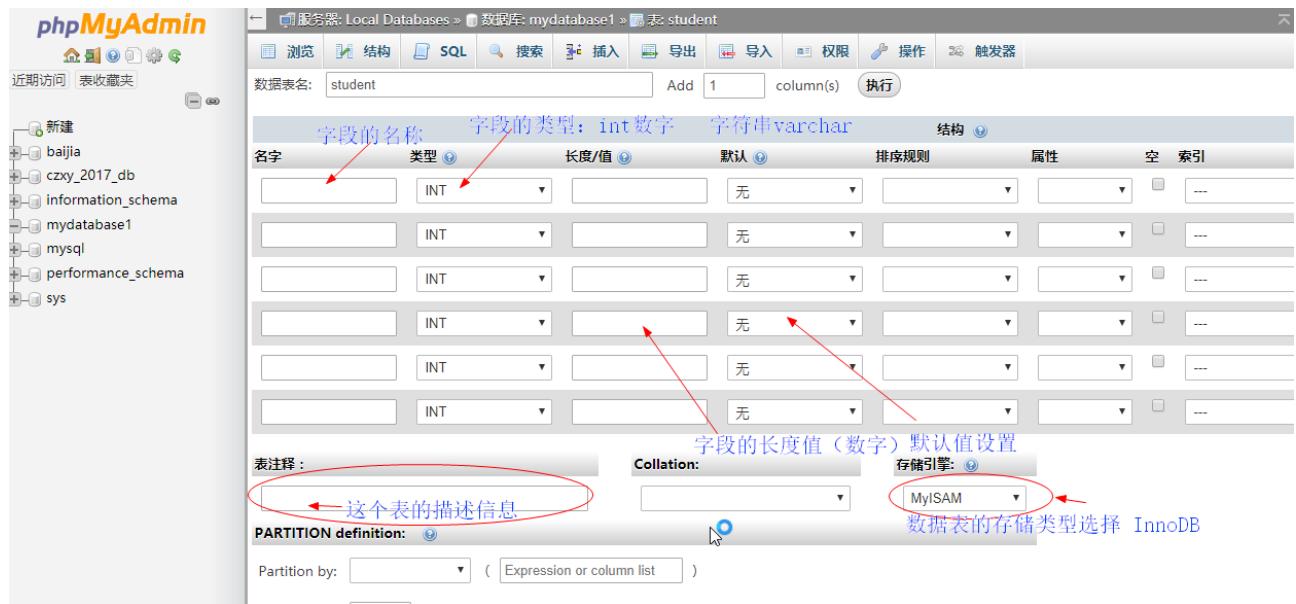
第二步：输入数据表的名称，如：student，表示这个表中存放的都是学生的信息

第三步：设置字段数，如 4，表示有 4 个字段（Field）

第四步：点击“执行”按钮，开始执行

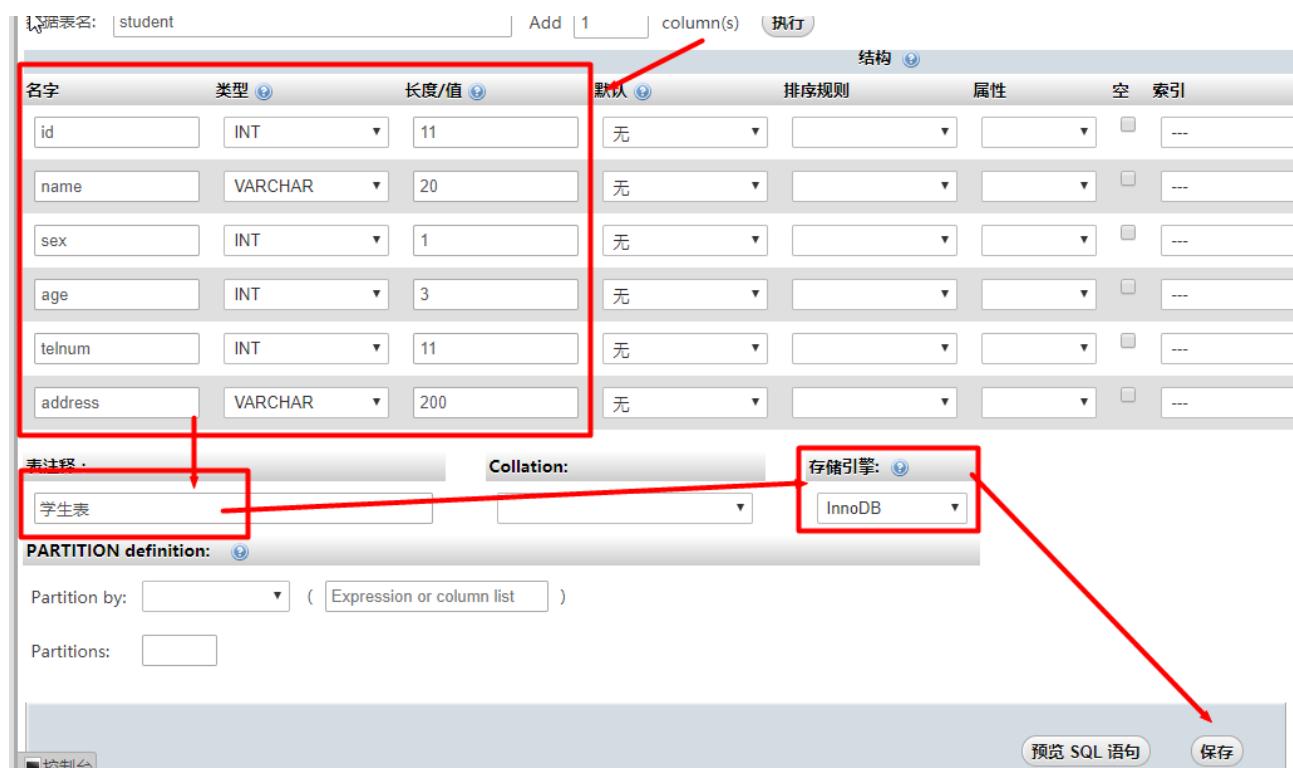


4.5 为表设置字段



字段的名称
字段的类型: int数字
字符集: varchar
名字
类型
长度/值
默认
排序规则
属性
空
索引
表注释:
这个表的描述信息
Collation:
存储引擎: MyISAM
数据表的存储类型选择 InnoDB
PARTITION definition:
Partition by: Expression or column list

设置内容如下:



名字	类型	长度/值	默认	排序规则	属性	空	索引
id	INT	11	无				
name	VARCHAR	20	无				
sex	INT	1	无				
age	INT	3	无				
telnum	INT	11	无				
address	VARCHAR	200	无				

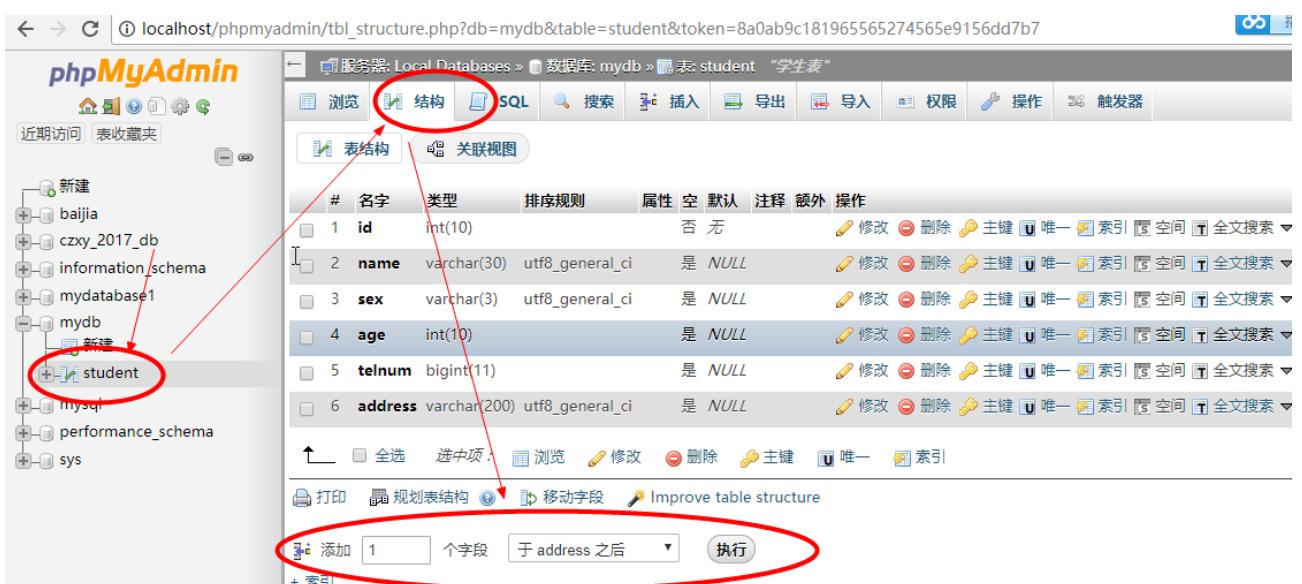
表注释:
学生表
存储引擎:
InnoDB

创建完成后的效果:



The screenshot shows the 'Structure' tab of the 'student' table in the 'mydb' database. The table has six columns: id, name, sex, age, telnum, and address. The 'age' column is currently selected. The interface includes a toolbar at the top with various operations like '修改' (Edit), '删除' (Delete), and '索引' (Index). Below the table, there are buttons for '添加' (Add), '执行' (Execute), and a dropdown for adding fields. The bottom navigation bar includes links for '打印' (Print), '规划表结构' (Structure Planner), and 'Improve table structure'.

4.6 修改表结构



This screenshot shows the 'Structure' tab for the 'student' table. Several UI elements are highlighted with red circles: the 'Structure' tab itself, the '添加' (Add) button at the bottom left, and the '执行' (Execute) button at the bottom right. The table structure is identical to the one in the previous screenshot, with columns id, name, sex, age, telnum, and address. The 'name' column is the current selection.

4.7 插入数据

第一步：点击插入按钮：

服务器: Local Databases > 数据库: mydatabase1 > 表: student “学生表”

浏览 结构 SQL 搜索 插入 导出 导入 权限 操作 触发器

表结构 关联视图 插入数据 (行/记录)

#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	id	int(11)			否	无			修改 删除 主键 唯一 索引 空间 全文搜索 更多
2	name	varchar(20) gbk_chinese_ci			否	无			修改 删除 主键 唯一 索引 空间 全文搜索 更多
3	sex	int(1)			否	无			修改 删除 主键 唯一 索引 空间 全文搜索 更多
4	age	int(3)			否	无			修改 删除 主键 唯一 索引 空间 全文搜索 更多
5	telnum	int(11)			否	无			修改 删除 主键 唯一 索引 空间 全文搜索 更多
6	address	varchar(200) gbk_chinese_ci			否	无			修改 删除 主键 唯一 索引 空间 全文搜索 更多

↑ 全选 挑选项: 浏览 修改 删除 主键 唯一 索引

第二步：填写字段对应的值

phpMyAdmin

近期访问 表收藏夹

- 新建
- baijia
- czxy_2017_db
- information_schema
- mydatabase1
 - 新建
 - student**
- mysql
- performance_schema
- sys

插入

字段	类型	函数	空值
id	int(11)		<input type="text"/>
name	varchar(20)		<input type="text"/>
sex	int(1)		<input type="text"/>
age	int(3)		<input type="text"/>
telnum	bigint(11)		<input type="text"/>
address	varchar(200)		<input type="text"/>

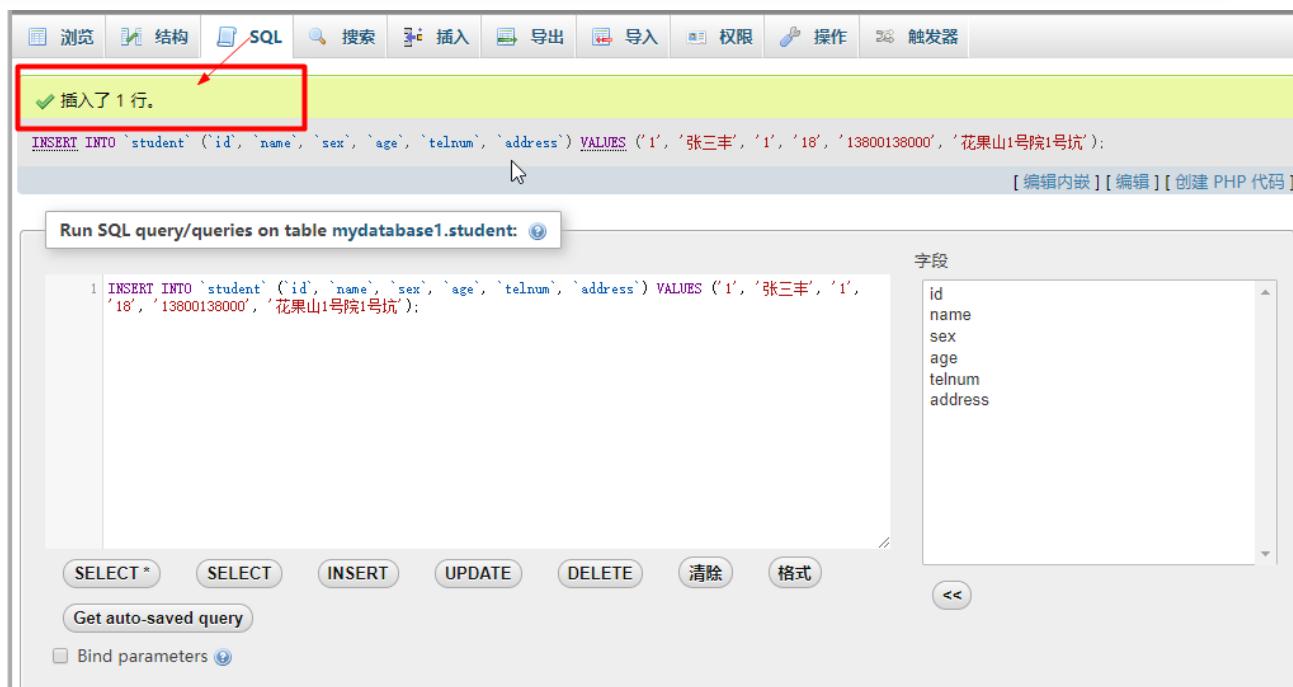
执行

插入

字段	类型	函数	空值
id	int(11)		<input type="text" value="1"/>
name	varchar(20)		<input type="text" value="张三丰"/>
sex	int(1)		<input type="text" value="1"/>
age	int(3)		<input type="text" value="18"/>
telnum	bigint(11)		<input type="text" value="13800138000"/>
address	varchar(200)		<input type="text" value="花果山1号院1号坑"/>

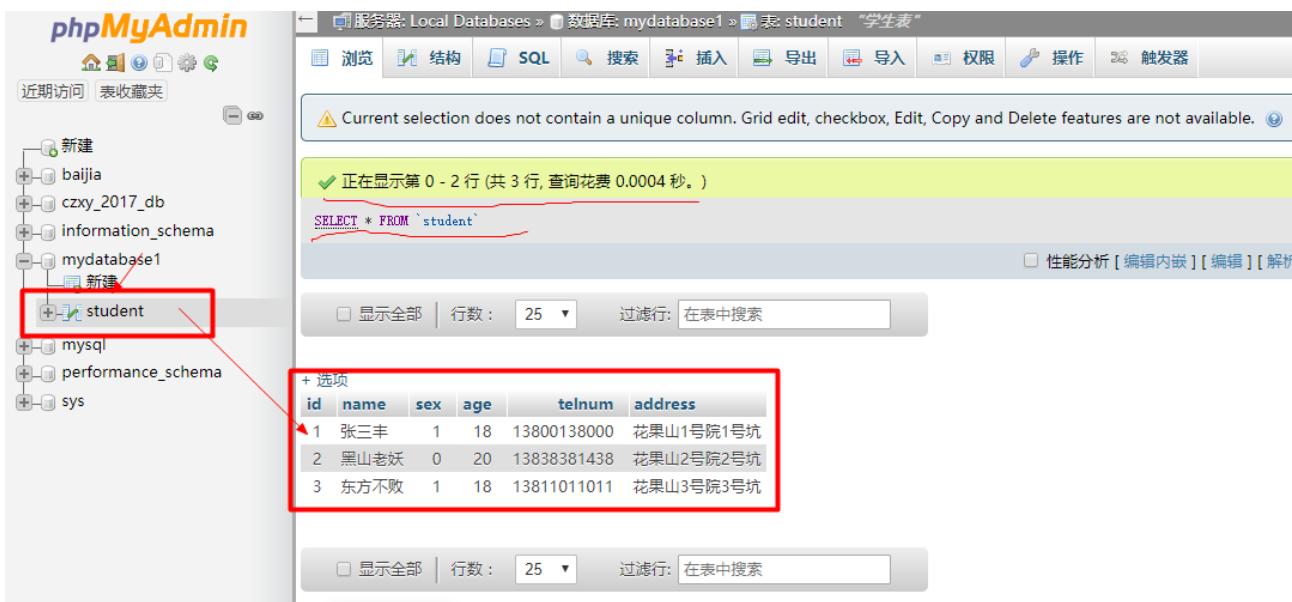
执行

第三步：保存



4.8 查看数据

- 在左侧导航，点击表名称则可以查询表中的数据



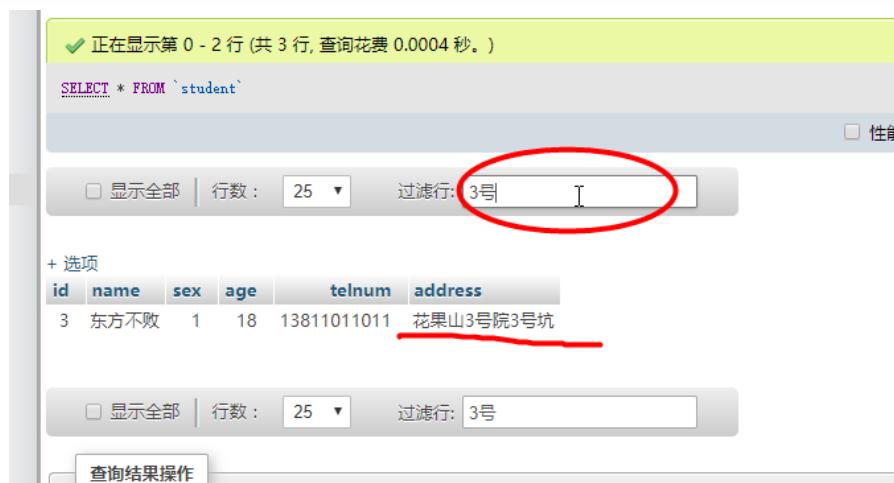
正在显示第 0 - 2 行 (共 3 行, 查询花费 0.0004 秒。)

```
SELECT * FROM `student`
```

+ 选项

	id	name	sex	age	telnum	address
1	张三丰	1	18	13800138000	花果山1号院1号坑	
2	黑山老妖	0	20	13838381438	花果山2号院2号坑	
3	东方不败	1	18	13811011011	花果山3号院3号坑	

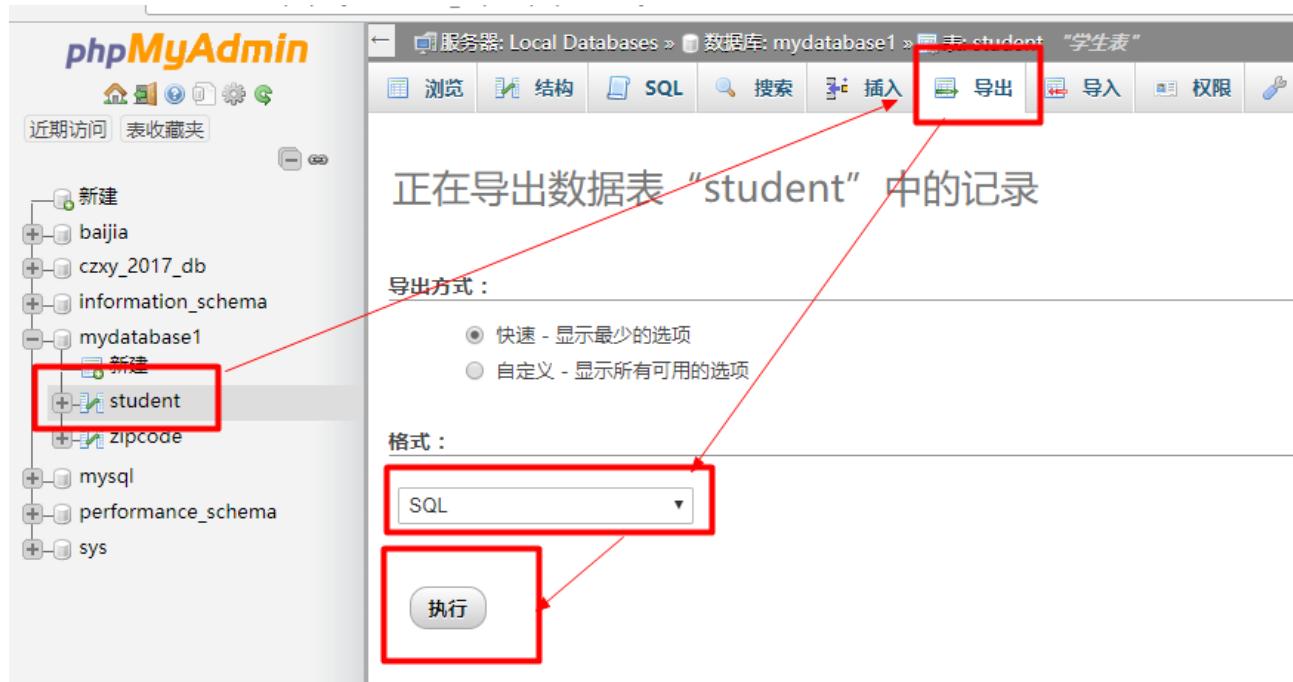
- 数据比较多的时候，可以过滤以便显示所需要的数据



+ 选项	id	name	sex	age	telnum	address
	3	东方不败	1	18	13811011011	花果山3号院3号坑

4.9 导出数据

导出数据的主要目的是进行数据备份，确保数据安全。



正在导出数据表 “student” 中的记录

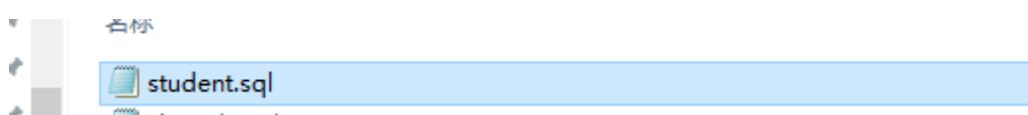
导出方式：

● 快速 - 显示最少的选项
● 自定义 - 显示所有可用的选项

格式：
SQL

执行

导出文件后缀为.sql 的数据库脚本文件



文件内容就是数据库指令

```
student.sql * | [1] 1 -- phpMyAdmin SQL Dump
2 -- version 4.6.4
3 -- https://www.phpmyadmin.net/
4 --
5 -- Host: 127.0.0.1
6 -- Generation Time: 2017-07-30 12:45:01
7 -- 服务器版本: 5.7.14
8 -- PHP Version: 5.6.25
9
10 SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
11 SET time_zone = "+00:00";
12
13
14 /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
15 /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
16 /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
17 /*!40101 SET NAMES utf8mb4 */;
18
19 --
20 -- Database: `mydatabase1`
21 --
22
23 --
24
25 --
26 -- 表的结构 `student`
27 --
28
29 CREATE TABLE `student` (
30   `id` int(11) NOT NULL,
31   `name` varchar(20) NOT NULL,
32   `sex` int(1) NOT NULL,
33   `age` int(3) NOT NULL,
34   `telnum` bigint(11) NOT NULL,
35   `address` varchar(200) NOT NULL
36 ) ENGINE=InnoDB DEFAULT CHARSET=gbk COMMENT='学生表';
37
38 --
39 -- 转存表中的数据 `student`
40 --
41
42 INSERT INTO `student` (`id`, `name`, `sex`, `age`, `telnum`, `address`) VALUES
43 (1, '张三丰', 1, 18, 13800138000, '花果山1号院1号坑'),
44 (2, '黑山老妖', 0, 20, 13838381438, '花果山2号院2号坑'),
45 (3, '东方不败', 1, 18, 13811011011, '花果山3号院3号坑');
46
47 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

```
-- phpMyAdmin SQL Dump
-- version 4.6.4
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1
-- Generation Time: 2017-07-30 12:45:01
-- 服务器版本: 5.7.14
```

```
-- PHP Version: 5.6.25

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

-- 
-- Database: `mydatabase1`
-- 

-- -----
-- 

-- 表的结构 `student` 
-- 

CREATE TABLE `student` (
  `id` int(11) NOT NULL,
  `name` varchar(20) NOT NULL,
  `sex` int(1) NOT NULL,
  `age` int(3) NOT NULL,
  `telnum` bigint(11) NOT NULL,
  `address` varchar(200) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=gbk COMMENT='学生表';

-- 
-- 转存表中的数据 `student` 
-- 

INSERT INTO `student`(`id`, `name`, `sex`, `age`, `telnum`, `address`) VALUES
(1, '张三丰', 1, 18, 13800138000, '花果山 1 号院 1 号坑'),
(2, '黑山老妖', 0, 20, 13838381438, '花果山 2 号院 2 号坑'),
(3, '东方不败', 1, 18, 13811011011, '花果山 3 号院 3 号坑');

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

4.10 导入数据

一条一条的输入比较麻烦，可以使用导入功能，批量添加数据

实例：导入全国所有的省市县的数据

- 选择数据库，并点击菜单上的“导入”



- 选择.sql文件



- 点击执行按钮，开始执行



● 导入成功！

```
✓ 导入成功，执行了 11 个查询。 (student.sql)

✓ MySQL 返回的查询结果为空 (即零行)。 (查询花费 0.0007 秒。)
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO"

[ 编辑内嵌 ] [ 编辑 ] [ 创

✓ MySQL 返回的查询结果为空 (即零行)。 (查询花费 0.0003 秒。)
SET time_zone = "+00:00"

[ 编辑内嵌 ] [ 编辑 ] [ 创

✓ MySQL 返回的查询结果为空 (即零行)。 (查询花费 0.0003 秒。)
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */

[ 编辑内嵌 ] [ 编辑 ] [ 创

✓ MySQL 返回的查询结果为空 (即零行)。 (查询花费 0.0003 秒。)
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */

[ 编辑内嵌 ] [ 编辑 ] [ 创

✓ MySQL 返回的查询结果为空 (即零行)。 (查询花费 0.0003 秒。)
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */

[ 编辑内嵌 ] [ 编辑 ] [ 创

SQL 返回的查询结果为空 (即零行)。 (查询花费 0.0003 秒。)

控制台
```

导入成功后，数据表显示在数据库中

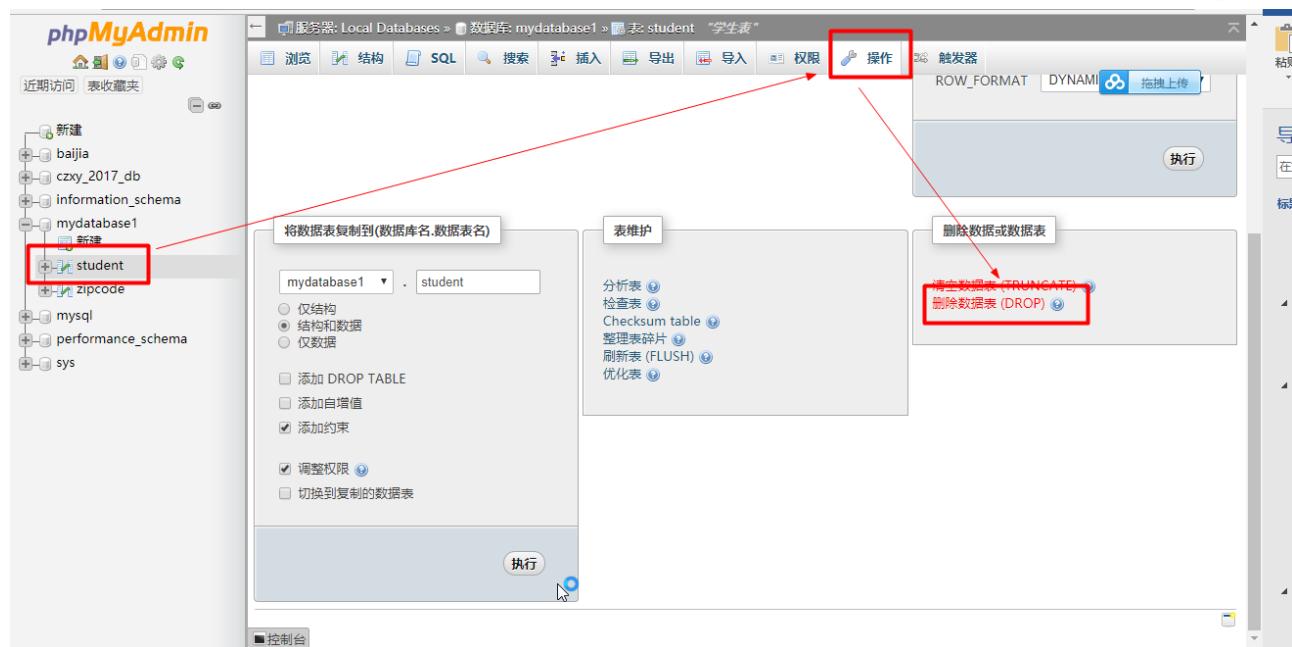


4.11 删除数据表

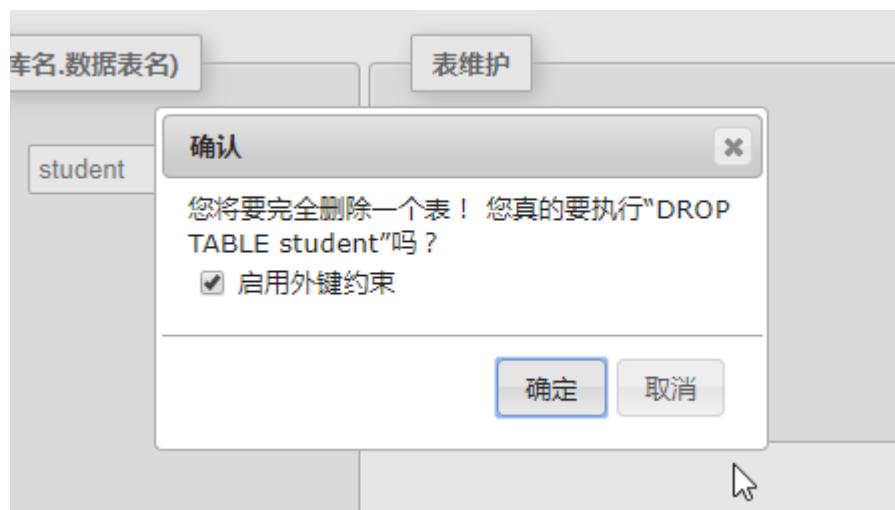
如果一个数据表想要进行删除，可以使用如下方法进行删除

清除数据表：清空表中的数据，但是结构还在（慎用）

删除数据表：清空表中的数据，同时删除表结构（慎用）



弹出提示，确认是否真的要删除



点击确定即可删除！

4.12 删除数据库

删除数据库：同时删除数据库中所有的表结构及表中的数据（非常危险，慎用！慎用！慎用！）

5. 数值类型

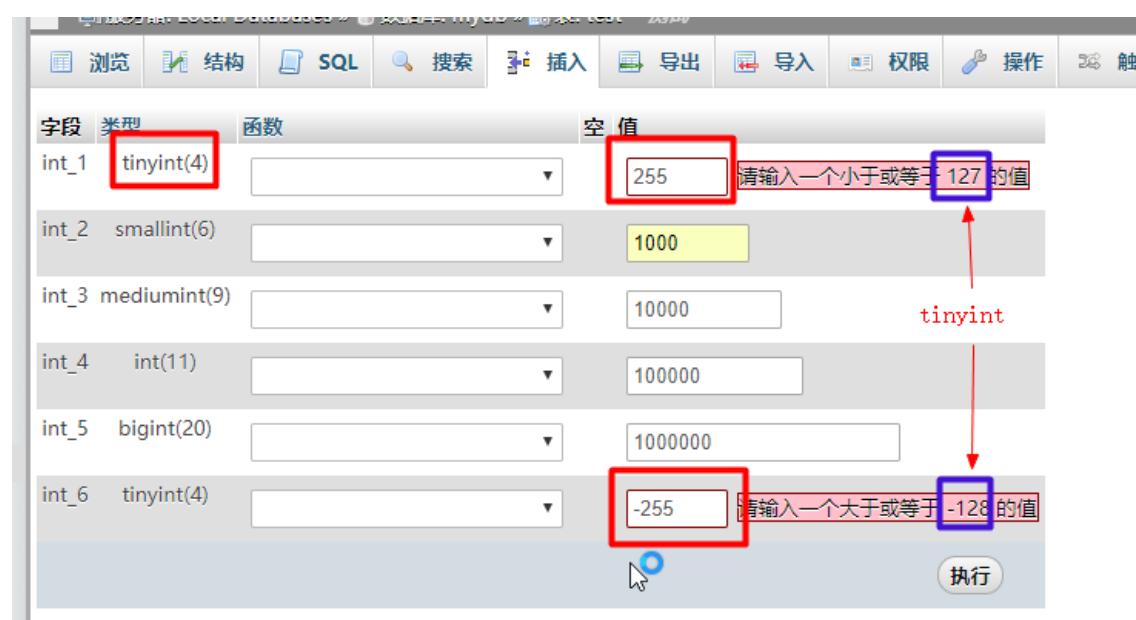
5.1 整数类型

整型类型名称及表示的范围

类型	存储	最小值	最大值
	(Bytes)	(Signed/Unsigned)	(Signed/Unsigned)
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

5.1.1 通过 tinyint 看数据长度问题

tinyint 表示迷你整型，字段长度默认为 4，表示数字范围：-127 ~ 128



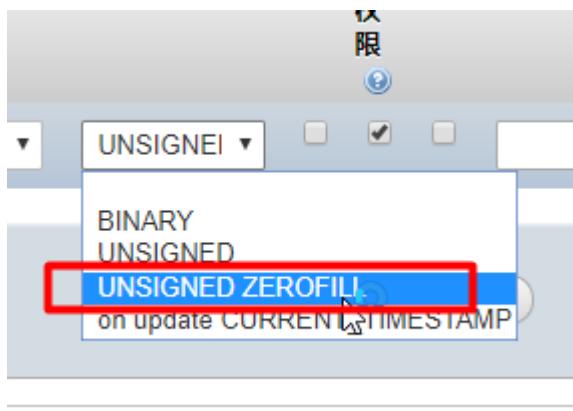
-127 含“-”，总长度为 4，可以通过设置属性为 unsigned 来查看、验证。



#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	int_1	tinyint(4)			是	NULL			修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)
2	int_2	smallint(6)			是	NULL			修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)
3	int_3	mediumint(9)			是	NULL			修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)
4	int_4	int(11)			是	NULL			修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)
5	int_5	bigint(20)			是	NULL			修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)
6	int_6	tinyint(4)			是	NULL			修改 删除 主键 唯一 索引 空间 全文搜索 非重复值 (DISTINCT)

5.1.2 关于整型的补零问题

zerofill 0 填充



5.1.3 整型数据的常见应用场景

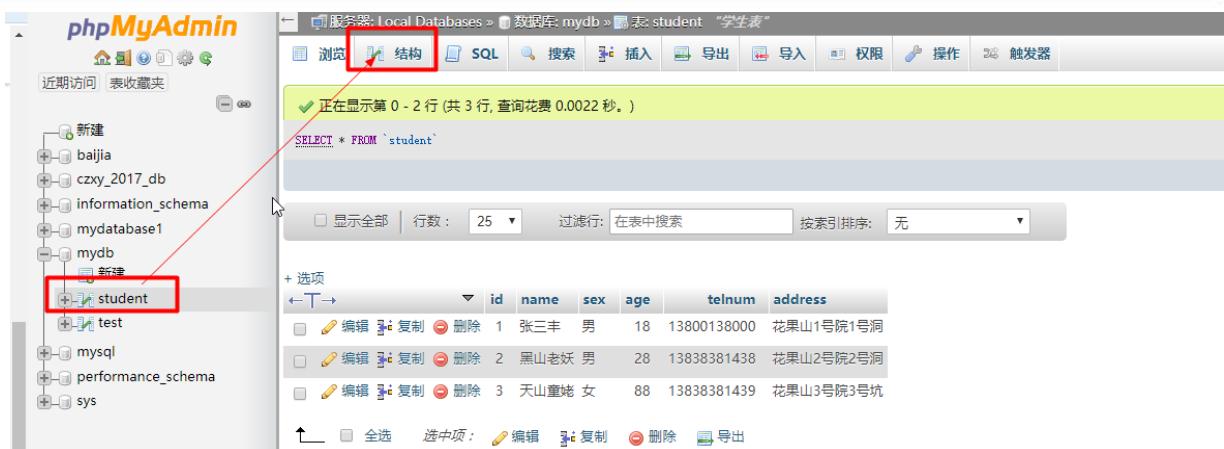
保存人的年龄（1-100）：tinyint

保存某个状态值（0、1）：tinyint

小型项目的编号：int

5.1.4 关于 id 字段的自增 auto_increment 问题

第一步：点击表名，并点击结构



正在显示第 0 - 2 行 (共 3 行, 查询花费 0.0022 秒.)

```
SELECT * FROM `student`
```

		name	sex	age	telnum	address
<input type="checkbox"/>	1	张三丰	男	18	13800138000	花果山1号院1号洞
<input type="checkbox"/>	2	黑山老妖	男	28	13838381438	花果山2号院2号洞
<input type="checkbox"/>	3	天山童姥	女	88	13838381439	花果山3号院3号坑

第二步：设置表的主键为 id

第三步：设置主键为自动增长



#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	<input checked="" type="checkbox"/> id	int(10)			否	无			<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="checkbox"/> 主键 <input type="checkbox"/> 唯一 <input type="checkbox"/> 索引 <input type="checkbox"/> 空间 <input type="checkbox"/> 全文搜索 <input type="checkbox"/> 非重复值 (DISTINCT)
2	<input type="checkbox"/> name	varchar(30)	utf8_general_ci		是	NULL			<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="checkbox"/> 主键 <input type="checkbox"/> 唯一 <input type="checkbox"/> 索引 <input type="checkbox"/> 空间 <input type="checkbox"/> 全文搜索 <input type="checkbox"/> 非重复值 (DISTINCT)
3	<input type="checkbox"/> sex	varchar(3)	utf8_general_ci		是	NULL			<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="checkbox"/> 主键 <input type="checkbox"/> 唯一 <input type="checkbox"/> 索引 <input type="checkbox"/> 空间 <input type="checkbox"/> 全文搜索 <input type="checkbox"/> 非重复值 (DISTINCT)
4	<input type="checkbox"/> age	int(10)			是	NULL			<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="checkbox"/> 主键 <input type="checkbox"/> 唯一 <input type="checkbox"/> 索引 <input type="checkbox"/> 空间 <input type="checkbox"/> 全文搜索 <input type="checkbox"/> 非重复值 (DISTINCT)
5	<input type="checkbox"/> telnum	bigint(11)			是	NULL			<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="checkbox"/> 主键 <input type="checkbox"/> 唯一 <input type="checkbox"/> 索引 <input type="checkbox"/> 空间 <input type="checkbox"/> 全文搜索 <input type="checkbox"/> 非重复值 (DISTINCT)
6	<input type="checkbox"/> address	varchar(200)	utf8_general_ci		是	NULL			<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="checkbox"/> 主键 <input type="checkbox"/> 唯一 <input type="checkbox"/> 索引 <input type="checkbox"/> 空间 <input type="checkbox"/> 全文搜索 <input type="checkbox"/> 非重复值 (DISTINCT)

然后，勾选 A.I 复选框，点击保存即可。



名字	类型	长度/值	默认	排序规则	属性	空	调整权限	A.I.	注释	Virtuality	移动字段
id	INT	10	无				<input type="checkbox"/>	<input checked="" type="checkbox"/>			



传智专修学院
CHUANZHI SPECIALIZED COLLEGE

力学笃行 志存高远

MySQL (二)

1. 浮点型

浮点型，不精确的一个大概的小数。

1.1 两种浮点型数

float: 单精度型。

double: 双精度型。

字段名称	有符号数范围	无符号数范围	占硬盘空间
float	(-3.402 823 466 E+38, -1.175 494 351 E-38), 0, (1.175 494 351 E-38, 3.402 823 466 351 E+38)	0, (1.175 494 351 E-38, 3.402 823 466 E+38)	4个字节
double	(-1.797 693 134 862 315 7 E+308, -2.225 073 858 507 201 4 E-308), 0, (2.225 073 858 507 201 4 E-308, 1.797 693 134 862 315 7 E+308)	0, (2.225 073 858 507 201 4 E-308, 1.797 693 134 862 315 7 E+308)	8个字节

1.2 浮点数的范围

可以在定义浮点数时设置 M 和 D 两个参数：

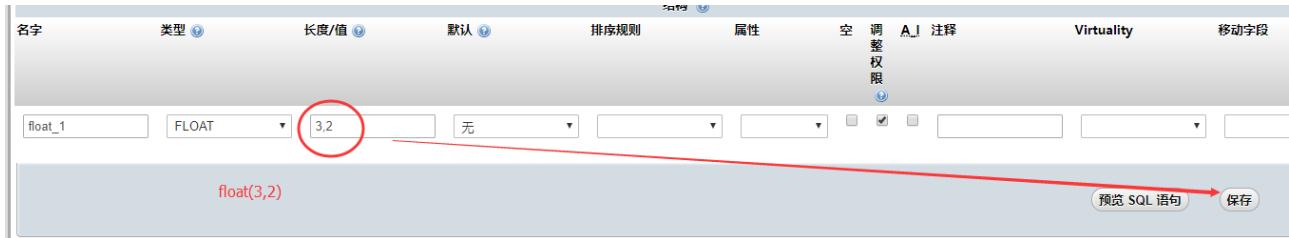
```
float(M,D)  
double(M,D)
```

M 代表总数字的位数，最大值是 255。

D 代表其中小数的位数。

比如：float(7,4) 代表最大可以保存 7 位数据，其中 4 位是小数。也就是范围为：
0.0000~999.9999

- 在 phpMyAdmin 中如何为浮点型设置范围:

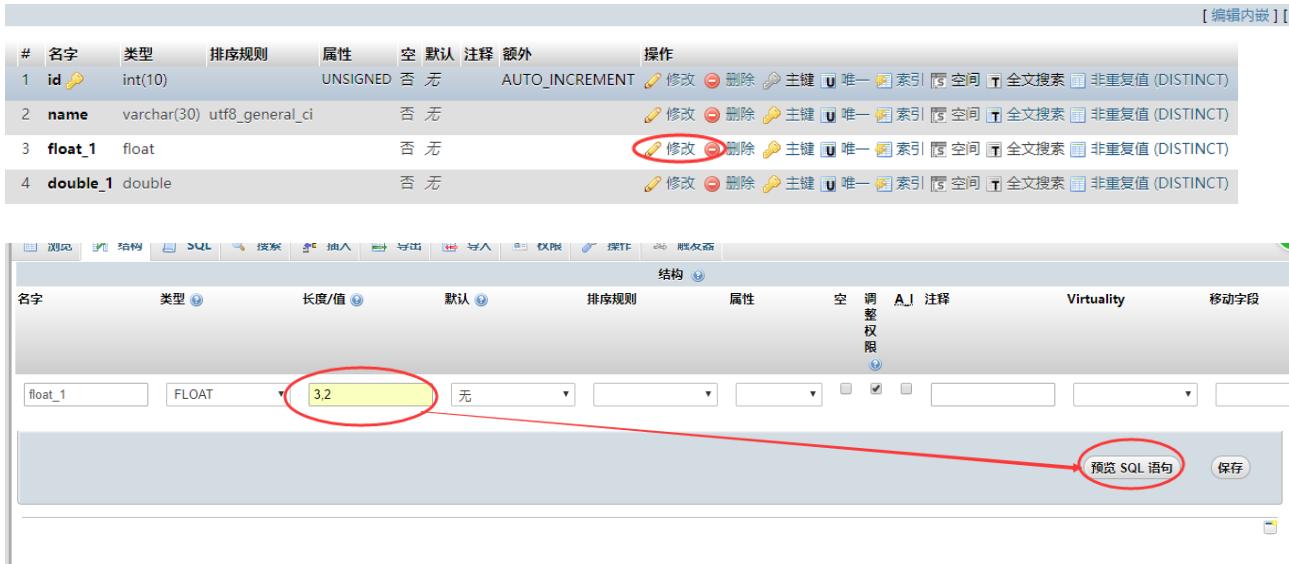


The screenshot shows the 'Structure' tab of a table in phpMyAdmin. A new column 'float_1' is being created with the type 'FLOAT'. The 'Default' field contains '3.2'. A red circle highlights this value. Below the table structure, the generated SQL code is shown: `float(3,2)`. At the bottom right, there are 'Preview SQL' and 'Save' buttons.

然而并没有什么用:

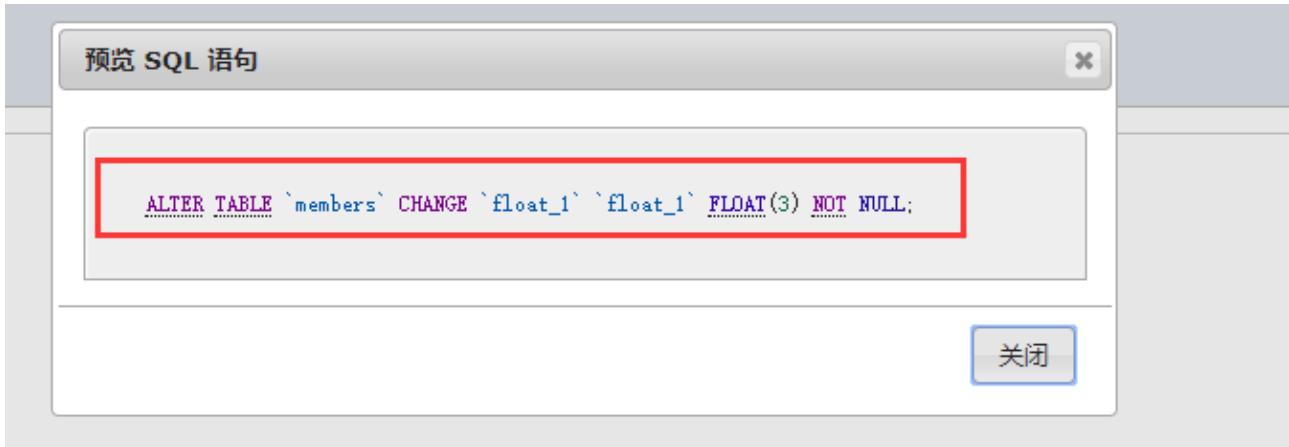
#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	id	int(10)		UNSIGNED	否	无		AUTO_INCREMENT	修改 删除 主
2	name	varchar(30)	utf8_general_ci		否	无			修改 删除 主
3	float_1	float			否	无			修改 删除 主
4	double_1	double			否	无			修改 删除 主

解决办法: 使用 SQL 语句来决定这个问题。



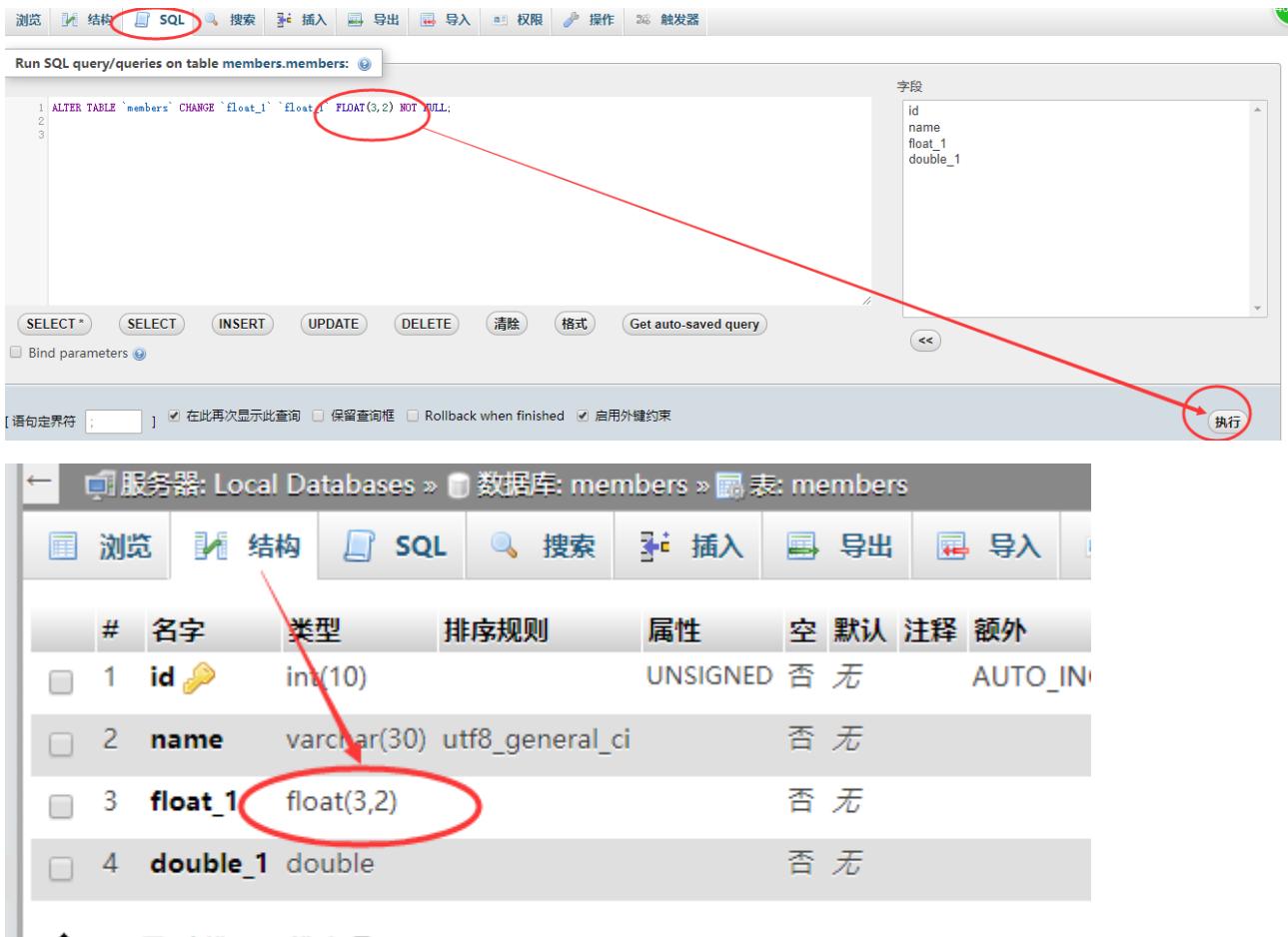
The screenshot shows the 'Structure' tab of a table in phpMyAdmin. The 'float_1' column is selected and its properties are being modified. A red circle highlights the 'Edit' button. Below the table structure, the generated SQL code is shown: `ALTER TABLE `members` CHANGE `float_1` `float_1` FLOAT(3) NOT NULL;`. At the bottom right, there is a 'Preview SQL' button highlighted with a red circle.

复制以下指令:



The screenshot shows the 'Preview SQL' dialog box. It displays the SQL command: `ALTER TABLE `members` CHANGE `float_1` `float_1` FLOAT(3) NOT NULL;`. This command adds a range constraint (FLOAT(3)) to the 'float_1' column. At the bottom right of the dialog, there is a 'Close' button.

打开 SQL 面板：



The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various tabs: 浏览 (Browse), 结构 (Structure), SQL (selected), 搜索 (Search), 插入 (Insert), 导出 (Export), 导入 (Import), 权限 (Permissions), 操作 (Operations), 触发器 (Triggers). Below the toolbar is a query editor window titled "Run SQL query/queries on table members.members:". It contains the following SQL code:

```
1 ALTER TABLE `members` CHANGE `float_1` `float` FLOAT(3,2) NOT NULL;
```

A red circle highlights the "float" keyword in the ALTER TABLE statement. A red arrow points from this circle to the "float_1" column definition in the table structure below. To the right of the query editor is a "字段" (Fields) panel showing the columns of the "members" table:

	id	name	float_1	double_1
--	----	------	---------	----------

Below the query editor are several buttons: SELECT*, SELECT, INSERT, UPDATE, DELETE, 清除 (Clear), 格式 (Format), Get auto-saved query, Bind parameters, and a "执行" (Execute) button.

The main window below shows the "members" table structure. The "结构" (Structure) tab is selected. The table has four columns:

#	名字	类型	排序规则	属性	空	默认	注释	额外
1	id	int(10)		UNSIGNED	否	无		AUTO_INCREMENT
2	name	varchar(30)	utf8_general_ci		否	无		
3	float_1	float(3,2)			否	无		
4	double_1	double			否	无		

1.3 浮点数特点

- 保存的数据并**不精确**只是一个大概的数。

price
8888880

保存 8888881 时变成了

- 小数会四舍五入。

price
2.55556

保存 2.555555 时变成了

说明：由于保存的小数并不精确，只是一个近似值，所以不适合精确数字的情况。

2. 定点数

定点数，保留以保存精确的小数。

2.1 定点数的范围

`decimal(M,D)`, M 代表总的数字位数【最大为 65】，D 代表其中的小数位。

比如：`decimal(5,2)` 代表共 5 位数字，其中 2 位是小数，比如：888.88

2.2 phpMyAdmin 中定义定点数

注意：phpMyAdmin 中设置范围的问题：



名字	类型	长度/值	默认	排序规则	属性	空	索引	A_I	注释	Virtuality
price	DECIMAL	10,2	无							

实际变成了

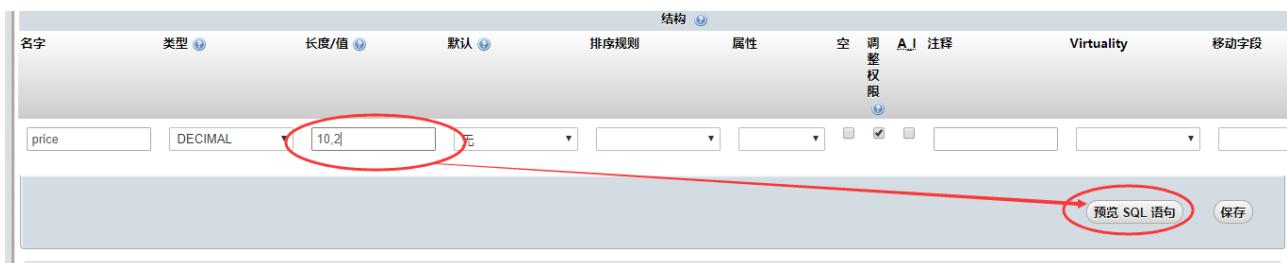


#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	id	int(10)		UNSIGNED	否	无	AUTO_INCREMENT		唯一 空间 非重复值 (DISTINCT)
2	name	varchar(30)	utf8_general_ci		否	无			唯一 空间 非重复值 (DISTINCT)
3	float_1	float(3,2)			否	无			唯一 空间 非重复值 (DISTINCT)
4	double_1	double			否	无			唯一 空间 非重复值 (DISTINCT)
5	float_2	float			否	无			唯一 空间 非重复值 (DISTINCT)
6	price	decimal(10,0)			否	无			唯一 空间 非重复值 (DISTINCT)

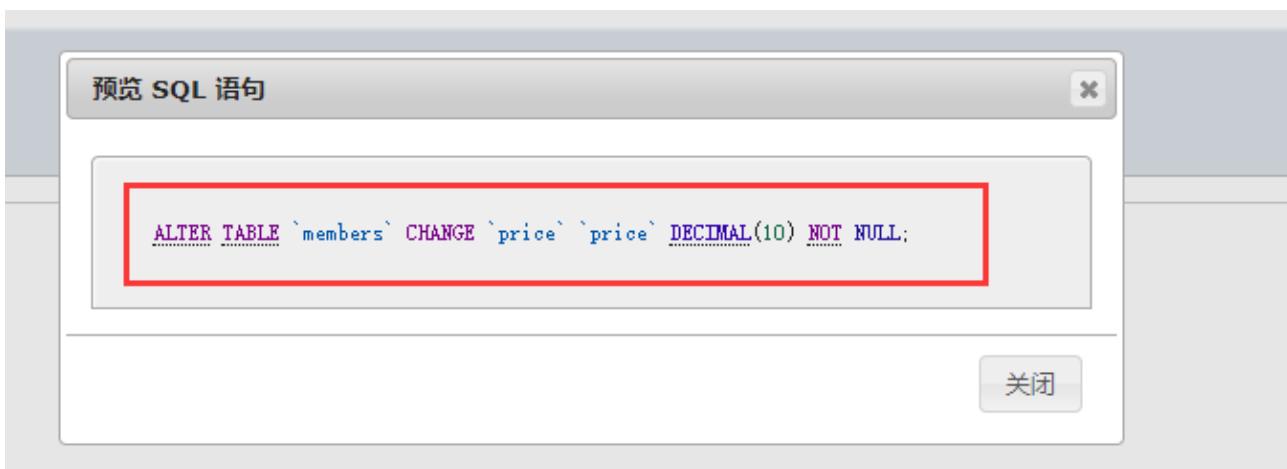
也就是说 phpMyAdmin 中设置这种长度时有问题，解决办法：使用 SQL 语句来解决：



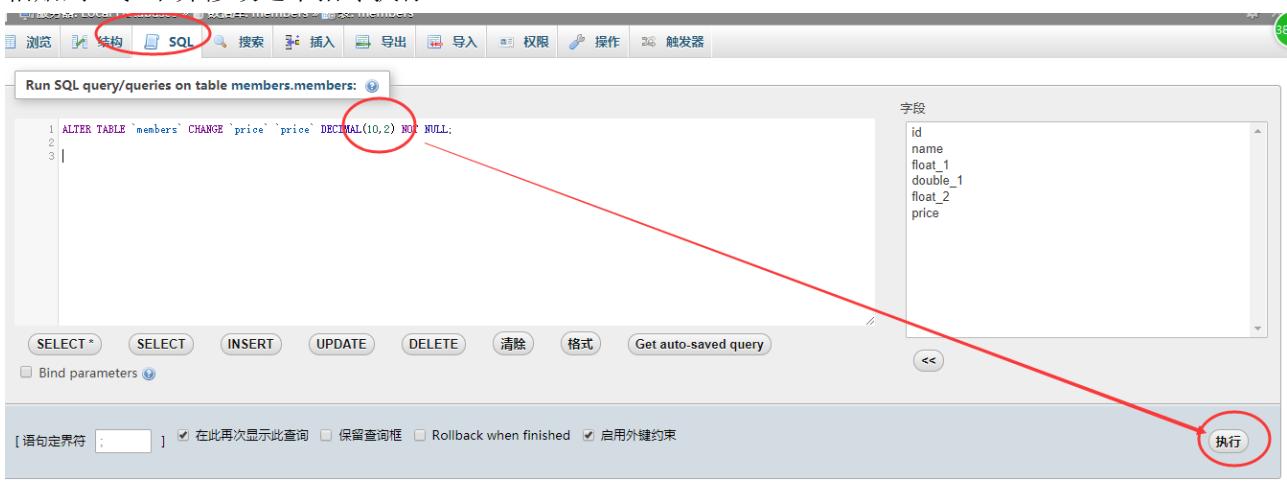
#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	id	int(10)		UNSIGNED	否	无	AUTO_INCREMENT		唯一 空间 非重复值 (DISTINCT)
2	name	varchar(30)	utf8_general_ci		否	无			唯一 空间 非重复值 (DISTINCT)
3	float_1	float(3,2)			否	无			唯一 空间 非重复值 (DISTINCT)
4	double_1	double			否	无			唯一 空间 非重复值 (DISTINCT)
5	float_2	float			否	无			唯一 空间 非重复值 (DISTINCT)
6	price	decimal(10,0)			否	无			唯一 空间 非重复值 (DISTINCT)



查看 SQL 语句并复制：



粘贴到 SQL 中并修改这个指令执行：



使用 SQL 就可以了：



The screenshot shows a MySQL Workbench interface. The top menu bar includes '浏览' (Browse), '结构' (Structure) (which is highlighted with a red circle), 'SQL', '搜索' (Search), '插入' (Insert), '导出' (Export), '导入' (Import), '权限' (Permissions), '操作' (Operations), and '触发器' (Triggers). Below the menu is a table structure view with columns: #, 名字 (Name), 类型 (Type), 排序规则 (Sort Rule), 属性 (Properties), 空 (Null), 默认 (Default), 注释 (Comment), 额外 (Extra), and 操作 (Operations). The table contains six rows:

#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	id	int(10)		UNSIGNED	否	无		AUTO_INCREMENT	修改 删除 主键
2	name	varchar(30)	utf8_general_ci		否	无			修改 删除 主键
3	float_1	float(3,2)			否	无			修改 删除 主键
4	double_1	double			否	无			修改 删除 主键
5	float_2	float			否	无			修改 删除 主键
6	price	decimal(10,2)			否	无			修改 删除 主键

Below the table are several buttons: 全选 (Select All), 选中项 (Selected Item), 浏览 (Browse), 编辑 (Edit), 删除 (Delete), 主键 (Primary Key), 唯一 (Unique), 索引 (Index), 全文搜索 (Full Text Search), 打印 (Print), 规划表结构 (Plan Table Structure), 移动字段 (Move Field), and Improve table structure.

2.3 应用场景

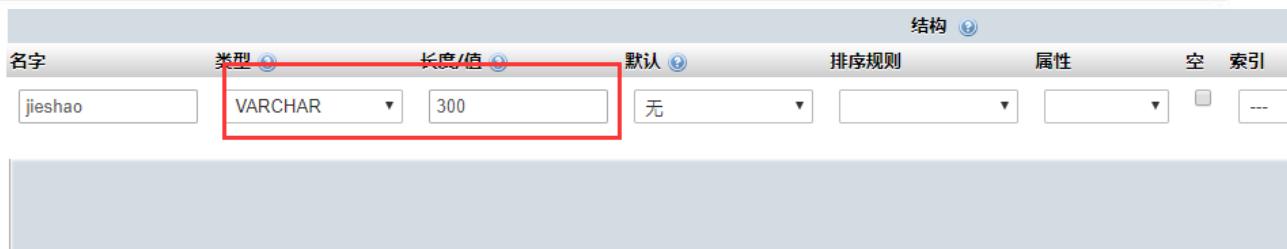
用在需要精确的小数时，比如价格。

3. 字符串

3.1 几种字符串

字段名称	长度	用途
char	最大 255 个字符。	255 个字符以下的字符串。
varchar	最大 65535 个字节。	65535 个字符以下的字符串。
tinytext	最多 255 个字节。	
text	最多 65535 个字节。	
mediumtext	最多 16777215 个字节。	大段文本时，比如新闻、文章、论文等。
longtext	最多 4294967295 个字节。	

示例：定义一个只能保存 300 个汉字的字段；



3.2 char 和 varchar 的区别

3.2.1 最大长度

char 最多能保存 255 个字符【无论中文还是英文还是任何符号】。

比如：可以保存 255 个汉字、英文字母、数字、符号等等。

varchar 最多能保存 65535 个字节 【UTF8 编码时，一个字符占 3 个字节;GBK 编码时一个字符占 2 个字节】。

最多保存多个字符呢？

utf8 时

大约是 $65535/3$ 个=2 万多个字符

gbk 时

大约是 $65535/2$ 个=3 万多个字符

3.2.2 占硬盘空间

● char: 定长字符串

char 也叫做定长字符串，指的是在创建表时，char 字段占用硬盘空间的大小就已经固定了。比如，我们定义 name char(10)，代表 name 字段将占 10 个字符的硬盘空间，那么具体是多少个字节要看字段是什么编码的，如果是 utf8 编码，那么一个字符占 3 个字节，所以 char(10) 将占 30 个字节，无论内容够不够 10 个字节都占 30 个字节，比如只存 abc 三个字符也要占 30 个字节。

● varchar: 变长字符串

`varchar` 也叫做变长字符串，指的是字段占用硬盘空间的大小并不是固定的，而是由内容决定的，等于内容的长度+1 个字节。

比如，我们定义 `name varchar(10)`, 代表最多保存 10 个字符，如果表是 `utf8` 编码，则最多占用空间 30 个字节，但是，如果内容不够 10 个字符，比如内容是 `abc`，那么只占三个字符 9 字节再+1，共 10 占 10 个字节。

● 对比

表是 `utf8` 编码：

`name char(10)` 如果内容是 `abc`，占用硬盘空间为 30 个字节。

`name varchar(10)` 如果内容是 `abc`，占用硬盘空间为 10 个字节。

总结：

1. `char` 定义时是多少，就占多少，无论内容多长。
2. `varchar` 定义时定义的是最大长度，而实际占有的空间由内容的长度+1 决定，--》节省空间

如何选择使用哪个？

1. 如果字符串的长度都是固定的可以用 `char`
2. 如果字符串内容的长度不固定，比如商品名称，每件商品的名字长度都不同，所以可变确定，这时使用 `varchar`

3.3 `char`、`varchar` 和 `text` 应该如何选择

选择的原则：

1. `varchar`: 长度不固定时使用，比如商品名称、用户名称、文章标题、家庭住址、简短的评论等。
2. `char`: 长度固定时使用，应用场景较少，比如 IP 地址、成语、MD5 之后的密码等。
3. `text`: 大文本时使用，比如：文章的内容、小说的内容、论文、商品描述、帖子内容等。

4. 枚举

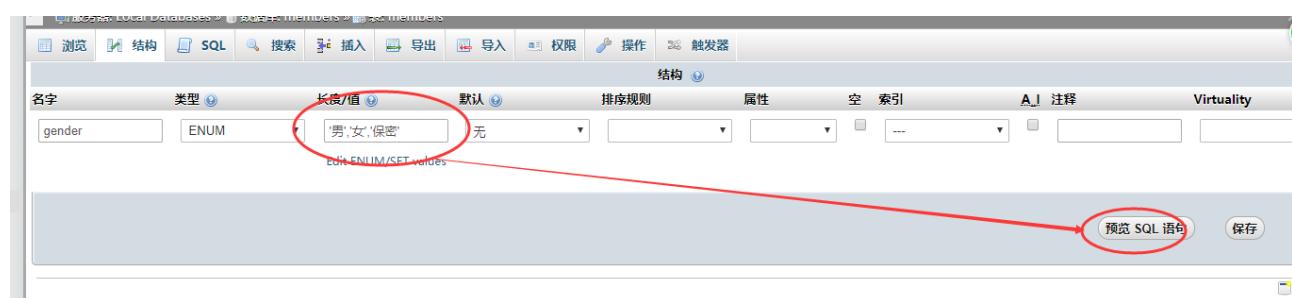
4.1 什么是枚举？

枚举类型【enum】，在定义字段时就预告规定好固定的几个值，然后插入记录时值只能这几个固定好的值中选择一个。

语法：

```
gender enum('男','女','保密')
```

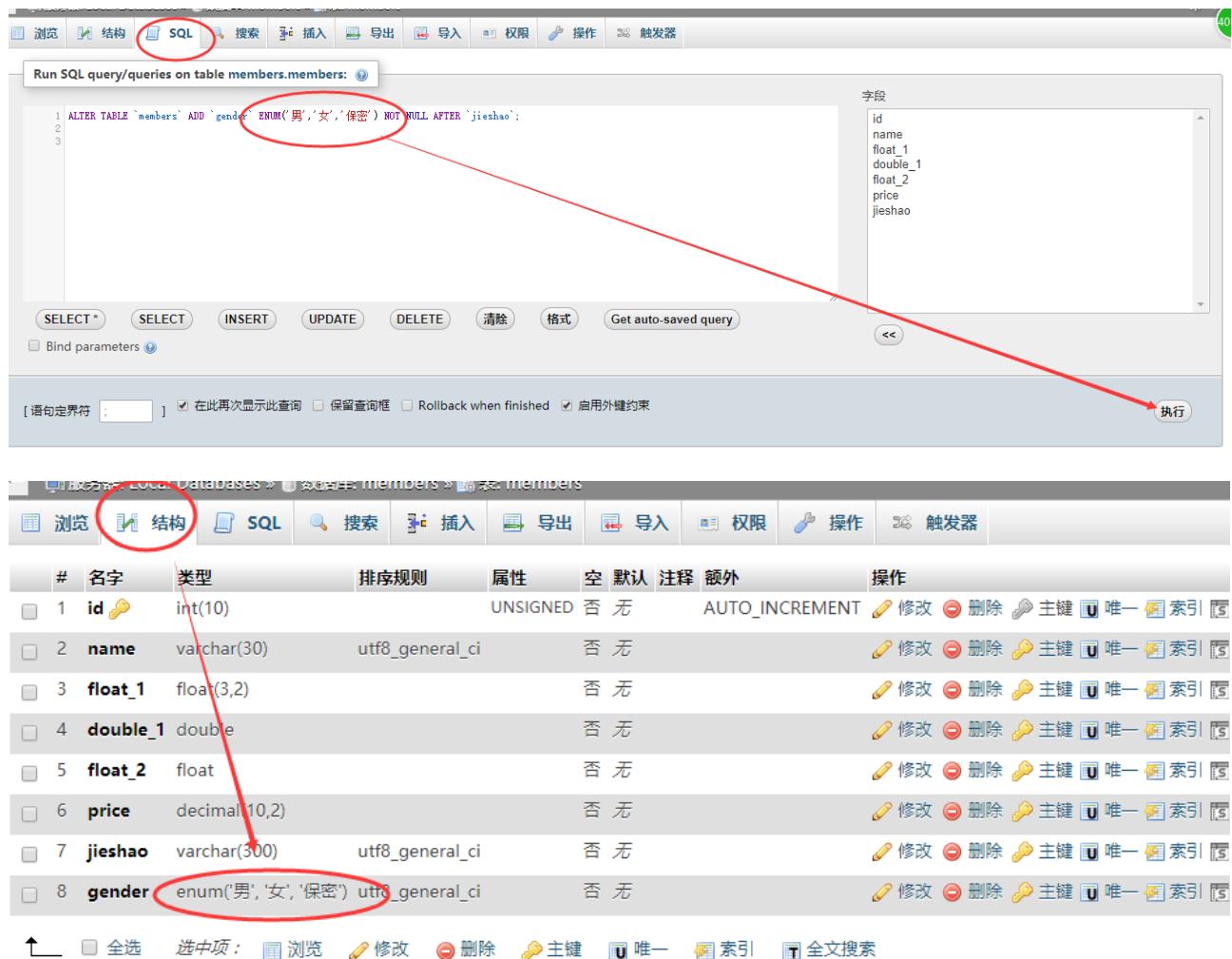
4.2 phpMyAdmin 中使用 enum



查看 SQL 语句并复制



到 SQL 面板中粘贴并修改执行：



Run SQL query/queries on table members.members:

```

1 ALTER TABLE `members` ADD `gender` ENUM('男','女','保密') NOT NULL AFTER `jieshao`;
2
3
    
```

Fields

id
name
float_1
double_1
float_2
price
jieshao

SELECT * | SELECT | INSERT | UPDATE | DELETE | Clear | Format | Get auto-saved query | Bind parameters | << | >> | Execute

[Statement delimiter :] Show this query again Save query Rollback when finished Enable foreign key constraint

MySQL Databases Local Databases > 表名: members > 表: members

浏览 结构 SQL 搜索 插入 导出 权限 操作 触发器

#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	id	int(10)		UNSIGNED	否	无	AUTO_INCREMENT		
2	name	varchar(30)	utf8_general_ci		否	无			
3	float_1	float(3,2)			否	无			
4	double_1	double			否	无			
5	float_2	float			否	无			
6	price	decimal(10,2)			否	无			
7	jieshao	varchar(300)	utf8_general_ci		否	无			
8	gender	enum('男','女','保密')	utf8_general_ci		否	无			

↑ 全选 选中项: 浏览 唯一 索引 全文搜索

这个字段里面的值，只能是这三个值之一。

4.3 特点

1. 一个 enum 最多可以设置 65535 个值
2. 这个字段最终只占 1 到 2 个字节【要看设置值的多少】【节省空间】

4.4 应用场景

当值是几个固定可选时，比如：性别、星期、月份、表示状态时【比如是、否】。

5. 集合

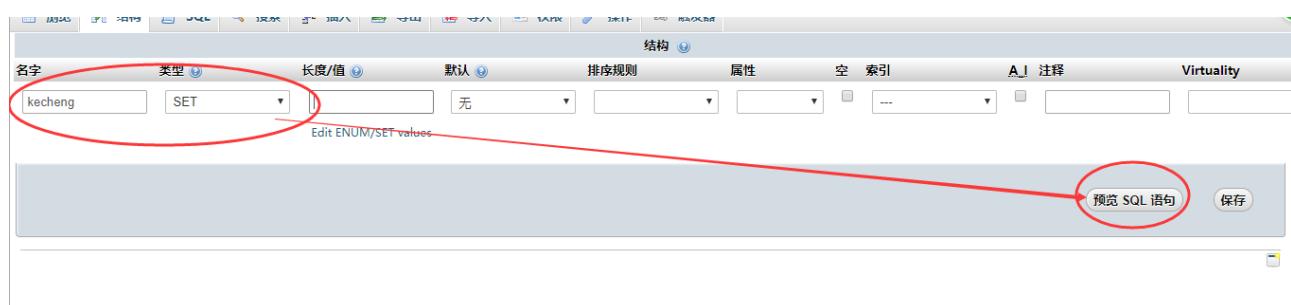
5.1 什么是集合

集合【set】，在定义字段时预告规定好几个值，然后在添加记录时可以从中选出多个做为值，多个值之间用,隔开。

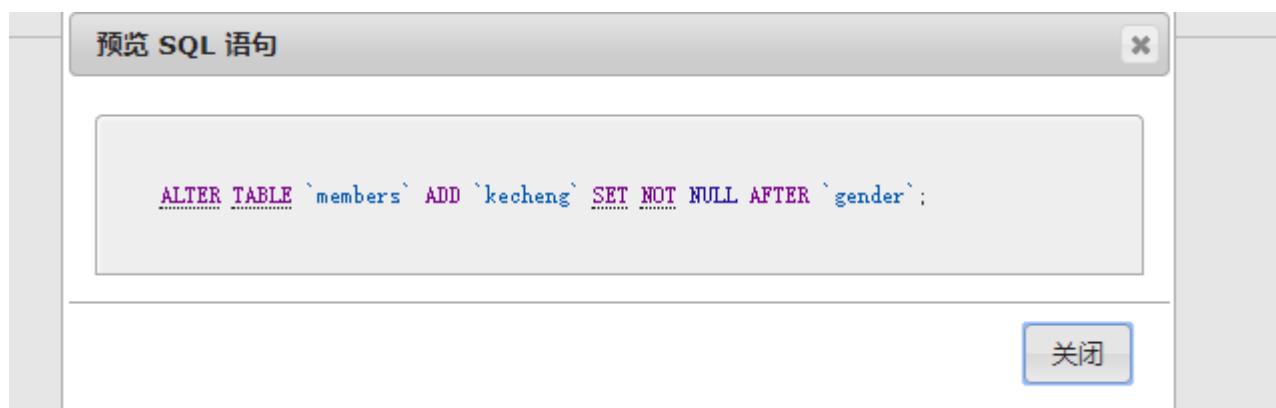
语法：

```
字段名 set('值 1','值 2','值 3'....)
```

5.2 在 phpMyAdmin 中设置集合



复制 SQL:



粘贴到 SQL 面板中并修改执行：

SQL tab circled in red.

```
Run SQL query/queries on table members.members: ⓘ
1 ALTER TABLE `members` ADD `kecheng` SET('html','css','javascript','jquery','bootstrap','mysql','php') NOT NULL AFTER `gender`;
```

Fields panel on the right shows:

id
name
float_1
double_1
float_2
price
jieshao
gender

Buttons below the SQL panel:

- SELECT*
- SELECT
- INSERT
- UPDATE
- DELETE
- 清除
- 格式
- Get auto-saved query

Checkboxes at the bottom:

- [语句定界符: ;]
- 在此再次显示此查询
- 保留查询框
- Rollback when finished
- 启用外键约束

Execution button on the far right.

Structure tab circled in red.

#	名字	类型	排序规则	属性	空	默认	注释
1	id	int(10)		UNSIGNED	否	无	
2	name	varchar(30)	utf8_general_ci		否	无	
3	float_1	float(3,2)			否	无	
4	double_1	double			否	无	
5	float_2	float			否	无	
6	price	decimal(10,2)			否	无	
7	jieshao	varchar(300)	utf8_general_ci		否	无	
8	gender	enum('男','女','保密')	utf8_general_ci		否	无	
9	kecheng	set('html','css','javascript','jquery','bootstrap','mysql','php')	utf8_general_ci		否	无	

添加记录时可以从这些值里面选择多个:

kecheng field values circled in red.

选项	ID	id	name	float_1	double_1	float_2	price	jieshao	gender	kecheng
<input type="checkbox"/>	1	比尔盖	6565.23	9843.54	0	0.00		男		
<input type="checkbox"/>	2	孙司空	3.56	333	8888900	9.99		保密		html,css,javascript,jquery,bootstrap,php

5.3 特点

- 集合中最多定义 64 个值

2. 占硬盘空间数量为：1, 2, 3, 4 或者 8 个字节，由值的个数决定

6. 时间类型

MySQL 提供了几个专门用来保存时间的类型。

6.1 几种时间类型

字段名称	格式
date	年-月-日 。 比如 2010-10-10
datetime	年-月-日 时:分:秒 。 比如： 2010-10-10 10:10:10
timestamp	年-月-日 时:分:秒 。 比如： 2010-10-10 10:10:10
time	时:分:秒 。 比如： 10:10:10
year	年 。 比如： 2010

添加一个日期字段：



可以在表中保存日期：

id	name	float_1	double_1	float_2	price	jieshao	gender	kecheng	birthday
1	比尔盖	6565.23	9843.54	0	0.00		男		2017-08-14
2	孙司空	3.56	333	8888900	9.99		保密	html,css,javascript,jquery,bootstrap,php	2017-08-10

6.2 datetime 和 timestamp 的区别

1. 范围不同

datetime 保存时间的范围: '1000-01-01 00:00:00' 到 '9999-12-31 23:59:59'
timestamp 保存时间的范围: '1970-01-01 00:00:01' 到 '2038-01-19 03:14:07'

2. 存储空间不同

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
<u>YEAR</u>	1 byte	1 byte
<u>DATE</u>	3 bytes	3 bytes
<u>TIME</u>	3 bytes	3 bytes + fractional seconds storage
<u>DATETIME</u>	8 bytes	5 bytes + fractional seconds storage
<u>TIMESTAMP</u>	4 bytes	4 bytes + fractional seconds storage

6.3 自动当前时间

可以在默认属性中设置 current_timestamp, 这时, 这个字段的值会被自动添加入当前时间。

只对 timestamp 和 datetime 这两种时间类型有效。



7. 字段属性

每个字段除了可以设置名称、数据类型之外还可以设置其他信息，比如属性。

7.1 默认值

可以为每个字段设置一个默认值，当没有为这个字段设置值时，就用这个默认值填充：

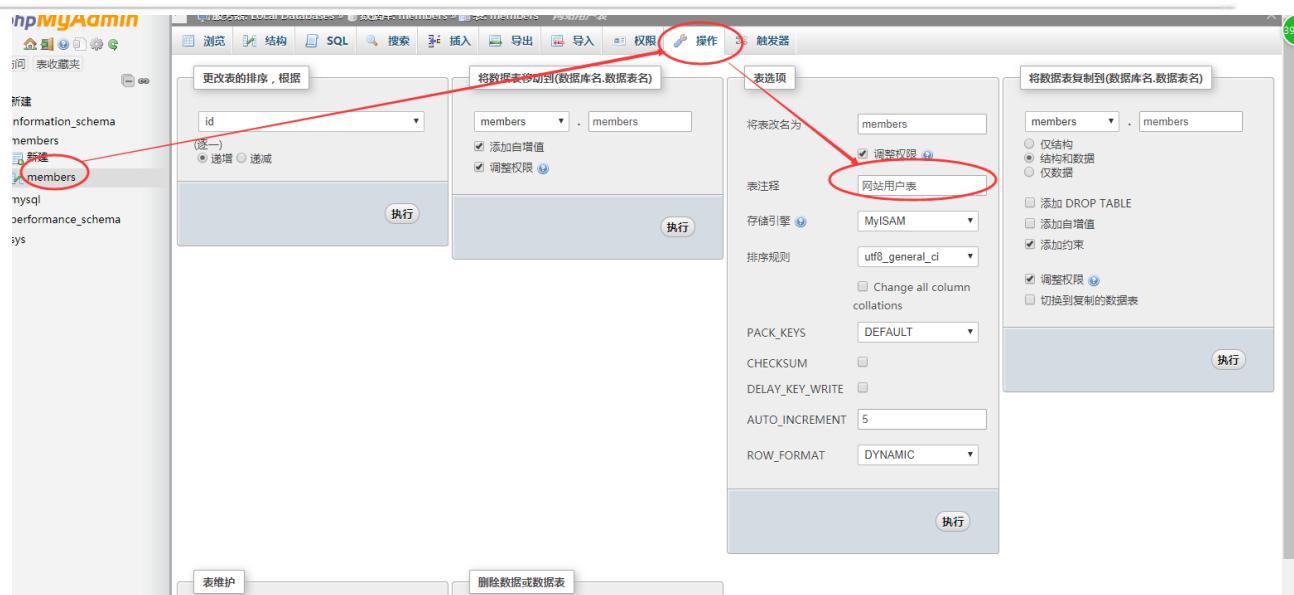
名字	类型	长度/值	默认	排序规则
age	INT	11	定义： 10	

7.2 注释

每个字段可以使用注释对字段进行注解说明；

名字	类型	长度/值	默认	排序规则	属性	空	调整权限	AJ	注释	Virtuality
age	INT	11	定义： 10				<input checked="" type="checkbox"/>	<input type="checkbox"/>	年龄字段	

可以为表也添加注释：



7.3 是否为 null

可以设置一个值中的值是否可以是 null【空】:

勾选: 可以为 null。

不勾选: 不允许为 null。



#	名字	类型	排序规则	属性	空	默认	注释	额外	操作
1	id 📄	int(10)		UNSIGNED	否	无		AUTO_INCREMENT	修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
2	name	varchar(30)		utf8_general_ci	否	无名氏			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
3	float_1	float(3,2)			否	无			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
4	double_1	double			否	无			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
5	float_2	float			否	无			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
6	price	decimal(10,2)			否	无			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
7	jieshao	varchar(300)		utf8_general_ci	否	无	个人信息介绍		修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
8	gender	enum('男', '女', '保密')		utf8_general_ci	否	无			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
9	kecheng	set('html', 'css', 'javascript', 'jquery', 'bootstrap')		utf8_general_ci	否	无			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
10	birthday	date			否	无			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多
11	regtime	timestamp			否	CURRENT_TIMESTAMP			修改 ⚒ 删除 ⚒ 主键 🖤 唯一 🔍 索引 🏃 空间 ▾ 更多

8. 编码和校对规则

不同的编码支持不同的字符集，所以要根据表中要保存的数据来选择编码。

8.1 支持中文的编码

gb2312：简体中文。

big5：繁体中文。

GBK：简体+繁体中文。

utf8：万国码，包含所有其他字符集。

优点：包含所有字符集。

缺点：同样的字符占的硬盘空间更大。

我们一般选择 utf8 编码。

8.2 库、表、字段的编码

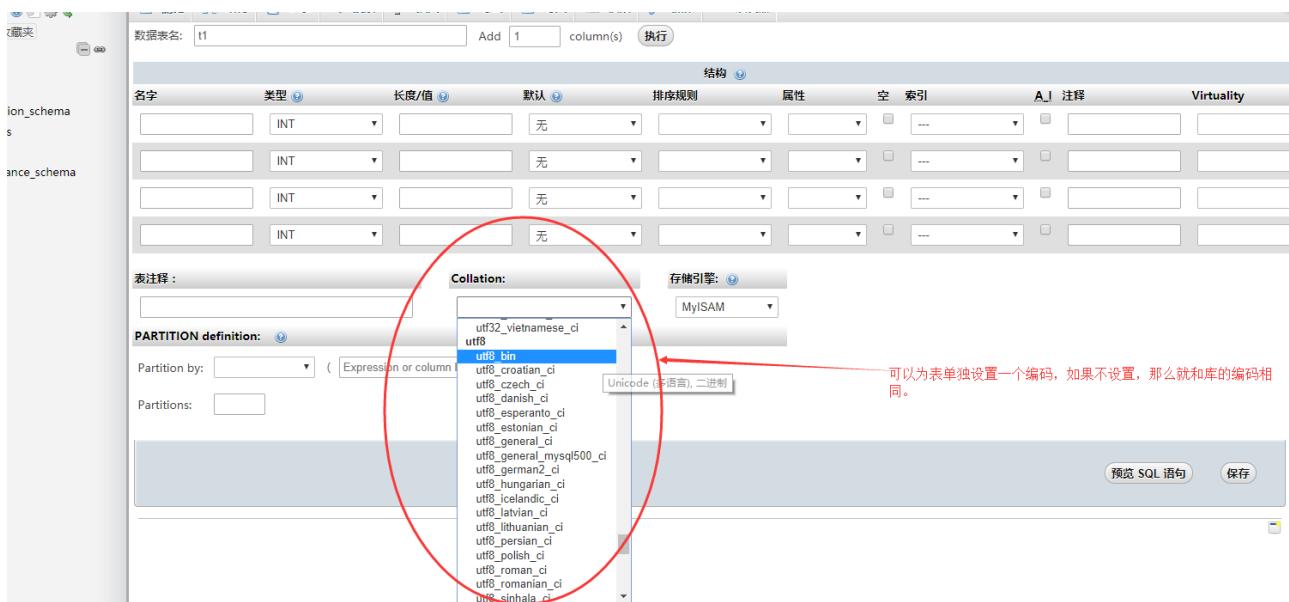
可以分别为库、表和每个字段设置不同的编码和校对规则，一般我们都设置为统一的。

技术：当只有库设置编码时，库中所有表和字段默认和库编码一致。

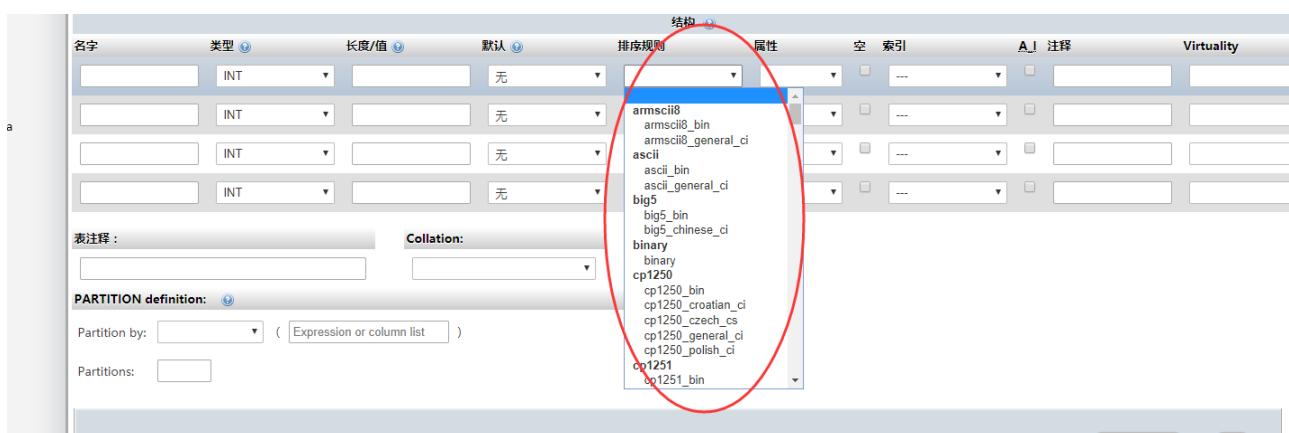
- 库可以设置编码：



- 表也可以设置编码，如果不设置就和库的编码相同：



- 表中每个字段还可以单独设置一个编码，如果不设置就和表的编码一致



8.3 校对规则

校对规则就是排序、比较时的规则，常用的两个：

utf8_general_ci：区分大小写，比如 a 等于 A。

utf8_bin：区分大小写，比如 a 并不等于 A。

9. 表的尺寸限制

MySQL 中规定一个表中所有字段的尺寸加到一起不能超过 **65535** 个字节。

9.1 计算表的字节

表名: members	编码: utf8	
n1 tinyint	--->	1
n2 int	--->	4
n3 varchar(200)	--->	600
n4 varchar(2000)	--->	60000
n5 varchar(2000)	--->	6000 <---失败无法创建这个字段了，尺 寸超了

9.2 在表中保存大字段

当表中需要使用超过 65535 个字节时，应该把 `varchar` 改成 `text`。因为 `text` 字段比较特殊，它中的数据并不是保存在表中的，无论 `text` 中有多少的数据，它都只占这个表的 9~12 个字节，所以一个表中可以有大量的 `text`，当然注意 `text` 速度比较慢，当不保存大字符串时不要使用。

10. 案例、CMS 系统建表

创建两张表，表名分别是：authors【作家表】、articles【文章表】。

author【作家表】：

字段名称	字段类型	字段含义	是否为空	是否主键	备注
id	int (10)	编号	否	是	自动增长
username	varchar (50)	作家账号名称			
passwd	char (32)	密码			md5,32位
user_type	enum('个人','媒体')	作家类型			个人、媒体
author_domain	varchar (255)	作家领域			
nicheng	varchar (50)	作家昵称			
head_img	varchar (255)	作家头像			图片名称
personal_introduction	varchar (255)	作家签名			
province	varchar (25)	所在省			
city	varchar (30)	所在市			
county	varchar (30)	所在县			
phone_num	bigint	手机号码			
invite_code	varchar (64)	邀请码			
author_state	tinyint (1)	作者状态			1-正常 0-禁用 2-已经选择完作家类型 3-已经完善作家信息
operator_name	varchar (50)	运营者姓名			
operator_ids	char (18)	运营者身份证			
operator_img	varchar (255)	运营者照片			
add_time	datetime	添加时间			
last_update_time	datetime	最后更新时间			
last_update_user	varchar (50)	最后更新用户			

articles【文章表】：

字段名称	字段类型	字段含义	是否为空	是否主键	备注

id	int (10)	编号	否	是	自动增长
article_type_id	int (10)	文章类型 id			
author_id	tinyint (10)	作者编号			
title	varchar(255)	标题			
description	varchar(255)	文章简述			
article_content	text	文章正文内容			
tags	varchar(255)	标签			
article_img	varchar(255)	文章图片			
video_url	varchar(255)	视频链接			
read_num	mediumint	阅读次数			
favorable_num	mediumint	好评次数			
difference_assessment	mediumint	差评次数			
add_time	datetime	添加时间			
state	tinyint (1)	文章状态			
is_top	tinyint (1)	是否置顶			
last_update_time	datetime	最后更新时间			
last_update_user	varchar (50)	最后更新人			

11. 今日总结



传智专修学院
CHUANZHI SPECIALIZED COLLEGE

力学笃行 志存高远

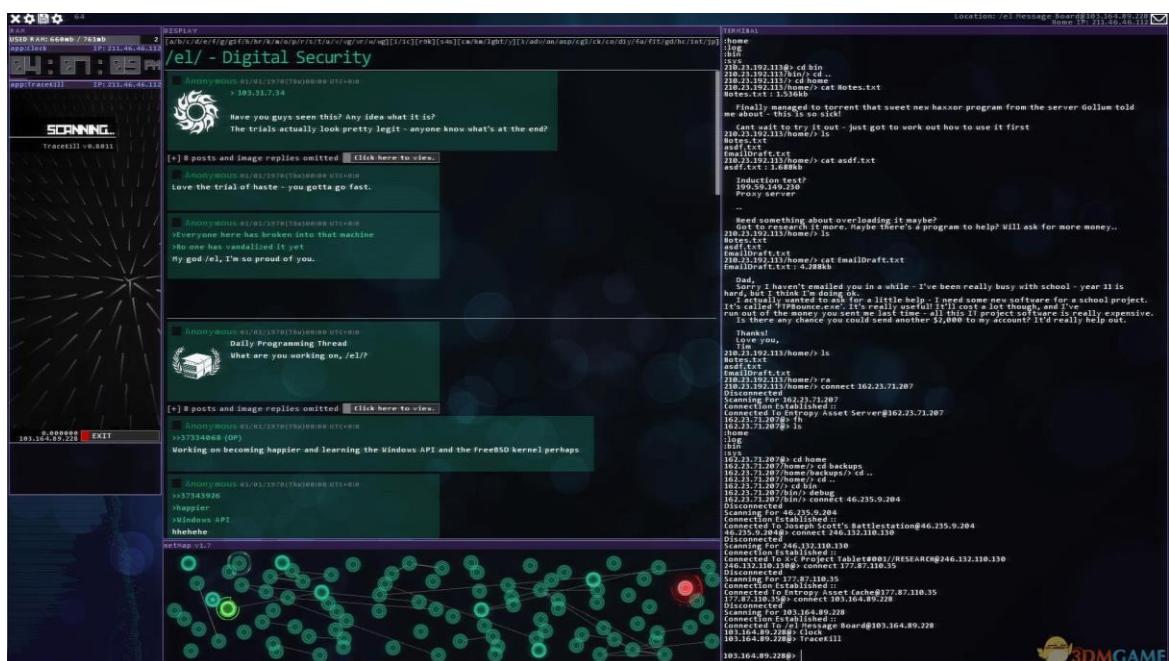


MySQL (三)

全球著名的勒索病毒：



攻击指令：这些指令用来操控一台或多台计算机

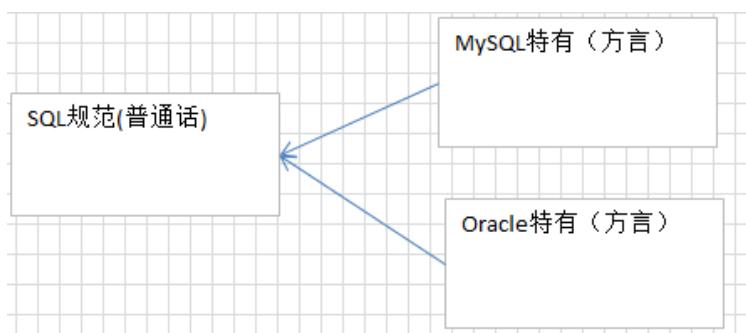


1. SQL 概述

1.1 SQL 语句介绍

数据库是不认识 PHP 语言的，但是我们同样要与数据库交互，这时需要使用到数据库认识的语言 SQL 语句，它是数据库的代码。

结构化查询语言(Structured Query Language)简称 SQL，是关系型数据库管理系统都需要遵循的规范。不同的数据库生产厂商都支持 SQL 语句，但都有特有内容。



1.2 SQL 语句分类

1.2.1 数据定义语言

简称 DDL(Data Definition Language)，用来定义数据库对象：数据库 database，表 table，列 column 等。关键字：创建 create，修改 alter，删除 drop 等（结构）

1.2.2 数据操作语言

简称 DML(Data Manipulation Language)，用来对数据库中表的记录进行更新。关键字：插入 insert，删除 delete，更新 update 等（数据）

1.2.3 数据查询语言

简称 DQL(Data Query Language), 用来查询数据库中表的记录。关键字: select, from, where 等

1.2.4 数据控制语言

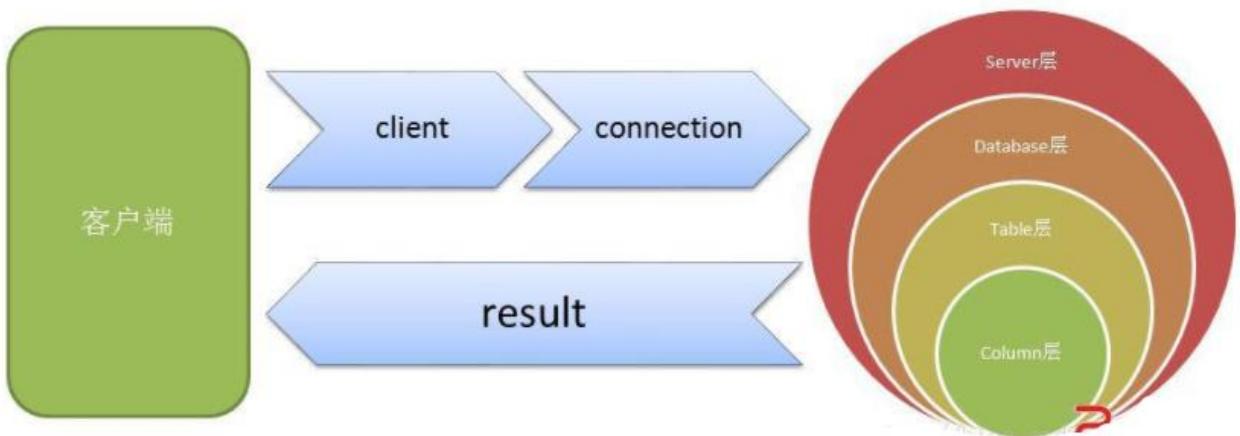
简称 DCL(Data Control Language), 用来定义数据库的访问权限和安全级别, 及创建用户。

1.3 SQL 通用语法

```
SELECT * FROM `student`;
```

- SQL 语句可以单行或多行书写, 以分号结尾
- 一般会使用空格和缩进来增强语句的可读性
- MySQL 数据库的 SQL 语句不区分大小写, 关键字建议使用大写
例如: SELECT * FROM user。
- 可以使用 “--” 的方式进行注释

2. MySQL 客户端



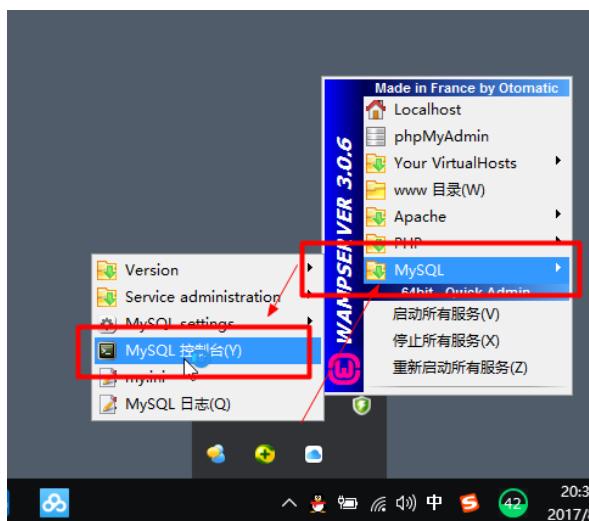
MySQL 是一个需要账户名密码登录的数据库, 登陆后使用, 它提供了一个默认的 root 账号, 使用安装时设置的密码即可登录。

mysql.exe 客户端 (mysql 软件安装后, 自带的客户端)

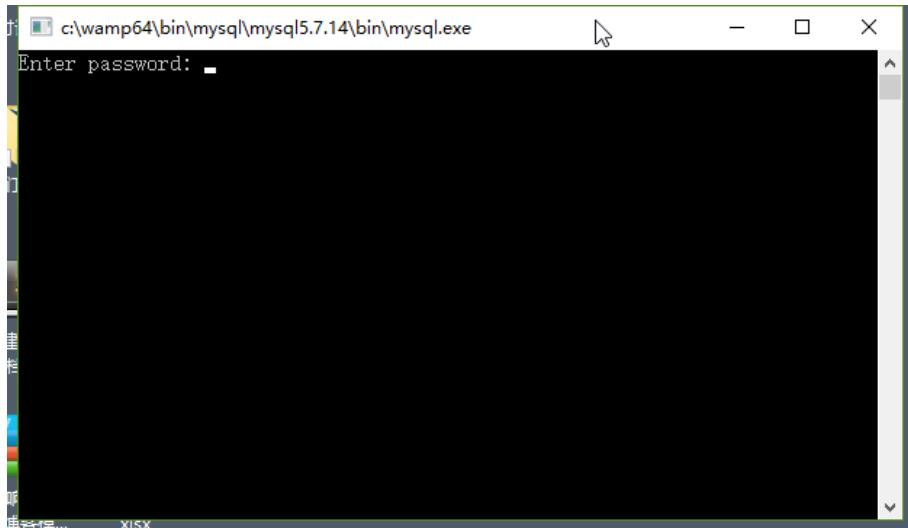
mysqld.exe 服务端

方式一：wamp 的方式登录：

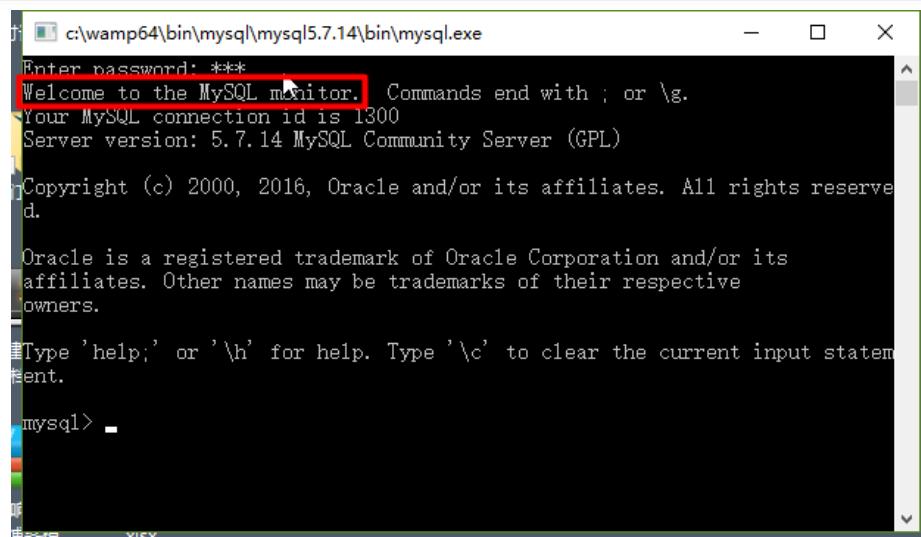
第一步：打开 MySQL 客户端



第二步：输入密码



第三步：输入密码登录



```
c:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe
Enter password: ***
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1300
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

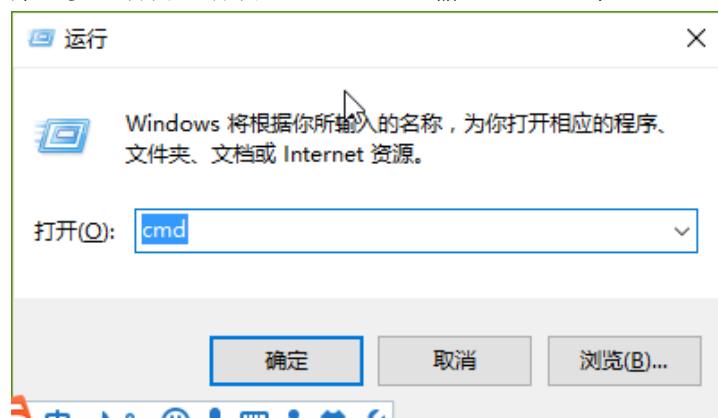
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help,' or '\h' for help. Type '\c' to clear the current input statement.

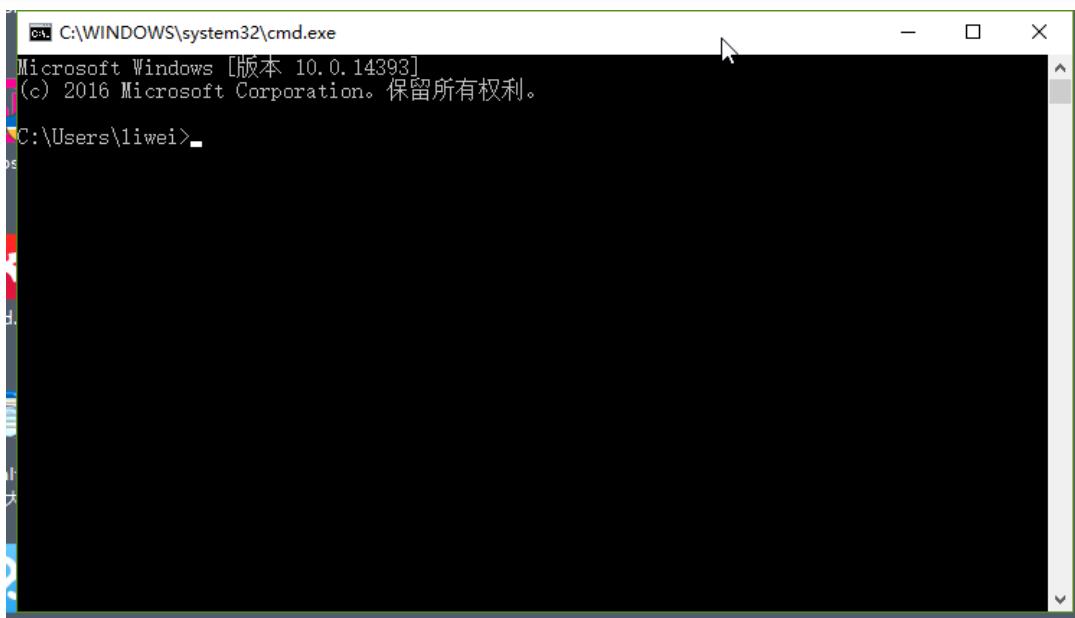
mysql> -
```

方式二：Dos 命令行中打开

第一步：打开运行窗口（win+R），输入 CMD 回车



第二步：打开 Dos 命令行



第三步：登录 mysql 服务器

格式 1: cmd> mysql.exe 路径 -u 用户名 -p 密码

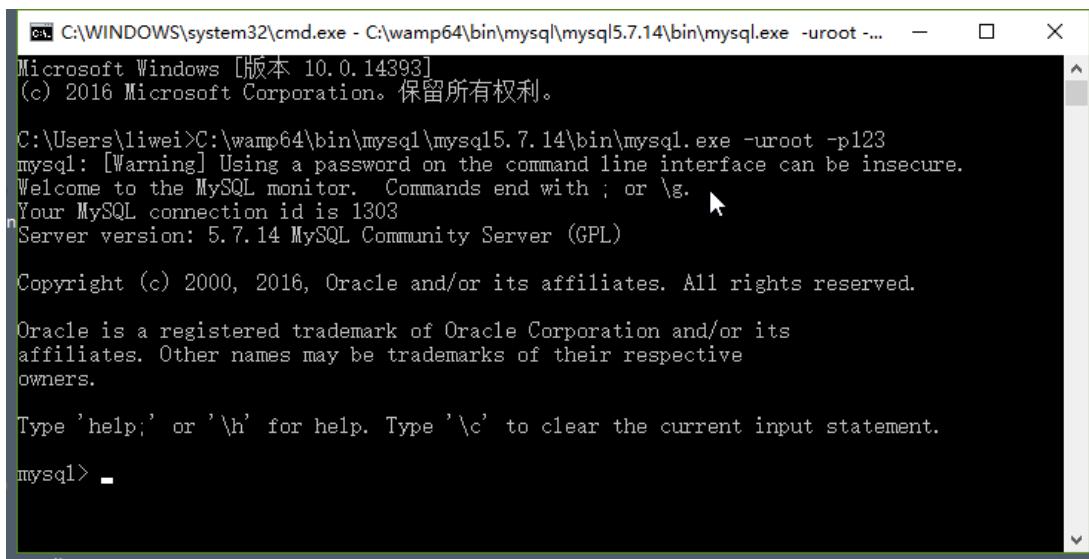
或者: cmd> mysql.exe 路径 -u 用户名 -p 回车

输入密码: ***

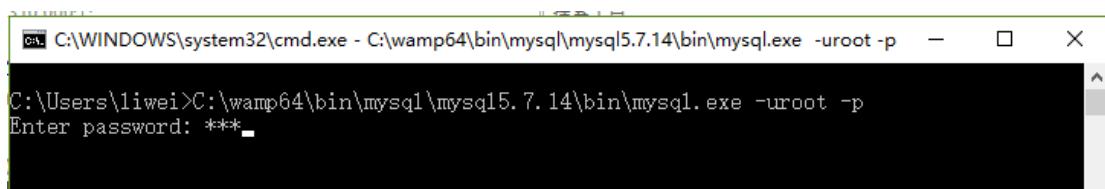
例如: C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p123

输入 mysqld.exe 的文件路径

如: C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p123

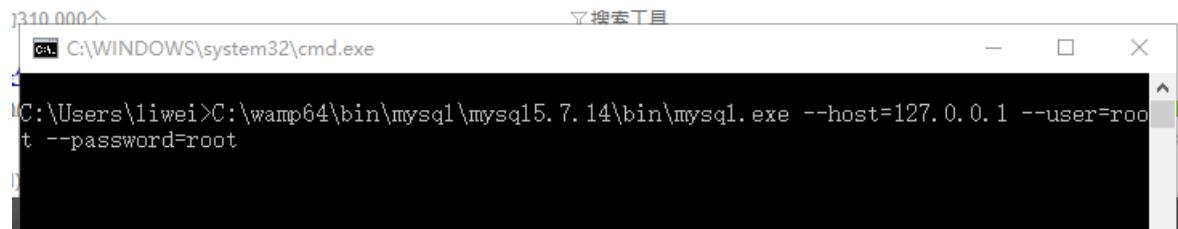


输入时也可以隐藏密码:



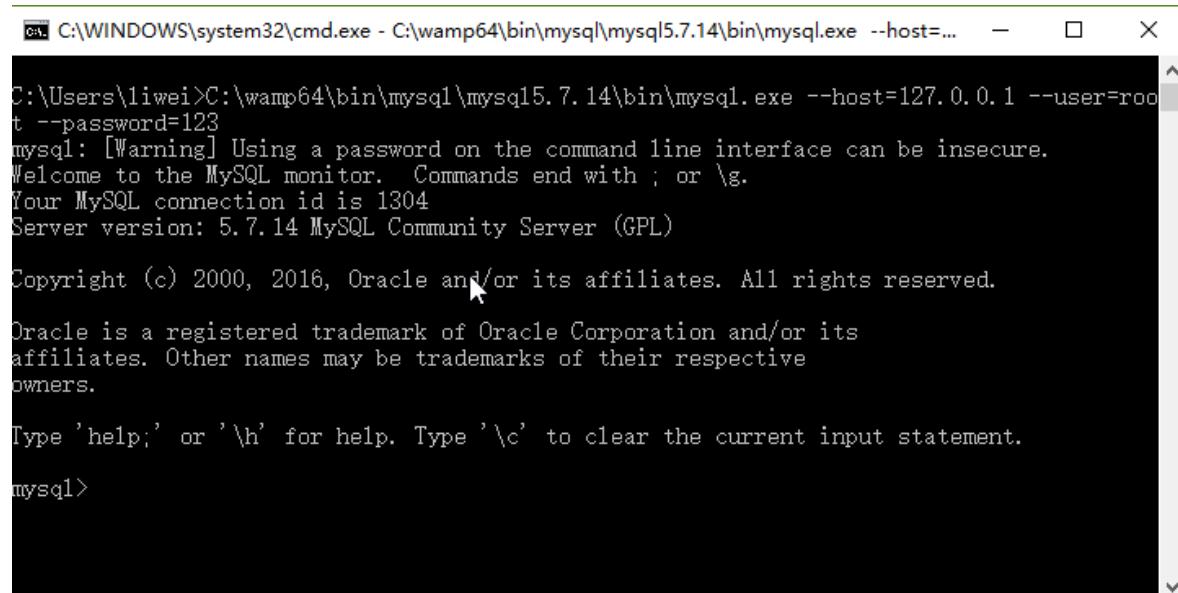
```
C:\WINDOWS\system32\cmd.exe - C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p
C:\Users\liwei>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p
Enter password: ***
```

格式 2: cmd> mysql.exe 路径 --host=ip 地址 --user=用户名 --password=密码
例如: C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe --host=127.0.0.1
--user=root --password=123



```
C:\WINDOWS\system32\cmd.exe
C:\Users\liwei>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe --host=127.0.0.1 --user=root --password=root
```

登录成功!



```
C:\WINDOWS\system32\cmd.exe - C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe --host=... -p
C:\Users\liwei>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe --host=127.0.0.1 --user=root --password=123
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1304
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

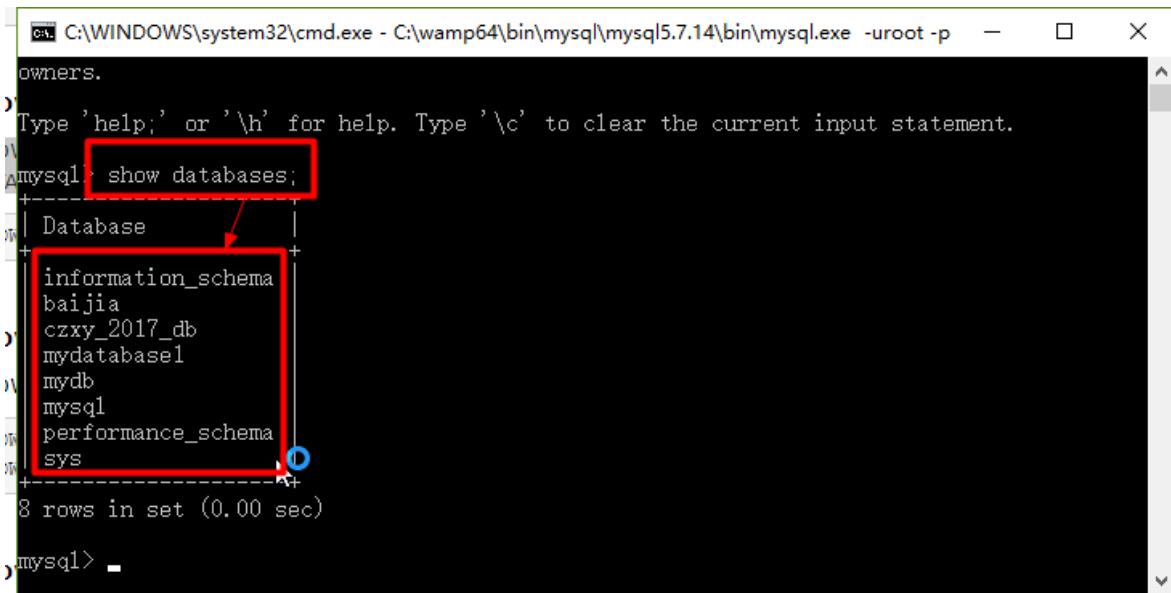
mysql>
```

3. show 指令

3.1 显示数据库

```
show databases;
```

show databases 可以在 MySQL 服务器主机上所有数据库



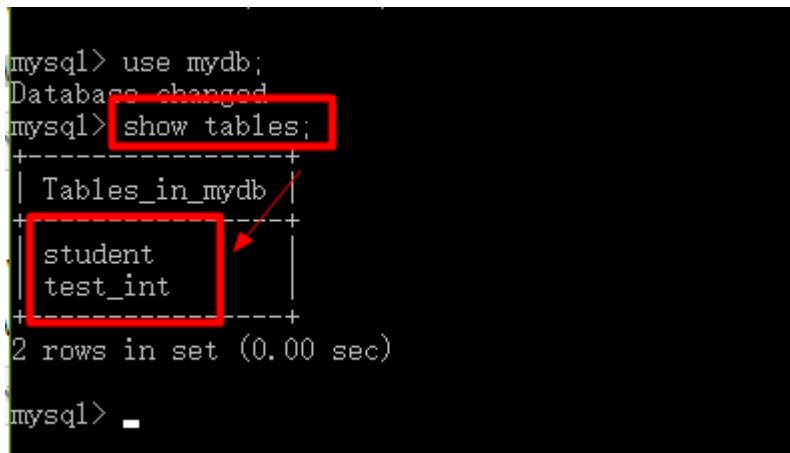
C:\WINDOWS\system32\cmd.exe - C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p
owners.
> Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
> mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| baijia |
| czxy_2017_db |
| mydatabase1 |
| mydb |
| mysql |
| performance_schema |
| sys |
+-----+
8 rows in set (0.00 sec)

mysql> -

3.2 显示数据库中的所有表

```
show tables;
```

show tables 用于查看数据库中的所有的表



mysql> use mydb;
Database changed
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| student |
| test_int |
+-----+
2 rows in set (0.00 sec)

mysql> -

3.3 显示指定表的列

```
show columns from 表名;
```

show columns from 表名，可以查看一个表的字段信息

```

C:\WINDOWS\system32\cmd.exe - C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p
+-----+
| student
| test_st
+-----+
2 rows in set (0.00 sec)

mysql> show columns from student;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+-----+
| id    | int(10) | NO   | PRI  | NULL    | auto_increment |
| name  | varchar(30)| YES  |       | NULL    |               |
| sex   | varchar(3) | YES  |       | NULL    |               |
| age   | int(10)  | YES  |       | NULL    |               |
| telnum| bigint(11)| YES  |       | NULL    |               |
| address| varchar(200)| YES  |       | NULL    |               |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>

```

1. `show tables` 或 `show tables from database_name;` -- 显示当前数据库中所有表的名称。
2. `show databases;` -- 显示 mysql 中所有数据库的名称。
3. `show columns from table_name from database_name;` 或 `show columns from database_name.table_name;` -- 显示表中列名称。
4. `show grants for user_name;` -- 显示一个用户的权限，显示结果类似于 grant 命令。
5. `show index from table_name;` -- 显示表的索引。
6. `show status;` -- 显示一些系统特定资源的信息，例如，正在运行的线程数量。
7. `show variables;` -- 显示系统变量的名称和值。
8. `show processlist;` -- 显示系统中正在运行的所有进程，也就是当前正在执行的查询。大多数用户可以查看他们自己的进程，但是如果他们拥有 process 权限，就可以查看所有人的进程，包括密码。
9. `show table status;` -- 显示当前使用或者指定的 database 中的每个表的信息。信息包括表类型和表的最新更新时间。
10. `show privileges;` -- 显示服务器所支持的不同权限。
11. `show create database database_name;` -- 显示 create database 语句是否能够创建指定的数据库。
12. `show create table table_name;` -- 显示 create database 语句是否能够创建指定的数据库。
13. `show engines;` -- 显示安装以后可用的存储引擎和默认引擎。
14. `show innodb status;` -- 显示 InnoDB 存储引擎的状态。
15. `show logs;` -- 显示 BDB 存储引擎的日志。
16. `show warnings;` -- 显示最后一个执行的语句所产生的错误、警告和通知。
17. `show errors;` -- 只显示最后一个执行语句所产生的错误。
18. `show [storage] engines;` -- 显示安装后的可用存储引擎和默认引擎。

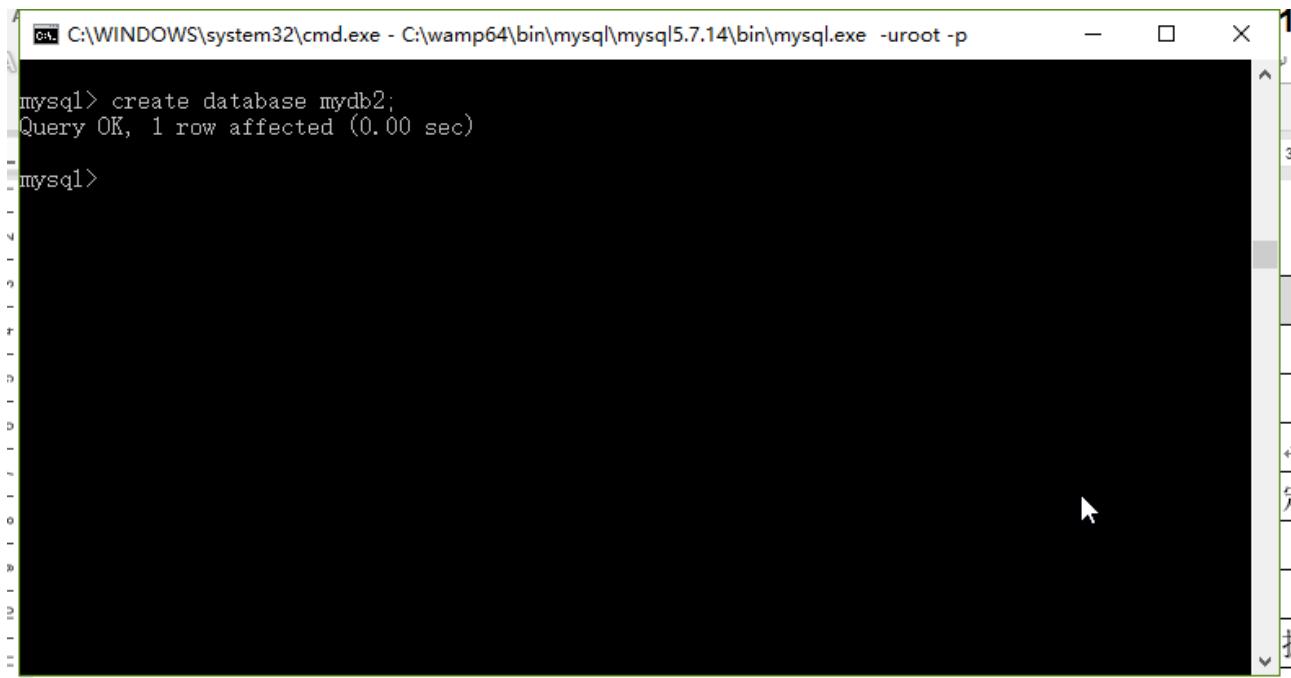
4. DDL SQL 之 database

分类	SQL 语句	描述
创建数据库	<code>create database 数据库名;</code>	

	create database 数据库名 character set 字符集	
查看数据库	show databases;	查看所有的数据库
	show create database 数据库名;	查看某个数据库的定义的信息
删除数据库	drop database 数据库名称	
	use 数据库名	切换数据库
其他命令		查看正在使用的数据库

4.1 创建数据库

- create database 数据库名;

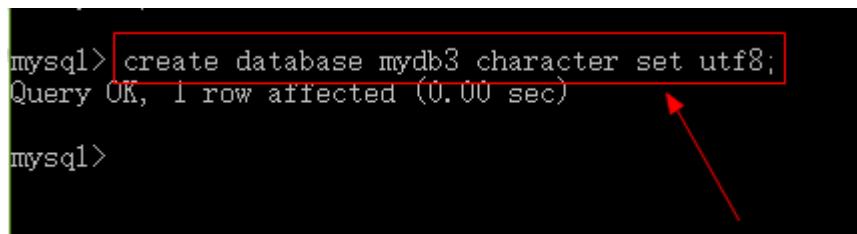


```
C:\WINDOWS\system32\cmd.exe - C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p
mysql> create database mydb2;
Query OK, 1 row affected (0.00 sec)

mysql>
```

A screenshot of a Windows Command Prompt window titled 'cmd'. The command 'create database mydb2;' is entered and executed, resulting in a 'Query OK, 1 row affected (0.00 sec)' message. The MySQL prompt 'mysql>' appears again at the bottom.

- create database 数据库名 character set 字符集;



```
mysql> create database mydb3 character set utf8;
Query OK, 1 row affected (0.00 sec)

mysql>
```

A screenshot of a MySQL command-line interface. A red box highlights the command 'create database mydb3 character set utf8;'. An arrow points from this highlighted area to the right side of the screen, likely indicating where the output or error messages would appear.

4.2 查看数据库创建的语句

- show database;

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| baijia |
| czxy_2017_db |
| mydatabase1 |
| mydb |
| mydb2 |
| mydb3 |
| mysql |
| performance_schema |
| sys |
+-----+
10 rows in set (0.00 sec)

mysql>
```

- show create database mydb2;

```
mysql> show create database mydb2;
+-----+-----+
| Database | Create Database |
+-----+-----+
| mydb2   | CREATE DATABASE `mydb2` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+-----+
1 row in set (0.00 sec)

mysql> _
```

4.3 切换数据库

use 数据库名;
可以切换要操作的数据库

4.4 删除数据

```
drop database mydb2;

1 row in set (0.00 sec)

mysql> drop database mydb2;
Query OK, 0 rows affected (0.01 sec)

mysql> _
```

5. DDL 之 table

分类	SQL 语句	描述
创建表	create table 表名(字段名 类型(长度) [约束], 字段名 类型(长度) [约束]);	
查看表	show tables;	查看数据库中的所有表
	desc 表名	查看表结构
删除表	drop table 表名	
修改表	alter table 表名 add 列名 类型(长度) [约束]	修改表添加列
	alter table 表名 modify 列名 类型(长度) 约束	修改表修改列的类型长度及约束
	alter table 表名 change 旧列名 新列名 类型 约束	修改表修改列名
	alter table 表名 drop 列名;	修改表删除列
	rename table 表名 to 新表名	修改表名
	alter table 表名 character set 字符集	修改表的字符集

5.1 创建表

创建 MySQL 数据表需要以下信息：

- 1) 表名
- 2) 表字段名
- 3) 定义每个表字段的类型

```
CREATE TABLE table_name (
    字段名称 字段类型(长度) 属性...,  

    字段名称 字段类型(长度) 属性...,  

);
```

```
-- 创建一个学生表
create table student (
    id int(10) unsigned,
    name varchar(20),
    age tinyint(3) unsigned
);
```

```
A mysql> -- 创建一个学生表
mysql> create table student (
    -> id int(10) unsigned,
    -> name varchar(20),
    -> age tinyint(3) unsigned
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
```

5.2 查看表

- show tables; 查看所有的表

```
mysql> show tables;
+-----+
| Tables_in_mydb3 |
+-----+
| student         |
+-----+
1 row in set (0.00 sec)

mysql>
```

- show create table student 查看建表语句

- desc 表名; 显示表结构

```
mysql> desc student;
+-----+-----+-----+-----+-----+-----+
| Field | Type            | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(10) unsigned | YES  |     | NULL    |       |
| name  | varchar(20)      | YES  |     | NULL    |       |
| age   | tinyint(3) unsigned | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

5.3 修改表

- 为表增加一个新的列

```
alter table 表名 add 字段名 字段类型(长度) 属性约束;
```

如: alter table student add sex enum("男","女");

```
mysql> alter table student add sex enum("男","女");
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> -
```

再如： alter table student add address varchar(255);

```
mysql> alter table student add sex enum("男","女");
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> alter table student add address varchar(255);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> desc student;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	YES		NULL	
name	varchar(20)	YES		NULL	
age	tinyint(3) unsigned	YES		NULL	
sex	enum('男','女')	YES		NULL	
address	varchar(255)	YES		NULL	

```
5 rows in set (0.00 sec)
```

查看表结构

```
mysql> desc student;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(10) unsigned | YES |   | NULL    |       |
| name  | varchar(20)      | YES |   | NULL    |       |
| age   | tinyint(3) unsigned | YES |   | NULL    |       |
| sex   | enum('男','女')  | YES |   | NULL    |       |
| address | varchar(255)     | YES |   | NULL    |       |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> -
```

- 修改列的长度

```
alter table 表名 modify 字段名(长度) 约束;
```

如：修改 student 表中的 address 字段长度为 300

```
alter table student modify address varchar(300);
```



```
mysql> alter table student modify address varchar(300);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> desc student;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	YES		NULL	
name	varchar(20)	YES		NULL	
age	tinyint(3) unsigned	YES		NULL	
sex	enum('男','女')	YES		NULL	
address	varchar(300)	YES		NULL	

```
5 rows in set (0.00 sec)
```

```
mysql>
```

- 修改列名称

```
alter table 表名 change 旧列名 新列名 类型(长度) 约束;
```

```
如: alter table student change sex gender enum("男","女");
```

```
mysql> alter table student change sex gender enum("男","女");
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> desc student;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	YES		NULL	
name	varchar(20)	YES		NULL	
age	tinyint(3) unsigned	YES		NULL	
gender	enum('男','女')	YES		NULL	
address	varchar(300)	YES		NULL	

```
5 rows in set (0.00 sec)
```

```
mysql>
```

- 删除列

```
alter table 表名 drop 字段名;
```

```
如: alter table student drop address;
```

```
mysql> alter table student drop address;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc student;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(10) unsigned | YES  |     | NULL    |       |
| name  | varchar(20)      | YES  |     | NULL    |       |
| age   | tinyint(3) unsigned | YES  |     | NULL    |       |
| gender | enum('男','女') | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

- 修改表名称

```
rename table 旧表名 to 新表名;
如: rename table student to student1;
```

```
mysql> rename table student to student1;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_mydb3 |
+-----+
| student1        |
+-----+
1 row in set (0.00 sec)

mysql>
```

- 修改字符集

```
alter table 表名 character set 新字符集;
```

```
mysql> alter table student1 character set gbk;
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> show create table student1;
+-----+-----+
| Table | Create Table
+-----+-----+
| student1 | CREATE TABLE `student1` (
  `id` int(10) unsigned DEFAULT NULL,
  `name` varchar(20) CHARACTER SET utf8 DEFAULT NULL,
  `age` tinyint(3) unsigned DEFAULT NULL,
  `gender` enum('男','女') CHARACTER SET utf8 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=gbk |
+-----+-----+
1 row in set (0.00 sec)
```

5.4 删表

```
drop table 表名;
```

为了配合测试，我们先创建一个名为 test 的表，然后再删除表
create table test (name varchar(10));

```
mysql> create table test (name varchar(10));
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+-----+
| Tables_in_mydb3 |
+-----+
| student1         |
| test             |
+-----+
2 rows in set (0.00 sec)
```

删除 test 表

```
drop table test;
```

```

mysql> drop table test;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+-----+
| Tables_in_mydb3 |
+-----+
| student1         |
+-----+
1 row in set (0.00 sec)

mysql>
    
```

6. DML 之数据增删改

6.1 插入 insert

插入	语法
向表中插入某些列	insert into 表 (列名 1,列名 2,列名 3...) values (值 1,值 2,值 3...)
向表中插入所有列	insert into 表 values (值 1,值 2,值 3...)

- 所有列的情况

```
insert into student1(id,name,age,gender) values(1,'czxy',18,'男');
```

```

mysql> insert into student1(id,name,age,gender) values(1,'czxy',18,'男');
Query OK, 1 row affected (0.00 sec)

mysql> select * from student1;
+----+----+----+----+
| id | name | age | gender |
+----+----+----+----+
| 1  | czxy | 18 | 男      |
+----+----+----+----+
1 row in set (0.00 sec)
    
```

- 部分列的情况

```
insert into student1(id,name) values(2,'东方不败');
```

```

mysql> insert into student1(id,name) values(2,'东方不败');
Query OK, 1 row affected (0.00 sec)
    
```

```

mysql> select * from student1;
+----+----+----+----+
| id | name     | age | gender |
+----+----+----+----+
| 1  | czxy     | 18 | 男      |
| 2  | 东方不败 | NULL | NULL   |
+----+----+----+----+
2 rows in set (0.00 sec)
    
```

- 书写顺序无关，但要对应

```
insert into student1(id,name) values(2,'东方不败');
```

```
mysql> insert into student1(name, id) values('黑山老妖', 3);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from student1;
+----+-----+-----+-----+
| id | name | age | gender |
+----+-----+-----+-----+
| 1  | czxy | 18  | 男   |
| 2  | 东方不败 | NULL | NULL |
| 3  | 黑山老妖 | NULL | NULL |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

注意：

1. 列名数与 values 后面的值个数必须保持一致
2. 列的顺序与插入的值得顺序必须保持一致
3. 列名的类型与插入的值要一致.
4. 插入值的时候不能超过最大长度.
5. 值如果是字符串或者日期需要加引号''（一般是单引号）

6.2 更新 update

更新	语法
更新所有记录的指定列	update 表名 set 字段名=值,字段名=值 ...
更新符合条件所有记录的指定列	update 表名 set 字段名=值,字段名=值 ... where 条件

- 更新所有记录的字段值

```
update student1 set age=20,gender='男';
```

```
mysql> update student1 set age=20, gender='男';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

```
mysql> select * from student1;
+----+-----+-----+-----+
| id | name | age | gender |
+----+-----+-----+-----+
| 1  | czxy | 20  | 男   |
| 2  | 东方不败 | 20 | 男   |
| 3  | 黑山老妖 | 20 | 男   |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

- 更新复合条件的记录字段值

```
update student1 set age=18,gender='女' where id=2;
```

```
mysql> update student1 set age=18,gender='女' where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from student1;
+----+-----+-----+-----+
| id | name | age | gender |
+----+-----+-----+-----+
| 1  | czxy | 20  | 男   |
| 2  | 东方不败 | 18  | 女   |
| 3  | 黑山老妖 | 20  | 男   |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

注意：

1. 列名的类型与修改的值要一致.
2. 修改值得时候不能超过最大长度.
3. 值如果是字符串或者日期需要加' '.

6.3 删除 delete

删除	语法
删除指定表的所有数据	delete from 表名
删除指定表符合条件的所有数据	delete from 表名 where 条件

- 删除表中复合条件记录

```
delete from student1 where id=3;
```

```
mysql> delete from student1 where id=3;
Query OK, 1 row affected (0.00 sec)

mysql> select * from student1;
+----+-----+-----+-----+
| id | name | age | gender |
+----+-----+-----+-----+
| 1  | czxy | 20  | 男   |
| 2  | 东方不败 | 18  | 女   |
+----+-----+-----+-----+
2 rows in set (0.00 sec)
```

- 删除表中的所有数据

```
delete from student1
```



```
mysql> delete from student1
-> .
Query OK, 2 rows affected (0.00 sec)

mysql> select * from student1;
Empty set (0.00 sec)

mysql>
```

7. 今日总结

MySQL (四)

1. 查询数据

select 是一个功能非常功能的指令，可以从一张或者多张表中查询出各种结构的数据。

1.1 准备数据

```
#创建商品表:  
create table product(  
    pid int unsigned not null auto_increment comment '编号',  
    pname varchar(50) not null comment '商品名称',  
    price decimal(10,2) not null comment '商品价格',  
    is_on_sale enum('y','n') not null default 'y' comment '是否上架,y:是,n:否',  
    primary key (pid)  
);  
#自动增长列: auto_increment, 要求: 1.必须整形 2.必须主键 primary key  
  
# 插入数据  
insert into product(pid,pname,price,is_on_sale)  
values(1,'联想',5000,'y'),  
(2,'海尔',3000,'y'),  
(3,'雷神',5000,'y'),  
(4,'JACK JONES',800,'y'),  
(5,'真维斯',200,'n'),  
(6,'花花公子',440,'n'),  
(7,'劲霸',2000,'y'),  
(8,'香奈儿',800,'n'),  
(9,'相宜本草',200,'y'),  
(10,'面霸',5,'y'),  
(11,'神州',2000,'y'),  
(12,'印度神油',363,'y'),  
(13,'高级檀香',45,'y');
```

准备完成后，表格如下：

```
mysql> select * from product;
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 1   | 联想   | 5000.00 | y
| 2   | 海尔   | 3000.00 | y
| 3   | 雷神   | 5000.00 | y
| 4   | JACK JONES | 800.00 | y
| 5   | 真维斯   | 200.00 | n
| 6   | 花花公子   | 440.00 | n
| 7   | 劲霸   | 2000.00 | y
| 8   | 香奈儿   | 800.00 | n
| 9   | 相宜本草   | 200.00 | y
| 10  | 面霸   | 5.00 | y
| 11  | 神州   | 2000.00 | y
| 12  | 印度神油   | 363.00 | y
| 13  | 高级檀香   | 45.00 | y
+----+-----+-----+-----+
13 rows in set (0.00 sec)
```

1.2 select 的语法

select 是一个非常复杂的指令，可以查询一张表，也可以同时从多张表中查询数据，我们先看在查询一张表时的语法：

```
select 字段 [as] 别名 ....
from 表名 [as] 别名
[where 条件
group by 字段名
having 过滤条件
order by 字段名 desc|asc
limit 起始记录,记录数]
```

说明：

1. []中的内容是可以省略。
2. |是或者的意思。
3. 如果同时写了多个指令，那么顺序是固定的 select->from->where->group by->order by->having->limit

2. 基本查询

简单查询	SQL语句
查询所有的商品	select * from product;
查询商品名和商品价格	select pname,price from product;
表别名, select ... from 表名 [as] 别名	select * from product as p
列别名, select 字段 [as] 别名, ... from 表名	select pname as pn from product
去掉重复值	select distinct price from product
查询结果是表达式: 将所有商品的价格+10 元进行显示	select pname,price+10 from product
使用函数	select version();

2.1 查询所有字段信息

```
select * from 表名; # * 表示所有字段
```

select:选择

from: 从..里

说明: 从表中取出所有的字段。

```
mysql> select * from product;
+----+-----+-----+-----+
| pid | pname      | price   | is_on_sale |
+----+-----+-----+-----+
| 1  | 联想        | 5000.00 | y         |
| 2  | 海尔        | 3000.00 | y         |
| 3  | 雷神        | 5000.00 | y         |
| 4  | JACK JONES | 800.00  | y         |
| 5  | 真维斯      | 200.00  | n         |
| 6  | 花花公子    | 440.00  | n         |
| 7  | 劲霸        | 2000.00 | y         |
| 8  | 香奈儿      | 800.00  | n         |
| 9  | 相宜本草    | 200.00  | y         |
| 10 | 面霸        | 5.00    | y         |
| 11 | 神州        | 2000.00 | y         |
| 12 | 印度神油    | 363.00  | y         |
| 13 | 高级檀香    | 45.00   | y         |
+----+-----+-----+-----+
13 rows in set (0.00 sec)
```

2.2 查询指定的字段

```
select 字段1, 字段2, ... from 表名;
```

```
mysql> select pname, price from product;
+-----+-----+
| pname      | price   |
+-----+-----+
| 联想        | 5000.00 |
| 海尔        | 3000.00 |
| 雷神        | 5000.00 |
| JACK JONES | 800.00  |
| 真维斯      | 200.00  |
| 花花公子    | 440.00  |
| 劲霸        | 2000.00 |
| 香奈儿      | 800.00  |
| 相宜本草    | 200.00  |
| 面霸        | 5.00    |
| 神州        | 2000.00 |
| 印度神油    | 363.00  |
| 高级檀香    | 45.00  |
+-----+-----+
13 rows in set (0.00 sec)
```

2.3 查询表达式

```
select 表达式 from 表名;
```

说明：表达式可以使用加(+)、减(-)、乘(*)、除(/)、求余(%)算术运算符。

可以对字段中的字段进行运算：

```
mysql> select pname, price, price/2, price%7 from product;
+-----+-----+-----+-----+
| pname | price | price/2 | price%7 |
+-----+-----+-----+-----+
| 联想 | 5000.00 | 2500.000000 | 2.00 |
| 海尔 | 3000.00 | 1500.000000 | 4.00 |
| 雷神 | 5000.00 | 2500.000000 | 2.00 |
| JACK JONES | 800.00 | 400.000000 | 2.00 |
| 真维斯 | 200.00 | 100.000000 | 4.00 |
| 花花公子 | 440.00 | 220.000000 | 6.00 |
| 劲霸 | 2000.00 | 1000.000000 | 5.00 |
| 香奈儿 | 800.00 | 400.000000 | 2.00 |
| 相宜本草 | 200.00 | 100.000000 | 4.00 |
| 面霸 | 5.00 | 2.500000 | 5.00 |
| 神州 | 2000.00 | 1000.000000 | 5.00 |
| 印度神油 | 363.00 | 181.500000 | 6.00 |
| 高级檀香 | 45.00 | 22.500000 | 3.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)
```

还可以做单纯的数字运算：

```
mysql>
mysql> select (17*37*12+20)/3;
+-----+
| (17*37*12+20)/3 |
+-----+
| 2522.6667 |
+-----+
1 row in set (0.00 sec)
```

2.4 别名

罗纳尔多[巴西足球运动员]

开放分类：人物 | 足球运动员

罗纳尔多·路易斯·纳扎里奥·达·利马（Ronaldo Luiz Nazario De Lima），1976年9月18日出生在巴西里约热内卢，巴西足球运动员，场上司职前锋。青少年时期成名于克鲁塞罗，1996、1997、2002年三度获得世界足球先生，1998年当选世界杯最佳球员，2002年获得世界杯金靴奖，2010年罗纳尔多获得巴西传奇巨星奖。罗纳尔多的欧洲足球生涯始于埃因霍温，此后先后效力过巴塞罗那、国际米兰、皇家马德里、AC米兰四支豪门球队，共拿到1次西甲冠军奖杯，1次欧洲联盟杯、1次欧洲优胜者杯、1次西班牙国王杯以及2次西班牙超级杯冠军奖杯。2011年2月14日，饱受伤病折磨的罗纳尔多宣布退役，终结了18年的职业生涯。2015年2月，罗纳尔多宣布复出，加盟美国二级联赛的劳德代尔堡前锋足球俱乐部。

[编辑](#) | [讨论](#) | [分享](#) | [8029](#)

此词条还可添加 [信息模块](#)



[罗纳尔多\[巴西足球运动员\]图册](#)

2.4.1 给列加别名

```
select 字段1 [as] 别名1, 字段2 [as] 别名2, ... from 表名;
```

```
mysql> select pname, price, price/2 as banjia, price%7 yu7 from product;
+-----+-----+-----+-----+
| pname | price | banjia | yu7  |
+-----+-----+-----+-----+
| 联想   | 5000.00 | 2500.000000 | 2.00 |
| 海尔   | 3000.00 | 1500.000000 | 4.00 |
| 雷神   | 5000.00 | 2500.000000 | 2.00 |
| JACK JONES | 800.00 | 400.000000 | 2.00 |
| 真维斯   | 200.00  | 100.000000 | 4.00 |
| 花花公子 | 440.00 | 220.000000 | 6.00 |
| 劲霸   | 2000.00 | 1000.000000 | 5.00 |
| 香奈儿   | 800.00  | 400.000000 | 2.00 |
| 相宜本草 | 200.00 | 100.000000 | 4.00 |
| 面霸   | 5.00   | 2.500000 | 5.00 |
| 神州   | 2000.00 | 1000.000000 | 5.00 |
| 印度神油 | 363.00 | 181.500000 | 6.00 |
| 高级檀香 | 45.00  | 22.500000 | 3.00 |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)
```

2.4.2 给表加别名

也可以为表起别名，在多表查询时经常使用：

```
select * from 表名 [as] 别名
```

```
mysql> select * from product as p;
+----+-----+-----+-----+
| pid | pname      | price    | is_on_sale |
+----+-----+-----+-----+
| 1   | 联想        | 5000.00  | y          |
| 2   | 海尔        | 3000.00  | y          |
| 3   | 雷神        | 5000.00  | y          |
| 4   | JACK JONES  | 800.00   | y          |
| 5   | 真维斯      | 200.00   | n          |
| 6   | 花花公子    | 440.00   | n          |
| 7   | 劲霸        | 2000.00  | y          |
| 8   | 香奈儿      | 800.00   | n          |
| 9   | 相宜本草    | 200.00   | y          |
| 10  | 面霸        | 5.00     | y          |
| 11  | 神州        | 2000.00  | y          |
| 12  | 印度神油    | 363.00   | y          |
| 13  | 高级檀香    | 45.00    | y          |
+----+-----+-----+-----+
13 rows in set (0.00 sec)
```

2.5 distinct 去掉重复值

```
select distince 字段名 from 表名;
```

```
mysql> select distinct price from product;
+-----+
| price |
+-----+
| 5000.00 |
| 3000.00 |
| 800.00  |
| 200.00  |
| 440.00  |
| 2000.00 |
| 5.00    |
| 363.00  |
| 45.00   |
+-----+
9 rows in set (0.00 sec)
```

注意：

1. distinct 要写在所有字段的前面

2. 只取出来的字段值都相同时才算是重复

```
mysql>
mysql> select distinct price , pname from product;
+-----+-----+
| price | pname   |
+-----+-----+
| 5000.00 | 联想
| 3000.00 | 海尔
| 5000.00 | 雷神
| 800.00  | JACK JONES
| 200.00  | 真维斯
| 440.00  | 花花公子
| 2000.00 | 劲霸
| 800.00  | 香奈儿
| 200.00  | 相宜本草
| 5.00    | 面霸
| 2000.00 | 神州
| 363.00  | 印度神油
| 45.00   | 高级檀香
+-----+-----+
13 rows in set (0.00 sec)
```

2.6 使用函数

语法:

函数名()

例 1、查看当前 MySQL 版本号:

```
mysql>
mysql> select version();
+-----+
| version() |
+-----+
| 5.7.14    |
+-----+
1 row in set (0.00 sec)
```

查看当前数据库名:

```
mysql> select database();
+-----+
| database() |
+-----+
| db1      |
+-----+
1 row in set (0.00 sec)
```

3. 条件查询

查询时设置一个条件，只有满足条件的记录才会被取出来。

语法：

```
select * from 表名 where 条件
```

条件查询时需要用到的运算符：

分类	符号	描述
比较运算符	> < <= >= = <>	大于、小于、小于等于、大于等于、等于、不等于
	BETWEEN ...AND...	显示在某一区间的值(含头含尾)
	IN(set)	显示在 in 列表中的值，例：in(100,200)
	LIKE '张 pattern'	模糊查询，Like 语句中， % 代表零个或多个任意字符， _ 代表一个字符， 例如：first_name like '_a%' ;
	IS NULL 和 IS NOT NULL	判断是否为空
逻辑运算符	and	多个条件同时成立
	or	多个条件任一成立
	not	不成立，例：where not(salary>100);

3.1 比较运算符

3.1.1 > < <= >= 等于【=】 不等于【!=或者<>】

例 1、取出价格小于 1000 元的商品：

```
mysql> select * from product where price < 1000;
+----+-----+-----+-----+
| pid | pname      | price   | is_on_sale |
+----+-----+-----+-----+
| 4   | JACK JONES | 800.00 | y          |
| 5   | 真维斯     | 200.00 | n          |
| 6   | 花花公子    | 440.00 | n          |
| 8   | 香奈儿     | 800.00 | n          |
| 9   | 相宜本草    | 200.00 | y          |
| 10  | 面霸        | 5.00    | y          |
| 12  | 印度神油    | 363.00 | y          |
| 13  | 高级檀香    | 45.00   | y          |
+----+-----+-----+-----+
8 rows in set (0.00 sec)
```

例 2、取出商品名称等于“花花公子”的商品：

```
mysql> select * from product where pname='花花公子';
+----+-----+-----+-----+
| pid | pname      | price   | is_on_sale |
+----+-----+-----+-----+
| 6   | 花花公子    | 440.00 | n          |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

例 3、取出所有上架的商品【提示：is_on_sale='y'】。

3.1.2 between ... and ... :【在 xxx 和 xxx 之间】 范围查询

例 1、取出价格在 300 到 3000 之间的商品【包含 300 和 3000】：

```
mysql> select * from product where price between 300 and 3000;
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 2   | 海尔  | 3000.00 | y
| 4   | JACK JONES | 800.00 | y
| 6   | 花花公子  | 440.00 | n
| 7   | 劲霸  | 2000.00 | y
| 8   | 香奈儿  | 800.00 | n
| 11  | 神州  | 2000.00 | y
| 12  | 印度神油  | 363.00 | y
+----+-----+-----+-----+
7 rows in set (0.00 sec)
```

3.1.3 like：模糊查询

在 like 查询是可以使用两个特殊符号：

_：代表 1 个任意字符。

%：代表 0 个或者多个做任意字符。

例 1. 取出商品名称只有两个字，并且第二个字是霸的商品：

```
mysql> select * from product where pname like '_霸';
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 7   | 劲霸  | 2000.00 | y
| 10  | 面霸  | 5.00   | y
+----+-----+-----+-----+
2 rows in set (0.00 sec)
```

例 2、取出商品名称只有三个字的商品：

```
mysql> select * from product where pname like '___';
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 5   | 真维斯 | 200.00 | n
| 8   | 香奈儿 | 800.00 | n
+----+-----+-----+-----+
2 rows in set (0.00 sec)
```

例 3、取出所有商品名称中带“神”字的商品：

```
mysql> select * from product where pname like '%神%';
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
|   3 | 雷神 | 5000.00 | y
| 11 | 神州 | 2000.00 | y
| 12 | 印度神油 | 363.00 | y
+----+-----+-----+
3 rows in set (0.00 sec)
```

例 4、取出所有商品名称以“神”字结尾的商品：

```
mysql> select * from product where pname like '%神';
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
|   3 | 雷神 | 5000.00 | y
+----+-----+-----+
1 row in set (0.00 sec)
```

3.1.4 in

取出一个集合中给定的商品。

语法：

```
where 字段 in (数据集合)
```

例：取出所有价格等于 300, 800, 2000, 3000 的商品：

```
mysql> select * from product where price in(300,800,2000,3000);
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
|   2 | 海尔 | 3000.00 | y
|   4 | JACK JONES | 800.00 | y
|   7 | 劲霸 | 2000.00 | y
|   8 | 香奈儿 | 800.00 | n
| 11 | 神州 | 2000.00 | y
+----+-----+-----+
5 rows in set (0.00 sec)
```

3.1.5 is null

例：取出价格是 null 的商品：【因为表中没有满足条件的记录，所以取出空】

```
mysql> select * from product where price is null;  
Empty set (0.00 sec)
```

3.2 逻辑运算符

3.2.1 and：并且【同个条件同时满足】

例 1、取出价格大于 1000 并且商品名称中带“神”这个字的：

```
mysql> select * from product where price>1000 and pname like '%神%';  
+----+----+----+----+  
| pid | pname | price | is_on_sale |  
+----+----+----+----+  
| 3 | 雷神 | 5000.00 | y |  
+----+----+----+----+  
1 row in set (0.00 sec)
```

3.2.2 or：或者【只要有一个条件满足】

例 1：取出价格大于 3000 或者商品名称以“神”结尾的：

```
mysql> select * from product where price > 3000 or pname like '%神';
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 1 | 联想 | 5000.00 | y
| 3 | 雷神 | 5000.00 | y
+----+-----+-----+
2 rows in set (0.00 sec)
```

例 2：取出商品名称以“神”结尾并且价格大于 3000，或者，商品名称由 3 个字组成并且价格小于 2000：

```
mysql> select * from product where (pname like '%神' and price > 3000) or (pname like ___ and price < 2000);
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 3 | 雷神 | 5000.00 | y
| 5 | 真维斯 | 200.00 | n
| 8 | 香奈儿 | 800.00 | n
+----+-----+-----+
3 rows in set (0.00 sec)
```

3.2.3 not: 非

例 1：取出商品名称不能 null 的：

```
mysql> select * from product where pname is not null;
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 1 | 联想 | 5000.00 | y
| 2 | 海尔 | 3000.00 | y
| 3 | 雷神 | 5000.00 | y
| 4 | JACK JONES | 800.00 | y
| 5 | 真维斯 | 200.00 | n
| 6 | 花花公子 | 440.00 | n
| 7 | 劲霸 | 2000.00 | y
| 8 | 香奈儿 | 800.00 | n
| 9 | 相宜本草 | 200.00 | y
| 10 | 面霸 | 5.00 | y
+----+-----+-----+
10 rows in set (0.00 sec)
```

4. 数据排序



4.1 根据一个字段排序

语法:

```
order by 字段名 asc|desc
```

说明:

1. **asc**: 升序。
2. **desc**: 降序。

例 1、取出所有商品并按价格降序排列:

```
mysql> select * from product order by price desc;
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 1   | 联想   | 5000.00 | y
| 3   | 雷神   | 5000.00 | y
| 2   | 海尔   | 3000.00 | y
| 7   | 劲霸   | 2000.00 | y
| 4   | JACK JONES | 800.00 | y
| 8   | 香奈儿   | 800.00 | n
| 6   | 花花公子   | 440.00 | n
| 5   | 真维斯   | 200.00 | n
| 9   | 相宜本草   | 200.00 | y
| 10  | 面霸   | 5.00 | y
+----+-----+-----+-----+
10 rows in set (0.00 sec)
```

例 2、取出所有带“神”的商品，并且按价格升序排列

4.2 根据多个字段排序

语法：

```
order by 字段1 asc|desc , 字段2 asc|desc , 字段3 asc|desc....
```

说明：先根据第一个字段排序，当第一个字段的值相同时再根据第二个字段排序，依次类推：

例 1：取出所有的商品以价格降序排列，价格相同的，以 pid 降序排列：

```
mysql> select * from product order by price desc, pid desc;
+----+-----+-----+-----+
| pid | pname | price | is_on_sale |
+----+-----+-----+-----+
| 3   | 雷神   | 5000.00 | y
| 1   | 联想   | 5000.00 | y
| 2   | 海尔   | 3000.00 | y
| 7   | 劲霸   | 2000.00 | y
| 8   | 香奈儿   | 800.00 | n
| 4   | JACK JONES | 800.00 | y
| 6   | 花花公子   | 440.00 | n
| 9   | 相宜本草   | 200.00 | y
| 5   | 真维斯   | 200.00 | n
| 10  | 面霸   | 5.00 | y
+----+-----+-----+-----+
10 rows in set (0.00 sec)
```

4.3 随机排序

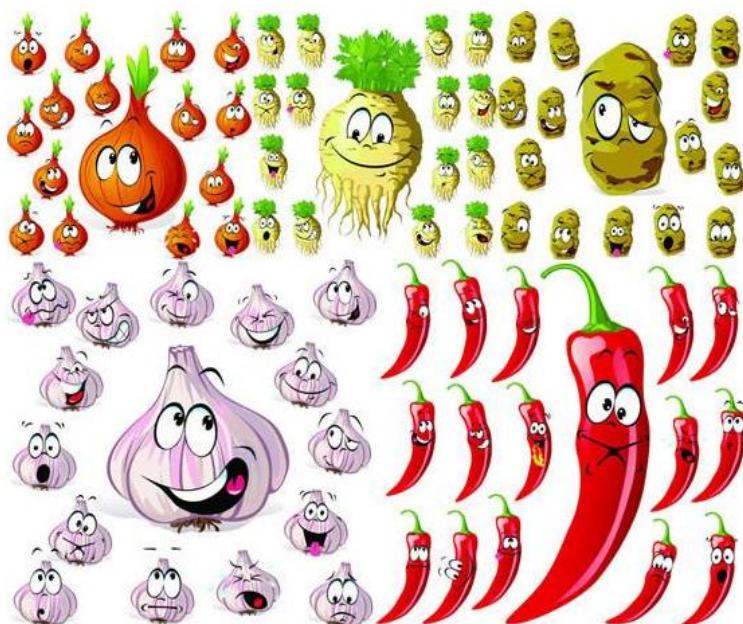
语法：

```
order by rand()
```

rand() : 随机函数

5. 数据分组

分组，根据一个规则把事物归类，相同的在一起。



5.1 分组语法

```
group by 字段名1,字段名2.....
```

5.2 分组的特点

1. 字段值相同的归为一组
2. select 的字段中，不能出现未在分组中的字段

比如：根据 is_on_sale 字段分组，那么 select 只能取这个字段否则出错：

```
mysql> select * from product group by is_on_sale;
ERROR 1055 (42000): Expression #1 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'db1.product.pid' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by
```

只能 select 分组的字段：

```
mysql> select is_on_sale from product group by is_on_sale;
+-----+
| is_on_sale |
+-----+
| y          |
| n          |
+-----+
2 rows in set (0.00 sec)
```

3. 最终结果只保留每一组中的第一条记录

总结：单纯的分组，和去重实际上是一样的。

5.3 聚合函数

MySQL 中提供了一套聚合函数：

max(字段名)：取表中这个字段的最大值。
min(字段名)：取表中这个字段的最小值。
avg(字段名)：取表中这个字段的平均值。
sum(字段名)：取表中这个字段所有值的和。
count(*或者字段名)：统计记录数。

重点说明：以上函数在不分组时，是对所有数据进行统计，返回一个结果，当分组时，是对每一组进行统计，返回每一组的结果。

例 1、取出表中的最大价格：

```
mysql> select max(price) from product;
+-----+
| max(price) |
+-----+
| 5000.00 |
+-----+
1 row in set (0.00 sec)
```

例 2、取出“上架”商品和“下架”商品分别的最大价格：

```
mysql> select is_on_sale,max(price) from product group by is_on_sale;
+-----+-----+
| is_on_sale | max(price) |
+-----+-----+
| y          | 5000.00  |
| n          | 800.00   |
+-----+-----+
2 rows in set (0.00 sec)
```

例 3、取出表中总的记录数：

```
mysql> select count(*) from product;
+-----+
| count(*) |
+-----+
|      10 |
+-----+
1 row in set (0.00 sec)
```

例 4、取出“上架”商品和“下架”商品分别的记录数：

```
mysql> select count(*),is_on_sale from product group by is_on_sale;
+-----+-----+
| count(*) | is_on_sale |
+-----+-----+
|      7  | y        |
|      3  | n        |
+-----+-----+
2 rows in set (0.00 sec)
```

✧ 扩展： **count(*)** 和 **count(字段名)** 的区别

count(*): 取表中记录数。

count(字段名): 取表中这个字段值不为 null 的记录数。

```
abc
id    name
1     abc
2     null
3     null
```

```
select count(name) from abc;    --> 1
```

6. having 过滤数据

having 和 **where** 很像都可以用来实现数据的过滤，它们的区别是：

1. 位置不同，**where** 写在 **group by** 前面、**having** 写在 **group by** 后面。
2. 执行时机不同，**where** 是在聚合函数之前执行，**having** 是在聚合函数之后执行。

```
mysql> select is_on_sale,max(price) mp from product group by is_on_sale
->;
+-----+-----+
| is_on_sale | mp      |
+-----+-----+
| y        | 5000.00 |
| n        | 800.00  |
+-----+-----+
2 rows in set (0.00 sec)

mysql> select is_on_sale,max(price) mp from product where mp > 2000 group by is_on_sale
->;
ERROR 1054 (42S22): Unknown column 'mp' in 'where clause'
mysql>
mysql>
mysql> select is_on_sale,max(price) mp from product group by is_on_sale having mp > 2000;
+-----+-----+
| is_on_sale | mp      |
+-----+-----+
| y        | 5000.00 |
+-----+-----+
1 row in set (0.00 sec)
```

使用 where 时报错，因为执行 where 时还没有 计算出 mp 呢，因为 where 是在 max 聚合函数之前执行的。

having 的应用场景：一般 having 都是和 group by 以及聚合函数一起使用的。

总结：平时都使用 where，但是当需要用一个聚合函数的结果做为条件时需要使用 having。

7. 限制记录数

可以使用 limit 限制取出记录的数量，limit 要写在 sql 语句的最后。

语法：

```
limit 起始记录,记录数
```

说明：

1. 起始记录是指从第几条记录开始取，第一条记录的下标是 0。
2. 记录数是指从起始记录开始向后依次取的记录数。

比如： limit 0,10 ，意思是：从第 1 条记录开始取，取 10 条。

例 1：取出表中前 3 件商品

```
mysql> select * from product limit 0,3;
+----+----+----+----+
| pid | pname | price | is_on_sale |
+----+----+----+----+
| 1 | 联想 | 5000.00 | y
| 2 | 海尔 | 3000.00 | y
| 3 | 雷神 | 5000.00 | y
+----+----+----+----+
3 rows in set (0.00 sec)
```

注意：当起始值是 0 时，可以省略为 limit 记录数，所以取出前三条记录：

```
mysql> select * from product limit 3;
+----+----+----+----+
| pid | pname | price | is_on_sale |
+----+----+----+----+
| 1 | 联想 | 5000.00 | y
| 2 | 海尔 | 3000.00 | y
| 3 | 雷神 | 5000.00 | y
+----+----+----+----+
3 rows in set (0.00 sec)
```

例 2、从第 3 条记录开始取，取 3 条

```
mysql> select * from product limit 2,3;
+----+----+----+----+
| pid | pname | price | is_on_sale |
+----+----+----+----+
| 3 | 雷神 | 5000.00 | y
| 4 | JACK JONES | 800.00 | y
| 5 | 真维斯 | 200.00 | n
+----+----+----+----+
3 rows in set (0.00 sec)
```

8. 今日总结



传智专修学院
CHUANZHI SPECIALIZED COLLEGE

力学笃行 志存高远

MySQL (五)

1. 多表关系

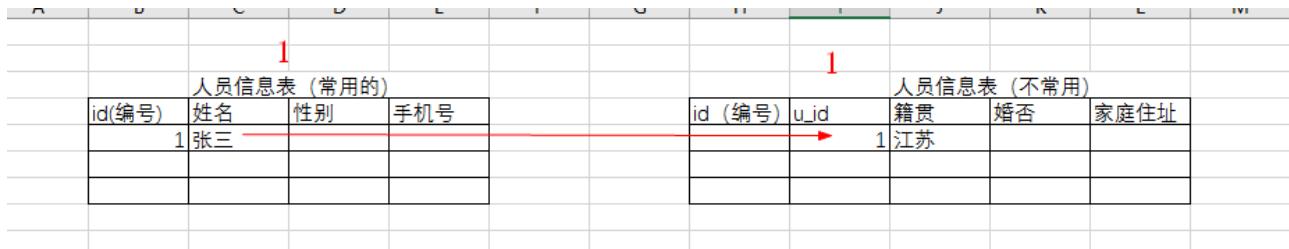
在 MySQL 数据库中表和表之间存在一些必要的关系，他们分别是：一对一、一对多、多对多关系，下面我们来具体学习一下。

1.1 一对一关系

一对一：一张表中的一条记录与另外一张表中最多有一条明确的对应关系

如何设置一对一：

在另外一个表《人员信息表（不常用）》中增加一个字段，设置字段的值和另外一表《人员信息表》中的 id 相等，而且在《人员信息表（不常用）》中有且只有一条数据对应



人员信息表(常用的)				人员信息表(不常用)			
id(编号)	姓名	性别	手机号	id(编号)	u_id	籍贯	婚否
1	张三				1	江苏	

应用场合：

把一个很复杂的表，拆分为多个表，会用一对一的关系

```
-- 创建 person 表 (常用表)
create table person(
    id int(10) unsigned primary key auto_increment,
    name varchar(10) not null,
    sex enum("男", "女"),
    telnum bigint(11)
);
-- 插入数据到 person 表中
insert into person values(null, '张三', '男', 13800138000);
insert into person values(null, '李四', '男', 13800138000);
insert into person values(null, '王五', '男', 13800138000);

-- 创建人员信息详情表 (不常用)
```

```

create table person_detail(
    id int(10) unsigned primary key auto_increment,
    u_id int(10) unsigned not null,
    jiguang varchar(10) not null,
    is_merry_state enum("y","n"),
    address varchar(200)
);

-- 添加人员详细数据
insert into person_detail values(null,1,'山东','y','山东省济南市');
insert into person_detail values(null,2,'河南','n','河南省郑州市');
insert into person_detail values(null,3,'河北','y','河北省石家庄市');

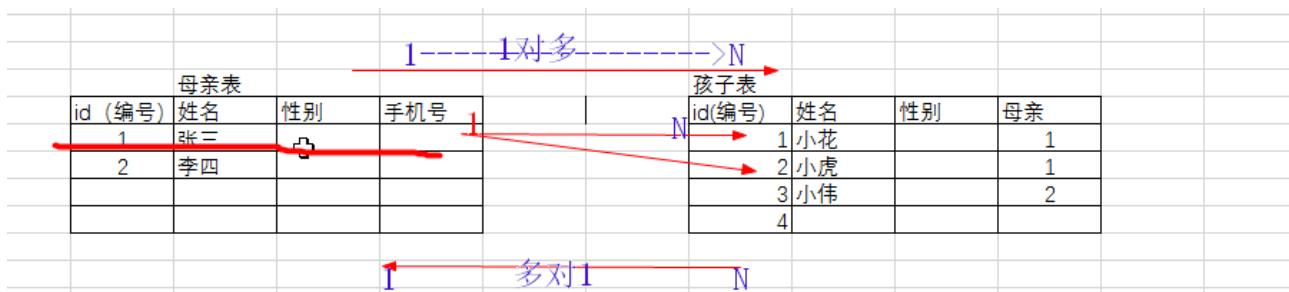
```

1.2 一对多关系

一对多：在一个表中的一条数据和另外一个表中的多条数据形成关联关系，他就是一对多的关系，反过来当一个表中多条数据和另外一个表中的一条数据对应，这就是多对一的关系

如何建立一对多的关系

1) 在“多”的表中，增加一个字段，这个字段可以和“一”表中的编号相等，“多”表中可以有多条记录和“一”表中的编号相等



```

-- 创建母亲表
create table mother(
    id int(10) unsigned primary key auto_increment,
    name varchar(10) not null,
    telnum bigint(11)
);

insert into mother values(null,'兰兰',13800138000);
insert into mother values(null,'花花',13800138000);
insert into mother values(null,'美美',13800138000);

-- 创建孩子表

```

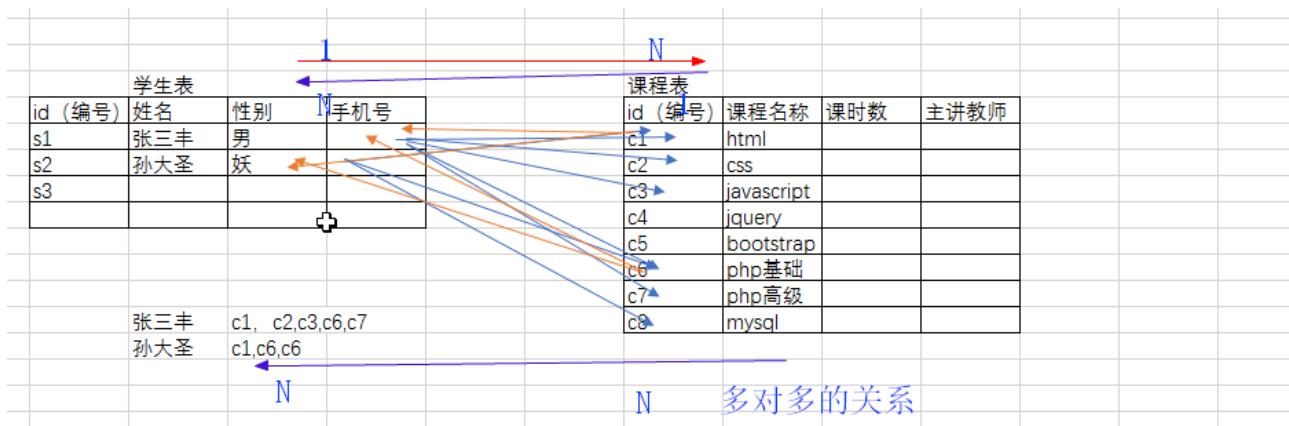
```

create table children(
    id int(10) unsigned primary key auto_increment,
    name varchar(10) not null,
    sex enum("男","女"),
    mother_id int(10) unsigned not null,
);
insert into children values(null,'可可','男');
insert into children values(null,'小米粒','女');
insert into children values(null,'小丸子','女');

```

1.3 多对多关系

多对多：一张表中的一条记录在另外一张表中可以匹配到多条记录，反过来也一样。



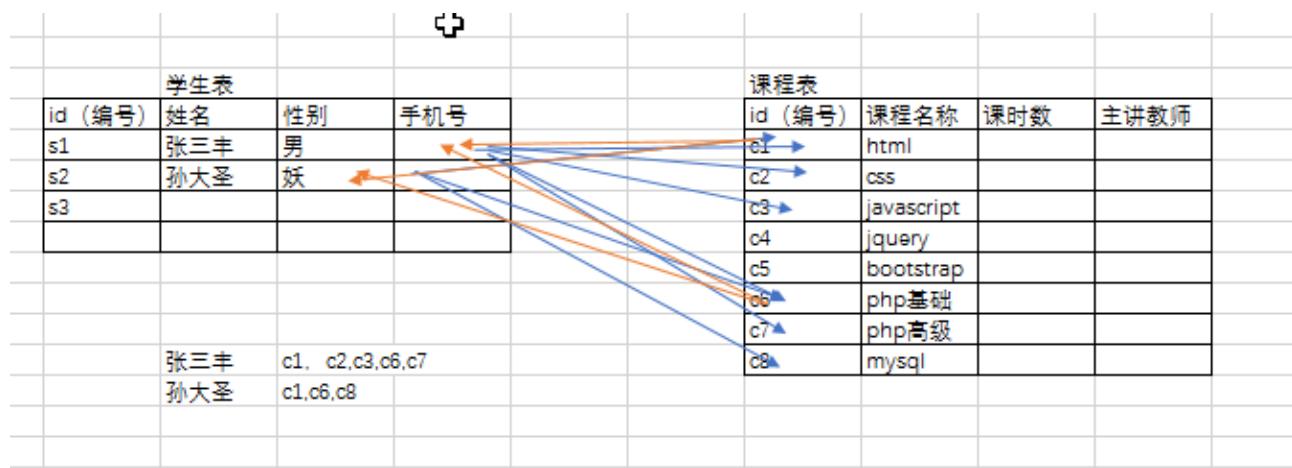
多对多的关系如果按照多对一的关系维护：就会出现一个字段中有多个其他表的主键，在访问的时候就会带来不便。

既然通过两张表自己增加字段解决不了问题，那么就通过第三张表来解决。

师生关系

- 1、一个学生学习多个课程；
- 2、一个课程被多个学生选课学习；

首先得有两个实体：学生表和课程表



从中间设计一张表：维护两张表对应的联系：每一种联系都包含



多对多解决方案；增加一个中间表，让中间表与对应的其他表形成两个多对一的关系：多对一的解决方案是在“多”表中增加“一”表对应的主键字段。



学生表

```
create table students
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '姓名',
    gender enum('男','女') not null comment '性别',
    tel bigint unsigned not null comment '手机号',
    primary key(id)
) comment='学生表';
```

```
insert into students
values
(1, '八戒', '男', 13912345555),
(2, '大师兄', '男', 13912346666),
(3, '金角大王', '男', 13922222222);
```

课程表

```
create table course
(
    id int unsigned not null auto_increment comment '编号',
    course_name varchar(50) not null comment '课程名称',
    primary key(id)
) comment='课程表';
```

```
insert into course
values(1,'HTML'),(2,'CSS'),(3,'JavaScript'),
(4,'jQuery'),(5,'Bootstrap'),(6,'PHP'),(7,'MySQL');

# 学生选课表
create table student_course
(
    student_id int unsigned not null comment '学生Id',
    course_id int unsigned not null comment '课程Id'
) comment='学生选课表';

# id=1,八戒      学习了      PHP、MySQL
# id=2,大师兄    学习了      HTML、CSS、JavaScript、jQuery、Bootstrap、
PHP、MySQL
# id=3,金角大五    学习了      HTML、CSS、JavaScript
insert into student_course
values
(1, 6), (1, 7),
(2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7),
(3, 1), (3, 2), (3, 3);
```

2. 联合查询

2.1 基本概念

联合查询是可合并多个相似的 `select` 查询的结果集。等同于将一个表追加到另一个表，从而实现将两个表的查询组合到一起，使用谓词为 `UNION` 或 `UNION ALL`。

联合查询：将多个查询的结果合并到一起（纵向合并）：字段数不变，多个查询的记录数合并。

2.2 应用场景

- 1、将同一张表中不同的结果（需要对应多条查询语句来实现），合并到一起展示数据
男生身高升序排序，女生身高降序排序
- 2、最常见：在数据量大的情况下，会对表进行分表操作，需要对每张表进行部分数据统计，

使用联合查询来将数据存放到一起显示。

2.3 基本语法

基本语法：

Select 语句

Union [union 选项]

Select 语句；

Union 选项：与 select 选项基本一样

Distinct：去重，去掉完全重复的数据（默认的）

```
mysql> select * from person;
+----+-----+---+-----+
| id | name | sex | telnum |
+----+-----+---+-----+
| 1  | 张三 | 男 | 13800138000 |
| 2  | 李四 | 男 | 13800138000 |
| 3  | 王五 | 男 | 13800138000 |
+----+-----+---+-----+
3 rows in set (0.00 sec)

mysql> select * from student;
ERROR 1146 (42S02): Table 'czxy.student' doesn't exist
mysql> select * from students;
+----+-----+-----+-----+
| id | name | gender | tel |
+----+-----+-----+-----+
| 1  | 八戒 | 男 | 13912345555 |
| 2  | 大师兄 | 男 | 13912346666 |
| 3  | 金角大王 | 男 | 13922222222 |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select * from person
      -> union
      -> select * from students;
+----+-----+---+-----+
| id | name | sex | telnum |
+----+-----+---+-----+
| 1  | 张三 | 男 | 13800138000 |
| 2  | 李四 | 男 | 13800138000 |
| 3  | 王五 | 男 | 13800138000 |
| 1  | 八戒 | 男 | 13912345555 |
| 2  | 大师兄 | 男 | 13912346666 |
| 3  | 金角大王 | 男 | 13922222222 |
+----+-----+---+-----+
6 rows in set (0.00 sec)
```

All：保存所有的结果

注意细节：

- 1、联合后，字段显示的以第一个表的字段为准
- 2、如果使用 `select *` 进行多表联合，必须保证两个表的字段要一致，否则报错：

```
mysql> select * from person
      -> union
      -> select * from course;
ERROR 1222 (HY000): The used SELECT statements have a different number of columns
mysql>
```

- 3、如果两个表字段个数不相同，但是又希望联合在一起，此时需要使用 `select` 指定相同个数的字段

```
mysql> select id, name from person
      -> union
      -> select * from course;
+----+-----+
| id | name |
+----+-----+
| 1  | 张三  |
| 2  | 李四  |
| 3  | 王五  |
| 1  | HTML |
| 2  | CSS   |
| 3  | JavaScript |
| 4  | jQuery |
| 5  | Bootstrap |
| 6  | PHP   |
| 7  | MySQL |
+----+-----+
10 rows in set (0.00 sec)
```

指定两个字段

表本身只有两个字段

- 4、`union` 默认情况下，对查询后联合的结果进行 `distinct`（去除重复了）

```
mysql> select name, sex from person
      -> union
      -> select name, gender from students;
+-----+-----+
| name  | sex   |
+-----+-----+
| 张三  | 男    |
| 李四  | 男    |
| 王五  | 男    |
| 金角大王 | 男    |
| 八戒  | 男    |
| 大师兄 | 男    |
+-----+-----+
6 rows in set (0.00 sec)
```

少了一个 金角大王

如果不希望，我们可以使用 `union all` 来进行联合

```
mysql> select name,sex from person
-> union all
-> select name,gender from students;
+-----+-----+
| name | sex |
+-----+-----+
| 张三 | 男 |
| 李四 | 男 |
| 王五 | 男 |
| 金角大王 | 男 |
| 八戒 | 男 |
| 大师兄 | 男 |
| 金角大王 | 男 |
+-----+
7 rows in set (0.00 sec)
```

students 表中的

person表

```
mysql> -
```

union 理论上只要保证字段数一样，不需要每次拿到的数据对应的字段类型一致。永远只保留第一个 **select** 语句对应的字段名字。

2.4 Order by 的使用

1、在联合查询中，如果要使用 **order by**，那么对应的 **select** 语句必须使用括号括起来

```
-> select id, name from students;
+----+-----+
| id | name |
+----+-----+
| 1  | 张三   | 5
| 2  | 李四   | 4
| 3  | 王五   | 3
| 4  | 金角大王 | 2
| 1  | 八戒   | 1
| 2  | 大师兄 | 1
| 3  | 金角大王 | 2
+----+-----+
7 rows in set (0.00 sec)
mysql>
```

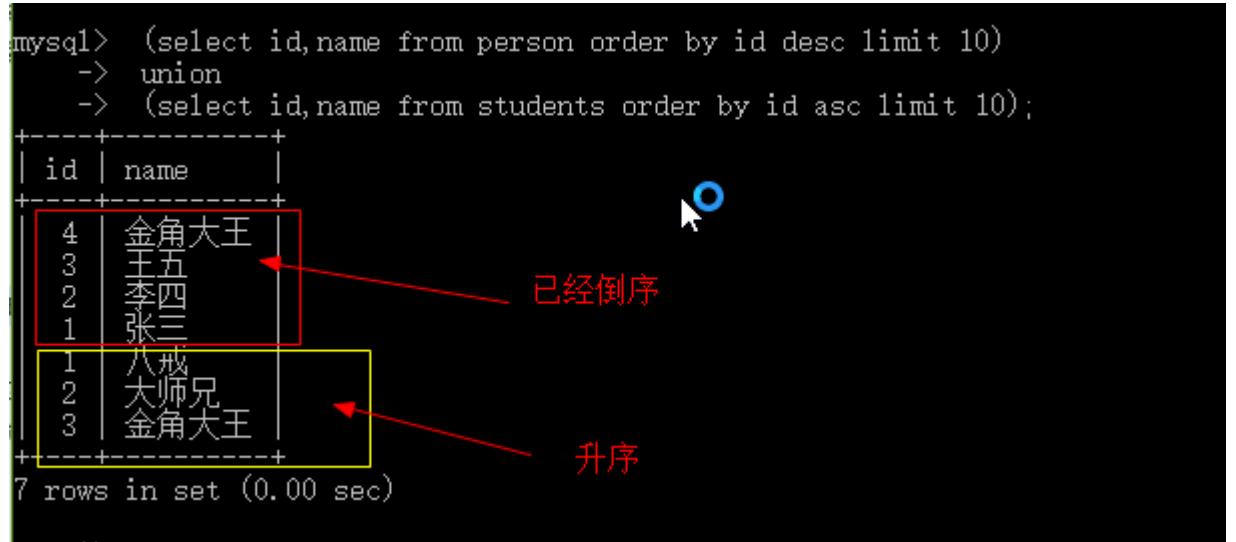
```
mysql> select * from member order by id asc
      -> union all
      -> select * from user order by id desc;
ERROR 1221 (HY000): Incorrect usage of UNION and ORDER BY
mysql>
```

正确的语法：加上括号

```
mysql> (select * from member order by id asc)
      -> union all
      -> (select * from user order by id desc);
+----+-----+-----+
| id | nicheng    | password |
+----+-----+-----+
| 1  | 老吊线     | 123
| 2  | 二师兄说得对 | 123
| 1  | 老吊线     | 123
| 2  | 孙长老     | 123
| 3  | 二师兄     | 123
| 4  | 大师兄说得对 | 123
| 5  | 农夫三拳   | 123
+----+-----+-----+
7 rows in set (0.00 sec)
```

2、`orderby` 在联合查询中若要生效，必须配合使用 `limit`；而 `limit` 后面必须跟对应的限制数量（通常可以使用一个较大的值：大于对应表的记录数）

```
mysql> (select id,name from person order by id desc limit 10)
-> union
-> (select id,name from students order by id asc limit 10);
+----+-----+
| id | name |
+----+-----+
| 4  | 金角大王 |
| 3  | 王五      |
| 2  | 季四      |
| 1  | 张三      |
+----+-----+
| 1  | 八戒      |
| 2  | 大师兄    |
| 3  | 金角大王 |
+----+-----+
7 rows in set (0.00 sec)
```



3. 连接查询

连接查询：将多张表连到一起进行查询（会导致记录数行和字段数列发生改变）

3.1 连接查询的意义

在关系型数据库设计过程中，实体（表）与实体之间是存在很多联系的。在关系型数据库表的设计过程中，遵循着关系来设计：一对一，一对多和多对多，通常在实际操作的过程中，需要利用这层关系来保证数据的完整性。

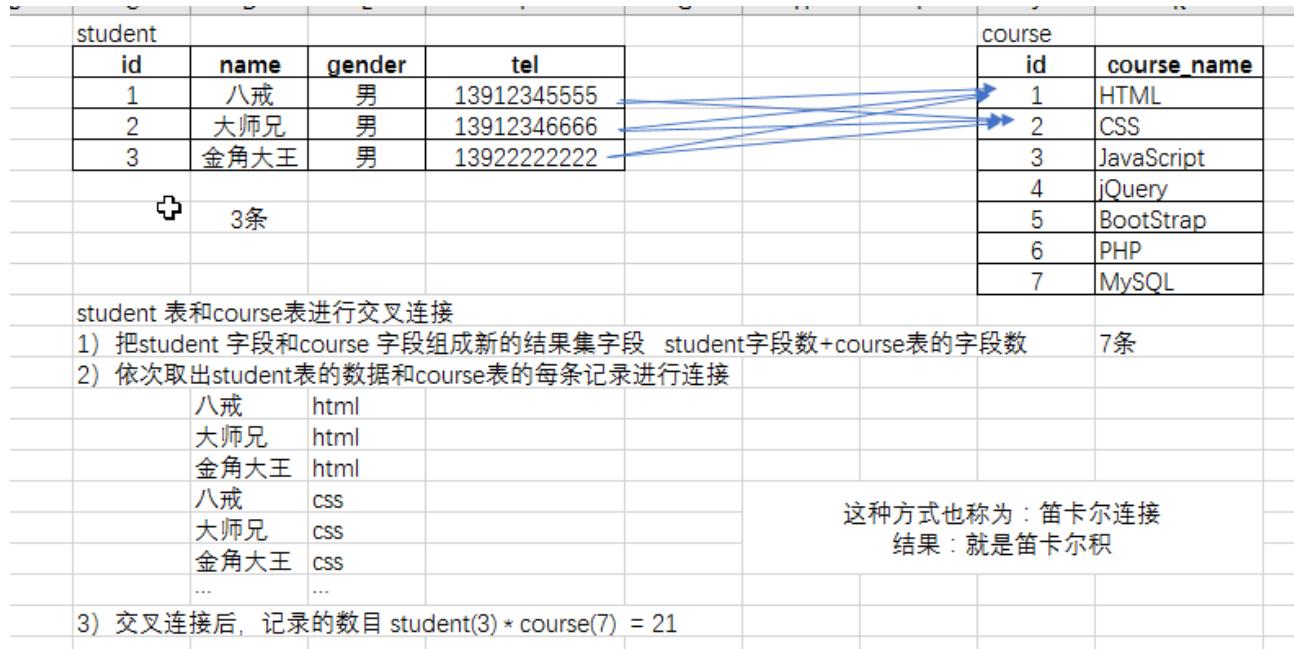
3.2 连接查询分类

连接查询一共有以下几类：

- 1) 交叉连接
- 2) 内连接
- 3) 外连接：左外连接（左连接）和右外连接（右连接）
- 4) 自然连接

4. 交叉连接

交叉连接：将一张表的数据与另外一张表彼此交叉



student 表和course表进行交叉连接

1) 把student 字段和course 字段组成新的结果集字段 student字段数+course表的字段数 7条

2) 依次取出student表的数据和course表的每条记录进行连接

八戒	html
大师兄	html
金角大王	html
八戒	css
大师兄	css
金角大王	css
...	...

3) 交叉连接后, 记录的数目 $student(3) * course(7) = 21$

这种方式也称为：笛卡尔连接
结果：就是笛卡尔积

4.1 原理

- 1、从第一张表依次取出每一条记录
- 2、每条记录都会和第二张表的所有记录进行连接
- 3、记录数 = 第一张表记录数 * 第二张表记录数;
字段数 = 第一张表字段数 + 第二张表字段数（笛卡尔积）

4.2 语法

4.2.1 基本语法：表 1 cross join 表 2;

把 students 表和 course 表进行连接，写法如下：

```
select * from students cross join course;
```

mysql> select * from students cross join course;							
	id	name	gender	tel		id	course_name
1	1	八戒	男	13912345555		1	HTML
2	2	大师兄	男	13912346666		1	HTML
3	3	金角大王	男	13922222222		1	HTML
1	1	八戒	男	13912345555		2	CSS
2	2	大师兄	男	13912346666		2	CSS
3	3	金角大王	男	13922222222		2	CSS
1	1	八戒	男	13912345555		3	JavaScript
2	2	大师兄	男	13912346666		3	JavaScript
3	3	金角大王	男	13922222222		3	JavaScript
1	1	八戒	男	13912345555		4	jQuery
2	2	大师兄	男	13912346666		4	jQuery
3	3	金角大王	男	13922222222		4	jQuery
1	1	八戒	男	13912345555		5	Bootstrap
2	2	大师兄	男	13912346666		5	Bootstrap
3	3	金角大王	男	13922222222		5	Bootstrap
1	1	八戒	男	13912345555		6	PHP
2	2	大师兄	男	13912346666		6	PHP
3	3	金角大王	男	13922222222		6	PHP
1	1	八戒	男	13912345555		7	MySQL
2	2	大师兄	男	13912346666		7	MySQL
3	3	金角大王	男	13922222222		7	MySQL

4.3 应用

交叉连接产生的结果是笛卡尔积，没有实际应用。

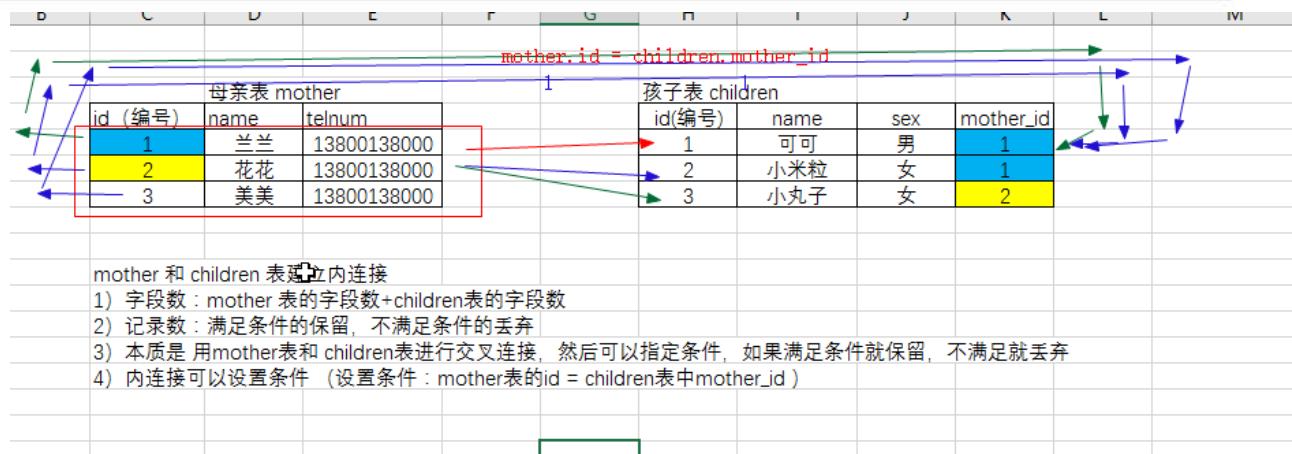
本质：from 表 1,表 2;

5. 内连接

内连接：inner join，从一张表中取出所有的记录去另外一张表中匹配：利用匹配条件进行匹配，成功了则保留，失败了放弃。

5.1 原理

1、从第一张表中取出一条记录，然后去另外一张表中进行匹配



C	D	E	F	G	H	I	J	K	L
id (编号)	name	telnum			id(编号)	name	sex	mother_id	
1	兰兰	13800138000			1	可可	男	1	
2	花花	13800138000			2	小米粒	女	1	
3	美美	13800138000			3	小丸子	女	2	

mother 和 children 表建立内连接

- 字段数：mother 表的字段数+children表的字段数
- 记录数：满足条件的保留，不满足条件的丢弃
- 本质是用mother表和 children表进行交叉连接，然后可以指定条件，如果满足条件就保留，不满足就丢弃
- 内连接可以设置条件（设置条件：mother表的id = children表中mother_id）

查询结果：

1	兰兰	13800138000	1	可可	男	1
1	兰兰	13800138000	2	小米粒	女	1
2	花花	13800138000	3	小丸子	女	2

2、利用匹配条件进行匹配：

- 匹配到：保留，继续向下匹配
- 匹配失败：向下继续，如果全表匹配失败，结束

5.2 语法

基本语法：表 1 [inner] join 表 2 on 匹配条件；

5.2.1 如果内连接没有条件（允许），那么其实就是交叉连接（避免）

```

mysql> use czxy;
Database changed
mysql> select * from mother inner join children;
+----+-----+-----+----+-----+----+-----+
| id | name | telnum | id | name | sex | mother_id |
+----+-----+-----+----+-----+----+-----+
| 1  | 兰兰 | 13800138000 | 1  | 可可 | 男   |      1      |
| 2  | 花花 | 13800138000 | 1  | 可可 | 男   |      1      |
| 3  | 美美 | 13800138000 | 1  | 可可 | 男   |      1      |
| 1  | 兰兰 | 13800138000 | 2  | 小米粒 | 女 |      1      |
| 2  | 花花 | 13800138000 | 2  | 小米粒 | 女 |      1      |
| 3  | 美美 | 13800138000 | 2  | 小米粒 | 女 |      1      |
| 1  | 兰兰 | 13800138000 | 3  | 小丸子 | 女 |      2      |
| 2  | 花花 | 13800138000 | 3  | 小丸子 | 女 |      2      |
| 3  | 美美 | 13800138000 | 3  | 小丸子 | 女 |      2      |
+----+-----+-----+----+-----+----+-----+
9 rows in set (0.00 sec)

mysql> -

```

5.2.2 使用匹配条件进行匹配，但可能会报条件“模棱两可”

ambiguous-模糊的，不明确的

```

mysql> select * from mother inner join children on id = mother_id;
ERROR 1052 (23000): Column 'id' in on clause is ambiguous
mysql> -

```

5.2.3 因为表的设计通常容易产生同名字段，尤其是 ID，所以为了避免重名出现错误，通常使用表名.字段名，来确保唯一性

```

mysql> select * from mother inner join children on mother.id = children.mother_id;
+----+-----+-----+----+-----+----+-----+
| id | name | telnum | id | name | sex | mother_id |
+----+-----+-----+----+-----+----+-----+
| 1  | 兰兰 | 13800138000 | 1  | 可可 | 男   |      1      |
| 1  | 兰兰 | 13800138000 | 2  | 小米粒 | 女 |      1      |
| 2  | 花花 | 13800138000 | 3  | 小丸子 | 女 |      2      |
+----+-----+-----+----+-----+----+-----+
3 rows in set (0.00 sec)

```

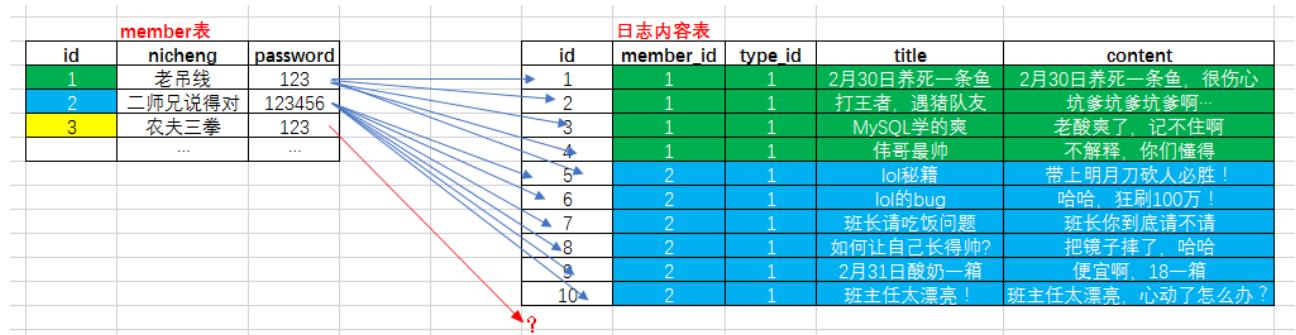
5.2.4 通常，如果条件中使用到对应的表名，而表名通常比较长，所以可以通过表别名来简化

```
mysql> select * from mother m inner join children c on m.id = c.mother_id;
+----+-----+-----+----+-----+-----+
| id | name | telnum | id | name | sex | mother_id |
+----+-----+-----+----+-----+-----+
| 1  | 三三  | 13800138000 | 1  | 可可  | 男  | 1          |
| 1  | 三三  | 13800138000 | 2  | 小米粒 | 女  | 1          |
| 2  | 花花  | 13800138000 | 3  | 小丸子 | 女  | 2          |
+----+-----+-----+----+-----+-----+
3 rows in set (0.00 sec)
```

5.2.5 内连接匹配的时候，必须保证匹配到才会保存

比如：此时模拟注册一个用户“农夫三拳”

```
insert into member values(null, '农夫三拳', '123');
```



因为此时在 blog 表中并没有“农夫三拳”的匹配，所以内连接后依然不会显示“农夫三拳”

```
mysql> insert into member values(null, '农夫三拳', '123');
Query OK, 1 row affected (0.00 sec)

mysql> select * from member as m inner join blog as b on m.id = b.member_id;
+----+-----+-----+----+-----+-----+-----+-----+
| id | nicheng | password | id | member_id | type_id | title | content |
+----+-----+-----+----+-----+-----+-----+-----+
| 1  | 老吊线  | 123     | 1  | 1         | 1       | 日记：2月30日 晴 | 今天一天都没出太阳，真不好，爸爸买回两条鱼，养在水缸里淹死一条，我很伤心！
| 1  | 老吊线  | 123     | 2  | 1         | 1       | 打王者，遇猪队友 | 坑爹坑爹坑爹啊...
| 1  | 老吊线  | 123     | 3  | 1         | 1       | MySQL学的爽      | 老酸爽了，记不住啊
| 1  | 老吊线  | 123     | 4  | 1         | 1       | 伟哥最帅          | 不解释，你们懂得
| 2  | 二师兄说得对 | 123456 | 5  | 2         | 1       | lol秘籍          | 带上明月刀砍人必胜！
| 2  | 二师兄说得对 | 123456 | 6  | 2         | 1       | lol的bug          | 哈哈，狂刷100万！
| 2  | 二师兄说得对 | 123456 | 7  | 2         | 1       | 班长请吃饭问题    | 班长你到底请不请
| 2  | 二师兄说得对 | 123456 | 8  | 2         | 1       | 如何让自己长得帅？ | 把镜子摔了，哈哈
| 2  | 二师兄说得对 | 123456 | 9  | 2         | 1       | 2月31日酸奶一箱   | 便宜啊，18一箱
| 2  | 二师兄说得对 | 123456 | 10 | 2         | 1       | 班主任太漂亮！      | 班主任太漂亮，心动了怎么办？
```

5.2.6 内连接因为不强制必须使用匹配条件（on）因此可以在数据匹配完成之后，使用 where 条件来限制，效果与 on 一样（建议使用 on）

```
mysql> select * from mother m inner join children c where m.id = c.mother_id and c.name='小米粒';
+----+-----+-----+----+-----+-----+
| id | name | telnum | id | name | sex | mother_id |
+----+-----+-----+----+-----+-----+
| 1  | 兰兰 | 13800138000 | 2 | 小米粒 | 女 | 1          |
+----+-----+-----+----+-----+-----+
1 row in set (0.00 sec)
```

使用 on ... and... 多个条件

5.2.7 内连接更普遍的写法

select 字段列表 from 表 a, 表 b where a.字段 = b.字段

```
mysql> select * from mother m ,children c where m.id = c.mother_id;
+----+-----+-----+----+-----+-----+
| id | name | telnum | id | name | sex | mother_id |
+----+-----+-----+----+-----+-----+
| 1  | 兰兰 | 13800138000 | 1 | 可可 | 男 | 1          |
| 1  | 兰兰 | 13800138000 | 2 | 小米粒 | 女 | 1          |
| 2  | 花花 | 13800138000 | 3 | 小丸子 | 女 | 2          |
+----+-----+-----+----+-----+-----+
3 rows in set (0.00 sec)
```

添加条件

```
mysql> select * from mother m ,children c where m.id = c.mother_id and c.name='小米粒';
+----+-----+-----+----+-----+-----+
| id | name | telnum | id | name | sex | mother_id |
+----+-----+-----+----+-----+-----+
| 1  | 兰兰 | 13800138000 | 2 | 小米粒 | 女 | 1          |
+----+-----+-----+----+-----+-----+
1 row in set (0.00 sec)
```

5.2.8 内连接后，可以查询指定字段的信息，而不是所有的信息

```

c:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe
+-----+-----+-----+-----+-----+-----+-----+-----+
| 3 | 王五 | 男 | 13800138000 | 2 | 2 | 河南 | n |
| 3 | 王五 | 男 | 13800138000 | 3 | 3 | 河北 | y |
| 4 | 金角大王 | 男 | 13922222222 | 1 | 1 | 山东 | y |
| 4 | 金角大王 | 男 | 13922222222 | 2 | 2 | 河南 | n |
| 4 | 金角大王 | 男 | 13922222222 | 3 | 3 | 河北 | y |
+-----+-----+-----+-----+-----+-----+-----+-----+
12 rows in set (0.00 sec)

mysql> select * from person p , person_detail pd where p.id = pd.u_id;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | sex | telnum | id | u_id | jiguang | is_merry_state | address |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 张三 | 男 | 13800138000 | 1 | 1 | 山东 | y | 山东省济南市 |
| 2 | 李四 | 男 | 13800138000 | 2 | 2 | 河南 | n | 河南省郑州市 |
| 3 | 王五 | 男 | 13800138000 | 3 | 3 | 河北 | y | 河北省石家庄市 |
+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select p.id,p.name,p.sex,p.telnum,pd.jiguang,pd.is_merry_state,pd.address from person p , person_detail pd where p.id = pd.u_id;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | sex | telnum | jiguang | is_merry_state | address |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | 张三 | 男 | 13800138000 | 山东 | y | 山东省济南市 |
| 2 | 李四 | 男 | 13800138000 | 河南 | n | 河南省郑州市 |
| 3 | 王五 | 男 | 13800138000 | 河北 | y | 河北省石家庄市 |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>

```

5.3 应用

内连接通常是在对数据有精确要求的地方使用：必须保证两种表中都能进行数据匹配。

6. 外连接

外链接：outer join，按照某一张表作为主表（表中所有记录在最后都会保留），根据条件去连接另外一张表，从而得到目标数据。

外连接分为两种：左外连接（left join），右外连接（right join）

左连接：左表是主表

右连接：右表是主表

6.1 原理

母亲表 mother			孩子表 children			
id (编号)	name	telnum	id(编号)	name	sex	mother_id
1	兰兰	13800138000	1	可可	男	1
2	花花	13800138000	2	小米粒	女	1
3	美美	13800138000	3	小丸子	女	2

主表

mother 表和 children 表建立外连接

- 1) 外连接, outer join
- 2) 外连接分为 : 左外连接 left join 和 右外连接 right join
- 3) 外连接有主表, 连接的过程是 : 从主表中取出数据去连接的表中依次查找, 如果满足条件则表保留
- 4) 外连接一定会保存主表中的所有的记录
- 5) 连接条件必须写, on mother.id = children.mother_id;
- 6) 关于主表 如果使用的 left join 左边的表就是主表, 如果使用的 right join 那么右侧的表是主表
- 7) 如果主表的记录, 在连接的表中没有查找到匹配的记录, 那么此时主表的记录依然会保存, 但是连接的表的信息全为 NULL
- 8) 外连接后, 表中的记录等于主表的记录数

6.2 语法

基本语法:

左连接: 主表 left join 从表 on 连接条件;

右连接: 从表 right join 主表 on 连接条件;

6.2.1 左连接对应的主表数据在左边;右连接对应的主表数据在右边:

```
mysql> select * from mother left join children on mother.id = children.mother_id;
+----+-----+-----+----+-----+-----+-----+
| id | name | telnum | id | name | sex | mother_id |
+----+-----+-----+----+-----+-----+-----+
| 1  | 兰兰 | 13800138000 | 1  | 可可 | 男   | 1       |
| 1  | 兰兰 | 13800138000 | 2  | 小米粒 | 女   | 1       |
| 2  | 花花 | 13800138000 | 3  | 小丸子 | 女   | 2       |
| 3  | 美美 | 13800138000 | NULL | NULL | NULL | NULL    |
+----+-----+-----+----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> -
```

6.2.2 右连接查看数据

```
mysql> select * from mother right join children on mother.id = children.mother_id;
+----+----+----+----+----+----+----+
| id | name | telnum | id | name | sex | mother_id |
+----+----+----+----+----+----+----+
| 1  | 兰兰 | 13800138000 | 1  | 可可 | 男  | 1        |
| 1  | 兰兰 | 13800138000 | 2  | 小米粒 | 女 | 1        |
| 2  | 花花 | 13800138000 | 3  | 小丸子 | 女 | 2        |
| NULL | NULL | NULL | 4  | 小爱 | 女 | 4        |
+----+----+----+----+----+----+----+
4 rows in set (0.00 sec)
```

外连接的时候，也可以使用别名

```
mysql> select * from mother m right join children c on m.id = c.mother_id;
+----+----+----+----+----+----+----+
| id | name | telnum | id | name | sex | mother_id |
+----+----+----+----+----+----+----+
| 1  | 兰兰 | 13800138000 | 1  | 可可 | 男  | 1        |
| 1  | 兰兰 | 13800138000 | 2  | 小米粒 | 女 | 1        |
| 2  | 花花 | 13800138000 | 3  | 小丸子 | 女 | 2        |
| NULL | NULL | NULL | 4  | 小爱 | 女 | 4        |
+----+----+----+----+----+----+----+
4 rows in set (0.00 sec)
```

特点

- 1、外连接中主表数据记录一定会保存：连接之后不会出现记录数少于主表（内连接可能）
- 2、左连接和右连接其实可以互相转换，但是数据对应的位置（表顺序）会改变

6.3 应用

非常常用的一种获取的数据方式：作为数据获取对应主表以及其他数据（关联）

7. Using 关键字

是在连接查询中用来代替对应的 on 关键字的，进行条件匹配。

7.1 原理

- 1、在连接查询时，使用 on 的地方用 using 代替
- 2、使用 using 的前提是对应的两张表连接的**字段是同名**（类似自然连接自动匹配）

mysql> select * from mother m inner join children c on m.id = c.mother_id;						
mother_id	id	name	telnum	id	name	sex
1	1	兰兰	13800138000	1	可可	男
1	1	兰兰	13800138000	2	小米粒	女
2	2	花花	13800138000	3	小丸子	女

3、如果使用 using 关键字，那么对应的同名字段，最终在结果中只会保留一个。

7.2 语法

基本语法：表 1 [inner, left, right] join 表 2 using(同名字段列表); //连接字段

第一步：修改 mother 的 id 字段名称为 mother_id(为了和 children 表中的 mother_id 字段一致)

```
alter table mother change id mother_id int(10) unsigned auto_increment;
```

Field	Type	Null	Key	Default	Extra
mother_id	int(10) unsigned	NO	PRI	NULL	auto_increment
name	varchar(10)	NO		NULL	
telnum	bigint(11)	YES		NULL	

7.2.1 使用内连接演示

mysql> select * from mother inner join children using(mother_id);						
mother_id	name	telnum	id	name	sex	
1	兰兰	13800138000	1	可可	男	
1	兰兰	13800138000	2	小米粒	女	
2	花花	13800138000	3	小丸子	女	

mother表和children表有相同的字段 mother_id, 此时就可以使用using

7.2.2 对比查看结果

```
mysql> select * from mother inner join children using(mother_id);
+-----+-----+-----+-----+-----+
| mother_id | name | telnum | id | name | sex |
+-----+-----+-----+-----+-----+
| 1 | 兰兰 | 13800138000 | 1 | 可可 | 男 |
| 1 | 兰兰 | 13800138000 | 2 | 小米粒 | 女 |
| 2 | 花花 | 13800138000 | 3 | 小丸子 | 女 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from mother inner join children on mother.mother_id = children.mother_id;
+-----+-----+-----+-----+-----+-----+
| mother_id | name | telnum | id | name | sex | mother_id |
+-----+-----+-----+-----+-----+-----+
| 1 | 兰兰 | 13800138000 | 1 | 可可 | 男 | 1 |
| 1 | 兰兰 | 13800138000 | 2 | 小米粒 | 女 | 1 |
| 2 | 花花 | 13800138000 | 3 | 小丸子 | 女 | 2 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

8. 整库数据备份与还原

整库数据备份也叫 SQL 数据备份：备份的结果都是 SQL 指令

在 Mysql 中提供了一个专门用于备份 SQL 的客户端：mysqldump.exe

 mysql.exe	2016/7/12 16:07	应用程序	38,951 KB
 mysql.pdb	2016/7/12 16:07	PDB 文件	169,180 KB
 mysql_multi.pl	2016/7/12 14:41	PL 文件	28 KB
 mysqldump.exe	2016/7/12 15:55	应用程序	5,268 KB
 mysql dumpslow.pl	2016/7/12 14:41	PL 文件	8 KB
 mysqlimport.exe	2016/7/12 15:55	应用程序	5,191 KB
 mysqlpump.exe	2016/7/12 15:57	应用程序	6,261 KB
 mysqlshow.exe	2016/7/12 15:55	应用程序	5,188 KB

8.1 应用场景

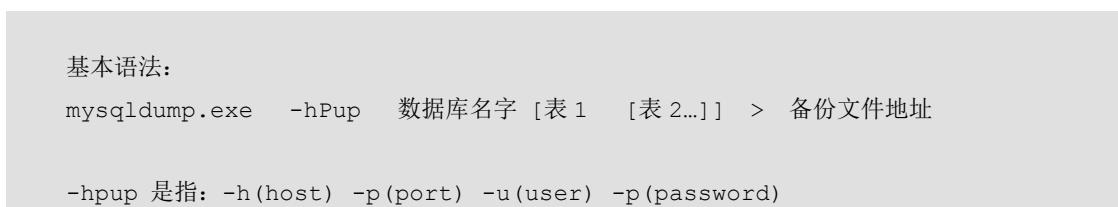
SQL 备份是一种 mysql 非常常见的备份与还原方式，SQL 备份不只是备份数据，还备份对应的 SQL 指令（表结构）：即便是数据库遭到毁灭性的破坏（数据库被删），那么利用 SQL 备份依然可以实现数据还原。

SQL 备份因为需要备份结构，因此产生的备份文件特别大，因此不适合特大型数据备份，也不适合数据变换频繁型数据库备份。

8.2 应用方案

8.2.1 SQL 备份

SQL 备份用到的是专门的备份客户端，因此还没与数据库服务器进行连接。



备份可以有三种形式：

- 整库备份（只需要提供数据库名字）

备份数据库需要更高的操作权限，如下图要以管理员身份运行命令行



打开命令提示行，输入导出指令

```
C:\wamp64\bin\mysql\mysql5.7.14\bin\mysqldump.exe -uroot -p123 czxy >  
c:\czxy.sql
```



```
C:\>管理员:命令提示符
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation. 保留所有权利。

C:\WINDOWS\system32>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysqldump.exe -uroot -p123 czxy > c:\czxy.sql
mysqldump: [Warning] Using a password on the command line interface can be insecure
C:\WINDOWS\system32>
```

- 单表备份：数据库后面跟一张表

导出时可以不在命令中书写密码，就没有上述的警告了

```
C:\wamp64\bin\mysql\mysql5.7.14\bin\mysqldump.exe -uroot -p czxy students >
c:\students.sql
```

导出单表：

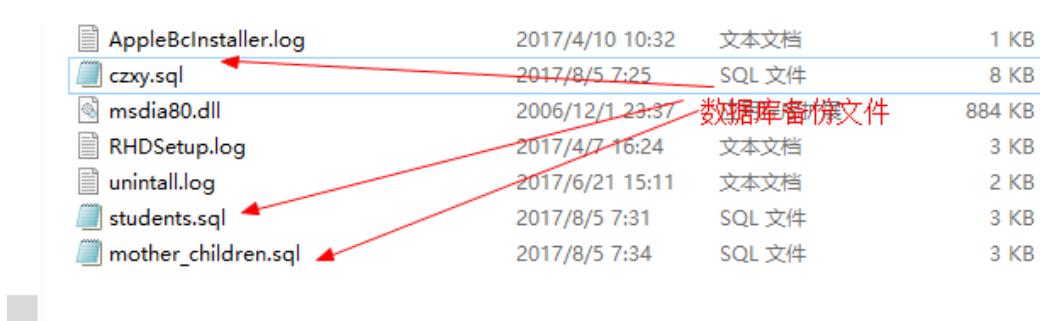
```
C:\WINDOWS\system32>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysqldump.exe -uroot -p czxy students > c:\students.sql
Enter password: ***
C:\WINDOWS\system32>
```

- 多表备份：数据库后跟多张表

```
C:\wamp64\bin\mysql\mysql5.7.14\bin\mysqldump.exe -uroot -p czxy mother
children > c:\mother_children.sql
```

```
C:\WINDOWS\system32>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysqldump.exe -uroot -p czxy mother children > c:\mother_children.sql
Enter password: ***
C:\WINDOWS\system32>
```

查看备份的成果



AppleBcInstaller.log	2017/4/10 10:32	文本文档	1 KB
czxy.sql	2017/8/5 7:25	SQL 文件	8 KB
msdia80.dll	2006/12/1 23:37	数据库备份文件	884 KB
RHDSetup.log	2017/4/7 16:24	文本文档	3 KB
uninstall.log	2017/6/21 15:11	文本文档	2 KB
students.sql	2017/8/5 7:31	SQL 文件	3 KB
mother_children.sql	2017/8/5 7:34	SQL 文件	3 KB

查看备份文件中的具体内容

```
20 --
21 |DROP TABLE IF EXISTS `children`;| 如果表存在，则先删除
22 /*!40101 SET @saved_cs_client      = @@character_set_client */;
23 /*!40101 SET character set client = utf8 */;
24 CREATE TABLE `children` (
25   `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
26   `name` varchar(10) NOT NULL,
27   `sex` enum('男','女') DEFAULT NULL,
28   `mother_id` int(10) unsigned NOT NULL,
29   PRIMARY KEY (`id`)
30 ) ENGINE=MyISAM AUTO_INCREMENT=5 DEFAULT CHARSET=utf8;
31 /*!40101 SET character_set_client = @saved_cs_client */;
32
33 --
34 -- Dumping data for table `children`
35 --
36
37 LOCK TABLES `children` WRITE;
38 /*!40000 ALTER TABLE `children` DISABLE KEYS */;
39 INSERT INTO `children` VALUES (1,'可可','男',1),(2,'小米粒','女',1),(3,'小丸子','女',2),(4,'小
40 /*!40000 ALTER TABLE `children` ENABLE KEYS */;
41 UNLOCK TABLES;
42
43
44 --
45 -- Table structure for table `course`
```

8.2.2 数据还原

MySQL 提供了多种方式来实现：两种

mysqldump 备份的数据中没有关于数据库本身的操作，都是针对表级别的操作：当进行数据（SQL 还原），必须指定数据库

模拟数据库误删除了

```

+-----+
| information_schema
| baijia
| czxy
| czxy_2017_db
| czxy_new
| mydatabase1
| mydb
| mydb3
| mysql
| performance_schema
| sys
+-----+
11 rows in set (0.00 sec)

mysql> drop database czxy;
Query OK, 7 rows affected (0.03 sec)

mysql> show databases;
+-----+
| Database
+-----+
| information_schema
| baijia
| czxy_2017_db
| czxy_new
| mydatabase1
| mydb
| mydb3
| mysql
| performance_schema
| sys
+-----+
10 rows in set (0.00 sec)

```

模拟czxy数据库被误删除

此处已经没有数据库了

- 利用 mysql.exe 客户端：没有登录之前，可以直接用该客户端进行数据还原
 mysql.exe -hPup 数据库 < 文件位置

注意：此时要先创建数据库并且切换到该库中，才能执行还原操作

```

mysql> show databases;
+-----+
| Database
+-----+
| information_schema
| baijia
| czxy
| czxy_2017_db
| czxy_new
| mydatabase1
| mydb
| mydb3
| mysql
| performance_schema
| sys
+-----+
11 rows in set (0.00 sec)

mysql> use czxy;
Database changed
mysql> show tables;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'tables' at line 1
mysql> show tables;
Empty set (0.00 sec)

```

Enter password: ***
 C:\WINDOWS\system32>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -uroot -p czxy < c:\czxy.sql
 Enter password: ***
 C:\WINDOWS\system32>

到入成功！

```
mysql> show tables;
+----------------+
| Tables_in_czxy |
+----------------+
| children        |
| course          |
| mother          |
| person          |
| person_detail   |
| student_course  |
| students         |
+----------------+
7 rows in set (0.00 sec)

mysql>
```

- 在 SQL 指令，提供了一种导入 SQL 指令的方式

Source SQL 文件位置； //必须先进入到对应的数据库

```
mysql> source c:\mother_children.sql;
ERROR:
Unknown command '\m'.
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

- 人为操作：打开备份文件，复制所有 SQL 指令，然后到 mysql.exe 客户端中去粘贴执行。
(不推荐)

```
mysql> show tables;
+-----+
| Tables_in_czxy |
+-----+
| children      |
| course        |
| mother        |
| person         |
| person_detail |
| student_course|
| students       |
+-----+
7 rows in set (0.00 sec)

mysql> select * from mother;
+-----+-----+-----+
| mother_id | name   | telnum |
+-----+-----+-----+
| 1         | 兰兰   | 13800138000 |
| 2         | 花花   | 13800138000 |
| 3         | 美美   | 13800138000 |
+-----+-----+-----+
3 rows in set (0.00 sec)                                数据已经导入成功

mysql> select * from children;
+-----+-----+-----+-----+
| id   | name   | sex    | mother_id |
+-----+-----+-----+-----+
| 1     | 可可   | 男     | 1          |
| 2     | 小米粒 | 女     | 1          |
| 3     | 小丸子 | 女     | 2          |
| 4     | 小爱   | 女     | 4          |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)                                数据已经导入成功
```

9. 今日总结



传智专修学院
CHUANZHI SPECIALIZED COLLEGE

力学笃行 志存高远

MySQL (六)

1. 子查询

盒子里面放盒子，里面的盒子可以叫子盒子。



1.1 什么是子查询

子查询，在一条 SQL 语句中嵌入另一条 SQL 语句。

- 主查询
我们管最外层的查询叫做主查询。
- 子查询
嵌入的 SQL 叫做子查询。

1.2 子查询分类

1.2.1 按返回结果分

标量子查询：结果返回一个值。（一行一列）

列子查询：返回结果是一列。（多行一列）

行子查询：返回结果是一行。（一行多列）

表子查询：返回结果是多行多列。

exists 子查询：返回的结果是 1 或者 0。（类似布尔操作）

1.2.2 按位置分

where 子查询：子查询出现在 where 的位置上，用子查询的结果做为条件。

from 子查询：子查询出现在 from 的位置上，用子查询的结果做为一张表。

select 子查询：出现在 select 后面列的位置上，用子查询的结果做为一列。

1.3 准备数据

```
mysql> select * from class;
+----+-----+
| id | name |
+----+-----+
| 1  | 全栈开发班 |
| 2  | Java开发班 |
| 3  | 服装设计班 |
| 4  | 美容美发班 |
| 5  | 挖掘机精英班 |
| 6  | 美国总统速成班 |
+----+-----+
6 rows in set (0.00 sec)

mysql> select * from students;
+----+-----+-----+-----+-----+-----+-----+
| id | name   | gender | tel      | class_id | age   | height |
+----+-----+-----+-----+-----+-----+-----+
| 1  | 八戒   | 男     | 13912345555 | 1        | 20    | 180   |
| 2  | 大师兄 | 男     | 13912346666 | 1        | 21    | 175   |
| 3  | 金角大王 | 男    | 13922222222 | 2        | 204   | 195   |
| 4  | 西施   | 女     | 13912342343 | 4        | 17    | 165   |
| 5  | 大乔   | 女     | 13912346666 | 3        | 22    | 168   |
| 6  | 貂蝉   | 女     | 13922233222 | 1        | 21    | 164   |
| 7  | 大乔   | 男     | 13922225454 | 2        | 19    | 183   |
+----+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

```
create database subquery default character set utf8;
use subquery;

create table class
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '班级',
    primary key(id)
) comment='班级表';

insert into class
values
(1,'全栈开发班'),
(2,'Java 开发班'),
(3,'服装设计班'),
(4,'美容美发班'),
(5,'挖掘机精英班'),
(6,'美国总统速成班');

# 学生表
create table students
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '姓名',
    gender enum('男','女') not null comment '性别',
    tel bigint unsigned not null comment '手机号',
    class_id int unsigned not null comment '班级 Id',
    age tinyint unsigned not null comment '年龄',
    height tinyint unsigned not null comment '身高, 单位: 厘米',
    primary key(id)
) comment='学生表';

insert into students
values
(1,'八戒','男',13912345555,1,20,180),
(2,'大师兄','男',13912346666,1,21,175),
(3,'金角大王','男',13922222222,2,204,195),
(4,'西施','女',13912342343,4,17,165),
(5,'大乔','女',13912346666,3,22,168),
(6,'貂蝉','女',13922233222,1,21,164),
(7,'大乔','男',13922225454,2,19,183);
```

2. 标量子查询

标量子查询，子查询返回的结果是一个值【一行一列】

- 示例：取出“八戒”所在的班级的名称。

```
mysql> select name from class where id = (select class_id from students where name = '八戒');
+-----+
| name |
+-----+
| 全栈开发班 |
+-----+
1 row in set (0.00 sec)
```

说明：子查询返回的是一个值：【一行一列】

```
mysql> select class_id from students where name = '八戒';
+-----+
| class_id |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

所以我们叫它标量子查询。

3. 列子查询

列子查询，子查询返回结果是一列。

注意：如果子查询返回的结果是一列的话，那么需要使用 `in` 连接不能是`=`。

- 示例：取出所有女同学所在的班级名称。

第一步：取出女同学所在的班级的 ID 【结果是一列】

```
mysql> select class_id from students where gender = '女';
+-----+
| class_id |
+-----+
|      4 |
|      3 |
|      1 |
+-----+
3 rows in set (0.00 sec)
```

第二步：从班级表中取出班级名称

```
mysql> select name from class where id in (select class_id from students where gender = '女');
+-----+
| name   |
+-----+
| 美容美发班 |
| 服装设计班 |
| 全栈开发班 |
+-----+
3 rows in set (0.00 sec)
```

因为子查询的结果是一列，所以需要使用 in 而不是 =

4. 行子查询

行子查询，子查询返回的是一行的数据。

- 示例：找出年龄最大并且身高也是最高的同学。

```
-- 找出这样一个同学：年龄是最大，并且身高也是最高
# 第一步：取出班级最大的年龄 和 最大的身高
select max(age),max(height) from students;
+-----+-----+
| max(age) | max(height) |
+-----+-----+
|      204 |        195 |
+-----+-----+
1 row in set (0.00 sec)
# 第二步：取出年龄=204，并且身高=195
select * from students where (age,height) = (select max(age),max(height) from students);
```

返回的结果是一行

5. 表子查询

表子查询，子查询返回的结果是多行多列的数据。

- 示例：取出每个班级中年龄最大的同学。

思路：先通过子查询把数据根据年龄降序排列，然后再套一层 SQL 对其进行分组。【因为一条 SQL 中是先分组后排序，无法实现先排序后分组】

```
# 第一步：把同一个班级的同学放到时一起【根据class_分组】
select * from students group by class_id

1. 先分组
+-----+-----+-----+-----+-----+-----+
| id | name | gender | tel      | class_id | age | height |
+-----+-----+-----+-----+-----+-----+
| 1  | 八戒  | 男     | 13912345555 | 1        | 20  | 180   |
| 2  | 大师兄 | 男     | 13912346666 | 1        | 21  | 175   |
| 6  | 貂蝉  | 女     | 13922233222 | 1        | 21  | 164   |
| 3  | 金角大王 | 男     | 13922222222 | 2        | 204 | 195   |
| 7  | 大乔    | 男     | 13922225454 | 2        | 19  | 183   |
| 4  | 西施    | 女     | 13912342343 | 4        | 17  | 165   |
| 5  | 大乔    | 女     | 13912346666 | 3        | 22  | 168   |
+-----+-----+-----+-----+-----+-----+
2. mysql只保留每一组中的第一条记录
+-----+-----+-----+-----+-----+
| id | name | gender | tel      | class_id | age | height |
+-----+-----+-----+-----+-----+
| 1  | 八戒  | 男     | 13912345555 | 1        | 20  | 180   |
| 3  | 金角大王 | 男     | 13922222222 | 2        | 204 | 195   |
| 4  | 西施    | 女     | 13912342343 | 4        | 17  | 165   |
| 5  | 大乔    | 女     | 13912346666 | 3        | 22  | 168   |
+-----+-----+-----+-----+-----+
```

3. 因为分组只保留每一组中的第一条记录，而第一条记录并不是年龄最大，所以如果我们先根据age字段子段降序排列，然后再分组，这时候每一组中的第一条记录肯定是这一组中年龄最大的。
所以我们应该先排序再分组，但是，因为在一条SQL语句中是先分组然后才排序的，而我们要先排序后分组，所以一条SQL不行，我们需要写两条。

`select * from students order by age desc`

```
+-----+-----+-----+-----+-----+
| id | name | gender | tel      | class_id | age | height |
+-----+-----+-----+-----+-----+
| 3  | 金角大王 | 男     | 13922222222 | 2        | 204 | 195   |
| 5  | 大乔    | 女     | 13912346666 | 3        | 22  | 168   |
| 2  | 大师兄 | 男     | 13912346666 | 1        | 21  | 175   |
| 6  | 貂蝉  | 女     | 13922233222 | 1        | 21  | 164   |
| 1  | 八戒  | 男     | 13912345555 | 1        | 20  | 180   |
| 7  | 大乔    | 男     | 13922225454 | 2        | 19  | 183   |
| 4  | 西施    | 女     | 13912342343 | 4        | 17  | 165   |
+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

4. 我们把上面这个整个结果做为一个子查询，再其基础上，进行分组，

注意：当子查询做为数据来源时，需要起个别名，比如 c

```
select * from (select * from students order by age desc) as c group by class_id
```

id	name	gender	tel	class_id	age	height
1	八戒	男	13912345555	1	20	180
3	金角大王	男	13922222222	2	204	195
5	大乔	女	13912346666	3	22	168
4	西施	女	13912342343	4	17	165

6. exists 子查询

子查询是否返回结果，返回结果为真，没有结果为假。

- 示例：取出有人报名的班级名称。

```
--取出有人报名的班级名称
```

思路：用班级的id到学生表中去找，如果有学生的class_id和这个班级的ID相同，说明有人报名。

每个班级都到 子查询中找有没有 对应的学生，如果有就显示，否则就过滤掉

```
select * from class a where exists (select id from students b where a.id=b.class_id);
```

id	name
1	全栈开发班
2	Java开发班
3	服装设计班
4	美容美发班

7. 表引擎



7.1 查看 MySQL 中可用的表引擎

可以使用 `show engines` 指令查看 MySQL 服务器支持的表引擎：

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	YES	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
MyISAM	DEFAULT	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL

7.2 MyISAM 引擎

<i>Storage limits</i>	256TB	<i>Transactions</i>	No	<i>Locking granularity</i>	Table
<i>MVCC</i>	No	<i>Geospatial data type support</i>	Yes	<i>Geospatial indexing support</i>	Yes
<i>B-tree indexes</i>	Yes	<i>T-tree indexes</i>	No	<i>Hash indexes</i>	No
<i>Full-text search indexes</i>	Yes	<i>Clustered indexes</i>	No	<i>Data caches</i>	No
<i>Index caches</i>	Yes	<i>Compressed data</i>	Yes [a]	<i>Encrypted data</i> [b]	Yes
<i>Cluster database support</i>	No	<i>Replication support</i> [c]	Yes	<i>Foreign key support</i>	No
<i>Backup / point-in-time recovery</i> [d]	Yes	<i>Query cache support</i>	Yes	<i>Update statistics for data dictionary</i>	Yes

存储限制：256TB。【1TB=1024GB】

事务：不支持

锁的粒度：表级

外键：不支持

7.3 InnoDB 引擎

注意说明：MySQL 官方网站推荐我们使用这种引擎。

<i>Storage limits</i>	64TB	<i>Transactions</i>	Yes	<i>Locking granularity</i>	Row
<i>MVCC</i>	Yes	<i>Geospatial data type support</i>	Yes	<i>Geospatial indexing support</i>	Yes [a]
<i>B-tree indexes</i>	Yes	<i>T-tree indexes</i>	No	<i>Hash indexes</i>	No [b]
<i>Full-text search indexes</i>	Yes [c]	<i>Clustered indexes</i>	Yes	<i>Data caches</i>	Yes
<i>Index caches</i>	Yes	<i>Compressed data</i>	Yes [d]	<i>Encrypted data</i> [e]	Yes
<i>Cluster database support</i>	No	<i>Replication support</i> [f]	Yes	<i>Foreign key support</i>	Yes
<i>Backup / point-in-time recovery</i> [g]	Yes	<i>Query cache support</i>	Yes	<i>Update statistics for data dictionary</i>	Yes

存储限制：64TB。【1TB=1024GB】

事务：支持

锁的粒度：行级

外键：支持

7.4 设置表引擎

在 `create table` 创建表时可以通过 `engine` 属性设置表引擎：

```
create table 表名 (
    字段...
) type|engine=引擎名;
```

8. 事务

8.1 什么是事务？

事务，就是“要完成的一件事情”，比如购物、转账、做饭、洗澡等等。
做一件事情通常由多个步骤组成，如果某个步骤失败了，那就尴尬了……



有时我们要完成一个功能，需要执行多个 SQL 语句，如果这些 SQL 执行到一半突然停电了，那么就会导致这个功能只完成了一半，这种情况很多时候是不允许出现的，比如：银行转账时。

MySQL 中提供了事务的功能，可以确保一个事务中的多条 SQL 语句要就都成功，要就都失败，不会在中途失败。

注意：MySQL 中只有 InnoDB 引擎的表才支持事务功能。

8.2 转账功能

8.2.1 创建表结构

```
# 数据库
create database shiwu default character set utf8;
use shiwu;
# 账号表
create table users
(
    id mediumint unsigned not null auto_increment comment '编号',
    username varchar(50) not null comment '用户名',
    password char(32) not null comment '密码',
    money decimal(10,2) not null default '0.00' comment '钱',
    primary key (id)
)engine=InnoDB;
# 插入测试数据
insert into users
values(1,'八戒','123456',0),
(2,'大师兄','123456',2000);
```

8.2.2 转账功能

“大师兄给八戒转账 100 元钱”。

实现这个功能需要以下两条 SQL 语句：

```
update users set money=money-100 where id = 2; # 大师兄减 100 元钱
update users set money=money+100 where id = 1; # 八戒加 100 元钱
```

如果第一条记录执行完之后突然停电了，第二条 SQL 没有执行，那么大师兄白白丢了 100 元钱。

8.3 事务的使用

可以使用三个指令使用事务：

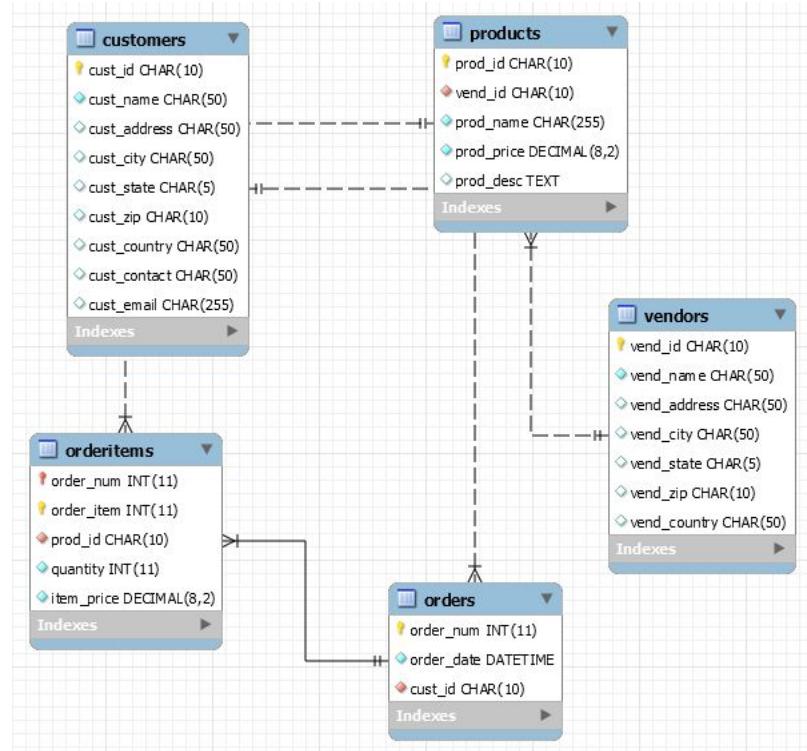
命令	描述
start transaction	开启一个事务。
commit	提交一个事务。
rollback	回滚【取消】一个事务。

```
start transaction;          # 开启一个事务
update users set money=money-100 where id = 2; # 大师兄减 100 元钱
update users set money=money+100 where id = 1; # 八戒加 100 元钱
commit;                  # 提交事务
```

实现原理：MySQL 在执行一个事务之前会先把 SQL 写到“事务日志”中，如果所有 SQL 都成功了才会把数据同步到数据表中，如果中途发生错误，MySQL 会根据事务日志回滚已经完成的操作。

9. 外键

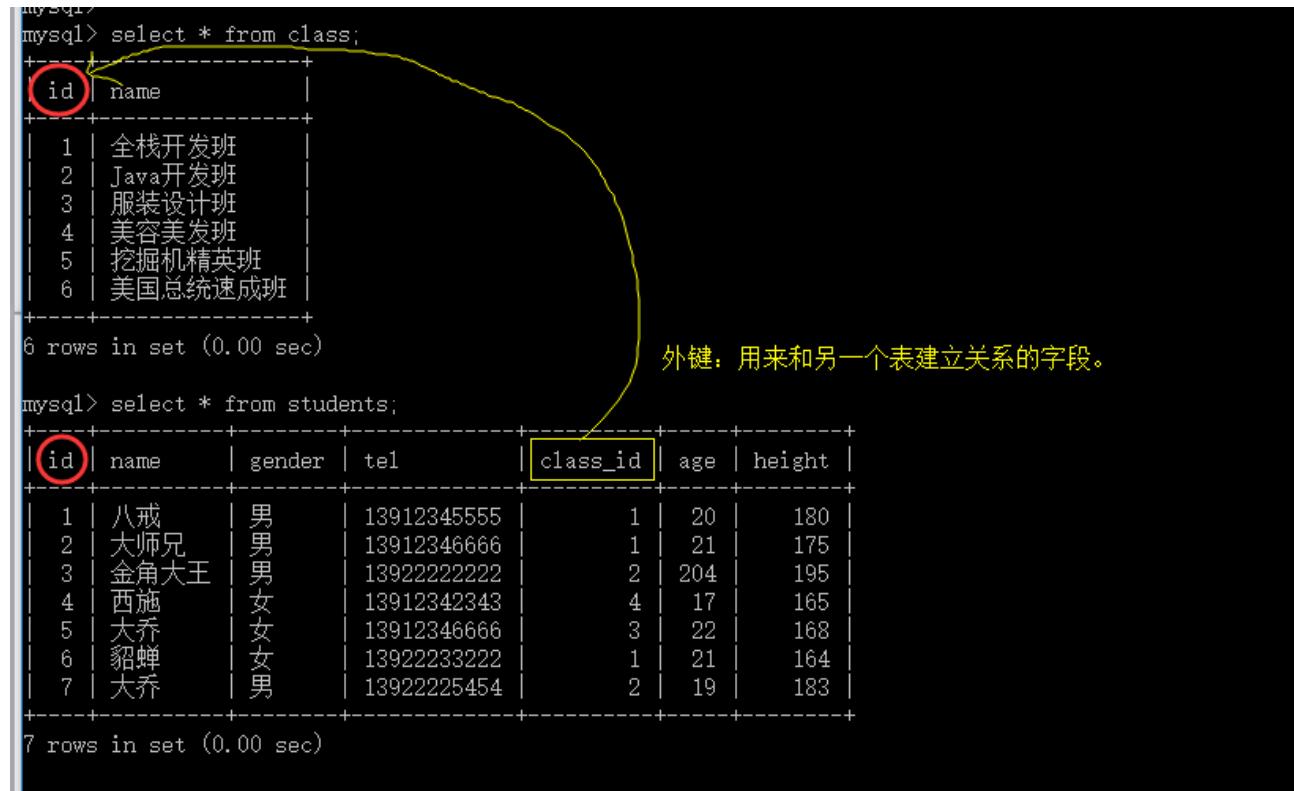
外键，建立表间关系时必不可少的东西。



9.1 什么是外键

外键 (foreign key)，就是另一个表的主键，用来和另一个表建立关系。

注意：只有 InnoDB 引擎支持外键。



外键：用来和另一个表建立关系的字段。

```

mysql> select * from class;
+---+-----+
| id | name |
+---+-----+
| 1 | 全栈开发班 |
| 2 | Java开发班 |
| 3 | 服装设计班 |
| 4 | 美容美发班 |
| 5 | 挖掘机精英班 |
| 6 | 美国总统速成班 |
+---+-----+
6 rows in set (0.00 sec)

mysql> select * from students;
+---+-----+-----+-----+-----+-----+-----+
| id | name | gender | tel | class_id | age | height |
+---+-----+-----+-----+-----+-----+-----+
| 1 | 八戒 | 男 | 13912345555 | 1 | 20 | 180 |
| 2 | 大师兄 | 男 | 13912346666 | 1 | 21 | 175 |
| 3 | 金角大王 | 男 | 13922222222 | 2 | 204 | 195 |
| 4 | 西施 | 女 | 13912342343 | 4 | 17 | 165 |
| 5 | 大乔 | 女 | 13912346666 | 3 | 22 | 168 |
| 6 | 貂蝉 | 女 | 13922233222 | 1 | 21 | 164 |
| 7 | 大乔 | 男 | 13922225454 | 2 | 19 | 183 |
+---+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
  
```

9.2 定义外键

- 添加外键

建添时加外键

```
[constraint 名称] foreign key (字段名) references 表名(字段名)
```

constraint 为外键起一个名字。如果没有起名，MySQL 会默认给起个名字。

```
# 学生表
create table students
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '姓名',
    gender enum('男','女') not null comment '性别',
    tel bigint unsigned not null comment '手机号',
    class_id int unsigned not null comment '班级Id',
    age tinyint unsigned not null comment '年龄',
    primary key(id),
    foreign key class_id references class(id)      → class_id字段是一个外键。
) comment='学生表';                           用来关系class表中的id字段。
```

为一个现有的表添加外键

```
alter table 表名 add [constraint 名称] foreign key (字段名) references 表名(字段名)
```

- 修改外键

注意不能修改键，如果要修改我们只能先删除再重新添加。

- 删除外键

```
alter table 表名 drop foreign key 外键名
```

9.3 外键的基本要求

1. 外键只能和另一个表中的主键建立关系

```
mysql> desc class;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id    | int(10) unsigned | NO   | PRI  | NULL    | auto_increment |
| name  | varchar(20)       | NO   |       | NULL    |                |
| id1   | int(10) unsigned | NO   |       | 0       |                |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> alter table students add foreign key(class_id) references class(id1);
ERROR 1215 (HY000): Cannot add foreign key constraint
mysql>
```

建立失败，因为id1不是另一个表的主键

2. 建立关系的这两个字段必须字段的类型和属性都相同才可以。

```
mysql> desc class;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id    | int(10) unsigned | NO   | PRI  | NULL    | auto_increment |
| name  | varchar(20)       | NO   |       | NULL    |                |
| id1   | int(10) unsigned | NO   |       | 0       |                |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> desc students; 建立外键时类型和属性必须要相同
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| id    | int(10) unsigned | NO   | PRI  | NULL    | auto_increment |
| name  | varchar(20)       | NO   |       | NULL    |                |
| gender | enum('男','女') | NO   |       | NULL    |                |
| tel   | bigint(20) unsigned | NO  |       | NULL    |                |
| class_id | int(10) unsigned | NO   | MUL  | NULL    |                |
| age   | tinyint(3) unsigned | NO  |       | NULL    |                |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

9.4 外键约束

语法：

```
foreign key (字段名) references 主表(主键) on 约束时机 约束模式
```

注意：约束的主键所在的表的操作。

约束时机：

update: 修改时的约束。

delete: 删除时的约束。

约束模式：

restrict: 严格模式，不允许操作。

cascade: 级别操作，从表和主表执行同样的操作。

set null: 置空模式。

no action: 什么也不做。

示例：当外键所在的表中有数据和键所在的表中的数据相关联，那么主键表中的数据是不允许被删除。

```
mysql> alter table students add foreign key (class_id) references class(id) on delete restrict;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

10. 今日总结



传智专修学院
CHUANZHI SPECIALIZED COLLEGE

力学笃行 志存高远

MySQL (七)

1. 用户权限管理

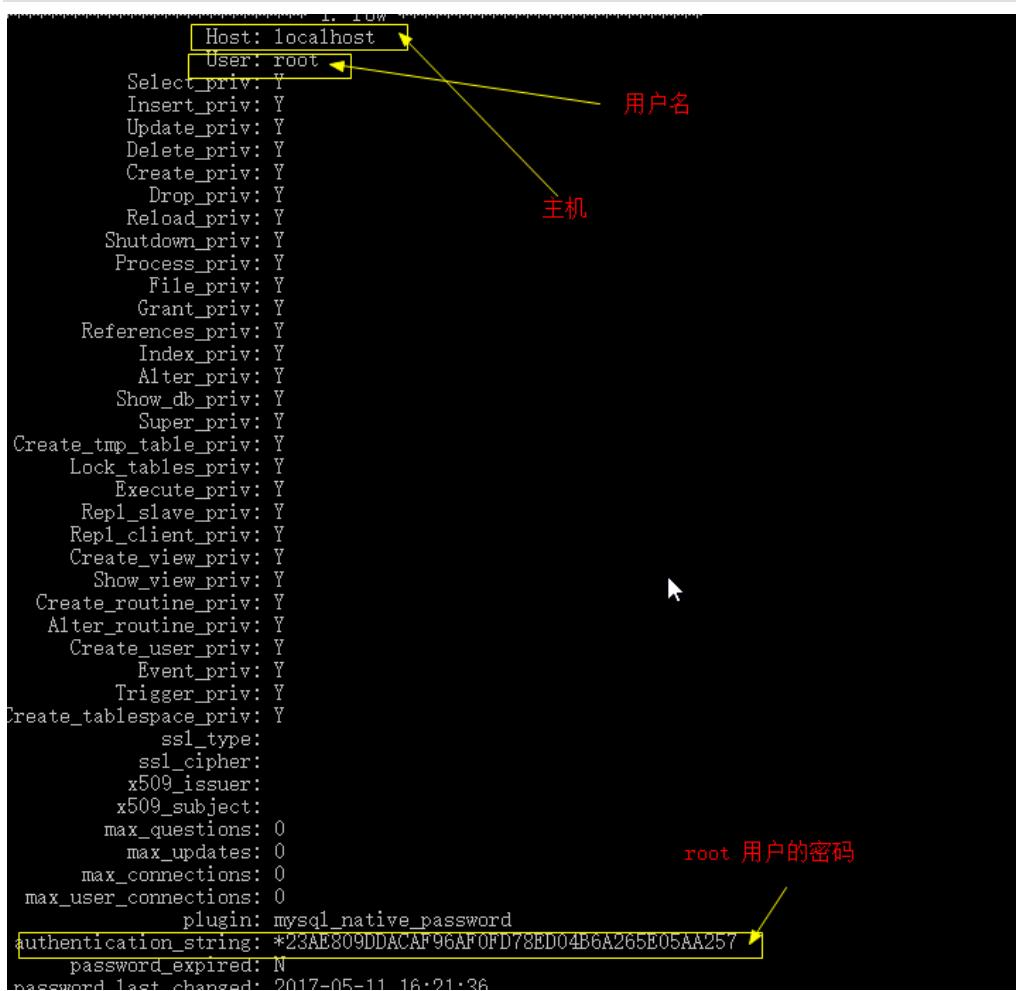
用户权限管理：在不同的项目中给不同的角色（开发者）不同的操作权限，为了保证数据库数据的安全。

通常，一个用户的密码不会长期不变，所以需要经常性的变更数据库用户密码来确保用户本身安全（mysql 客户端用户）

1.1 用户管理

MySQL 需要客户端进行连接认证才能进行服务器操作：需要用户信息。MySQL 中所有的用户信息都是保存在 mysql 数据库下的 user 表中。

```
select * from mysql.user; 可以查看当前数据库的用户
```



```

+-----+-----+
| User | Host |
+-----+-----+
| root | localhost |
|      |          |
|      | Select_priv: Y |
|      | Insert_priv: Y |
|      | Update_priv: Y |
|      | Delete_priv: Y |
|      | Create_priv: Y |
|      | Drop_priv: Y |
|      | Reload_priv: Y |
|      | Shutdown_priv: Y |
|      | Process_priv: Y |
|      | File_priv: Y |
|      | Grant_priv: Y |
|      | References_priv: Y |
|      | Index_priv: Y |
|      | Alter_priv: Y |
|      | Show_db_priv: Y |
|      | Super_priv: Y |
|      | Create_tmp_table_priv: Y |
|      | Lock_tables_priv: Y |
|      | Execute_priv: Y |
|      | Repl_slave_priv: Y |
|      | Repl_client_priv: Y |
|      | Create_view_priv: Y |
|      | Show_view_priv: Y |
|      | Create_routine_priv: Y |
|      | Alter_routine_priv: Y |
|      | Create_user_priv: Y |
|      | Event_priv: Y |
|      | Trigger_priv: Y |
|      | Create_tablespace_priv: Y |
|      |           ssl_type: |
|      |           ssl_cipher: |
|      |           x509_issuer: |
|      |           x509_subject: |
|      |           max_questions: 0 |
|      |           max_updates: 0 |
|      |           max_connections: 0 |
|      |           max_user_connections: 0 |
|      |           plugin: mysql_native_password |
| authentication_string: *23AE809DDACAF96AF0FD78ED04B6A265E05AA257 |
| password_expired: N |
| password_last_changed: 2017-05-11 16:21:36 |
+-----+-----+

```

默认的，在安装 MySQL 的时候，如果不选择创建匿名用户，那么意味着所有的用户只有一个：

root 超级用户

在 mysql 中，对用的用户管理中，是由对应的 Host 和 User 共同组成主键来区分用户。

Field	Type	Null	Key	Default	Extra
Host	char(60)	NO	PRI		
User	char(32)	NO	PRI		
Select_priv	enum('N','Y')			N	
Insert_priv	enum('N','Y')			N	
Update_priv	enum('N','Y')			N	
Delete_priv	enum('N','Y')			N	
Create_priv	enum('N','Y')			N	
Drop_priv	enum('N','Y')			N	
Reload_priv	enum('N','Y')			N	
Shutdown_priv	enum('N','Y')			N	
Process_priv	enum('N','Y')			N	
File_priv	enum('N','Y')			N	
Grant_priv	enum('N','Y')			N	
References_priv	enum('N','Y')			N	

User：代表用户的用户名

Host：代表本质是允许访问的客户端（IP 或者主机地址）。如果 host 使用%代表所有的用户（客户端）都可以访问

1.1.1 创建用户

理论上讲可以采用两种方式创建用户：

- 直接使用 root 用户在 mysql.user 表中插入记录（不推荐）
- 专门创建用户的 SQL 指令

➤ 基本语法：create user 用户名 identified by ‘明文密码’；

用户名：用户名@主机地址

主机地址：‘’ 或者 ‘%’、或者 ip 地址

```
如: create user 'stu1'@'%' identified by '123';
```

创建成功！

```
mysql> create user 'stu1'@'%' identified by '123';
Query OK, 0 rows affected (0.01 sec)
```

```
mysql>
```

查看 mysql.user 表中是否存在新增的用户



```
account_locked: 1
***** 3. row *****
    Host: %
        User: stu1
    Select_priv: N
    Insert_priv: N
    Update_priv: N
    Delete_priv: N
    Create_priv: N
        Drop_priv: N
    Reload_priv: N
    Shutdown_priv: N
    Process_priv: N
        File_priv: N
        Grant_priv: N
    References_priv: N
        Index_priv: N
        Alter_priv: N
    Show_db_priv: N
        Super_priv: N
Create_tmp_table_priv: N
    Lock_tables_priv: N
        Execute_priv: N
    Repl_slave_priv: N
    Repl_client_priv: N
    Create_view_priv: N
        Show_view_priv: N
Create_routine_priv: N
    Alter_routine_priv: N
    Create_user_priv: N
        Event_priv: N
        Trigger_priv: N
Create_tablespace_priv: N
    ssl_type:
    ssl_cipher:
    x509_issuer:
    x509_subject:
    max_questions: 0
        max_updates: 0
    max_connections: 0
    max_user_connections: 0
        plugin: mysql_native_password
authentication_string: *23AE809DDACAF96AF0FD78ED04B6A265E05AA257
    password_expired: N
password_last_changed: 2017-08-05 17:14:28
    password_lifetime: NULL
        account_locked: N
```

➤ 简化版创建用户（谁都可以访问，不需要密码）

```
mysql> create user user2;
Query OK, 0 rows affected (0.00 sec)
```

1、不限定客户端IP
2、没有密码的用户

```

account_locked: N
***** 4. row *****
      Host: %
      User: stu2
Select_priv: N
Insert_priv: N
Update_priv: N
Delete_priv: N
Create_priv: N
Drop_priv: N
Reload_priv: N
Shutdown_priv: N
Process_priv: N
File_priv: N
Grant_priv: N
References_priv: N
Index_priv: N
Alter_priv: N
Show_db_priv: N
Super_priv: N
Create_tmp_table_priv: N
Lock_tables_priv: N
Execute_priv: N
Repl_slave_priv: N
Repl_client_priv: N
Create_view_priv: N
Show_view_priv: N
Create_routine_priv: N
Alter_routine_priv: N
Create_user_priv: N
Event_priv: N
Trigger_priv: N
Create_tablespace_priv: N
      ssl_type:
      ssl_cipher:
      x509_issuer:
      x509_subject:
max_questions: 0
max_updates: 0
max_connections: 0
max_user_connections: 0
      plugin: mysql_native_password
authentication_string:                          密码为空
      password_expired: N
password_last_changed: 2017-08-05 17:18:55
      password_lifetime: NULL
  
```

- 当用户创建完成之后，用户是否可以使用？

```

C:\Users\liwei>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -ustu1 -p
Enter password: ***
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>                         
  
```

```
C:\Users\liwei>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -ustu2
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

1.1.2 删除用户

注意：mysql 中 user 是带着 host 本身的（具有唯一性）

基本语法：drop user 用户名;

```
mysql> drop user stu2;
Query OK, 0 rows affected (0.00 sec)

mysql> 
```

1.1.3 修改用户密码

MySQL 中提供了多种修改的方式：基本上都必须使用对应提供的一个系统函数：password()，需要靠该函数对密码进行加密处理。

- 使用专门的修改密码的指令

基本语法：set password for 用户 = password('新的明文密码');

```
-- 修改用户密码
mysql> set password for 'user1'@'%' = password('654321');
Query OK, 0 rows affected (0.00 sec)

mysql> 
```

修改后的数据测试

```
C:\Users\liwei>C:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe -ustu1 -p123456
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help,' or '\h' for help. Type '\c' to clear the current input statement.

mysql> - 使用新密码登录成功!
```

- 使用更新语句 update 来修改表

基本语法: update mysql.user set authentication_string= password('新的明文密码') where user = " and host= ";

1.2 权限管理

在 mysql 中将权限管理分为三类:

- 1、数据权限: 增删改查 (select\update\delete\insert)
- 2、结构权限: 结构操作 (create\drop)
- 3、管理权限: 权限管理 (create user\grant\revoke): 通常只给管理员如此权限



1.2.1 授予权限：grant

将权限分配给指定的用户

基本语法：grant 权限列表 on 数据库[*].表名[*] to 用户；

权限列表：使用逗号分隔，但是可以使用 all privileges 代表全部权限

数据库.表名：可以是单表（数据库名字.表名），可以是具体某个数据库（数据库.*），也可以整库（*.*）

给 stu1 用户授权 czxy 数据库的 students 表

```
grant select,update,insert,delete on czxy.students to 'stu1'@'%';
```

```
mysql> grant select,update,insert,delete on czxy.students to 'stu1'@'%';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
```

给 stu1 用户授权 mydb 数据库的所有表

```
grant all privileges on mydb.* to 'stu1'@'%';
```

```
mysql> grant all privileges on mydb.* to 'stu1'@'%';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
```

用户被分配权限以后不需要退出就可以看到效果

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| czxy          |
| mydb          |
+-----+
3 rows in set (0.00 sec)
```

```

| czxy
+-----+
2 rows in set (0.00 sec)  stu1

mysql> use czxy;
Database changed
mysql> show tables;
+-----+
| Tables_in_czxy |
+-----+
| students |
+-----+
1 row in set (0.00 sec)

mysql>

```

y因为我们只给stu1 授权了czxy中的students的权限
所以, czxy即便有很多表, 但是stu1仍然只能看到students这个表

```

mysql> use czxy;
Database changed
mysql> show tables;
+-----+
| Tables_in_czxy |
+-----+
| children
| course
| mother
| person
| person_detail
| student_course
| students |
+-----+
7 rows in set (0.00 sec)

mysql>

```

具体权限查看：单表权限只能看到数据库中的一张表

```

mysql> use czxy;
Database changed
mysql> show tables;
+-----+
| Tables_in_czxy |
+-----+
| students |
+-----+
1 row in set (0.00 sec)

mysql> select * from students;
+----+-----+-----+-----+
| id | name   | gender | tel   |
+----+-----+-----+-----+
| 1  | 八戒   | 男     | 13912345555 |
| 2  | 大师兄 | 男     | 13912346666 |
| 3  | 金角大王 | 男     | 13922222222 |
+----+-----+-----+-----+
3 rows in set (0.01 sec)

```

```

mysql> use mydb;
Database changed
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| student
| test_int |
+-----+
2 rows in set (0.00 sec)

```

1.2.2 取消权限：revoke

权限收回：将权限从用户手中收回

基本语法：revoke 权限列表/all privileges on 数据库/*表/* from 用户；

```
revoke all privileges on czxy.students from 'stu1'@'%';
```

```
mysql> revoke all privileges on czxy.students from 'stu1'@'%';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> -
```

权限收回，同样不需要刷新，用户马上就会感受到

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mydb |
+-----+
2 rows in set (0.00 sec)

mysql>
```

1.2.3 刷新权限：flush

flush：刷新，将当前对用户的权限操作，进行一个刷新，将操作的具体内容同步到对应的表中。

基本语法：flush privileges;

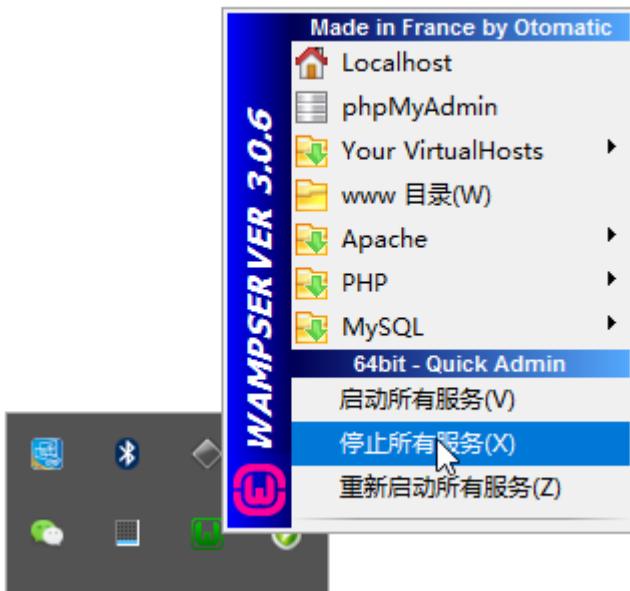
```
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> -
```

1.3 密码丢失的解决方案

如果忘记了 root 用户密码，就需要去找回或者重置 root 用户密码

1、停止服务



2、重新启动服务：mysql.exe --skip-grant-tables //启动服务器但是跳过权限

```
C:\Users\liwei>C:\wamp64\bin\mysql\mysql15.7.14\bin\mysql.exe --skip-grant-tables
```

3、当前启动的服务器没有权限概念：非常危险，任何客户端，不需要任何用户信息都可以直接登录，而且是 root 权限：新开客户端，使用 mysql.exe 登录即可

```
C:\Users\liwei>C:\wamp64\bin\mysql\mysql15.7.14\bin\mysql.exe
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| baijia |
| czxy |
| czxy_2017_db |
| czxy_new |
| mydatabase1 |
| mydb |
| mydb3 |
| mysql |
| performance_schema |
| sys |
+-----+
11 rows in set (0.00 sec)
```

此时已经拥有最高的权限

4、修改 root 用户的密码：指定 用户名@host

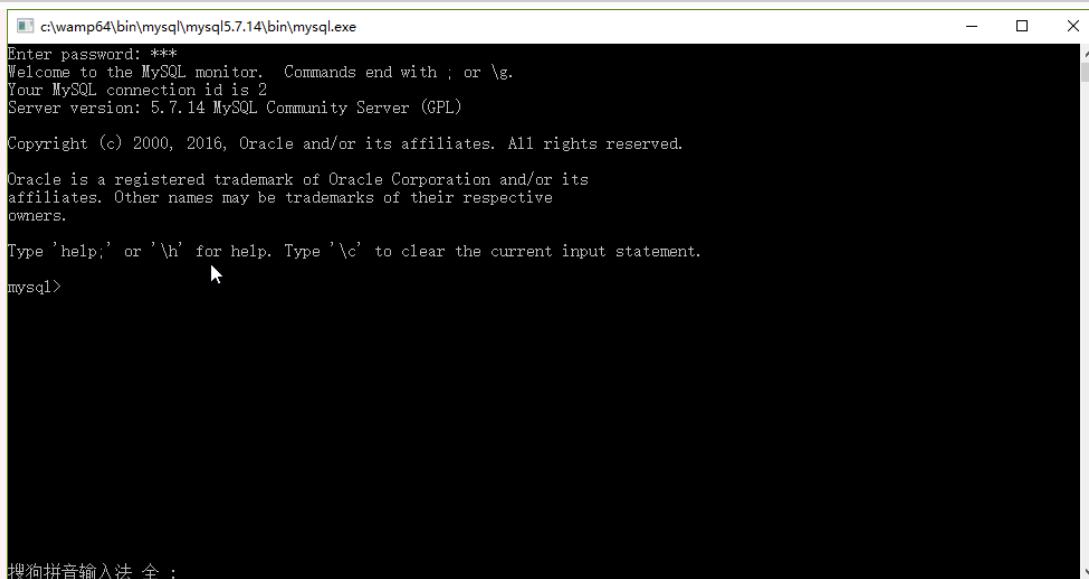
```
mysql> set password for 'root'@'localhost' = password('123');
ERROR 1290 (HY000): The MySQL server is running with the --skip-grant-tables option so it cannot execute this statement
mysql> update mysql.user set authentication_string = password(123) where user='root' and host='localhost';
Query OK, 0 rows affected, 1 warning (0.01 sec)
Rows matched: 1  Changed: 0  Warnings: 1
```

在这种情况下，不能使用 set password for ...
使用update语句可以进行密码更新！

5、赶紧关闭服务器，重启服务



登录成功！



```
c:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe
Enter password: ***
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.14 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

2. 变量

MySQL 本质是一种编程语言，需要很多变量来保存数据。MySQL 中很多的属性控制都是通过 mysql 中固有的变量来实现的。

2.1 系统变量

系统内部定义的变量，系统变量针对所有用户（MySQL 客户端）有效。

- 显示系统所有变量：`show variables [like 'pattern'];`



```
+-----+-----+
| warning_count | 0
+-----+-----+
504 rows in set, 1 warning (0.00 sec)

mysql>
```

`show variables like 'auto%'; --` 这是模糊查找

```

mysql> 
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| autocommit | OFF |
| automatic_sp_privileges | ON |
+-----+-----+
4 rows in set, 1 warning (0.00 sec)

--> 事务自动提交
    
```

- select 查询变量的数据值（系统变量）

基本语法: select @@变量名;

```

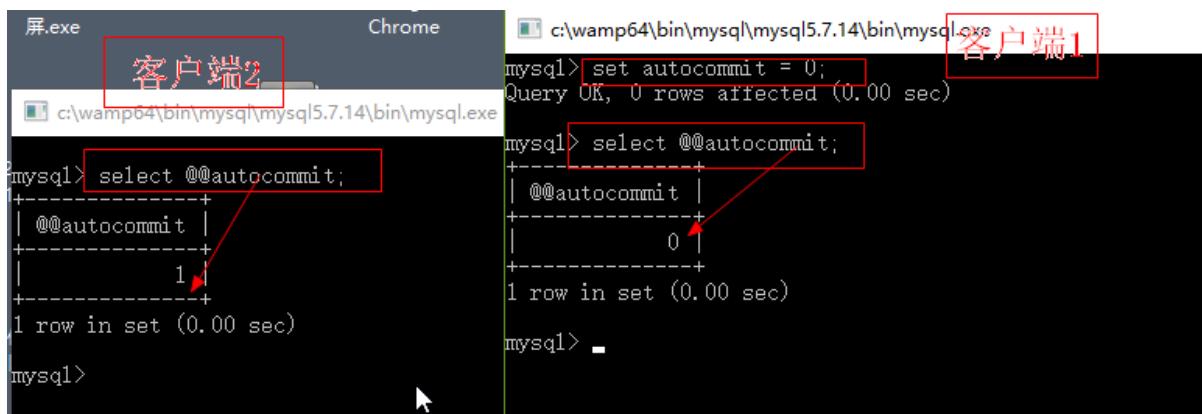
mysql> -- 查看系统变量
mysql> select @@autocommit;
+-----+
| @@autocommit |
+-----+
|          1 |
+-----+
1 row in set (0.00 sec)
    
```

修改系统变量：分为两种修改方式

2.1.1 局部修改（会话级别）

只针对当前自己客户端当次连接有效

基本语法: set @@变量名 = 新值;



```

客户端2
c:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe
mysql> set autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

客户端1
c:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe
mysql> select @@autocommit;
+-----+
| @@autocommit |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)

    
```

2.1.2 全局修改

针对所有的客户端，“所有时刻”都有效

基本语法: set global 变量名 = 值; 或者 set @@global.变量名 = 值;



```

mysql> -- 全局修改系统变量
mysql> set global autocommit = 0;
Query OK, 0 rows affected <0.00 sec>

mysql> set @@global.auto_increment_increment = 2;
Query OK, 0 rows affected <0.00 sec>

mysql> show variables like 'auto_increment%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| auto_increment_increment | 1    |
| auto_increment_offset | 1    |
+-----+-----+
2 rows in set <0.00 sec>

mysql>半:

```

mysql>

```

mysql> select @@autocommit;
+-----+
| @@autocommit |
+-----+
| 1           |
+-----+
1 row in set <0.00 sec>

mysql> select @@autocommit,@@auto_increment_increment;
+-----+-----+
| @@autocommit | @@auto_increment_increment |
+-----+-----+
| 1           | 1                   |
+-----+-----+
1 row in set <0.00 sec>

mysql>

```

全局修改之后：所有连接的客户端并没发现改变？全局修改只针对新客户端生效（正在连着的无效）

```

mysql> -- 全局修改系统变量
mysql> set global autocommit = 0;
Query OK, 0 rows affected <0.00 sec>

mysql> set @@global.auto_increment_increment = 2;
Query OK, 0 rows affected <0.00 sec>

mysql> show variables like 'auto_increment%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| auto_increment_increment | 1    |
| auto_increment_offset | 1    |
+-----+-----+
2 rows in set <0.00 sec>

mysql>半:

```

mysql>

```

mysql> select @@autocommit;
+-----+
| @@autocommit |
+-----+
| 1           |
+-----+
1 row in set <0.00 sec>

mysql> select @@autocommit,@@auto_increment_increment;
+-----+-----+
| @@autocommit | @@auto_increment_increment |
+-----+-----+
| 1           | 1                   |
+-----+-----+
1 row in set <0.00 sec>

mysql>半:

```

mysql>

```

C:\Users\window>mysql -uroot -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ;
Your MySQL connection id is 9
Server version: 5.5.28 MySQL Community Server <
Copyright (c) 2000, 2012, Oracle and/or its affiliates.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their
owners.
Type 'help;' or '\h' for help. Type '\s' to clear
mysql> select @@autocommit,@@auto_increment_increment;
+-----+-----+
| @@autocommit | @@auto_increment_increment |
+-----+-----+
| 0           | 2                   |
+-----+-----+
1 row in set <0.00 sec>

mysql>

```

注意：如果想要本次连接对应的变量修改有效，那么不能使用全局修改，只能使用会话级别修改（`set 变量名 = 值`）；

```

mysql> set global autocommit = 1;
Query OK, 0 rows affected <0.00 sec>

mysql> select @@autocommit;
+-----+
| @@autocommit |
+-----+
| 0           |
+-----+
1 row in set <0.00 sec>

```

2.2 会话变量

会话变量也称之为用户变量，会话变量跟 mysql 客户端是绑定的，设置的变量，只对当前用户使用的客户端生效。

2.2.1 定义用户变量：`set @变量名 = 值;`

```

set @name="helloworld!";

```

```
mysql> set @name="helloworld!";
Query OK, 0 rows affected (0.00 sec)

mysql> select @name;
+-----+
| @name      |
+-----+
| helloworld! |
+-----+
1 row in set (0.00 sec)
```

在 mysql 中因为没有比较符号 ==，所以是用 = 代替比较符号：有时候在赋值的时候，会报错：
mysql 为了避免系统分不清是赋值还是比较：特定增加一个变量的赋值符号： :=

2.2.2 Set @变量名 := 值;

```
-- 使用专用赋值符号
mysql> set @age := 50;
Query OK, 0 rows affected (0.00 sec)

mysql> select @age;
+-----+
| @age      |
+-----+
|    50     |
+-----+
1 row in set (0.00 sec)
```

MySQL 是专门存储数据的：允许将数据从表中取出存储到变量中：查询得到的数据必须只能是一行数据（一个变量对应一个字段值）：MySQL 没有数组。

2.2.3 使用 select 赋值且查看赋值过程

语法：select @变量 1 := 字段 1, @变量 2 := 字段 2 from 数据表 where 条件；

```
select @name:=name,@telnum:=telnum from person limit 1;
```

```
a mysql> select @name:=name,@telnum:=telnum from person limit 1;
+-----+-----+
| @name:=name | @telnum:=telnum |
+-----+-----+
| 张三         | 13800138000 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select @name,@telnum;
+-----+-----+
| @name | @telnum |
+-----+-----+
| 张三 | 13800138000 |
+-----+-----+
1 row in set (0.00 sec)

mysql>
```

错误语法：使用“=”当做赋值了，系统会当做比较符号（MySQL 中“=”是关系运算符，`where xxx='xx'`）来处理

```
mysql> select @name=name,@telnum=telnum from person limit 1;
+-----+-----+
| @name=name | @telnum=telnum |
+-----+-----+
|      NULL |          NULL |
+-----+-----+
1 row in set (0.00 sec)
```

当做关系运算符“==”处理了

2.2.4 使用 select 只赋值，不看过程

语法格式：

`select 字段 1, 字段 2... from 数据源 where 条件 into @变量 1, @变量 2...`

```
select name,telnum from person limit 1 into @myName,@myTelnum;
```

```

for the right syntax to use near ' to @myName, @myTelnum' at line 1
mysql> select name, telnum from person limit 1;
+-----+-----+
| name | telnum |
+-----+-----+
| 张三 | 13800138000 |
+-----+
1 row in set (0.00 sec)

mysql> select name, telnum from person limit 1 into @myName, @myTelnum
Query OK, 1 row affected (0.00 sec)

mysql> select @myName, @myTelnum;
+-----+-----+
| @myName | @myTelnum |
+-----+-----+
| 张三 | 13800138000 |
+-----+
1 row in set (0.00 sec)

```

客户端2

```

c:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe 客户端2
1 row in set (0.00 sec)

mysql> select @myName, @myTelnum;
+-----+-----+
| @myName | @myTelnum |
+-----+-----+
| NULL | NULL |
+-----+
1 row in set (0.00 sec)

mysql> 客户端2中无法使用客户端1的会话变量

```

客户端1

```

c:\wamp64\bin\mysql\mysql5.7.14\bin\mysql.exe 客户端1
for the right syntax to use near ' to @myName, @myTelnum' at line 1
mysql> select name, telnum from person limit 1;
+-----+-----+
| name | telnum |
+-----+-----+
| 张三 | 13800138000 |
+-----+
1 row in set (0.00 sec)

mysql> select name, telnum from person limit 1 into @myName, @myTelnum
Query OK, 1 row affected (0.00 sec)

mysql> select @myName, @myTelnum;
+-----+-----+
| @myName | @myTelnum |
+-----+-----+
| 张三 | 13800138000 |
+-----+
1 row in set (0.00 sec)

mysql>

```

客户端1定义的会话变量，只能在客户端1中使用

2.3 局部变量

作用范围在 `begin` 到 `end` 语句块之间。在该语句块里设置的变量，`declare` 语句专门用于定义局部变量。

- 1、 局部变量是使用 `declare` 关键字声明
- 2、 局部变量 `declare` 语句出现的位置一定是在 `begin` 和 `end` 之间（`beginend` 是在大型语句块中使用：函数/存储过程/触发器）
- 3、 声明语法： `declare 变量名 数据类型 default 值;`

3. 流程结构

流程结构：代码的执行顺序

3.1 if 分支

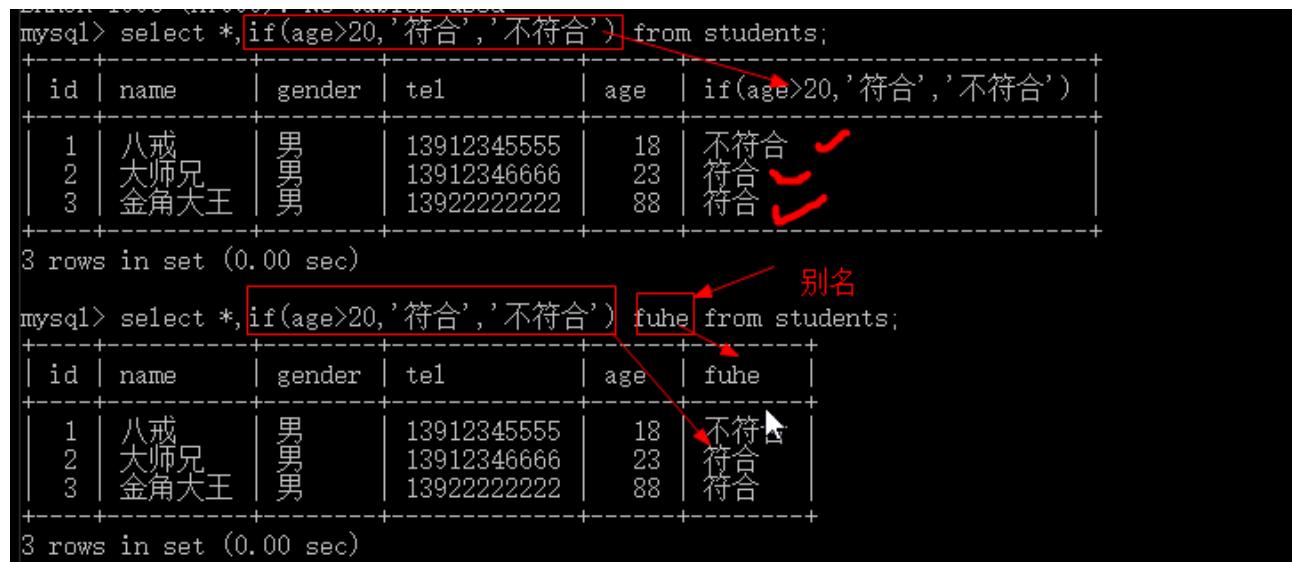
3.1.1 基本语法

If 在 MySQL 中有两种基本用法

- 用在 select 查询当中，当做一种条件来进行判断

基本语法：if(条件,为真结果,为假结果)

```
-- 未起别名
select *,if(age>20,'符合','不符合') from students;
-- 起别名 fuhe
select *,if(age>20,'符合','不符合') fuhe from students;
```



mysql> select *,if(age>20,'符合','不符合') from students;				
id	name	gender	tel	age if(age>20,'符合','不符合')
1	八戒	男	13912345555	18 不符合 ✓
2	大师兄	男	13912346666	23 符合 ✓
3	金角大王	男	13922222222	88 符合 ✓

mysql> select *,if(age>20,'符合','不符合') fuhe from students;				
id	name	gender	tel	age fuhe
1	八戒	男	13912345555	18 不符合
2	大师兄	男	13912346666	23 符合
3	金角大王	男	13922222222	88 符合

- 用在复杂的语句块中（函数/存储过程/触发器）

基本语法

mysql中的if

```
if 条件表达式 then
    满足条件要执行的语句;
end if;
```

php、javascript

```
if( 条件 ) {
    满足条件要执行的语句;
}
```

3.1.2 复合语法

复合语法：代码的判断存在两面性，两面都有对应的代码执行。

基本语法：

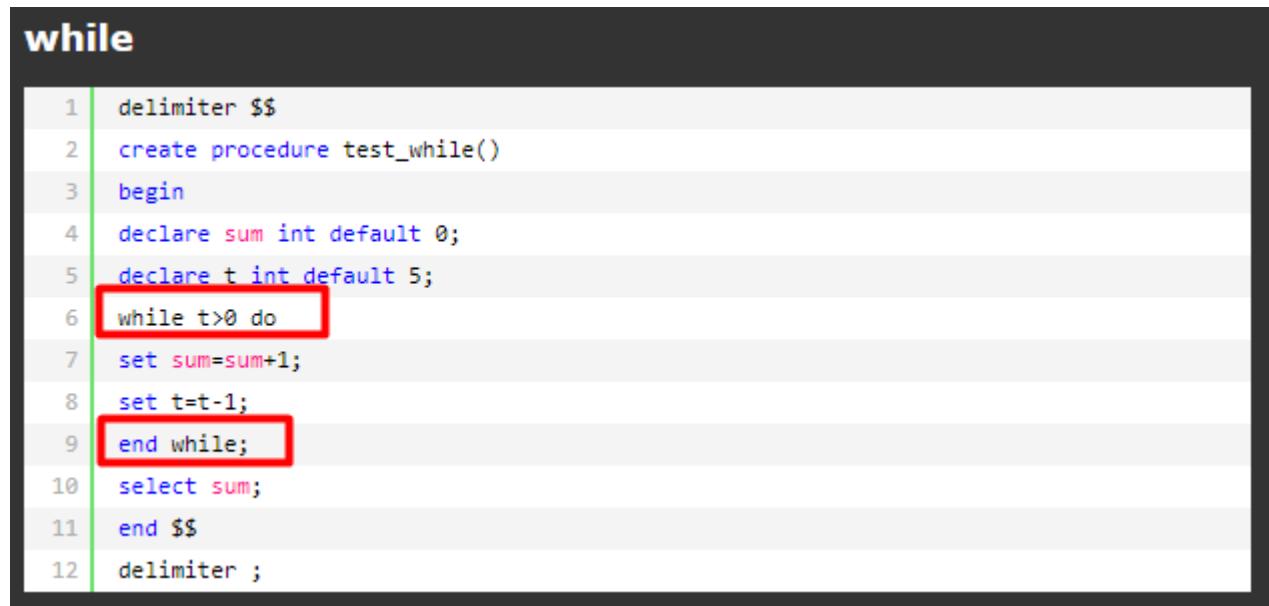
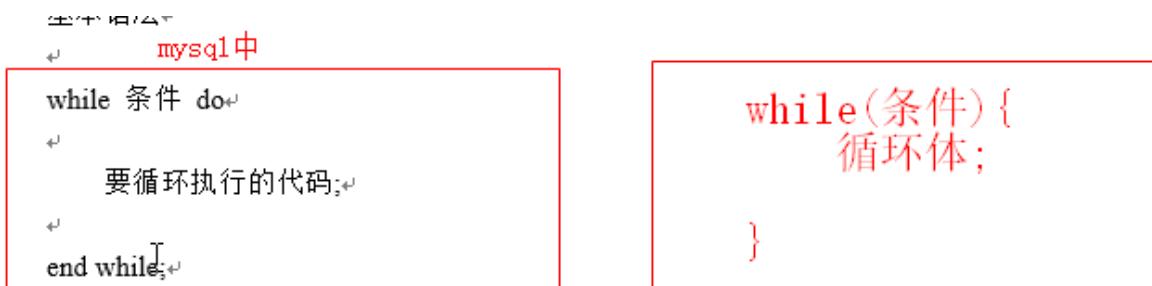
```
if 条件表达式 then
    满足条件要执行的语句;
else
    不满足条件要执行的语句;
//如果还有其他分支（细分），可以在里面再使用 if
    if 条件表达式 then
        //满足要执行的语句
    end if;
end if;
```

```
BEGIN
    declare result tinyint;
    set result=0;
    if period=0 then
        if date=periodData then
            set result=1;
        end if;
    elseif period=1 then
        set result=1;
    elseif period=2 then
        if WEEKDAY(date)+1=periodData then
            set result=1;
        end if;
    elseif period=3 then
        if DAYOFMONTH(date)=periodData then
            set result=1;
        end if;
    else
        if DAYOFMONTH(date)=periodData then
            set result=1;
        end if;
    end if;
    RETURN result;
END
```

4. While 循环

4.1 基本语法

循环体都是需要在大型代码块中使用
基本语法



```
1 delimiter $$ 
2 create procedure test_while()
3 begin
4 declare sum int default 0;
5 declare t int default 5;
6 while t>0 do
7     set sum=sum+1;
8     set t=t-1;
9 end while;
10 select sum;
11 end $$ 
12 delimiter ;
```

4.2 结构标识符(别名)

结构标识符：为某些特定的结构进行命名，然后为的是在某些地方使用名字

基本语法

标识名字:while 条件 do
 循环体
end while [标识名字];

标识符的存在主要是为了循环体中使用循环控制。在 mysql 中没有 continue 和 break，有自己的关键字替代：

iterate: 迭代，就是以下的代码不执行，重新开始循环（**continue**）

leave: 离开，整个循环终止（**break**）

标识名字:while 条件 do

 if 条件判断 then

 循环控制;

 Iterate/leave 标识名字;

 end if;

 循环体

end while [标识名字];

```
-- 别名 (结构化表示符)
-- w1.while i<=100 do
-- 循环体
-- end while w1          w1 是 while 循环的一个
                           别名
                           别名用途：比如在while 循环中，要跳出while循环。
                           就要使用别名。
```

5. 函数

`password('明文的密码')` → 生成加密的密码

在 mysql 中，函数分为两类：系统函数（内置函数）和自定义函数

不管是内置函数还是用户自定义函数，都是使用 `select` 函数名(参数列表);

5.1 内置函数

5.1.1 字符串函数

- `char_length()`: 判断字符串的字符数
- `length()`: 判断字符串的字节数（与字符集）



```
mysql>
mysql> select char_length('你好中国'),length('你好中国');
+-----+-----+
| char_length('你好中国') | length('你好中国') |
+-----+-----+
|           4           |          8          |
+-----+-----+
1 row in set (0.00 sec)
```

- concat(): 连接字符串
- instr(): 判断字符在目标字符串中是否存在，存在返回其位置，不存在返回 0

```
mysql> select concat('你好','中国'),instr('你好中国','中'),instr('你好中国','万');
+-----+-----+-----+
| concat('你好','中国') | instr('你好中国','中') | instr('你好中国','万') |
+-----+-----+-----+
|    你好中国           |          3          |          0          |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

- lcase(): 全部小写
- left(): 从左侧开始截取，直到指定长度的位置（位置如果超过长度，截取所有）
- right(): 从右侧开始截取，直到指定长度的位置（位置如果超过长度，截取所有）

```
mysql> select lcase('aBcD'),left('你好中国',2);
+-----+-----+
| lcase('aBcD') | left('你好中国',2) |
+-----+-----+
|     abcd      |      你好      |
+-----+-----+
1 row in set (0.00 sec)
```

使用 left 截取时间

```
mysql> select left('2017-08-08 10:10:10',10);
+-----+
| left('2017-08-08 10:10:10',10) |
+-----+
| 2017-08-08                         |
+-----+
1 row in set (0.00 sec)

mysql>
```

使用 right() 从右侧截取

```
mysql> select right('2017-08-08 10:10:10',8);
+-----+
| right('2017-08-08 10:10:10',8) |
+-----+
| 10:10:10                         |
+-----+
1 row in set (0.00 sec)
```

- `substring()` 函数

```
substring (str, pos)
```

```
substring (str, pos, length)
```

说明: `substring` (被截取字段, 从第几位开始截取)

`substring` (被截取字段, 从第几位开始截取, 截取长度)

例: `select substring (content,5) as abstract from my_content_t`

`select substring (content,5,200) as abstract from my_content_t`

(注: 如果位数是负数 如-5 则是从后倒数位数, 到字符串结束或截取的长度)

```
mysql> select substring("传智专修学院",3);
+-----+
| substring("传智专修学院", 3) |
+-----+
| 专修学院 |
+-----+
1 row in set (0.00 sec)

mysql> select substring("传智专修学院", 3, 2);
+-----+
| substring("传智专修学院", 3, 2) |
+-----+
| 专修 |
+-----+
1 row in set (0.00 sec)

mysql>
```

- `ltrim()`: 消除左边对应的空格

```
mysql> select ltrim(" 哈哈 沐阳 我来了! ");
+-----+
| ltrim(" 哈哈 沐阳 我来了! ") |
+-----+
| 哈哈 沐阳 我来了! |
+-----+
1 row in set (0.00 sec)
```

5.1.2 时间函数

`now()`: 返回当前时间, 日期 时间

`curdate()`: 返回当前日期

`curtime()`: 返回当前时间

```
mysql> select now(),curdate(),curtime();
+-----+-----+-----+
| now() | curdate() | curtime() |
+-----+-----+-----+
| 2017-03-04 00:53:06 | 2017-03-04 | 00:53:06 |
+-----+-----+-----+
```



datediff(): 判断两个日期之间的天数差距，参数日期必须使用字符串格式（用引号）

```
mysql> select datediff('2017-08-10', '2017-08-08');  
+-----+  
| datediff('2017-08-10', '2017-08-08') |  
+-----+  
| 2 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> -
```

此函数经常用于判断文章是否最新发布，并配合下面图标进行显示



Unix_timestamp(): 获取时间戳

```
mysql> select unix_timestamp();  
+-----+  
| unix_timestamp() |  
+-----+  
| 1488560184 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> select unix_timestamp();  
+-----+  
| unix_timestamp() |  
+-----+  
| 1488560189 |  
+-----+  
1 row in set (0.00 sec)
```

From_unixtime(): 将指定时间戳转换成对应的日期时间格式

```
mysql> select from_unixtime(1234567890);  
+-----+  
| from_unixtime(1234567890) |  
+-----+  
| 2009-02-14 07:31:30 |  
+-----+  
1 row in set (0.00 sec)
```

5.1.3 数学函数

abs(): 绝对值

ceil(): 向上取整

floor(): 向下取整

pow(): 求指数，谁的多少次方

rand(): 获取一个随机数（0-1 之间）

round(): 四舍五入函数

```
mysql> select abs(-1),ceiling(1.1),floor(1.1),pow(2,4),rand(),round(1.5);
+-----+-----+-----+-----+-----+
| abs(-1) | ceiling(1.1) | floor(1.1) | pow(2,4) | rand() | round(1.5) |
+-----+-----+-----+-----+-----+
|      1 |           2 |          1 |       16 | 0.8089020352893528 | 1
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

5.1.4 其他函数

md5(): 对数据进行 md5 加密（mysql 中的 md5 与其他任何地方的 md5 加密出来的内容是完全相同的）

version(): 获取版本号

database(): 显示当前所在数据库

uuid(): 生成一个唯一标识符（自增长）：自增长是单表唯一，UUID 是整库（数据唯一同时空间唯一）

```
mysql> select md5('a'),version(),database(),uuid();
+-----+-----+-----+-----+
| md5('a') | version() | database() | uuid() |
+-----+-----+-----+-----+
| 0cc175b9c0f1b6a831c399e269772661 | 5.5.28 | mydb | 412f8d9b-0033-11e7-beb5-080027bf2eaa |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

6. 自定义函数

自定义函数：用户自己定义的函数

函数：实现某种功能的语句块（由多条语句组成）

1、 函数内部的每条指令都是一个独立的个体：需要符合语句定义规范：需要语句结束符分号；

2、 函数是一个整体，而且函数是在调用的时候才会被执行，那么当设计函数的时候，意味着整体不能被中断；

3、 MySQL一旦见到语句结束符分号，就会自动开始执行

解决方案：在定义函数之前，尝试修改临时的语句结束符

基本语法： delimiter

修改临时语句结束符： delimiter 新符号[可以使用系统非内置即可\$\$]

中间为正常 SQL 指令： 使用分号结束（系统不会执行： 不认识分号）

使用新符号结束

修改回语句结束符： delimiter ;

6.1 创建函数

自定义函数包含几个要素： function 关键字， 函数名， 参数（形参和实参[可选]）， 确认函数返回值类型， 函数体， 返回值语句

6.1.1 函数定义基本语法

```
create function 函数名(形参) returns 返回值类型
begin
    //函数体
    return 返回值数据;    //数据必须与结构中定义的返回值类型一致
end
```

6.1.2 创建无参函数

```
-- 创建一个无参函数，返回值是int
create function myfun1() returns int
begin
    return 10;
end
```

```
-- 创建一个无参函数，返回值是 int
create function myfun1() returns int
begin
    return 10;
end
```

但执行 sql 指令时报错

```
1 row in set (0.00 sec)

mysql> create function myfun1() returns int
      -> begin
      -> return 10;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
for the right syntax to use near '' at line 3
mysql> end
```

错误原因为：

mysql 以“;”作为语句结束符，但是当执行到 `return 10;` 时认为指令结束，实际上函数并未结束。

解决办法：临时设置sql指令的结束符为其他符号，只要不是“;”和系统特殊含义的符号(`_`,`%`)就行

通常设置为：`$$`

具体操作：使用 `delimiter` 符号；可以修改 sql 指令结束符

6.1.3 修改语句结束符

```
-- at line 1
mysql> delimiter $$          // 表示语句结束
mysql> create function myfun1() returns int
      -> begin
      -> return 10;
      -> end
      -> -- $$ 表示语句结束
      -> $$

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;           // 表示语句结束
mysql>
```

6.1.4 函数体只有一条指令

并不是所有的函数都需要 `begin` 和 `end`：如果函数体本身只有一条指令（`return`），那么可以省略 `begin` 和 `end`

```
-- at line 1
mysql> create function myfun2() returns int
      -> return 100;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

因为“;”恰巧是函数语句结束，所以此类型函数不需要修改 sql 语句结束符。

6.1.5 创建有参函数

形参：在 mysql 中需要为函数的形参指定数据类型（形参本身可以有多个）

基本语法：变量名 字段类型

```
mysql> create function myfun3(a int,b int) returns int
      -> return a+b;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

6.2 查看函数

- 可以通过查看 function 状态，查看所有的函数

Show function status [like 'pattern'];

```
mysql> show function status like 'my%'\G
*****1. row *****
Db: czxy
Name: myfun1
Type: FUNCTION
Definer: root@localhost
Modified: 2017-08-05 23:23:48
Created: 2017-08-05 23:23:48
Security_type: DEFINER
Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
Database Collation: utf8_general_ci
*****2. row *****
Db: czxy
Name: myfun2
Type: FUNCTION
Definer: root@localhost
Modified: 2017-08-05 23:26:17
Created: 2017-08-05 23:26:17
Security_type: DEFINER
Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
Database Collation: utf8_general_ci
*****3. row *****
Db: czxy
```

- show 查看函数的创建语句：show create function 函数名字；

```
mysql> show create function myfun1 \G;
***** 1. row *****
Function: myfun1
sql_mode: STRICT_ALL_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER
Create Function: CREATE DEFINER='root'@'localhost' FUNCTION `myfun1`() RETURNS int(11)
begin
return 10;
end
character_set_client: gbk
collation_connection: gbk_chinese_ci
Database Collation: utf8_general_ci
1 row in set (0.00 sec)

ERROR:
No query specified
```

6.3 调用函数

自定义函数的调用与内置函数的调用是一样的：select 函数名(实参列表);
调用格式：

```
mysql> select myfun1(),myfun2(),myfun3(10,34);
+-----+-----+-----+
| myfun1() | myfun2() | myfun3(10,34) |
+-----+-----+-----+
|      10 |      100 |          44 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> -
```

6.4 删除函数

删除函数：drop function 函数名；

```

mysql> drop function myfun1;
Query OK, 0 rows affected (0.00 sec)

mysql> show function status like 'my%'\G
***** 1. row *****
      Db: czxy
      Name: myfun2
      Type: FUNCTION
      Definer: root@localhost
      Modified: 2017-08-05 23:26:17
      Created: 2017-08-05 23:26:17
      Security_type: DEFINER
      Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
Database Collation: utf8_general_ci
***** 2. row *****
      Db: czxy
      Name: myfun3
      Type: FUNCTION
      Definer: root@localhost
      Modified: 2017-08-05 23:28:43
      Created: 2017-08-05 23:28:43
      Security_type: DEFINER
      Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
Database Collation: utf8_general_ci
    
```

删除函数 myfun1()

6.5 注意事项

- 1、自定义函数是属于用户级别的：只有当前客户端对应的数据库中可以使用
- 2、可以在不同的数据库下看到对应的函数，但是不可以调用

```

mysql> use mydb;
Database changed
mysql> select myfun2();
ERROR 1305 (42000): FUNCTION mydb.myfun2 does not exist
mysql>
    
```

myfun2()函数在czxy库中有效

- 3、自定义函数：通常是为了将多行代码集合到一起解决一个重复性的问题
- 4、函数因为必须规范返回值：那么在函数内部不能使用 select
指令：select 一旦执行就会得到一个结果（result set）

例外的情况：**select 字段 into @变量;**

7. 函数流程结构案例

需求：从 1 开始，直到用户传入的对应的值为止，自动求和：凡是 5 的倍数都不要。

$$1+2+3+4+\textcolor{red}{5}+6+7+8 = 31$$

设计：

- 1、 创建函数
- 2、 需要一个形参：确定要累加到什么位置
- 3、 需要定义一个变量来保存对应的结果：set @变量名；

使用局部变量来操作：此结果是在函数内部使用

Declare 变量名 类型 [= 默认值]；

- 4、 内部需要一个循环来实现迭代累加
- 5、 循环内部需要进行条件判断控制：5 的倍数
- 6、 函数必须有返回值

函数指令代码：

```
-- 修改 sql 结束符
delimiter $$

-- 1、 创建函数
create function my_sum(end_value int) returns int
begin
    -- 2、 需要一个形参：确定要累加到什么位置
    -- 3、 需要定义一个变量来保存对应的结果：set @变量名；
    declare res int default 0;
    declare i int default 1;
    -- 使用局部变量来操作：此结果是在函数内部使用
    -- Declare 变量名 类型 [= 默认值]；
    -- 4、 内部需要一个循环来实现迭代累加
    mywhile:while i <= end_value do
        -- 5、 循环内部需要进行条件判断控制：5 的倍数
        if i % 5 = 0 then
            set i = i + 1;
            iterate mywhile;
        end if;
        -- 累加操作
        set res = res + i;
        set i = i + 1;
    end while mywhile;
    -- 6、 函数必须有返回值
    return res;
end
$$
delimiter ;
```

运行结果：

```
mysql> delimiter $$  
mysql> 1、创建函数  
mysql> create function my_sum(end_value int) returns int  
-> begin  
-> -- 2、需要一个形参：确定要累加到什么位置  
-> -- 3、需要定义一个变量来保存对应的结果：set @变量名；  
-> declare res int default 0;  
-> declare i int default 1;  
-> -- 使用局部变量来操作：此结果是在函数内部使用  
-> -- Declare 变量名 类型 [= 默认值];  
-> -- 4、内部需要一个循环来实现迭代累加  
-> mywhile:while i <= end_value do  
-> -- 5、循环内部需要进行条件判断控制：5的倍数  
-> if i % 5 = 0 then  
-> set i = i + 1;  
-> iterate mywhile;  
-> end if;  
-> -- 累加操作  
-> set res = res + i;  
-> set i = i + 1;  
-> end while mywhile;  
-> -- 6、函数必须有返回值  
-> return res;  
-> end  
-> $$  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> delimiter ;  
mysql>
```

函数的创建过程

调用函数：select 函数名(实参);

```
mysql> delimiter ;  
mysql> select my_sum(8);  
+-----+  
| my_sum(8) |  
+-----+  
| 31 |  
+-----+  
1 row in set (0.00 sec)
```

8. 变量作用域

变量作用域：变量能够使用的区域范围

8.1 局部作用域

使用 `declare` 关键字声明（在结构体内：函数/存储过程/触发器），而且只能在结构体内部使用

1、 `declare` 关键字声明的变量没有任何符号修饰，就是普通字符串，如果在外部访问该变量，系统会自动认为是字段

8.2 会话作用域

用户定义的，使用@符号定义的变量，使用 set 关键字

会话作用域：在当前用户当次连接有效，只要在本连接之中，任何地方都可以使用（可以在结构内部，也可以跨库）

会话变量可以在函数内部使用

```
mysql> set @name = "张三";
Query OK, 0 rows affected (0.00 sec)

mysql> create function myfun4() returns char(4)
-> return @name;
Query OK, 0 rows affected (0.00 sec)

mysql> select myfun4();
+-----+
| myfun4() |
+-----+
| 张三     |
+-----+
1 row in set (0.00 sec)

mysql>
```

此处定义的是一个会话级变量

定义一个函数，返回值是@name 会话变量值

说明会话级变量可以在函数内部使用

会话变量可以跨库

```
mysql> use mydb3;
Database changed
mysql> select @name;
+-----+
| @name |
+-----+
| 张三   |
+-----+
1 row in set (0.00 sec)
```

切换库

依然可以使用会话级变量

8.3 全局作用域

所有的客户端所有的连接都有效：需要使用全局符号来定义

Set global 变量名 = 值;

Set @@global.变量名 = 值;

通常，在 SQL 编程的时候，不会使用自定义变量来控制全局。一般都是定义会话变量或者在结构中使用局部变量来解决问题。

9. 今日总结



传智专修学院
CHUANZHI SPECIALIZED COLLEGE

力学笃行 志存高远

MySQL (八)

1. 准备数据

```
create database p_t_v default character set utf8;
use p_t_v;

create table class
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '班级',
    pcount smallint unsigned not null default '0' comment '班级人数',
    primary key(id)
) comment='班级表';

insert into class
values
(1,'全栈开发班',3),
(2,'Java 开发班',2),
(3,'服装设计班',1),
(4,'美容美发班',1),
(5,'挖掘机精英班',0),
(6,'美国总统速成班',0);

# 学生表
create table students
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '姓名',
    gender enum('男','女') not null comment '性别',
    tel bigint unsigned not null comment '手机号',
    class_id int unsigned not null comment '班级 Id',
    age tinyint unsigned not null comment '年龄',
    height tinyint unsigned not null comment '身高, 单位: 厘米',
    primary key(id)
) comment='学生表';

insert into students
values
(1,'八戒','男',13912345555,1,20,180),
(2,'大师兄','男',13912346666,1,21,175),
```

```
(3,'金角大王','男',13922222222,2,204,195),  
(4,'西施','女',13912342343,4,17,165),  
(5,'大乔','女',13912346666,3,22,168),  
(6,'貂蝉','女',13922233222,1,21,164),  
(7,'大乔','男',13922225454,2,19,183);
```

2. 存储过程

2.1 什么是存储过程

存储过程（Stored Procedure）和函数类似都是定义在数据库中用来实现一个功能的 SQL 语句集合，但函数实现的功能针对性比较强，一般只实现一个明确的小功能，比如：截取字符串、加密、向上取整、随机数等，而存储过程用来实现更加复杂的业务层面上的功能，比如：注册、购买商品、下订单等。

2.2 函数和存储过程的比较

● 相同点

- 1、预先定义在 MySQL 中的为了实现一个特定功能的 SQL 语句集合。
- 2、需要调用才可以执行。
- 3、调用时可以传参数。

● 不同点

- 1) 存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。
- 2) 存储过程的参数可以有 IN, OUT, INOUT 三种类型，而函数只能有 IN 类~
- 3) 存储过程声明时不需要返回类型，而函数声明时需要描述返回类型，且函数体中必须包含一个有效的 RETURN 语句。
- 3) 存储过程需要单独执行，不能用在 sql 语句中，而函数需要使用 select 来执行，可以用在 SQL 语句中。

2.3 存储过程操作

2.3.1 创建存储过程

语法：

```
create procedure 过程名(参数列表)
begin
    过程体
end
结束符
```

注意：如果过程体中只有一行指令，可以省略 begin 和 end。

示例：创建一个存储过程，用来输出 Hello World！

```
delimiter $$ 
create procedure prec1()
begin
    select 'Hello World !';
end
$$
delimiter ;
```

2.3.2 查看存储过程

查看所有存储过程

```
show procedure status [like 'xxx'];
```

查看某一个存储过程

```
show create procedure 过程名;
```

2.3.3 调用存储过程

需要使用 call 指令调用：

```
call 过程名();
```

注意：存储过程是没有返回值的。

2.3.4 删除存储过程

语法：

```
drop procedure 过程名
```

2.4 存储过程的参数

参数有三种类型 **in**（进入、输入）、**out**（出来、输出）、**inout**（即输入又输出）：

- **in**

把数据从存储过程外面拿到存储过程里面。

- **out**

从存储过程里面把数据拿到外面来。

输出类型的参数，特点：

1. 调用时必须要传一个变量，不能是一个直接量。

```
mysql> call proc1('Abc', 10, 8.45);  
ERROR 1414 (42000): OUT or INOUT argument 2 for routine p_t_v.proc1 is not a variable or NEW pseudo-variable in BEFORE trigger  
mysql>  
mysql> set @a=10;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> set @b=8.45  
-> ;  
Query OK, 0 rows affected (0.00 sec)  
  
mysql>  
mysql> call proc1('Abc', @a, @b);  
+-----+-----+-----+  
| p1  | p2  | p3  |  
+-----+-----+-----+  
| Abc | NULL | 8.45 |  
+-----+-----+-----+  
1 row in set (0.00 sec)  
  
Query OK, 0 rows affected (0.01 sec)
```

out和inout类型的参数，在调用时，不能直接传直接量。

必须先放到一个变量中。

必须用变量来传out和inout类型的参数

2. **out** 不能用来接收传入的值，如果传了也会被设置为 null



```

mysql> set @a=10;
Query OK, 0 rows affected (0.00 sec)

mysql> set @b=3.45
->;
Query OK, 0 rows affected (0.00 sec)

mysql> call proc1('Abc',@a,@b);
+-----+-----+-----+
| p1  | p2  | p3  |
+-----+-----+-----+
| Abc | NULL | 8.45 |
+-----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

mysql>

```

280 主体
281 end
282
283 注意：如果主体中只有一行SQL语句，那么begin和end可以省略。
284
285 delimiter \$\$
286 create procedure proc1(in p1 varchar(10), out p2 int, inout p3 decimal(5,2))
287 begin
288 # 输出三个参数
289 select p1,p2,p3;
290 end
291 \$\$
292 delimiter ;

out类型的参数，是用来向外传值的，所以这里@a的值无法传到存储过程里面来。
所有out类型传进来的值都会被设置为null

3. 当执行到 end 时过程会把过程内的值覆盖过程外对应的变量

```

mysql> set @a='abc';
Query OK, 0 rows affected (0.00 sec)

mysql> set @b=10;
Query OK, 0 rows affected (0.00 sec)

mysql> set @c=8.45;
Query OK, 0 rows affected (0.00 sec)

mysql> call proc1(@a,@b,@c);
+-----+-----+-----+
| p1  | p2  | p3  |
+-----+-----+-----+
| abc | NULL | 8.45 |
+-----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> select @a,@b,@c;
+-----+-----+-----+
| @a  | @b  | @c  |
+-----+-----+-----+
| abc | 200 | 9.99 |
+-----+-----+-----+
1 row in set (0.00 sec)

```

285
286 delimiter \$\$
287 create procedure proc1(in p1 varchar(10), out p2 int, inout p3 decimal(5,2))
288 begin
289 # 输出三个参数
290 select p1,p2,p3;
291 # 修改这三个参数的值
292 set p1='bcd';
293 set p2=200;
294 set p3='9.99';
295 end
296 \$\$
297 delimiter ;

因为p1是in类型的参数，所以不会修改外面的@a的值
把@b的值修改了
把对应的@c的值修改了
out、和inout类型的参数，在存储过程中修改时，也会同时修改存储过程外面对应的变量。

● inout

即可以输入又可以输出类型的值。

特点：

- 调用时必须要传一个变量，不能是一个直接量。

```

mysql> call proc1('Abc',10,8.45);
ERROR 1414 (42000): OUT or INOUT argument 2 for routine p_t_v.proc1 is not a variable or NEW pseudo-variable in BEFORE trigger
mysql>
mysql> set @a=10;
Query OK, 0 rows affected (0.00 sec)

mysql> set @b=8.45
->;
Query OK, 0 rows affected (0.00 sec)

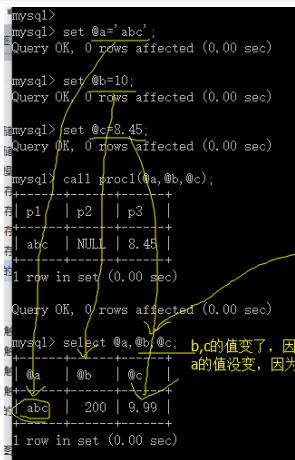
mysql>
mysql> call proc1('Abc',@a,@b);
+-----+-----+-----+
| p1  | p2  | p3  |
+-----+-----+-----+
| Abc | NULL | 8.45 |
+-----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

```

out和inout类型的参数，在调用时，不能直接传直接量。
必须先放到一个变量中。
必须用变量来传out和inout类型的参数

- 当执行到 end 时过程会把过程内的值覆盖过程外对应的变量



```

mysql> set @a='abc';
Query OK, 0 rows affected (0.00 sec)

mysql> set @b=10;
Query OK, 0 rows affected (0.00 sec)

mysql> set @c=8.45;
Query OK, 0 rows affected (0.00 sec)

mysql> call proc1(@a,@b,@c);
+-----+-----+-----+
| p1  | p2  | p3  |
+-----+-----+-----+
| abc | NULL | 8.45 |
+-----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> select @a,@b,@c;
+-----+-----+-----+
| @a  | @b  | @c  |
+-----+-----+-----+
| abc | 200 | 9.99 |
+-----+-----+-----+
1 row in set (0.00 sec)
    
```

MySQL command-line interface showing variable assignments and a stored procedure call. The stored procedure proc1 takes three parameters: p1 (in), p2 (out), and p3 (inout). The output shows that p1 is not modified, p2 is modified to 200, and p3 is modified to 9.99.



```

285
286 delimiter $$ 
287 create procedure proc1(in p1 varchar(10), out p2 int, inout p3 decimal(5,2))
288 begin
289     # 输出三个参数
290     select p1,p2,p3;
291     # 修改这三个参数的值
292     set p1='bcd';
293     set p2=200;
294     set p3='9.99';
295 end
296 $$ 
297 delimiter ;
    
```

Annotations explain the parameter types and modifications:

- # p1 is an in type parameter, so it won't modify the outer variable @a.
- # p2 is an out type parameter, so its value is modified.
- # p3 is an inout type parameter, so its value is modified.

总结：

in: 只能向存储过程中传值，存储过程中无法修改外面对应的变量。【向里传值】

out: 无法向存储过程中传值，但是在存储过程里面可以修改外面对应的变量。【向外写值】

inout: 即可以向过程中传值，又可以向存储过程外面写值。

2.5 案例、定义一个存储过程实现学生转班

3. 触发器



触发器（trigger），是一种特殊类型的存储过程，不同于之前的存储过程需要使用 call 来调用，触发器是在特定时机自动被调用执行的。【类似于 JS 里面的事件】

3.1 创建触发器

语法：

```
create trigger 触发器名称 触发时机 触发事件 on 表 for each row
begin
  ...
end
```

- 触发时机

before: 在事件之前触发。

after: 在事件之后触发。

- 触发事件

insert: 插入事件。

update: 修改事件。

delete: 删除事件。

- 触发器中的关键字

表中的每条记录在进行操作时都会触发一个触发器，而表中的每条记录都有操作前和操作后两个状态，分别保存在 new 和 old 两个关键字中。

示例：当向一个班级添加一个学生时，把这个班级的人数+1，当一个班级里面减少一个学生时，这个班级的人数-1。

```
delimiter $$  
create trigger update_stu_count after insert on students for each row  
begin  
  #这段SQL在向students表中插入数据之后执行。  
  #把这个学生所在的班级的人数+1  
  # 在触发器中可以使用new和old两个关键字。  
  # new: 新插入的数据的信息  
  update class set pcount=pcount+1 where id=new.class_id;  
end  
$$  
delimiter ;
```

```
mysql> insert into students values(8,'白马龙','男',19943438989',6,18,170);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from students;
```

id	name	gender	tel	class_id	age	height
1	八戒	男	13912345555	1	20	180
2	大师兄	男	13912346666	1	21	175
3	金角大王	男	13922222222	2	204	195
4	西施	女	13912342343	4	17	165
5	大乔	女	13912346666	3	22	168
6	貂蝉	女	13922233222	1	21	164
7	大乔	男	13922225454	2	19	183
8	白马龙	男	19943438989	6	18	170

8 rows in set (0.00 sec)

```
mysql> select * from class;
```

id	name	pcount
1	全栈开发班	3
2	Java开发班	2
3	服装设计班	1
4	美容美发班	1
5	挖掘机精英班	0
6	美国总统速成班	1

6 rows in set (0.00 sec)

当我们向学生表中插入一条记录时，这时会触发一个触发器中的代码，这个触发器根据这条记录学生所在的班级ID，修改了这个班级的学生人数。

注意：一张表的同一类触发器只能有一个，所以一张表最多可以定义六个触发器：before insert【插入前】、after insert【插入后】、before update【修改前】、after update【修改后】、before delete【删除前】、after delete【删除后】。

3.2 查看触发器

查看所有触发器：

```
show triggers
```

查看一个触发器详情：

```
show create trigger 触发器名
```

3.3 删除触发器

语法：

```
drop trigger 触发器名;
```

练习、当删除一个学生时，让这个学生所在的班级人数自动-1。

4. 视图

视图（view），是由 select 语句组成的一张虚拟表，可以当做表来使用。

4.1 什么是视图

视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值集形式存在。行和列数据来自定义视图的查询所引用的表，并且在引用视图时动态生成。

视图的好处：

1. 简单性，可以把复杂的 SQL 简单化。

```

# 取出所有的班级的名称以及每个班级中年龄最大的学生信息。
#第一步：先根据年龄排序
select * from students order by age desc;
+-----+-----+-----+-----+-----+
| id | name | gender | tel   | class_id | age | height |
+-----+-----+-----+-----+-----+
| 3  | 金角大王 | 男     | 13922222222 | 2 | 204 | 195 |
| 5  | 大乔   | 女     | 13912346666 | 3 | 22  | 168 |
| 2  | 大师兄 | 男     | 13912346666 | 1 | 21  | 175 |
| 6  | 貂蝉   | 女     | 13922233222 | 1 | 21  | 164 |
| 1  | 八戒   | 男     | 13912345555 | 1 | 20  | 180 |
| 7  | 大乔   | 男     | 13922225454 | 2 | 19  | 183 |
| 8  | 白马龙 | 男     | 19943438989 | 6 | 18  | 170 |
| 4  | 西施   | 女     | 13912342343 | 4 | 17  | 165 |
+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)

#第二步：根据班级分组
select * from (select * from students order by age desc) a group by class_id;
+-----+-----+-----+-----+-----+
| id | name | gender | tel   | class_id | age | height |
+-----+-----+-----+-----+-----+
| 1  | 八戒   | 男     | 13912345555 | 1 | 20  | 180 |
| 3  | 金角大王 | 男     | 13922222222 | 2 | 204 | 195 |
| 5  | 大乔   | 女     | 13912346666 | 3 | 22  | 168 |
| 4  | 西施   | 女     | 13912342343 | 4 | 17  | 165 |
| 8  | 白马龙 | 男     | 19943438989 | 6 | 18  | 170 |
+-----+-----+-----+-----+-----+
#第三步：根据class_id外连班级表，取出班级名称
select b.name class_name,a.* from (select * from students order by age desc) a left join class b on a.class_id=b.id group by a.class_id;
+-----+-----+-----+-----+-----+
| class_name | id | name | gender | tel   | class_id | age | height |
+-----+-----+-----+-----+-----+
| 全栈开发班 | 1  | 八戒   | 男     | 13912345555 | 1 | 20  | 180 |
| Java开发班 | 3  | 金角大王 | 男     | 13922222222 | 2 | 204 | 195 |
| 服装设计班 | 5  | 大乔   | 女     | 13912346666 | 3 | 22  | 168 |
| 美容美发班 | 4  | 西施   | 女     | 13912342343 | 4 | 17  | 165 |
| 美国总统速成班 | 8  | 白马龙 | 男     | 19943438989 | 6 | 18  | 170 |
+-----+-----+-----+-----+-----+

```

以上这个功能比较复杂，如果再添加更多功能这个 SQL 会越来越复杂，以至不利于维护，所以我们可以把这个复杂的 SQL 定义的一个视图，以后再用时直接使用视图即可。

2. 安全性，通过视图我们可以限制用户在一个数据的子集上。

4.2 视图的基本操作

4.2.1 创建视图

语法：

```
create view 视图名称 as select 语句
```

```

mysql> create view class_max_age as select b.name class_name,a.* from (select * from students order by age desc) a left
join class b on a.class_id=b.id group by a.class_id;
Query OK, 0 rows affected (0.01 sec)

```

说明：就是使用一个 select 语句的结果制作一张虚拟表。

4.2.2 查看视图

视图就是一张虚拟表，所以对视图的操作基本和表操作一样。

语法：

```
desc|show create table 视图名; # 查看某个视图  
show tables; # 查看所有视图
```

4.2.3 修改视图

语法：

```
alter view 视图名 as 新的 sql 语句
```

4.2.4 删除视图

语法：

```
drop view 视图名称
```

4.3 视图数据操作

就是使用一个 select 语句的结果制作一张虚拟表，所以操作视图中的数据和操作表的语法基本一样。

4.3.1 视图数据操作原理

视图是虚拟表，本身并没有数据，数据的操作实际上是通过视图找到对应的基表【select 语句中的表】进行操作的。

4.3.2 查看数据

```
mysql> select * from class_max_age;
+-----+-----+-----+-----+-----+-----+-----+
| class_name | id | name | gender | tel | class_id | age | height |
+-----+-----+-----+-----+-----+-----+-----+
| 全栈开发班 | 1 | 八戒 | 男 | 13912345555 | 1 | 20 | 180 |
| Java开发班 | 3 | 金角大王 | 男 | 13922222222 | 2 | 204 | 195 |
| 服装设计班 | 5 | 大乔 | 女 | 13912346666 | 3 | 22 | 168 |
| 美容美发班 | 4 | 西施 | 女 | 13912342343 | 4 | 17 | 165 |
| 美国总统速成班 | 8 | 白马龙 | 男 | 19943438989 | 6 | 18 | 170 |
+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from class_max_age where height between 170 and 180;
+-----+-----+-----+-----+-----+-----+-----+
| class_name | id | name | gender | tel | class_id | age | height |
+-----+-----+-----+-----+-----+-----+-----+
| 全栈开发班 | 1 | 八戒 | 男 | 13912345555 | 1 | 20 | 180 |
| 美国总统速成班 | 8 | 白马龙 | 男 | 19943438989 | 6 | 18 | 170 |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> ■
```

插入、修改、删除视图中的数据时有一个要求：视图必须是一个单表构成的视图。所以我们刚刚创建的视图是不能执行插入、修改和删除操作的，因为视图是由学生表和班级表两个表构成的。

4.3.3 插入数据

向视图中插入数据实际上会插入到 select 的基表中，不过要注意，如果 select 是从多表中查询出的数据，那么这个视图不允许插入数据。

4.3.4 修改数据

和操作表一样。

4.3.5 删除数据

和操作表一样，但是如果视图是由多张表中取的数据，那么不允许删除数据。

4.3.6 with check option

视图特有的一个属性，可以在创建视图时添加 `with check option` 属性，加了这个之后，对视图所有的数据操作时都会先使用这个条件验证一下，只有满足条件时才允许操作。

其实就是向这个表又额外加了一个限制。

语法：

```
create view 神图名称 as select 语句 [where 条件 with check option]
```

5. sql_mode

6. 今日总结



传智专修学院
CHUANZHI SPECIALIZED COLLEGE

力学笃行 志存高远