

目录

数据库简介	5
数据库概念	5
数据库分类	5
关系型数据库	5
典型关系型数据库	6
SQL 介绍	6
MySQL 基本介绍	7
启动和停止 MySQL 服务	7
命令行方式	8
系统服务方式	9
登录和退出 MySQL 系统	9
登录	10
退出	10
MySQL 服务端架构	11
数据库的基本操作	11
创建数据库	11
显示数据库	12
选择数据库	13
修改数据库	13
删除数据库	14
数据表操作	14
创建数据表	14
显示数据表	16
设置表属性	18
修改表结构	19
删除表结构	20
数据基础操作	21
插入操作	21
查询操作	22
删除操作	22
更新操作	23
字符集	23
字符编码概念	23
字符集概念	24
MySQL 字符集设置	24
设置客户端所有字符集	24
列类型	27
整数类型	27
tinyint	27
smallint	27
mediumint	27

int.....	27
bigint	27
无符号标识设定.....	28
显示长度	28
小数型.....	30
浮点型.....	30
定点数.....	32
时间日期型.....	32
字符串类型.....	34
MySQL 记录长度.....	38
列属性	38
null 属性.....	38
默认值.....	39
列描述.....	40
主键	40
自增长.....	43
唯一键.....	46
表关系	49
一对一	49
一对多	50
多对多	50
高级数据操作	51
新增数据	51
多数据插入.....	51
主键冲突	52
蠕虫赋值	52
更新数据	53
删除数据	53
查询数据	54
select 选项	54
字段列表	55
from 数据源	56
where 子句	57
group by 子句	57
having 子句	60
order by 子句	60
limit 子句	61
聚合函数	61
运算符	62
in 运算符	64
is 运算符	64
like 运算符	64
联合查询.....	65
基本概念	65

应用场景	65
基本语法	65
Order by 的使用	66
连接查询	67
连接查询概念	67
交叉连接	67
内连接	68
外连接	69
自然连接	70
using 关键字	71
子查询	71
子查询概念	71
主查询概念	71
子查询与主查询的关系	71
子查询分类	72
标量子查询	72
列子查询	72
行子查询	73
表子查询	73
exists 子查询	74
特定关键字	74
数据备份与还原	75
意义	75
整库备份与还原	75
应用场景	75
应用方案	76
用户权限管理	78
用户管理	78
创建用户	78
删除用户	79
修改用户密码	80
权限管理	80
授予权限：grant	80
取消权限：revoke	81
刷新权限：flush	82
密码丢失找回	82
外键	82
外键概念	82
外键的操作	82
增加外键	82
修改&删除外键	83
外键的基本要求	84
外键的约束	84
约束的基本概念	84

外键约束的概念	85
约束作用	86
视图	87
视图概念	87
视图基本操作	87
创建视图	87
使用视图	88
修改视图	88
删除视图	88
DML 操作视图	89
with check option	89
事务安全	90
事务概念	90
事务基本原理	90
自动事务	90
手动事务	92
开始事务	93
执行事务	93
提交事务	93
回滚点	93
事务特点	93
变量	94
系统变量	94
查看系统所有变量	94
修改系统变量	95
会话变量	96
局部变量	97
流程结构	98
if 分支	98
基本语法	98
复合语法	98
while 循环	99
case 循环	100
函数	101
内置函数	101
字符串函数	101
时间函数	101
数学函数	102
其他函数	102
自定义函数	103
创建函数	103
查看函数	105
调用函数	105
修改函数	105

删除函数	106
注意事项	106
变量作用域	109
局部作用域	109
会话作用域	109
全局作用域	110
存储过程	110
存储过程的概念	110
与函数的区别	110
相同点	110
不同点	110
存储过程操作	111
创建过程	111
查看过程	111
调用过程	112
删除过程	112
存储过程的形参类型	112
in	112
out	112
inout	112
触发器	115
触发器概念	115
基本概念	115
作用	115
触发器优缺点	115
触发器基本语法	116
创建触发器	116
查看触发器	117
触发触发器	117
删除触发器	117
触发器应用	117
记录关键字：new 、 old	117
商品自动扣除库存	118
游标	120
jdbc	120

MySQL 基础

数据库简介

数据库概念

数据库（database）是按照数据结构来组织、存储和管理数据的建立在计算机存储设备上的仓库。

数据库：存储数据的仓库。

数据库分类

网络数据库

网络数据库是指把数据库技术引入到计算机网络系统中，借助于网络技术将存储于数据库中的大量信息及时发布出去，而计算机网络借助于成熟的数据库技术对网络中的各种数据进行有效管理，并实现用户与网络中的数据库进行实时动态数据交互。

层级数据库

层次结构模型实质上是一种有根结点的定向有序树（在数学中‘树’被定义为一个无回的连通图）

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

关系型数据库

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。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: 指的是数据以什么方式来存储，是一种二维表的形式存储。

关系操作集合: 如何来关联和管理对应的存储数据，SQL 指令。

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。

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）

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：Microsoft SQL Server，MySQL（开源免费）

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。

SQL 介绍

SQL 基本介绍

结构化查询语言（Structured Query Language），简称 SQL，是一种特殊目的的编程语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统，同时也是数据库脚本文件的扩展名

SQL 就是专门为关系型数据库而设计出来的。

SQL 分类

1、数据查询语言（DQL: Data Query Language）：

其语句，也称为“数据检索语句”，用以从表中获得数据，确定数据怎样在应用程序中给出。保留字 SELECT 是 DQL（也是所有 SQL）用的最多的动词，其他 DQL 常用的保留字有 WHERE, ORDER BY, GROUP BY 和 HAVING。这些 DQL 保留字常与其他类型的 SQL 语句一起使用。

专门用于查询数据，代表语句为 select/show。

2、数据操作语言（DML: Data Manipulation Language）：

其语句包括动词 INSERT, UPDATE 和 DELETE。它们分别用于添加、修改和删除表中的行。也称为动作查询语言。

专门用于写数据: 代表指令为 insert、update 和 delete。

3、事务处理语言 (TPL):

它的语句能确保被 DML 语句影响的表的所有行及时得以更新。TPL 语句包括 BEGIN, TRANSACTION, COMMIT 和 ROLLBACK。(不是所有的关系型数据库都提供事务安全处理)

专门用于事务安全处理: transaction

4、数据控制语言 (DCL):

它的语句通过 GRANT 和 REVOKE 获得许可, 确定单个用户和用户组对数据库对象的访问。某些 ROBMS 可用 GRANT 和 REVOKE 控制对表单各列的访问。

专门用于权限管理: 代表指令为 grant 和 revoke

5、数据定义语言 (DDL):

其语句包括动词 CREAT 和 DROP。在数据库中创建新表或删除表(CREAT TABLE 或 DROP TABLE); 为表加入索引等。DDL 包括许多与人从数据库目录中获得数据有关的保留字。它也是动作查询的一部分。

专门用于结构管理: 代表指令 create 和 drop (alter)

MySQL 基本介绍

MySQL 是一个关系型数据库管理系统。

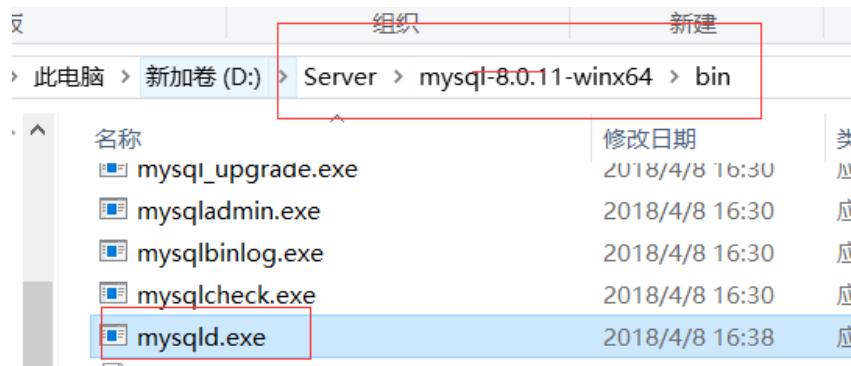
- 1、MySQL 是一种开源免费的数据库产品。
- 2、MySQL 对 PHP 的支持最好 (wamp 或者 lamp)

MySQL 中用到的操作指令就是 SQL 指令。

启动和停止 MySQL 服务

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



命令行方式

通过 Windows 下打开 cmd，然后使用命令进行管理

net start 服务 (mysql): 开启服务

错误情况：

```
统C:\Users\YZQ>net start mysql  
发生系统错误 5。  
拒绝访问。
```

以管理员身份运行即可解决，若想永久解决，去

C:\Windows\System32，找到 cmd. exe，先创建快捷方式，右键选择属性-快捷方式-高级-用管理员身份运行。

修改错误后：

```
(c) 2018 Microsoft Corporation。保留所有  
C:\WINDOWS\system32>net start mysql  
MySQL 服务正在启动 ..  
MySQL 服务已经启动成功。
```

net stop mysql : 关闭服务

```
C:\WINDOWS\system32>net stop mysql  
MySQL 服务正在停止..  
MySQL 服务已成功停止。
```

net start + 服务名 : 启动 Windows 中服务。

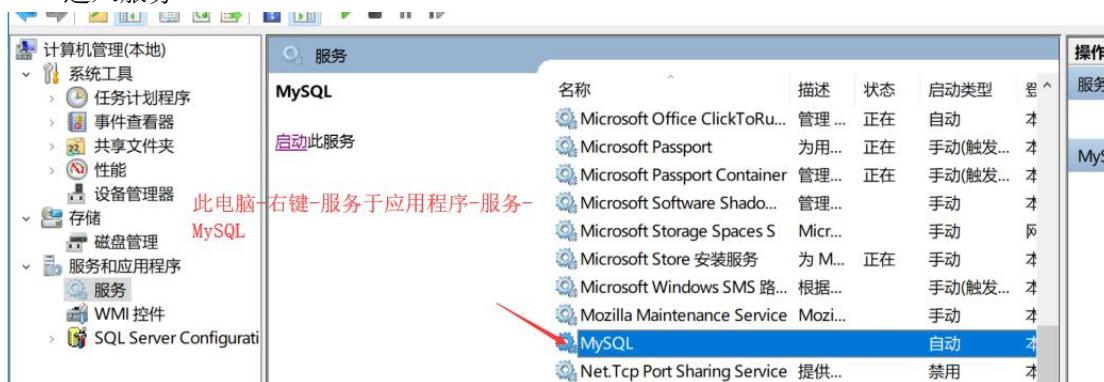
net stop + 服务名 : 关闭 Windows 中服务。

系统服务方式

前提：在安装 MySQL 的时候将 MySQL 添加到 Windows 的服务中去了。

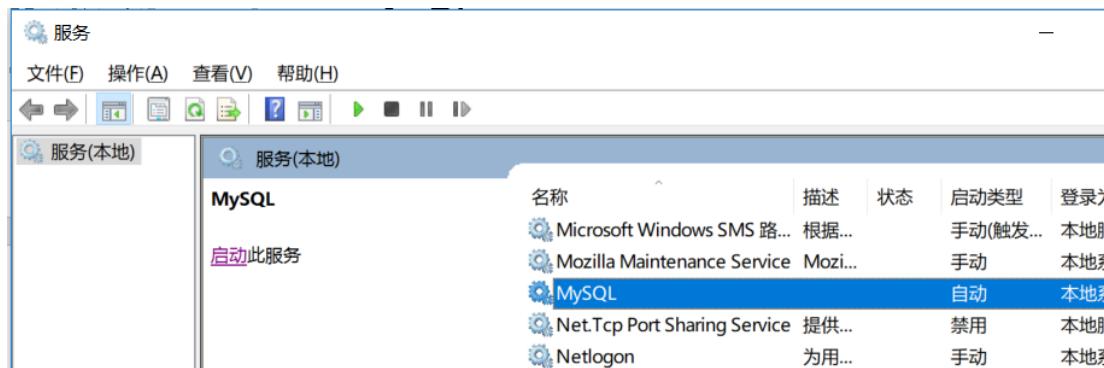
方式 1：

进入服务



方式 2：

进入服务：通过命令行：win+r – 输入 services.msc



通过服务对 MySQL 服务器进行管理

方案 1：右键服务，然后选择开启或者停止

方案 2：双击服务，进入到服务详情界面，点击开启或者停止按钮。

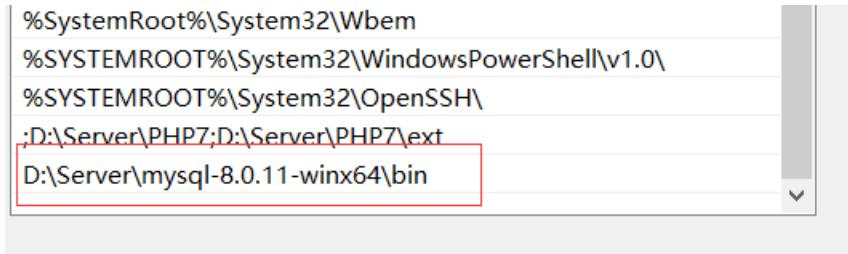
登录和退出 MySQL 系统

通过客户端（mysql.exe）与服务器进行连接认证，就可以进行操作。

通常：服务端和客户端不在同一台电脑上。

登录

1、找到 mysql.exe（通过 cmd 控制台：如果在安装的时候指定了 mysql.exe 所在的路径为环境变量，就可以直接访问，如果没有，那么就必须进入 mysql.exe 所在目录）



2、输入对应的服务器地址： -h [ip 地址/域名]

3、输入服务器中 mysql 监听的端口： -P:3306

4、输入用户名： -u:root

5、输入密码： -p:root

连接认证基本语法： mysql.exe -h 主机地址 -P 端口 -u 用户名 -p 密码

```
C:\Users\YZ0>mysql -uroot -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.11 MySQL Community Server - GPL

Copyright (c) 2000, 2018, 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> ■
```

注意事项：

- 1、通常端口都可以默认： mysql 的监听端口通常都是 3306
- 2、mysql 或者 mysql.exe
- 3、密码的输入可以先输入-p，直接换行，然后再以密文方式输入密码（安全）
- 4、-h 主机名，可以使用该参数指定主机名或 Ip，如果不指定，默认是 localhost。

退出

断开与服务器的链接：通常 mysql 提供的服务器数量有限，一点客户端用完，建议就应该断开连接。

建议方式： 使用 SQL 提供的指令

```
exit; //exit 带分号  
\q; /\q 带不带分号都可以  
quit; //quit 带不带都可以
```

MySQL 服务端架构

MySQL 服务端架构由以下几层构成：

- 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（可以有很多个仓库）
- 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）

数据库中常用的几个关键字

row：行

column：列（filed）

数据库的基本操作

数据库是数据存储的最外层（最大单元）

创建数据库

基本语法：create database 数据库名字[库选项]; --》 []代表可选参数

```
mysql> -- 创建数据库  
mysql> create database mydatabase;  
Query OK, 1 row affected (0.52 sec)  
mysql>
```

库选项：数据库的相关属性

- 1) 字符集：charset，代表着当前数据库下所有表存储的数据默认指定的字符集（如果当前不指定，那么采用 DBMS 默认的字符集）
- 2) 校对集：collate，

用法：create database 数据库名称 charset 字符集名称

```
Query OK, 1 row affected (0.52 sec)

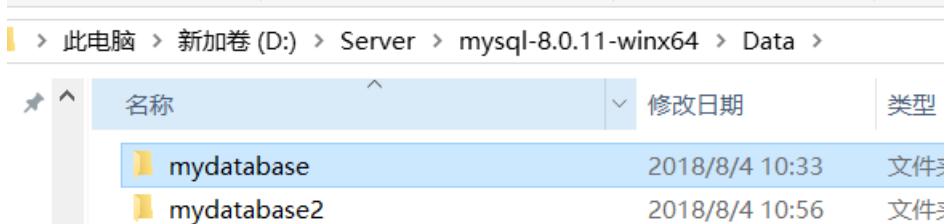
mysql> create database mydatabase2 charset gbk;
Query OK, 1 row affected (0.40 sec)

mysql>
```

显示数据库

每当用户通过 SQL 指令创建一个数据库，那么系统就会产生一个对应的存储数据的文件夹（关系型数据库保存在磁盘），位于 data 目录下，data 是安装时自己指定的。

```
# 设置mysql数据库的数据的存放目录
datadir=D:\Server\mysql-8.0.11-winx64\Data
```



创建数据库但是没有任何数据，只有一个目录文件夹。

显示全部

基本语法: show databases;

```
Query OK, 1 row affected (0.40 sec)

mysql> -- 显示所有数据库
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema
| mydatabase
| mydatabase2
| mysql
| performance_schema
| sys
+-----+
6 rows in set (0.45 sec)
```

解释图注：

- 保存数据库所有的结构信息（表，库）
- 自己建立的，剩下都是系统的
- 核心数据库：权限关系
- 效率库
- 数据库的相关配置

显示部分

基本语法: show databases like ‘匹配模式’;

_:匹配当前位置单个字符

%:匹配指定位置多个字符

获取以 my 开头的全部数据库: ‘my%’ ;

获取以 m 开头,后面第一个字母不确定，最后为 database 的数据库: ‘m_database’ ;

获取以 database 结尾的数据库：‘%database’；

```
mysql> -- 查看以my开头的数据库
mysql> show databases like 'my%';
+-----+
| Database (my%) |
+-----+
| mydatabase      |
| mydatabase2     |
| mysql           |
+-----+
3 rows in set (0.00 sec)
```

显示数据库创建语句

基本语法: show create database 数据库名字;

```
mysql> -- 查看数据库创建语句
mysql> show create database mydatabase;
+-----+-----+
| Database | Create Database |
+-----+-----+
| mydatabase | CREATE DATABASE `mydatabase` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+-----+-----+
1 row in set (0.00 sec)

mysql>
```

看到的指令并非原始指令，已经被系统加工过

选择数据库

为什么要选择数据库？因为数据是存储到数据表，表存在于数据库下。如果要操作数据，那么就必须进入到对应的数据表才行。

基本语法: use 数据库名字;

```
mysql> -- 选择数据库
mysql> use mydatabase;          表示当前已经进入到指定的数据库环境
Database changed ←
mysql>
```

修改数据库

修改数据库字符集（库选项）

基本语法: alter database 数据库名字 charset = 字符集;

```
Database changed
mysql> -- 修改数据库字符集
mysql> alter database mydatabase charset gbk; charset 可以不用等于
Query OK, 1 row affected (0.42 sec) 号直接指定也可以有
```

是否可以修改数据库名字？mysql5.5 之前可以修改 rename 命令，但是 5.5 之后不可以。

删除数据库

基本语法： drop database 数据库名字；

```
Query OK, 1 row affected (0.42 sec)
mysql> -- 删除数据库
mysql> drop database mydatabase2;
Query OK, 0 rows affected (0.23 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mydatabase |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

删除虽简单，但是切记要做好安全操作，确保里面数据没有问题（重要）

删除数据库后：对应的存储数据的文件夹也会被删除。

数据表操作

创建数据表

数据库中数据表的名字通常有前缀：取数据库的前两个字母加下划线，易于区分。

普通创建表

基本语法： create table 表名 (字段名 字段类型[字段属性], 字段名 字段类型[字段属性]...) [表选项]

```
attributes. Other names may be trademarks of their re-
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the
mysql> -- 创建数据表
mysql> create table class(
    -> -- 字段名 字段类型
    -> -- 字段与表其实是分不开的
    -> name varchar(10) -- 10个字符 (不能超过)
    -> );
ERROR 1046 (3D000): No database selected
mysql>
```

表必须放在对应的数据库下：有两种方式可以将表挂入到指定的数据库下

1、在数据表名字前面加上数据库名字，用“.”号连接即可。数据库.数据表

```
mysql> -- 将数据表挂到数据库下
mysql> create table mydatabase.class(
    -> name varchar(10)
    -> );
ERROR 1050 (42S01): Table 'class' already exists
mysql>
```

2、在创建数据表之前先进入到某个具体的数据库即可：use 数据库名字；

```
-- 进入数据库， 创建表
use mydatabase;          数据库名称
create table class(
name varchar(10)
);
默认创建到当前所在数据库
```

表选项：与数据库选项类似

engine: 存储引擎，mysql 提供的具体存储数据的方式，默认有一个 innodb (5.5 之前默认是 myisam)

charset: 字符集 – 只对当前自己表有效 (级别比数据库高)

collate: 校对集 –只对当前自己表有效 (级别比数据库高)

```
mysql> use mydatabase;
Database changed
mysql> -- 使用表选项
mysql> create table student(
    -> name varchar(10)
    -> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.67 sec)
```

复制已有表结构

从已经存在的表复制一份 (只复制结构，如果表中有数据不复制)

基本语法：create table 新表名 like 表名； //只要使用数据库.表名，就可以在任何数据库下

访问其他数据库的表名。

```
-- 在test数据库下创建一个与student一样的表
use test;
create table teacher like mydatabase.student;
```

显示数据表

每一张数据表创建，那么就会在对应的数据库下创建一些文件（与存储引擎有关）

注意：innodb 存储引擎所有文案都存储在外部的 ibdata 文件

The screenshot shows a Windows File Explorer window with the path: This PC > New Volume (D:) > Server > mysql-8.0.11-winx64 > Data. The 'Data' folder contains several files:

	名称	修改日期	类型
	ca.pem	2018/8/24 10:45	PEM 文件
	ca-key.pem	2018/7/24 10:45	PEM 文件
	client-cert.pem	2018/7/24 10:45	PEM 文件
	client-key.pem	2018/7/24 10:45	PEM 文件
	DESKTOP-FDEGBR1.err	2018/8/4 9:24	ERR 文件
	DESKTOP-FDEGBR1.pid	2018/8/4 9:39	PID 文件
	ib_buffer_pool	2018/8/4 9:24	文件
	ib_logfile0	2018/8/4 14:17	文件
	ib_logfile1	2018/7/24 10:44	文件
	ibdata1	2018/8/4 14:17	文件

显示所有表

基本语法: show tables;

```
mysql> -- 查看所有表
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| teacher        |
+-----+
1 row in set (0.06 sec)
```

显示匹配表

基本语法: show tables like '匹配模式';

```
Z rows in set (0.00 sec)
6 mysql> -- 查看匹配数据表
mysql> show tables like 'c%';
+-----+
| Tables_in_mydatabase (c%) |
+-----+
| class |
+-----+
1 row in set (0.00 sec)
```

显示表结构

本质含义: 显示表中所包含的字段信息 (名字, 类型, 属性等)

```
describe 表名 [column]; //列名称可以使用通配符
desc 表名
show columns from 表名 [from 表所在的数据库]
或者
show columns from 数据库.表名;
explain table;
```

```
mysql> -- 显示表结构
mysql> describe class;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name | varchar(10) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> desc student;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name | varchar(10) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
mysql> show columns from teacher;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name | varchar(10) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

字段名字
字段类型
值是否允许为空, yes表示允许
表示索引
默认值: null
额外的属性

显示表创建语句

查看数据表创建时的语句, 此语句看到的结果已经不是用户之前自己输入的。系统加工了。

基本语法: show create table 表名;

```
mysql> -- 查看表创建语句
mysql> show create table student;
+-----+
| Table | Create Table
+-----+
| student | CREATE TABLE `student` (
  `name` varchar(10) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+

```

数据表的多维显示

MySQL 中有多种语句结束符

;与\g 所表示的效果是一样的，都是字段在上排横着，下面跟着对应的数据。

\G 字段在左边竖着，数据在右边横着。

```
in
mysql> show create table student\G
***** 1. row *****
Table: student
Create Table: CREATE TABLE `student` (
  `name` varchar(10) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
e
```

设置表属性

表属性指的就是表选项: engine, charset 和 collate

基本语法:

alter table 表名 表选项 [=] 值; // [=] 代表可选参数

注意: 如果数据库已经确定了，里面有很多数据了，不要轻易修改表选项 (字符集影响不大)

修改表结构

修改表名: rename 旧表名 to 新表名;

修改表选项: alter table 表名 表选项 [=] 新值;

新增字段: alter table 表名 add [column] 新字段名 列类型 [列属性] [位置 first/after 字段名]

特点: 默认加到表的最后面

字段位置: 字段想要存放的位置

first: 在某某之前 (最前面), 第一个字段

after 字段名: 放在某个具体的字段之后, (默认)

修改字段名: alter table 表名 change 旧字段名 新字段名 字段类型 [列属性] [新位置];

修改字段类型 (属性): alter table 表名 modify 字段名 新类型 [新属性] [新位置];

删除字段: alter table 表名 drop 字段名;

```
mysql> desc my_student;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| name  | varchar(10) | YES | NULL |          |          |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

-- 给学生表增加age字段
mysql> alter table my_student add column age int;
Query OK, 0 rows affected (0.83 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc my_student;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| name  | varchar(10) | YES | NULL |          |          |
| age   | int(11)    | YES | NULL |          |          |
+-----+-----+-----+-----+-----+
0 rows in set (0.00 sec)
```

```
-- 增加字段, 放在第一个
mysql> alter table my_student add id int first;
Query OK, 0 rows affected (0.57 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc my_student;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id    | int(11) | YES |     | NULL    |          |
| name  | varchar(10) | YES |     | NULL    |          |
| age   | int(11)  | YES |     | NULL    |          |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> -- 修改字段名
mysql> alter table my_student change age nj int;
Query OK, 0 rows affected (0.41 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc my_student;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | YES  |     | NULL    |       |
| name  | varchar(10)| YES |     | NULL    |       |
| nj    | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
没 mysql> -- 修改字段类型
. e mysql> alter table my_student modify name varchar(20);
  Query OK, 0 rows affected (0.13 sec)
  Records: 0 Duplicates: 0 Warnings: 0
又
.e mysql> desc my_student;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | YES  |     | NULL    |       |
| name  | varchar(20)| YES |     | NULL    |       |
| nj    | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

```
之 mysql> -- 删除字段
之 mysql> alter table my_student drop nj;
Query OK, 0 rows affected (0.89 sec)
Records: 0 Duplicates: 0 Warnings: 0
之
之 mysql> desc my_student;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | YES  |     | NULL    |       |
| name  | varchar(20)| YES |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

删除表结构

基础语法: `drop table 表名[,表名 2...]`; 可以同时删除多个表。

数据基础操作

插入操作

本质含义：将数据以 SQL 的形式存储到指定的数据表（字段）里面

基本语法：向表中指定字段插入数据

insert into 表名 [(字段列表)] values (对应字段列表)

```
mysql> insert into my_teacher (name, age) values(' Jack', 30);
Query OK, 1 row affected (0.49 sec)
```

1、注意：后面（values 中）对应的值列表只需要与前面的字段列表相对应即可（不一定与表结构完全一致）

```
mysql> insert into my_teacher (age, name) values(40, ' Tom');
Query OK, 1 row affected (0.39 sec)
```

2、注意：字段列表并不一定非要所有的表中字段，只要对应即可

```
mysql> insert into my_teacher (name) values(' Han');
Query OK, 1 row affected (0.42 sec)
```

基本语法：向表中所有字段插入数据

insert into 表名 values (对应表结构) //值列表必须与字段列表一致，顺序

```
mysql> insert into my_teacher values ('Lilei', 28);
Query OK, 1 row affected (0.47 sec)
```

此时值列表必须对应表结构字段列表

insert into 表名 set 字段 1=“字段值”，字段 2=“字段值”...;

```
1 row in set (0.34 sec)

mysql> insert into my_stu set stu_id='stu006', stu_name='李宁', class_id='1', stu_age='18', stu_height='123', gender=1;
Query OK, 1 row affected (0.51 sec)

mysql> select * from my_stu;
```

查询操作

查询表中全部数据: select * from 表名;

```
mysql> -- 获取所有数据
mysql> select * from my_teacher;
+-----+-----+
| name | age  |
+-----+-----+
| Jack | 30   |
| Tom  | 40   |
| Han  | NULL |
| Lilei| 28   |
+-----+-----+
4 rows in set (0.00 sec)
```

查询表中部分字段: select 字段列表 from 表名; //字段列表使用逗号 “,” 隔开

```
mysql> -- 查询表中部分字段
mysql> select name from my_teacher;
+-----+
| name |
+-----+
| Jack |
| Tom  |
| Han  |
| Lilei|
+-----+
4 rows in set (0.00 sec)
```

简单条件查询数据: select 字段列表/* from 表名 where 字段名 = 值;

```
mysql> -- 简单条件查询:获取年龄为30的人的名字
mysql> select name from my_teacher where age = 30;
+-----+
| name |
+-----+
| Jack |
+-----+
1 row in set (0.00 sec)
```

在 select 语句之前添加 explain 关键字可查看 mysql 是如何评估并执行一个 select 查询。

删除操作

基本语法: delete from 表名 [where 条件]; // 如果没有 where 条件, 意味着系统会自动删除该表所有数据 (慎用)

```

mysql> -- 删除数据:删除年龄为30 岁的老师
mysql> delete from my_teacher where age = 30;
Query OK, 1 row affected (0.52 sec)

mysql>
mysql> select * from my_teacher;
+-----+-----+
| name | age |
+-----+-----+
| Tom  | 40  |
| Han  | NULL |
| Lilei | 28  |
+-----+-----+

```

更新操作

更新：将数据进行修改（通常是修改部分字段数据）

基本语法：update 表名 set 字段名 = 新值 [where 条件]; // 如果没有 where 条件，那么所有表中对应的那个字段都会被修改成同一值。

```

mysql> -- 更新年龄Han
mysql> update my_teacher set age = 28 where name = 'Han';
Query OK, 1 row affected (0.47 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from my_teacher;
+-----+-----+
| name | age |
+-----+-----+
| Tom  | 40  |
| Han  | 28  |
| Lilei | 28  |
+-----+-----+
3 rows in set (0.00 sec)

```

字符集

字符编码概念

字符（character）是各种文字和符号的总称，包括各国家文字、标识符号、图形符号、数字等。

在计算机中所看到的任何内容都是字符构成的。

字符编码（character code）是计算机针对各种符号，在计算机中的一种二进制存储代号。

字符集概念

字符集 (character set) 是多个字符的集合，字符集种类比较多，每个字符集包含的字符个数不同。

常见字符集名称：ASCII 字符集、GB2312 字符集、BIG5 字符集、GB18030 字符集、Unicode 字符集等。



MySQL 字符集设置

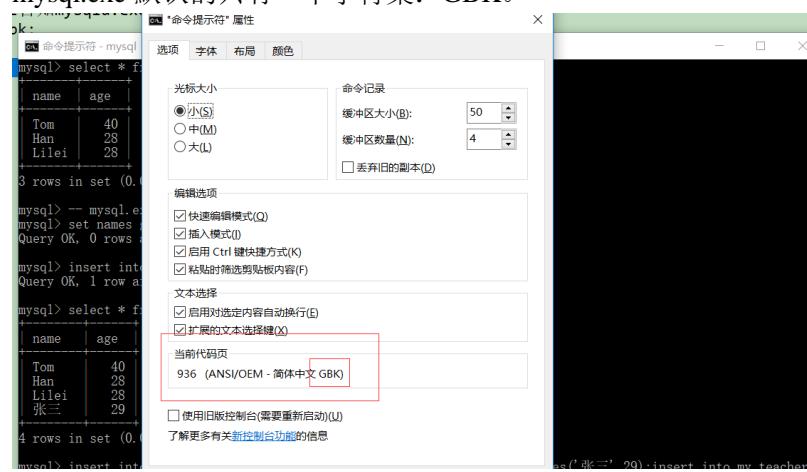
设置客户端所有字符集

如果直接通过 cmd 下的 mysql.exe 进行中文数据插入，那么可能会出错。

出错原因：

- 1、用户是通过 mysql.exe 来操作 mysqld.exe
- 2、真正的 SQL 执行时 mysqld.exe 来执行
- 3、mysql.exe 将数据传入 mysqld.exe 的时候，没有告知其对应的符号规则(字符集)，而 mysqld 也没有能力自己判断，就会使用自己默认的（字符集）

解决方案：mysql.exe 客户端在进行数据操作之前将自己使用的字符集告诉 mysqld，cmd 下 mysql.exe 默认的只有一个字符集：GBK。



mysql.exe 如何告知 mysqld.exe 对应的字符集类型为 gbk?

快捷方式: set names 字符集;

深层原理: 客户端, 服务端, 连接层 (show variables like 'character_set%')

mysql.exe 与 mysqld.exe 之间的处理关系一共分为三层:

客户端传入数据给服务端: client : character_set_client

服务端返回数据给客户端: server : character_set_results

客户端与服务端之间的连接: connection : character_set_connection

set names 字符集 的本质: 就是一次打通三层关系的字符集, 变得一致。

在系统中有三个变量来记录着这三个关系对应的字符集: show variables like 'character_set%';

Variable_name	Value
character_set_client	gbk
character_set_connection	gbk
character_set_database	gbk
character_set_filesystem	binary
character_set_results	gbk
character_set_server	utf8
character_set_system	utf8
character_sets_dir	D:\Server\mysql-8.0.11-winx64\share\charsets\

查看一个新的客户端对应的字符集关系

Variable_name	Value
character_set_client	utf8
character_set_connection	utf8
character_set_database	utf8
character_set_filesystem	binary
character_set_results	utf8
character_set_server	utf8
character_set_system	utf8
character_sets_dir	D:\Server\mysql-8.0.11-winx64\share\charsets\

Query OK, 1 row affected (0.06 sec)

Variable_name	Value
character_set_client	gbk
character_set_connection	gbk
character_set_database	gbk
character_set_filesystem	binary
character_set_results	gbk
character_set_server	utf8
character_set_system	utf8
character_sets_dir	D:\Server\mysql-8.0.11-winx64\share\charsets\

Query OK, 1 row affected (0.06 sec)

修改服务器端变量的值:

set 变量名 = 值;

set character_set_client = gbk;

重新进行数据插入和查看的结果: 插入 OK, 但是查看乱码。

```

mysql> -- 修改变量
mysql> set character_set_client = gbk;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'character_set%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | gbk |
| character_set_connection | utf8 |
| character_set_database | utf8 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | utf8 |
| character_set_system | utf8 |
| character_sets_dir | D:\Server\mysql-8.0.11-winx64\share\charsets\ |
+-----+-----+
8 rows in set, 1 warning (0.00 sec)

mysql> insert into my_teacher values('李四', 35); 客户端给服务器端的数据: 已经
ERROR 1046 (3D000): No database selected 正确(字符集识别)
mysql> use mydatabase;
Database changed
mysql> insert into my_teacher values('李四', 35);
Query OK, 1 row affected (0.10 sec)

mysql> select * from my_teacher;
+-----+-----+
| name | age |
+-----+-----+
| Tom | 40 |
| Han | 28 |
| Lilei | 28 |
| 李四 | 29 |
| 钱康源 | 35 |
+-----+-----+
8 rows in set (0.00 sec)

```

修改结果字符集为 gbk

```

mysql> set character_set_results = gbk;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'character_set%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | gbk |
| character_set_connection | utf8 |
| character_set_database | gbk |
| character_set_filesystem | binary |
| character_set_results | gbk |
| character_set_server | utf8 |
| character_set_system | utf8 |
| character_sets_dir | D:\Server\mysql-8.0.11-winx64\share\charsets\ |
+-----+-----+
8 rows in set, 1 warning (0.00 sec)

mysql> select * from my_teacher;
+-----+-----+
| name | age |
+-----+-----+
| Tom | 40 |
| Han | 28 |
| Lilei | 28 |
| 李四 | 35 |
+-----+-----+

```

connection 只是为了更方便客户端与服务端进行字符集转换而设。

```

set names gbk;
set character_set_client = gbk; //为了让服务端识别客户端传来的数据
set character_set_connection = gbk; //更好的帮助客户端与服务端之间进行字符集转换

```

```
set character_set_results = gbk; //为了告诉客户端服务端所有的返回的数据字符集
```

列类型

整数类型

tinyint

迷你整型，系统采用一个字节来保存的整型，一个字节=8byte，最大能表示的数据是 0-255。

smallint

小整型，系统采用两个字节来保存的整型，能表示 0-65535 之间的整型

mediumint

中整型，采用三个字节来保存数据

int

整型，(标准整型)，采用四个字节来保存数据。

bigint

大整型，采用八个字节来保存数据。

```

命令提示符 - mysql -uroot -p
mysql> set names gbk;
Query OK, 0 rows affected (0.00 sec)

mysql> use mydatabase;
Database changed
mysql> -- 创建数据表
mysql> create table my_int(
    -> int_1 tinyint,
    -> int_2 smallint,
    -> int_3 mediumint,
    -> int_4 int,
    -> int_5 bigint
    -> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.44 sec)

mysql> -- 插入数据
mysql> insert into my_int values(10,10000,100000,10000000,100000000000);
Query OK, 1 row affected (0.50 sec)

mysql> select * from my_int;
+-----+-----+-----+-----+-----+
| int_1 | int_2 | int_3 | int_4 | int_5 |
+-----+-----+-----+-----+-----+
|    10 | 10000 | 100000 | 10000000 | 100000000000 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> insert into my_int values(255,255,255,255,255);
ERROR 1264 (22003): Out of range value for column 'int_1' at row 1
mysql>

```

错误原因：并不是tinyint没有那么大的空间，而是因为mysql默认的为整形增加负数。实际表示区间为-128,127

超出范围

实际应用中，应该根据对应的数据范围来选定对应的整型类型，通常使用的比较多的是 tinyint 和 int。

无符号标识设定

无符号，表示存储的数据在当前字段中，没有负数（只有正数，如 tinyint 区间为 0-255）

基本语法：在列类型之后加上一个 unsigned。

显示长度

Field	Type	Null	Key	Default	Extra
int_6	tinyint(3) unsigned	YES		NULL	
int_1	tinyint(4)	YES		NULL	
int_2	smallint(6)	YES		NULL	
int_3	mediumint(9)	YES		NULL	显示长度
int_4	int(11)	YES		NULL	
int_5	bigint(20)	YES		NULL	

6 rows in set (0.00 sec)

显示长度：指数据在数据显示的时候，到底可以显示多长位。

tinyint (3)：表示最长可以显示 3 位，unsigned 说明只能是正数，0-255 永远不会超过三个字节长度。

tinyint (4)：表示最长可以显示 4 位，-128~127,带符号 (有符号位)，需要四个字节。

显示长度只是代表了数据是否可以达到指定的长度，但是不会自动满足到指定长度：如果想要数据显示的时候，保持最高位（显示长度），那么还需要给字段增加一个 zerofill 属性才可以。

zerofill：从左侧开始填充 0（左侧不会改变数值大小），所以负数的时候就不能使用 zerofill，一旦使用 zerofill 就相当于确认该字段为 unsigned。

```
mysql> alter table my_int add int_8 tinyint zerofill first;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
for the right syntax to use near '8 tinyint zerofill first' at line 1
mysql> alter table my_int add int_8 tinyint zerofill first;
Query OK, 0 rows affected (0.97 sec)
Records: 0 Duplicates: 0 Warnings: 0
          自动变为unsigned zerofill

mysql> desc my_int;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| int_8 | tinyint(3) | YES  | YES | NULL    |       |
| int_7 | tinyint(3) | YES  | YES | NULL    |       |
+-----+-----+-----+-----+-----+
```

数据显示的时候，zerofill 会在左侧填充 0 到指定位，如果不足 3 位，那么填充到 3 位，如果本身已经够了或者超出，那么就不再填充。

显示长度可以自己设定，超出长度不超出范围不会影响，只会对不够长度（显示长度）的进行补充

```
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| int_9 | tinyint(2) | YES  | YES | NULL    |       |
| int_8 | tinyint(3) | YES  | YES | NULL    |       |
| int_7 | tinyint(3) | YES  | YES | NULL    |       |
| int_6 | tinyint(3) | YES  | YES | NULL    |       |
| int_5 | tinyint(4) | YES  | YES | NULL    |       |
| int_4 | tinyint(4) | YES  | YES | NULL    |       |
| int_3 | smallint(6) | YES  | YES | NULL    |       |
| int_2 | mediumint(9) | YES  | YES | NULL    |       |
| int_1 | int(11)    | YES  | YES | NULL    |       |
| int_0 | bigint(20) | YES  | YES | NULL    |       |
+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)

mysql> insert into my_int values(100,1,1,1,1,1,1,1);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
mysql> insert into my_int values(1,1,1,1,1,1,1,1);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
mysql> insert into my_int values(100,1,1,1,1,1,1,1,1);
Query OK, 1 row affected (0.39 sec)

mysql> insert into my_int values(1,1,1,1,1,1,1,1,1);  
不会改变字段能表示的数据大小  
超出长度没办法，低于长度会自动补充
Query OK, 1 row affected (0.03 sec)

mysql> select * from my_int;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| int_9 | int_8 | int_7 | int_6 | int_5 | int_4 | int_3 | int_2 | int_1 | int_0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| NULL  | NULL  | NULL  | NULL  | -10   | 10000 | 100000 | 1000000 | 1000000000000 | 255   |
| NULL  | NULL  | NULL  | NULL  | 255  | 100   | 255  | 255  | 255  | 255   |
| NULL  | NULL  | NULL  | 001   | 1    | 1    | 1    | 1    | 1    | 1    |
| 100  | 001  | 001  | 001   | 1    | 1    | 1    | 1    | 1    | 1    |
| 01   | 001  | 001  | 001   | 1    | 1    | 1    | 1    | 1    | 1    |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

小数型

专门用来存储小数的。

在 mysql 中将小数类型又分为两类，浮点型和定点型。

浮点型

浮点型又称之为精度类型，是一种有可能丢失精度的数据类型，数据有可能不那么准确（尤其是在超出范围的时候）。

浮点型之所以能够存储较大的数值（不精确），原因就是利用存储数据的位来存储指数。

整型：所有位都为数据位。

浮点型：有部分用于存储数据，部分用于存指数。

float

float 又称之为单精度类型，系统提供 4 个字节用来存储数据，但是能表示的数据范围比整型大的多，大概是 10^{38} ；缺点：只能保证大概 7 个左右的精度（如果数据在 7 位数以内，那么基本是准确的，但是如果超过了 7 位数，那么就是不准确的）

基本语法：

float：表示不指定小数位的浮点数

float (M,D)：表示一共存储 M 个有效数字，其中小数部分占 D 位

float (10,2)：整数部分为 8 位，小数部分为 2 位

```

6 rows in set (0.00 sec)

mysql> create table my_float(
-> f1 float,
-> f2 float(10, 2)
-> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.50 sec)

mysql> desc my_float;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| f1    | float  | YES  |     | NULL    |       |
| f2    | float(10, 2) | YES |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
mysql> insert into my_float values(123.123, 12345678.90);
Query OK, 1 row affected (0.18 sec)

mysql> select * from my_float;
+-----+-----+
| f1  | f2   |
+-----+-----+
| 123.123 | 12345679.00 |
+-----+-----+

```

精度大概在7位左右，
系统进行四舍五入进位

注意：如果数据精度丢失，浮点型是按照四舍五入的方式进行计算。

```

mysql> -- 超出范围
mysql> insert into my_float values(123.1234567, 123456789.00);
ERROR 1264 (22003): Out of range value for column 'f2' at row 1
mysql>                               超出指定位数

ERROR 1264 (22003): Out of range value for column 'f2' at row 1
mysql> insert into my_float values(123.1234567, 99999999.99);
Query OK, 1 row affected (0.42 sec)

mysql> select * from my_float;
+-----+-----+
| f1  | f2   |
+-----+-----+
| 123.123 | 12345679.00 |
| 123.123 | 100000000.00 |
+-----+-----+

```

用户不能直接插入超过指定的整数部分长度，但是如果是系统自动进位导致，系统可以承担

说明：用户不能直接插入数据超过指定的整数部分长度，但是如果是系统自动进位导致，系统可以承担。

浮点数可以采用科学计数法存储数据

```

insert into my_float values(123.123, 10e5);

```

e前面为底数，e后面为指数

double

double 由称之为双精度类型，系统用 8 个字节来存储数据，表示的范围约为 10^{308} 次方，但是精度也只有 15 位左右。

定点数

定点数：能够保证数据精确的小数（小数部分可能不精确，超出长度会四舍五入），整数部分一定精确。

decimal

decimal 定点数：系统自动根据存储的数据来分配存储空间，每大概 9 个数就会分配四个字节来进行存储，同时整数和小数部分是分开的。

decimal (M, D): M 表示总长度，最大值不能超过 65 位；D 表示小数部分长度，最长不能超过 30 位。

时间日期型

date

日期类型：系统使用三个字节来存储数据，对应的格式为：YYYY-mm-dd，能表示的范围是从 1000-01-01 到 9999-12-12，初始值为 0000-00-00。

time

时间类型：能够表示某个指定的时间，但是系统同样是提供 3 个自己来存储，对应的格式为：HH:ii:ss，但是 mysql 中的 time 类型能够表示的时间范围要到得多，能表示从 -838:59:59~838:59:59，在 mysql 中具体的用处是用来描述时间段。

time 类型特殊性：本质是用来表示时间区间，能表示的范围比较大。在进行时间类型录入的时候，可以使用一个简单的日期代替时间，在时间格式之前加一个空格，然后指定一个数字（可以是负数），系统会自动将该数字转化成 天数*24 小时加上后面的时间。

```

5 rows in set (0.00 sec)

mysql> insert into my_date values('1900-01-01','512:12:12','1900-01-01 12:12:12');
Query OK, 1 row affected (0.45 sec)

mysql> insert into my_date values('1900-01-01','5 12:12:12','1900-01-01 12:12:12');
Query OK, 1 row affected (0.06 sec)

mysql> select * from my_date;
+-----+-----+-----+-----+
| d1   | d2   | d3   | d4   |
+-----+-----+-----+-----+
| 2000-01-01 | 12:12:12 | 1900-01-01 12:12:12 | 1990-01-01 12:12:12 |
| 1900-01-01 | 12:12:12 | 1900-01-01 12:12:12 | 1990-01-01 12:12:12 |
| 1900-01-01 | 12:12:12 | 1900-01-01 12:12:12 | 1990-01-01 12:12:12 |
| 1900-01-01 | 512:12:12 | 1900-01-01 12:12:12 | 1990-01-01 12:12:12 |
| 1900-01-01 | 132:12:12 | 1900-01-01 12:12:12 | 1990-01-01 12:12:12 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

datetime

日期时间类型：即时将 date 和 time 合并起来，表示的时间，使用 8 个字节来存储数据，格式为：YYYY-mm-dd HH-ii:ss，能表示的区间 1000-01-01 00:00:00 到 9999-12-12 23:59:59，其可以为 0 值：0000-00-00 00:00:00

timestamp

时间戳类型：mysql 中的时间戳只是表示从格林威治时间开始，但是其格式依然是：
YYYY-mm-dd HH-ii:ss

```

mysql> create table my_date(
    -> d1 date,
    -> d2 time,
    -> d3 datetime,
    -> d4 timestamp,
    -> d5 year
    -> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.78 sec)

mysql> desc my_date;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| d1   | date   | YES  |      | NULL    |       |
| d2   | time   | YES  | 可以为空 | NULL    |       |
| d3   | datetime | YES  |      | NULL    |       |
| d4   | timestamp | YES  |      | NULL    |       |
| d5   | year(4) | YES  |      | NULL    |       |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

year

年类型：占用一个字节来保存，能表示 1900~2155 年，但是 year 有两种数据插入方式：0-99

和四位数的具体年。

year 进行两位数插入的时候，有一个区间划分，零界点为 69 和 70；当输入 69 以下（包括 69），那么系统时间为 20+数字，如果是 70 以上（包括 70），那么系统时间为 19+数字

d1	d2	d3	d4	d5
1900-01-01	12:12:12	1900-01-01 12:12:12	1990-01-01 12:12:12	2069
1900-01-01	12:12:12	1900-01-01 12:12:12	1990-01-01 12:12:12	2020
1900-01-01	12:12:12	1900-01-01 12:12:12	1990-01-01 12:12:12	1970

PHP 中有非常强大的时间日期转换函数：date 将时间戳转换成想要的格式，strtotime 又可以将很多格式转换成对应的时间戳。PHP 通常不需要数据库来帮助处理这么复杂的时间日期，所以通常配合 PHP 的时候，时间的保存通常使用时间戳，从而用整型来保存。

字符串类型

char

定长字符：指定长度之后，系统一定会分配指定的空间用于存储数据

基本语法：char (L)，L 代表字符数（英文字母和中文一样），L 长度为 0 到 255

varchar

变长字符：指定长度后，系统会根据实际存储的数据来计算长度，分配合适的长度（数据没有超出长度）

基本语法：varchar(L)，L 代表字符数，L 的长度理论值为 0 到 65535

因为 varchar 要记录数据长度（系统根据数据长度自动分配长度），所以每两个 varchar 数据产生后，系统都会在数据后面增加 1-2 个字节的额外开销：是用来保存数据所占用的空间长度。如果数据本身小于 127 个字符，额外开销一个字节；如果大于 127 个，就开销两个字节。

char 和 varchar 数据存储对比（utf8，一个字符都回占用 3 个字节）：

存储数据	Char (2)	Varchar (2)	Char 所占字节	Varchar 所占字节
A	A	A	2*3=6	1*3+1 =4
AB	AB	AB	2*3=6	2*3+1=7

char 和 varchar 的 区别：

1、char 一定会使用指定的空间，varchar 是根据数据来定空间

2、char 的数据查询效率比 varchar 高：varchar 是需要通过后面的记录数来计算
如果确定数据一定是占指定长度，那么适应 char 类型。

如果不确定数据到底占多少，那么使用 varchar 类型

如果数据长度超过 255 个字符，不论是否固定长度，都回使用 text，不再使用 char 和 varchar

text

文本类型：本质上 mysql 提供了两种文本类型

text: 存储普通的字符文本

blob: 存储二进制文本（图片，文件），一般不会使用 blob 来存储文件本身，通常是使用一个链接来指向对应的文件本身。

text: 系统中提供四种 text

tinytext: 系统使用一个字节来保存，实际能够存储的数据为： $2^8 + 1$ 字符数

text: 使用两个字节保存，实际存储为： $2^{16} + 2$ 字符数

mediumtext: 使用三个字节保存，实际存储为： $2^{24} + 3$ 字符数

longtext: 使用四个字节保存，实际存储为： $2^{32} + 4$

注意：

1、在选择对应的存储文本的时候，不用刻意去选择 text 类型，系统会自动根据存储的数据长度来选择合适的文本类型

2、在选择字符存储的时候，如果数据超过 255 个字符，那么一定选择 text 存储

enum

枚举类型：在数据插入之前，先设定几个项，这几个项就是可能最终出现的数据结果。

如果确定某个字段的数据只有那么几个值：如性别，男、女、保密，系统就可以在设定字段的时候规定当前字段只能存放固定的几个值：使用枚举

基本语法：enum（数据值 1，数据值 2...）

系统提供了 1-2 个字节来存储枚举数据：通过计算 enum 列举的具体值来选择时机的存储空间；如果数据值列表在 255 个以内，那么一个字节就够了，如果超过 255 但是小于 65535，那么系统采用两个字节保存。

```

Query OK, 0 rows affected (0.51 sec)
mysql> set names gbk;           ← 设置了字符集
Query OK, 0 rows affected (0.00 sec)

mysql> create table my_enum(
    -> gender enum('男','女','保密')
    -> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.37 sec)

mysql> desc my_enum;
+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| gender | enum('男', '女', '保密') | YES | NULL |          |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> insert into my_enum values('男');           ← 部分乱码部分显示正常？？？
Query OK, 1 row affected, 1 warning (0.17 sec)

mysql> insert into my_enum values('女');           ← 插入枚举类型列表中指定的数据，其他不行。
Query OK, 1 row affected, 1 warning (0.04 sec)

mysql> insert into my_enum values('保密');
Query OK, 1 row affected (0.12 sec)

mysql> select * from my_enum;
+-----+
| gender |
+-----+
| 男   |
| 女   |
| 保密 |
+-----+
3 rows in set (0.00 sec)

```

枚举 enum 的存储原理：实际上字段上所存储的值并不是真正的字符串，而是字符串对应的下标；当系统设定枚举类型的时候，会给枚举中每个元素定义一个下标，这个下标规则从 1 开始

```
enum (1=>'男',2=> '女', 3=> '保密');
```

特性：在 mysql 中系统是自动进行类型转换的，如果数据用到到 “+、-、*、/” 系统就会自动将数据转换为数值，而普通字符串转换成数值为 0

```
select 字段名 + 0 from 表名;
```

```

mysql> select * from my_enum;
+-----+
| gender |
+-----+
| 男   |
| 女   |
| 保密 |
+-----+
3 rows in set (0.00 sec)

mysql> -- 将字段按照数值输出
mysql> select gender + 0 from my_enum;
+-----+
| gender + 0 |
+-----+
| 1           |
| 2           |
| 3           |
+-----+
3 rows in set (0.36 sec)

```

既然实际 enum 字段存储的结果是数值，那么在进行数据插入的时候，就可以使用对用的数值来进行

mysql> insert into my_enum values(3);
Query OK, 1 row affected (0.38 sec)

mysql> select * from my_enum;

gender
保密
保密

The screenshot shows the MySQL command-line interface. A red box highlights the value '3' in the first 'values' statement. Another red box highlights the second '保密' entry in the 'gender' column of the result table. A red arrow points from the highlighted '3' to the highlighted '保密' entry.

枚举的意义：

- 1、规范数据本身，限定只能插入规定的数据项
- 2、节省存储空间

set

集合：是一种将多个数据选项可以同时保存的数据类型，本质是将指定的项按照对应的二进制位来进行控制，1 表示该选项被选中，0 表示该选项没有被选中。

基本语法：set('值 1', '值 2'...);

系统为 set 提供了多个字节进行保存，但是系统会自动计算来选择具体的存储单元

1 个字节：set 中只能有 8 个选项

2 个字节：set 中只能有 16 个选项

3 个字节：set 中只能有 24 个选项

8 个字节：set 中只能有 64 个选项（最多）

set 和 enum 一样，最终存储大数据字段中的依然是数字而不是真实的字符串，可以数字插入数据。

插入数据：可以插入多个数据，使用逗号进行分割。

mysql> insert into my_set values('篮球,乒乓球');
Query OK, 1 row affected, 1 warning (0.40 sec)

The screenshot shows the MySQL command-line interface. A red box highlights the inserted value '篮球,乒乓球' in the 'values' statement. A red arrow points from this box to the same value in the result table below.

数据选项所在的顺序与数据插入顺序无关，最终都会变成选项对应的顺序。

系统按照数据选项顺序编排，从第一个开始进行占位，每一个对应一个二进制位。数据存储时，被选中的项对应的位值为 1，否则为 0，系统在进行存储的时候会自动将得到的最终的二进制颠倒过来，然后再进行转换为 10 进制（系统要补足八个，如果不够八位，则数据就会变得很大）

和 enum 类似，也可以进行数字插入。

注意：数字插入的前提是对应的二进制位上都有对应的数据项

set 的意义：

- 1、规范数据
- 2、节省存储空间

enum 和 set 的对比：

enum：单选框

set：复选框

MySQL 记录长度

在 mysql 中，mysql 的记录长度（record = 行 row）总长度不能超过 65535 个字节

varchar 能够存储的理论值为 65535 个字符：字符在不同的字符集下可能占用多个字节。

Varchar 能够存储的长度：（varchar 有 1~2 个放置长度的字节）

utf8 最多只能存储 21844 个字符（ $65535/3 - 1$ ）

gbk 最多只能存储 32766 个字符（ $65535/2 - 1$ ）

列属性

列属性又称之为字段属性，在 mysql 中一共有 6 个属性：null，默认值，列描述，主键，唯一键和自动增长。

null 属性

NULL 属性代表字段为空

Field	Type	Null	Key
name	varchar(10)	YES	
age	int(11)	YES	

如果对应的值为 YES，表示该字段可以为 NULL。

注意：

- 1、在设计表的时候，尽量不要让数据为空。
- 2、MySQL 的记录长度为 65535 字节，如果有一个表中有字段允许为 NULL，那么系统就会设计保留一个字节来存储 NULL，最终有效存储长度为 65534 个字节。

默认值

default：默认值，当字段被设计的时候，如果允许默认条件下，用户不进行数据的插入，那么就可以使用事先准备好的数据来填充：通常填充的是 NULL。

```
-- 创建数据表
create table my_default(
    name varchar(10) not null,          不能为空
    age int default 18                  默认值为18
)charset utf8;
```

mysql> insert into my_default(name) values('Tom');	values('Tom')
Query OK, 1 row affected (0.48 sec)	
mysql> select * from my_default;	
+-----+-----+	
name age	
+-----+-----+	
Tom 18	默认值
+-----+-----+	
1 row in set (0.00 sec)	

default 关键字的另一层使用：显示告知字段使用默认值：在进行数据插入的时候，对字段值直接使用 default。

```

mysql> insert into my_default values('Jack', default);
Query OK, 1 row affected (0.49 sec)

mysql> select * from my_default;
+-----+-----+
| name | age |
+-----+-----+
| Tom  | 18  |
| Jack | 18  |
+-----+-----+

```

列描述

列描述: comment, 专门用于给开发人员进行维护的一个注释说明。

基本语法: comment ‘字段描述’ ;

```

mysql> -- 创建表, 增加字段描述
mysql> create table my_comment(
    -> name varchar(10) not null comment '用户名不能为空',
    -> pass varchar(50) not null comment '密码不能为空',
    -> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.66 sec)

```

查看 comment: 必须通过表创建语句来查看。

```

mysql> show create table my_comment;
+-----+-----+
| Table      | Create Table
+-----+-----+
| my_comment | CREATE TABLE `my_comment` (
  `name` varchar(10) NOT NULL COMMENT '用户名不能为空',
  `pass` varchar(50) NOT NULL COMMENT '密码不能为空',
) ENGINE=InnoDB DEFAULT CHARSET=utf8
+-----+-----+

```

主键

顾名思义, 主要的键, primary key, 在一张表中, 有且只有一个字段, 里面的值具有唯一性, 同时, 系统默认为主键添加 index。可以用 constraint 来设置索引名字。

```
constraint t_user_id_pk primary key(id)
```

创建主键

随表创建

系统提供了两种增加主键的方式:

- 1、直接在需要当做主键的字段之后, 增加 primary key 属性来确定主键
- 2、在所有字段之后增加 primary key 选项: primary key (字段信息)

```

+-----+-----+-----+
-- 在字段后增加主键属性
create table my_primary(
username varchar(10) primary key
)charset utf8;

create table my_primary2(
username varchar(10),
primary key(username)
)charset utf8;

```

表后增加

基本语法: alter table 表名 add primary key(字段);

```

mysql> create table my_primary3(
    -> username varchar(10)
    -> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.36 sec)

mysql> -- 增加主键
mysql> alter table my_primary3 add primary key(username);
Query OK, 0 rows affected (0.63 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> desc my_primary3;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| username | varchar(10) | NO   | PRI  | NULL    |       |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

查看主键

方案 1: 查看表结构

```

mysql> desc my_primary3;
+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra |
+-----+-----+-----+-----+
| username | varchar(10) | NO   | PRI  | NULL    |       |
+-----+-----+-----+-----+

```

方案 2: 查看表的创建语句

```

mysql> show create table my_primary3;
+-----+-----+
| Table      | Create Table          | 后面加的主键系统也添加到
+-----+-----+-----+-----+
| my_primary3 | CREATE TABLE `my_primary3` (
    username varchar(10) NOT NULL,
    PRIMARY KEY (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+-----+

```

删除主键

基本语法: alter table 表名 drop primary key;

```
mysql> -- 删除主键
mysql> alter table my_priamry3 drop primary key;
Query OK, 0 rows affected (0.57 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc my_priamry3;
ERROR 1146 (42S02): Table 'mydatabase.my_priamry3' doesn't exist
mysql> desc my_priamry3;
ERROR 1146 (42S02): Table 'mydatabase.my_priamry3' doesn't exist
mysql> desc my_primary3;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| username | varchar(10) | NO   |       | NULL    |       |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

保留

复合主键

```
mysql> create table my_score(
    -> student_no char(10),
    -> course_no char(10),
    -> score tinyint not null,
    -> primary key(student_no, course_no)      复合主键
    ->)charset utf8;
Query OK, 0 rows affected, 1 warning (0.28 sec)

mysql> desc my_score;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| student_no | char(10) | NO   | PRI  | NULL    |       |
| course_no  | char(10) | NO   | PRI  | NULL    |       |
| score       | tinyint(4) | NO   |       | NULL    |       |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

主键约束

主键一旦增加，那么对对应的字段数据有要求:

- 1、当前字段对应的数据不能为空。
- 2、当前字段对应的数据不能有任何重复

```

3 rows in set (0.00 sec)

mysql> insert into my_score values('0000001','course001',100);
Query OK, 1 row affected (0.08 sec)

mysql> insert into my_score values('0000002','course001',90);
Query OK, 1 row affected (0.03 sec)

mysql> insert into my_score values('0000001','course002',95);
Query OK, 1 row affected (0.05 sec)

mysql> select * from my_score;
+-----+-----+-----+
| student_no | course_no | score |
+-----+-----+-----+
| 0000001    | course001 | 100   |
| 0000001    | course002 | 95    |
| 0000002    | course001 | 90    |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> insert into my_score values('0000002','course001',98);
ERROR 1062 (23000): Duplicate entry '0000002-course001' for key 'PRIMARY'
mysql>

```

主键约束，不能重复

主键分类

主键分类采用的是主键所对应的字段的业务意义分类：

业务主键：主键所在的字段，具有业务意义（学生 ID，课程 ID）

逻辑主键：自然增长的整型（应用广泛）

自增长

自动增长：auto increment，当给定某个字段的属性之后，该列的数据在没有提供确定数据的时候，系统会根据之前已经存在的数据进行自动增加后，填充数据

通常自动增长用于逻辑主键

原理

自动增长的原理：

- 1、在系统中有维护一组数据，用来保存当前使用了自动增长属性的字段，记住当前对应的数据值，再给定一个指定的步长。
- 2、当用户进行数据插入的时候，如果没有给定值，系统在原始值上加上步长变成新的数据
- 3、自动增长的触发：给定属性的字段没有提供值
- 4、自动增长只适用于数值
- 5、**自增的字段必须是主键**

```

→ )charset utf8;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key

```

使用自动增长

基本语法：在字段之后增加一个属性 auto_increment;

```
mysql> create table my_auto(
    -> id int primary key auto_increment,
    -> name varchar(10) not null comment '用户名',
    -> pass varchar(50) not null comment '密码',
    ->)charset utf8;
Query OK, 0 rows affected, 1 warning (0.78 sec)

mysql> desc my_auto;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra       |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI   | NULL    | auto_increment |
| name  | varchar(10)| NO  |        | NULL    |             |
| pass  | varchar(50)| NO  |        | NULL    |             |
+-----+-----+-----+-----+-----+
```

插入数据：触发自动增长，不能给定具体值（可以给 null）

```
mysql> insert into my_auto values(null, 'Tom', '12346');
Query OK, 1 row affected (0.20 sec)

mysql> select * from my_auto;
+----+----+----+
| id | name | pass |
+----+----+----+
| 1  | Tom  | 12346 |
+----+----+----+
1 row in set (0.00 sec)
```

修改自动增长

1、查看自增长：自增长一旦触发使用后，会自动地在表选项中增加一个选项（一张表最多只能拥有一个自增长）

```
+-----+
| my_auto | CREATE TABLE `my_auto` (
|   id` int(11) NOT NULL AUTO_INCREMENT,
|   name` varchar(10) NOT NULL COMMENT '用户名',
|   pass` varchar(50) NOT NULL COMMENT '密码',
|   PRIMARY KEY (`id`)
| ) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8
+-----+
```

2、表选项可以通过修改表结构来实现

基本语法： alter table 表名 auto_increment = 值；

```
mysql> -- 修改auto_increment
mysql> alter table my_auto auto_increment = 10;
Query OK, 0 rows affected (0.22 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> show create table my_auto;
+-----+
| Table | Create Table
+-----+
| my_auto | CREATE TABLE `my_auto` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) NOT NULL COMMENT '用户名',
  `pass` varchar(50) NOT NULL COMMENT '密码',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8
+-----+
```

2变为了10

删除和增加自动增长

删除自增长: 就是在字段属性之后不再保留 auto_increment, 当用户修改自增长所在字段时, 如果没有看到 auto_increment 属性, 系统会自动清除该自增长。

删除自增长: alter table my_auto modify id int;

增加自增长: alter table my_auto modify id int auto_increment;

```
mysql> -- 删除自增长
mysql> alter table my_auto modify id int;
Query OK, 1 row affected (0.90 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> -- 切记不要再次增加primary key
mysql> show create table my_auto;
+-----+
| Table | Create Table
+-----+
| my_auto | CREATE TABLE `my_auto` (
  `id` int(11) NOT NULL,
  `name` varchar(10) NOT NULL COMMENT '用户名',
  `pass` varchar(50) NOT NULL COMMENT '密码',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
+-----+
```

自增长属性被干掉

初始设置

在系统中, 有一组变量用来维护自增长的初始值和步长

查看: show variables like 'auto_increment%';

```
mysql> -- 查看自增长初始变量
mysql> show variables like 'auto_increment%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1      |
| auto_increment_offset    | 1      |
+-----+-----+
2 rows in set, 1 warning (0.00 sec)

mysql>
```

步长
初始值

修改自增长步长和初始值:

```
set auto_increment_increment = 值;
set auto_increment_offset = 值;
```

```
mysql> set auto_increment_increment = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'auto_increment%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 2      |
| auto_increment_offset    | 1      |
+-----+-----+
2 rows in set, 1 warning (0.00 sec)
```

可以进行修改

细节问题

- 1、一张表中只有一个自增长,: 自增长会上升到表选项中。
- 2、如果数据插入中没有触发自增长(给定了数据),那么自增长不会表现,用户指定数据之后,自增长不参与,但是自增长默默根据当前用户设定的值初始化下一个值。
- 3、自增长在修改的时候,值可以较大,但是不能比当前已有的自增长字段的值小。

唯一键

唯一键; unique key, 用来保证对应的字段中的数据唯一的。

主键也可以用保证字段数据唯一性,但是一张表只有一个主键。

唯一键特点:

- 1、唯一键在一张表中可以有多个。
- 2、唯一键允许字段数据为NULL, NULL 可以有多个 (NULL 不参与比较)

创建唯一键

创建唯一键和创建主键非常类似

- 1、直接在表字段之后增加唯一键标识符: unique[key]
- 2、在所有的字段之后使用 unique key (字段列表) ;
- 3、在创建完表之后也可以用增加唯一键
alter table 表名 add unique key (字段列表);

```
-- 唯一键

-- 方案一
create table my_unique1(
    id int primary key auto_increment,
    username varchar(10) unique
)charset utf8;

-- 方案二
create table my_unique2(
    id int primary key auto_increment,
    username varchar(10),
    unique key(username)
)charset utf8;

-- 方案三
create table my_unique3(
    id int primary key auto_increment,
    username varchar(10)
)charset utf8;

alter table my_unique3 add unique key(username);
```

创建唯一键的
三种方法

查看唯一键

唯一键效果：在不为空的情况下，不允许重复。

- 1、唯一键是属性，可以通过查看表结构来实现。

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
username	varchar(10)	YES	UNI	NULL	允许为空 标识符

- 2、查看表创建语句。

在查看表创建语句的时候，会看到与主键不同的一点，多出一个“名字”

```

mysql> show create table my_unique1;
+-----+-----+
| Table | Create Table |
+-----+-----+
| my_unique1 | CREATE TABLE `my_unique1`(
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+-----+

```

系统会为唯一键自动
创建一个名字（默认
是字段名）

删除唯一键

一个表中允许存在多个唯一键：

删除基本语法： alter table 表名 drop index 唯一键名字；
index 代表索引，唯一键是索引的一种（提升查询效率）

```

mysql> desc my_unique2;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI | NULL    | auto_increment |
| username | varchar(10) | YES  | UNI | NULL    |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> alter table my_unique2 drop index username;
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0          删除成功

mysql> desc my_unique2;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI | NULL    | auto_increment |
| username | varchar(10) | YES  |      | NULL    |                |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

修改唯一键

先删除后增加。

创建唯一索引

在创建表之后使用 CREATE 命令来创建

```

mysql> CREATE UNIQUE INDEX catename ON wb_blog(catid);
Query OK, 0 rows affected (0.47 sec)

```

如果不需唯一索引，则可以这样删除

```
mysql> ALTER TABLE wb_blog DROP INDEX catename;
Query OK, 0 rows affected (0.85 sec)
```

复合唯一键

唯一键与主键一样，可以使用多个字段来共同保存唯一性。

一般主键都是单一字段（逻辑字段），而其它需要唯一性的内容都是由唯一键来处理。

表关系

表关系：表与表之间（实体）有什么样的关系，每种关系应该如何设计表结构。

一对一

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关系，通常，此设计方案保证两张表中使用同样的主键即可。（唯一键可以为 null，无法匹配）

学生表：

学生 ID (PRI)	姓名	年龄	性别	籍贯	婚否	信仰

表的使用过程中，常用的信息会经常去查询，而不常用的信息会偶尔才会用到。

解决方案：将表拆分，常见的放一张表，不常见的放一张表。

常用表

学生 ID (PRI)	姓名	年龄	性别

不常用表

学生 ID (PRI)	籍贯	婚否	信仰

一对多

一对多，通常也叫作多对一的关系，通常一对多的关系设计的方案，在“多”关系的表中去维护一个字段，这个字段是“一”关系的主键。

母亲表

母亲 ID	姓名	年龄	身高
M1			
M2			

在孩子中加入一列母亲 ID（增加字段）

孩子表

孩子 ID	姓名	年龄	身高	母亲 ID
K1				M1
K2				M2

多对多

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中：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。

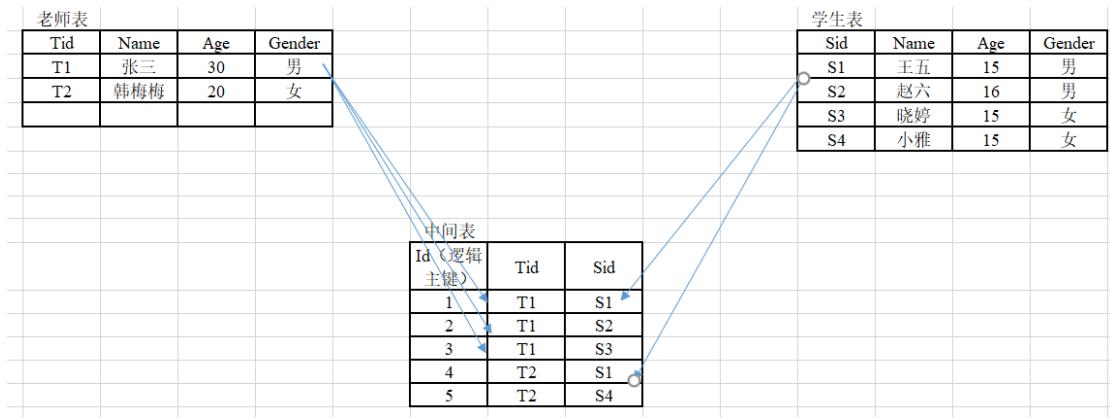
既然通过两张表自己增加字段解决不了问题，那么通过第三张表来解决。

师生关系

- 1、一个老师教过多个班级的 学生
- 2、一个学生听过多个老师讲的课

首先得有两个实体：老师表和学生表

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。

高级数据操作

新增数据

多数据插入

只要写一次 insert 指令，但是可以直接插入多条记录。

基本语法：

1) `insert into 表名 [(字段列表)] values(值列表), (值列表)...;`

```
-- 插入两条记录
insert into my_teacher values('王五',28),('赵四',35);
```

2) `insert into 表名 [(字段列表)] 子查询;`

```
insert into dept_age(Sdept,Avg_age) select Sdept,AVG(Sage) from student group by Sdept;
```

主键冲突

主键冲突：在有的表中，使用的是业务主键（字段有业务含义），但是往往在进行数据插入的时候，又不能确定数据表中是否已经存在对应的主键。

```
tmysql> select * from my_stu;
+-----+-----+
| stu_id | stu_name |
+-----+-----+
| stu001 | 张三   |
| stu002 | 张四   |
| stu003 | 张五   |
| stu004 | 张六   |
+-----+-----+
4 rows in set (0.00 sec)

mysql> insert into my_stu values('stu004','晓婷');
ERROR 1062 (23000): Duplicate entry 'stu004' for key 'PRIMARY'
mysql>
```

主键
主键冲突

主键冲突的解决方案：

1、主键冲突更新：类似插入数据语法，如果插入的过程中主键冲突，可以采用更新方法。

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

```
mysql> insert into my_stu values('stu004','晓婷') on duplicate key update stu_name = '晓婷';
Query OK, 2 rows affected (0.44 sec)

mysql> select * from my_stu;
+-----+-----+
| stu_id | stu_name |
+-----+-----+
| stu001 | 张三   |
| stu002 | 张四   |
| stu003 | 张五   |
| stu004 | 晓婷   |
+-----+-----+
4 rows in set (0.00 sec)
```

2、主键冲突替换：

当主键冲突之后，干掉原来的数据，重新插入进入。

replace into 表名 [(字段列表)] values(值列表);

```
mysql> replace into my_stu values('stu001','夏洛');
Query OK, 2 rows affected (0.42 sec)

mysql> select * from my_stu;
+-----+-----+
| stu_id | stu_name |
+-----+-----+
| stu001 | 夏洛   |
| stu002 | 张四   |
| stu003 | 张五   |
| stu004 | 晓婷   |
+-----+-----+
4 rows in set (0.00 sec)
```

蠕虫赋值

蠕虫赋值：一分为二，成倍的增加，从已有的数据中获取数据，并且将获取到的数据插入到

数据表中。

基本语法: insert into 表名 [(字段列表)] select */字段列表 from 其他表;

```
+-----+  
| 4 rows in set (0.00 sec)|  
+-----+  
- mysql> -- 蠕虫复制  
- mysql> insert into my_simple(name) select name from my_simple;  
- Query OK, 4 rows affected (0.48 sec)  
- Records: 4  Duplicates: 0  Warnings: 0  
+-----+  
- mysql> ^C  
- mysql>  
- mysql> -- 蠕虫复制  
- mysql> insert into my_simple(name) select name from my_simple;  
- Query OK, 8 rows affected (0.15 sec)  
- Records: 8  Duplicates: 0  Warnings: 0  
+-----+  
- mysql> select * from my_simple;
```

注意:

- 1、蠕虫复制的确通常是重复数据，没有太大业务意义：可以在短期内快速增加表的数据量，从而测试表的压力，还可以通过大量数据来测试表的效率（索引）
- 2、蠕虫复制虽好，但要注意主键冲突。

更新数据

1、在更新数据的时候，特别要注意，通常是跟随条件更新，而不是批量更新。

update 表名 set 字段名 = 新值 where 判断条件;

2、如果没有条件，是全表更新数据，但是可以使用 limit 来显示更新的数量；

update 表名 set 字段名 = 新值 [where 判断条件] limit 数量;

```
+-----+  
- mysql> -- 更新数据  
- mysql> update my_simple set name = 'e' where name = 'a' limit 2;  
- Query OK, 2 rows affected (0.43 sec)
```

删除数据

1、删除数据的时候尽量不要全部删除，应该使用 where 进行判定

2、删除数据的时候可以使用 limit 来限制要删除的具体数量。

delete 删除数据的时候无法重置 auto_increment

```

10 rows in set (0.00 sec)

mysql> delete from my_auto;
Query OK, 1 row affected (0.49 sec)

mysql> insert into my_auto values(null, 'tom', '123456');
Query OK, 1 row affected (0.39 sec)

mysql> select * from my_auto;
+----+-----+-----+
| id | name | pass |
+----+-----+-----+
| 2  | tom  | 123456 |
+----+-----+-----+
1 row in set (0.00 sec)

```

Mysql 有一个能够重置表选项中的自增长的语法:

truncate 表名; ==> drop ==> create

```

mysql> truncate my_auto;
Query OK, 0 rows affected (0.81 sec)

mysql> show create table my_auto;
+-----+
| Table      | Create Table
+-----+
| my_auto    | CREATE TABLE `my_auto` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) NOT NULL COMMENT '用户名',
  `pass` varchar(50) NOT NULL COMMENT '密码',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+
1 row in set (0.00 sec)

```

插入的时候就会
从1开始

查询数据

完整的查询指令:

select select 选项 字段列表 from 数据源 where 条件 group by 分组 having 条件 order by 排序 limit 限制

select 选项

系统该如何对待查询得到的结果:

1、all 默认的，表示保存所有的记录；

```

mysql> -- select 选项
mysql> select all * from my_simple;
+-----+
| name |
+-----+
| e    |
| b    |
| c    |
| d    |
| e    |
| b    |
| c    |
| d    |
| a    |
| b    |
| c    |
| d    |
| a    |
| b    |
| c    |
| d    |
+-----+
16 rows in set (0.00 sec)

```

2、distinct: 去重，去除重复的记录，只保留一条（所有的字段都相同）

```

ERROR 1140 (42S02): Table 'mydatabase.my_simple' u
mysql> select distinct * from my_simple;
+-----+
| name |
+-----+
| e    |
| b    |
| c    |
| d    |
| a    |
+-----+
5 rows in set (0.00 sec)

```

字段列表

有的时候需要从多张表获取数据，在获取数据的时候，可能存在不同表中有同名的字段，需要将同名的字段命名成不同的名； alias 别名

基本语法： 字段名 [as] 别名；

```

mysql> -- 字段别名
mysql> select distinct name as name1 ,name names from my_simple;
+-----+-----+
| name1 | names |
+-----+-----+
| e     | e      |
| b     | b      |
| c     | c      |
| d     | d      |
| a     | a      |
+-----+-----+
同一张表中字段名可以提取多次
5 rows in set (0.00 sec)

```

from 数据源

from 是为前面的查询提供数据，数据源只要是一个符合二维表结构的数据即可。

单表数据

from 表名

```
mysql> -- 字段别名
mysql> select distinct name as name1 ,name names from my_simple;
+-----+-----+
```

多表数据

从多张表获取数据：

基本语法：from 表 1, 表 2...;

结果：表的记录数相乘，字段数拼接

本质：从第一张表取出一条记录，去拼凑第二张表的所有记录，保留所有结果。得到的结果在数学上有一个专业的说法：笛卡尔积，这个结果除了给数据库造成压力，没有其他意义，应该尽量避免笛卡尔积。

动态数据

from 后面跟的数据不是一个实体表，而是一个从表中查询出来得到的二维结果表。

基本语法：select from (select 字段列表 from 表) [as] 别名;

```
0 rows in set (0.00 sec)
mysql> select * from (select int_1,int_2 from my_int) as int_my;
+-----+-----+
| int_1 | int_2 |
+-----+-----+
|    10 | 10000 |
| -128 |   255 |
|  100 |   255 |
|    1 |     1 |
|    1 |     1 |
|    1 |     1 |
+-----+-----+
6 rows in set (0.00 sec)
```

不是表，是一个
二维结果表

一定要有别名

where 子句

where 子句用来从数据表获取数据的时候，然后进行条件筛选。

数据获取原理：针对表去对应的磁盘除获取所有的记录（一条条），`where` 的作用就是在拿到一条结果就开始进行判断，判断是否符合条件；如果符合就保存下来，如果不符直接舍弃（不放到内存中）

`where` 是通过运算符进行结果比较来判断数据

group by 子句

`group by` 表示分组的含义：根据指定的字段，将数据进行分组：分组的目标是为了统计。

分组统计

基本语法：`group by` 字段名；

错误：提示 1055 错误。`only_full_group_by` 问题

https://blog.csdn.net/feng_idea/article/details/79957611

先查看，然后修改。（暂时修改）

执行：

```
select version(), @@sql_mode;
```

再执行：

```
SET sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY','));
```

[Err] 1055 - Expression #1 of ORDER BY clause is not in GROUP BY clause and contains nonaggregated column 'information_schema.PROFILING.SEQ' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by

原因分析：

only_full_group_by：使用这个就是使用和oracle一样的group 规则, select的列都要在group中,或者本身是聚合列(SUM,AVG,MAX,MIN)才行。5.6版本中没有这约束。5.7.21有。去掉就可以了。

解决方式：

linux登录mysql

mysql -uroot -p

输入密码

执行：select version(), @@sql_mode;

执行完这两句后再次执行你要执行的代码即可。

在执行：

SET sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY',''));

把ONLY_FULL_GROUP_BY去掉。

退出重启mysql服务：

service mysqld restart

不用退出重启，重启后又恢复成默认

```
mysql> -- 分组
mysql> select * from my_stu group by class_id;
+-----+-----+-----+
| stu_id | stu_name | class_id |
+-----+-----+-----+
| stu001 | 夏洛    |      2 |
| stu003 | 张五    |      1 |
+-----+-----+-----+
2 rows in set (0.01 sec)

mysql> select * from my_stu;
+-----+-----+-----+
| stu_id | stu_name | class_id |
+-----+-----+-----+
| stu001 | 夏洛    |      2 |
| stu002 | 张四    |      2 |
| stu003 | 张五    |      1 |
| stu004 | 晓婷    |      1 |
+-----+-----+-----+
```

group by 是为了分组进行数据统计的，如果只是想看数据显示，那么 group by 没什么意义：
group by 将数据按照指定的字段分组之后，只会保留每组的第一条记录。

利用一些统计函数（聚合函数）：

count():统计每组中的数量，如果统计的目标是字段，那么不统计空 NULL 字段，如果为* 代表统计记录。

avg():求平均值

sum():求和

max():求最大值

min():求最小值

```

mysql> select * from my_stu;
+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height |
+-----+-----+-----+-----+-----+
| stu001 | 夏洛 | 2 | 18 | 185 |
| stu002 | 张四 | 2 | 28 | 175 |
| stu003 | 张五 | 1 | 20 | 165 |
| stu004 | 晓婷 | 1 | 25 | 195 |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> -- 使用聚合函数 按照班级统计没班人数 最大年龄 最矮身高 平均年龄
mysql> select class_id, count(*), max(stu_age), min(stu_height), avg(stu_age) from my_stu group by class_id;
+-----+-----+-----+-----+-----+
| class_id | count(*) | max(stu_age) | min(stu_height) | avg(stu_age) |
+-----+-----+-----+-----+-----+
| 2 | 2 | 28 | 175 | 23.0000 |
| 1 | 2 | 25 | 165 | 22.5000 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

group_concat():为了将分组中指定的字段进行合并（字符串拼接）

```

mysql> select class_id, group_concat(stu_name), count(*), max(stu_age), min(stu_height), avg(stu_age) from my_stu group by class_id;
+-----+-----+-----+-----+-----+-----+
| class_id | group_concat(stu_name) | count(*) | max(stu_age) | min(stu_height) | avg(stu_age) |
+-----+-----+-----+-----+-----+-----+
| 1 | 张五,晓婷 | 2 | 25 | 165 | 22.5000 |
| 2 | 夏洛,张四 | 2 | 28 | 175 | 23.0000 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

多分组

将数据按照某个字段进行分组之后，对已经分组的数据进行再次分组

基本语法：group by 字段 1, 字段 2; //先按照字段 1 进行排序，之后将结果再按照字段 2 进行排序，以此类推

分组排序

Mysql 中，分组默认有排序的功能：按照分组字段进行排序，默认是升序

基本语法：group by 字段 [asc|desc],字段[asc|desc]; //默认是 asc 升序 desc 降序

回溯统计

当分组进行多分组之后，往上统计的过程中，需要进行层层上报，将这种层层上报统计的过程称之为回溯统计，每一次分组向上统计的过程都会产生一次新的统计数据，而且当前数据对应的分组字段为 NULL

基本语法：group by 字段 [asc|desc] with rollup;

```

mysql> select class_id,count(*) from my_stu group by class_id with rollup;
+-----+-----+
| class_id | count(*) |
+-----+-----+
|       1  |      3  |
|       2  |      2  |
|    NULL  |      5  |
+-----+-----+
3 rows in set (0.00 sec)

mysql> select class_id,count(*) from my_stu group by class_id asc;
+-----+-----+
| class_id | count(*) |
+-----+-----+
|       1  |      3  |
|       2  |      2  |
+-----+-----+
2 rows in set (0.00 sec)

```

回溯

有多少层就会回溯多少个。

having 子句

having 本质和 where 一样，是用来进行数据条件筛选

having 是在 group by 子句之后，可以针对分组数据进行筛选，但是 where 不行

where 不能使用聚合函数：聚合函数是在用在 group by 分组的时候，where 这个时候已经运行完毕。

having 在 group by 分组之后，可以使用聚合函数或者字段别名（where 是从表中取出数据，数据在表中只有字段名没有别名这一概念，别名是在数据进入到内存之后才有的）

强调：having 是在 group by 之后，groupby 是在 where 之后；where 的时候表示将数据从磁盘拿到内存，where 之后的所有操作都是内存操作。

order by 子句

order by 排序，根据校对规则对数据进行排序

基本语法：order by 字段[asc|desc]; // asc 升序 默认 desc 降序

```

mysql> select * from my_stu order by stu_height asc;
+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height |
+-----+-----+-----+-----+-----+
| stu003 | 张五     |     1  |    20  |     165  |
| stu005 | 小猪     |     1  |    30  |     172  |
| stu002 | 张四     |     2  |    28  |     175  |
| stu001 | 夏洛     |     2  |    18  |     185  |
| stu004 | 晓婷     |     1  |    25  |     195  |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

```

order by 和 group by 一样，也可以进行多字段排序：先按照第一个字段进行排序，再按照第二个字段进行排序，以此类推。

基本语法：order by 字段 1 规则，字段 2 规则...；//规则即 desc 和 asc ， asc 可以不写

limit 子句

limit 限制子句，主要是用来限制记录数来获取。

记录数限制

纯粹的限制获取的数量：从第一条到指定的数量。

基本语法：limit 数量；

limit 通常在查询的时候如果限定为一条记录的时候，使用的比较多，有时候获取多条记录并不能解决业务问题，但是会增加服务器压力

分页

利用 limit 来限制获取指定区间的数据。

基本语法: limit offset ,length; //offset P 偏移量，从哪开始，length 就是具体获取多少条记录

Mysql 中记录的数量从 0 开始。

limit 0,2; //获取前两条记录

注意：limit 后面的 length 表示最多获取对应数量，但是如果数量不够，系统不会强求

```
2  
3 select * from stu limit 0,1\G  
4 |  
5 select * from stu limit 1 offset 2\G
```

从第0个开始，选择一个
跳过前两个，选择一个

聚合函数

利用一些统计函数（聚合函数）：

count():统计每组中的数量，如果统计的目标是字段，那么不统计空 NULL 字段，如果为* 代表统计记录。

avg():求平均值

sum():求和
max():求最大值
min():求在最小值

```
mysql> select * from my_stu;
+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height |
+-----+-----+-----+-----+-----+
| stu001 | 夏洛 | 2 | 18 | 185 |
| stu002 | 张四 | 2 | 28 | 175 |
| stu003 | 张五 | 1 | 20 | 165 |
| stu004 | 晓婷 | 1 | 25 | 195 |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> -- 使用聚合函数 按照班级统计没班人数 最大年龄 最矮身高 平均年龄
mysql> select class_id, count(*), max(stu_age), min(stu_height), avg(stu_age) from my_stu group by class_id;
+-----+-----+-----+-----+-----+
| class_id | count(*) | max(stu_age) | min(stu_height) | avg(stu_age) |
+-----+-----+-----+-----+-----+
| 1 | 2 | 28 | 165 | 22.5000 |
| 2 | 2 | 25 | 175 | 23.0000 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

group_concat():为了将分组中指定的字段进行合并（字符串拼接）

```
mysql> select class_id, group_concat(stu_name), count(*), max(stu_age), min(stu_height), avg(stu_age) from my_stu group by class_id;
+-----+-----+-----+-----+-----+-----+
| class_id | group_concat(stu_name) | count(*) | max(stu_age) | min(stu_height) | avg(stu_age) |
+-----+-----+-----+-----+-----+-----+
| 1 | 张五,晓婷 | 2 | 28 | 165 | 22.5000 |
| 2 | 夏洛,张四 | 2 | 25 | 175 | 23.0000 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

运算符

算术运算符

+、-、*、/、%

基本算术运算，通常不在条件中使用，而是用于结果运算（select 字段中）

基本语法：select 运算 from 表名；

注意：

- 1、在 mysql 中，除法的运算结果是用浮点数表示的。
- 2、除法中除数如果为 0，系统会给 NULL。
- 3、NULL 进行任何的算术运算结果都为 NULL。

比较运算符

>、>=、<、<=、=、<>

通常是用来在条件中进行限定结果。

=: 在 mysql 中，没有对应的 == 比较符号，就是使用 = 来进行相等判断

<=>: 相等比较

特殊应用：就是在字段结果中进行比较运算

```
mysql> -- mysql中没有规定select必须有数据表
mysql> select '1'<=>1, 0.02 <=> 0, 0.02 <> 0;
+-----+-----+-----+
| '1'<=>1 | 0.02 <=> 0 | 0.02 <> 0 |
+-----+-----+-----+
|      1      |       0      |      1      |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

在 MySQL 中，数据会自动转换为同类型，再比较

在 mysql 中，没有布尔值，1 代表 true，0 代表 false

在条件判断的时候，还会有对应的比较运算符：计算区间。

between 条件 1 and 条件 2；

```
mysql> select * from my_stu where stu_age between 20 and 30;
+-----+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender |
+-----+-----+-----+-----+-----+-----+
| stu002 | 张四     | 2        | 28      | 175        | 男     |
| stu003 | 张五     | 1        | 20      | 165        | 女     |
| stu004 | 晓婷     | 1        | 25      | 195        | 男     |
| stu005 | 小猪     | 1        | 30      | 172        | 男     |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

between 中条件 1 必须小于条件 2，反过来不可以。

逻辑运算符

and (逻辑与)

or (逻辑或)

not (逻辑非)

```
mysql> -- 逻辑与解决
mysql> select * from my_stu where stu_age >= 20 and stu_age <= 30;
+-----+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender |
+-----+-----+-----+-----+-----+-----+
| stu002 | 张四     | 2        | 28      | 175        | 男     |
| stu003 | 张五     | 1        | 20      | 165        | 女     |
| stu004 | 晓婷     | 1        | 25      | 195        | 男     |
| stu005 | 小猪     | 1        | 30      | 172        | 男     |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

```
mysql> -- 逻辑或：查找身高高于175或者年龄大于20的学生
mysql> select * from my_stu where stu_height >= 175 or stu_age >= 20;
+-----+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender |
+-----+-----+-----+-----+-----+-----+
| stu001 | 夏洛     | 2        | 18      | 185        | 女     |
| stu002 | 张四     | 2        | 28      | 175        | 女     |
| stu003 | 张五     | 1        | 20      | 165        | 女     |
| stu004 | 晓婷     | 1        | 25      | 195        | 男     |
| stu005 | 小猪     | 1        | 30      | 172        | 男     |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

in 运算符

in: 在什么里面, 是用来替代=, 当结果不是一个值, 而是一个结果集的时候

基本语法: in (结果 1, 结果 2...) ;只要当前条件在结果集中出现过, 那么就成立。

```
mysql> select * from my_stu where stu_id in ('stu001', 'stu002', 'stu003')
```

stu_id	stu_name	class_id	stu_age	stu_height	gender
stu001	夏洛	2	18	185	男
stu002	张四	2	28	175	女
stu003	张五	1	20	165	男

is 运算符

is 是专门用于判断字段是否为 NULL 的运算符

基本语法: is null/is not null;

```
6 rows in set (0.37 sec)
mysql> select * from my_int where int_6 = null;           ← null与任何运算符运算都是null, 返回0
Empty set (0.00 sec)
```

```
mysql> select * from my_int where int_6 is null;          ←
+-----+-----+-----+-----+-----+-----+-----+-----+
| int_9 | int_8 | int_7 | int_6 | int_1 | int_2 | int_3 | int_4 | int_5 |
+-----+-----+-----+-----+-----+-----+-----+-----+
| NULL  | NULL  | NULL  | NULL  | -128  | 255   | 100000 | 10000000 | 1000000000000 |
| NULL  | NULL  | NULL  | NULL  |       |        |        |        |        |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

like 运算符

like 运算法是用来进行模糊匹配的

基本语法: like '匹配模式';

匹配模式中: 有两种占位符:

_:匹配单个字符

%:匹配多个字符

```
mysql> select * from my_stu where stu_name like '%';
ERROR 1267 (HY000): Illegal mix of collations (utf8_general_ci,IMPLICIT) and (gbk_chinese_ci,COERCIBLE) for operation 'like'`
```

如果匹配的目标中有_或者%, 则要对通配进行转义。有两种方式:

1、反斜杠是转义符, 通过反斜杠来转义%, 使其不再是通配符。这里第一个%是通配符, 第二个%不是通配符。

```
select percent from score where percent like '%0\%';
```

2、这种是通过 escape 关键字进行转义, 将特定符号后的字符进行转义, 这里斜杠后面的%就不再是通配符, 斜杠之前的%仍然起通配符作用。

```
select percent from score where percent like '%0/%' escape '/';
```

联合查询

基本概念

联合查询是可合并多个相似的**选择查询**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在一起，使用为此为 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

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

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

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）

all：保存所有的数据。

```
select * from my_stu where gender = '女' order by stu_id  
mysql> select * from my_stu  
-> union all  
-> select * from my_stu;  
+-----+-----+-----+-----+-----+  
| stu_id | stu_name | class_id | stu_age | stu_height |  
+-----+-----+-----+-----+-----+  
| stu001 | 夏洛    | 2       | 18      | 185     |  
| stu002 | 张四    | 2       | 28      | 175     |  
| stu003 | 张五    | 1       | 20      | 165     |  
| stu004 | 晓婷    | 1       | 25      | 195     |  
| stu005 | 小猪    | 1       | 30      | 172     |  
| stu001 | 夏洛    | 2       | 18      | 185     |  
| stu002 | 张四    | 2       | 28      | 175     |  
| stu003 | 张五    | 1       | 20      | 165     |  
| stu004 | 晓婷    | 1       | 25      | 195     |  
| stu005 | 小猪    | 1       | 30      | 172     |  
+-----+-----+-----+-----+-----+
```

```

mysql> select stu_id, stu_height from my_stu
-> union all
-> select stu_height, stu_id from my_stu;
+-----+
| stu_id | stu_height |
+-----+
| stu001 | 185
| stu002 | 175
| stu003 | 165
| stu004 | 195
| stu005 | 172
| 185    | stu001
| 175    | stu002
| 165    | stu003
| 195    | stu004
| 172    | stu005
+-----+

```

注意细节：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。

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

```

mysql> (select * from my_stu where gender = 1 order by stu_height asc)
-> union
-> (select * from my_stu where gender = 2 order by stu_height desc);
+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender |
+-----+
| stu001 | 夏洛     | 2         | 18      | 185        | ¢□      |
| stu003 | 张五     | 1         | 20      | 165        | ¢□      |
| stu002 | 张四     | 2         | 28      | 175        | ª¥      |
| stu004 | 晓婷     | 1         | 25      | 195        | ª¥      |
| stu005 | 小猪     | 1         | 30      | 172        | ª¥      |
+-----+

```

没有排序

2、order by 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 * from my_stu where gender = 1 order by stu_height asc limit 10)
-> union
-> (select * from my_stu where gender = 2 order by stu_height desc limit 10);
+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender |
+-----+
| stu003 | 张五     | 1         | 20      | 165        | ¢□      |
| stu001 | 夏洛     | 2         | 18      | 185        | ¢□      |
| stu004 | 晓婷     | 1         | 25      | 195        | ª¥      |
| stu002 | 张四     | 2         | 28      | 175        | ª¥      |
| stu005 | 小猪     | 1         | 30      | 172        | ª¥      |
+-----+

```

升序

降序

连接查询

连接查询概念

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）

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，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

连接查询的分类

连接查询一共有以下几类：

交叉连接

内连接

外连接：左外连接（左连接）和右外连接（右连接）

自然连接

交叉连接

交叉连接：将两张表的数据与另外一张表彼此交叉

原理

- 1、从第一张表依次取出每一条记录
- 2、取出每一条记录之后，与另外一张表的全部记录挨个匹配
- 3、没有任何匹配条件，所有的结果都会进行保留
- 4、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）

语法

基本语法：表 1 cross join 表 2;

应用

交叉连接产生那个的结果是笛卡尔积，没有实际应用。保证连接查询的整体完整性。

本质与 from 表 1, 表 2; 产生的结果一样

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

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
- 2、利用匹配条件进行匹配
 - 2.1 匹配成功, 保留, 继续向下匹配
 - 2.2 匹配失败, 向下继续, 如果全表匹配失败, 结束

语法

基本语法: 表 1[inner]join 表 2 on 匹配条件;

- 1、如果内连接没有匹配条件, 结果为笛卡尔积, 其实就是交叉连接(避免)
- 2、使用匹配条件进行匹配

stu_id	stu_name	class_id	stu_age	stu_height	gender	id	name
stu001	夏洛	2	18	185	♂	2	2
stu002	张四	2	28	175	♀	2	2
stu003	张五	1	20	165	♂	1	1
stu004	晓婷	1	25	195	♀	1	1
stu005	小猪	1	30	172	♀	1	1

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, 确保唯一性。

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。

```
-- 内连接
select * from my_stu inner join my_class; -- 可以没有条件, 结果为笛卡尔积
select * from my_stu inner join my_class on class_id = id;
select * from my_stu inner join my_class on my_stu.class_id = my_class.id;
select * from my_stu as s inner join my_class c on s.class_id = c.id;
```

5、内连接匹配的时候, 必须保证匹配到才会保存。

6、内连接因为可以不强制必须使用匹配条件, (on) 因此可以在完成数据匹配完成后, 所用 where 条件来限制, 效果与 on 一样 (建议使用 on)

```
select * from my_stu as s inner join my_class c where s.class_id = c.id;
```

应用

内连接通常是在对数据有精确要求的地方使用，必须保证两张表中都能进行数据匹配

外连接

外连接：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

原理

- 1、确定连接主表：左连接就是 left join 左边的表为主表；right join 就是右边为主表
- 2、拿主表的每一条记录，去匹配另外一张表（从表）的每一条记录
- 3、如果满足匹配条件，保留，不满足即不保留
- 4、如果主表记录在从表中一条都没有匹配成功，那么也要保留该记录，从表对应的字段值都为 NULL

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；

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 my_stu as s left join my_class c on s.class_id = c.id;
+-----+-----+-----+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender | id | name |
+-----+-----+-----+-----+-----+-----+-----+-----+
| stu001 | 夏洛 | 2 | 18 | 185 | 男 | 2 | 2 |
| stu002 | 张四 | 2 | 28 | 175 | 男 | 2 | 2 |
| stu003 | 张五 | 1 | 20 | 165 | 男 | 1 | 1 |
| stu004 | 晓婷 | 1 | 25 | 195 | 女 | 1 | 1 |
| stu005 | 小猪 | 1 | 30 | 172 | 女 | 1 | 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)      不论左右连接，表在左边数据就在左边

mysql> select * from my_stu as s right join my_class c on s.class_id = c.id;
+-----+-----+-----+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender | id | name |
+-----+-----+-----+-----+-----+-----+-----+-----+
| stu001 | 夏洛 | 2 | 18 | 185 | 男 | 2 | 2 |
| stu002 | 张四 | 2 | 28 | 175 | 男 | 2 | 2 |
| stu003 | 张五 | 1 | 20 | 165 | 男 | 1 | 1 |
| stu004 | 晓婷 | 1 | 25 | 195 | 女 | 1 | 1 |
| stu005 | 小猪 | 1 | 30 | 172 | 女 | 1 | 1 |
| NULL | NULL | NULL | NULL | NULL | NULL | 3 | 3 |
+-----+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)      右表尾主表：主表记录一定保存，从表没有数据对应，字段为NULL
mysql>

```

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、左连接和右连接可以互相转换，但是数据对应的位置（表顺序）会改变

应用

非常常用的一种获取数据的方式。作为数据获取对应主表以及其他数据（关联）

自然连接

自然连接： nature join，自然连接其实就是自动匹配连接条件，系统以两表中同名字段作为匹配条件，如果两表有多个同名字段，那就都作为匹配条件。在这里，自然连接可以分为自然内连接和自然外连接。

自然外连接：

基本语法： 左表 + nature + left/right + join + 右表；

自然内连接：

基本语法： 左表 + nature + join + 右表；

自然连接自动使用同名字段作为连接条件，而且在连接完成之后合并同名字段。

实际上，自然连接并不常用。而且，咱们可以用内连接和外连接来模拟自然连接，模拟的关键就在于使用同名字段作为连接条件及合并同名字段。

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
using 内部的字段名就是作为连接条件的字段，也是需要合并的同名字段。

原理

- 1、在连接查询中，使用 on 的地方使用 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）
- 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

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

子查询

子查询概念

子查询： sub query

子查询是一种计算机 SELECT-SQL 语言中嵌套查询下层的程序模块。当一个查询是另一个查询的条件时，称之为子查询

子查询：指在一条 select 语句中嵌入到另外一条 select 语句，那么被嵌入的 select 语句称之为子查询语句

主查询概念

主查询：主要的查询对象，第一条 select 语句，确定了用户所要获取的数据目标（数据源），以及要具体得到的字段信息。

子查询与主查询的关系

- 1、子查询是嵌入到主查询中的
- 2、子查询是辅助主查询的：要么作为条件，要么作为数据源

3、子查询其实是可以独立存在：是一条完整的 select 语句

子查询分类

按功能分类：

标量子查询：子查询返回的结果是一个数据（一行一列）

列子查询：返回的结果是一列（一列多行）

行子查询：返回的结果是一行（一行多列）

表子查询：返回的结果是多行多列（多行多列）

exists 子查询：返回的结果 1 或者 0（类似布尔操作）

按位置分类

where 子查询：子查询出现的位置在 where 条件中：标量子查询、列子查询和行子查询

from 子查询：子查询出现的位置在 from 数据源中（作数据源）：表子查询

标量子查询

标量子查询：子查询返回的结果是一个数据（一行一列）

基本语法：select * from 数据源 where 条件判断 =/=> (select 字段名 from 数据源 where 条件判断); //子查询得到的结果只有一个值

```
mysql> -- 知道一个学生的名字 小猪 想知道它在哪个班级 (班级名字)
mysql>
mysql> -- 1、通过学生表获取它所在班级id
mysql> -- 2、通过班级id获取班级名字
mysql>
mysql> -- 标量子查询
mysql> select * from my_class where id = (select class_id from my_stu where stu_name = '小猪');
+----+-----+
| id | name |
+----+-----+
| 1  | 1    |
+----+-----+
1 row in set (0.00 sec)

mysql>
```

需求决定主查询
条件决定子查询

列子查询

列子查询：返回的结果是一列（一列多行）

基本语法：主查询 where 条件 in (列子查询);

```
mysql> -- 想获取已经有学生在班的所有班级名字
mysql> -- 找出学生所在的所有班级id
mysql> -- 找出对应的班级表中对应的名字
mysql>
mysql> -- 列子查询
mysql> select name from my_class where id in (select class_id from my_stu);
+-----+
| name |
+-----+
| 2    |
| 1    |
+-----+
```

行子查询

行子查询：返回的结果是一行（一行多列）

行元素：字段元素是指一个字段对应的值。行元素对应的就是多个字段：多个字段合起来作为一个元素参与运算，把这种情况称之为行元素

基本语法：主查询 where 条件[(构造一个行元素)] = (行子查询);

```
mysql> select * from my_stu where (stu_age,stu_height) = (select max(stu_age),max(stu_height) from my_stu);
Empty set (0.00 sec)
mysql>                                     构造行元素
                                         子查询得到一行多字段
```

```
-- 需求：获取班级上年龄最大，且身高最高的学生
-- 1、求出班级年龄最大的值
-- 2、求出班级身高最高的值
-- 3、求出对应的学生
select * from my_stu where (stu_age,stu_height) = (select max(stu_age),max(stu_height) from my_stu);

-- 错误示例
select * from my_stu having stu_age = max(stu_age) and stu_height = max(stu_height);
-- 解释：
-- 1、having是在group by之后：使用having代表着前面的group by执行了一遍，（聚合函数）
-- 2、group by一旦执行：结构就是只返回一行记录：第一行|
```

表子查询

表子查询：返货的结果是多行多列（不一定），表子查询和行子查询非常相似，只是行子查询需要产生行元素，表子查询没有。

行子查询是用于 where 条件判断：where 子查询

表子查询是用于 from 数据源：from 子查询

基本语法：

```
select 字段表 from (表子查询) as 别名 [where][group by][having][order by][limit];
```

```

-- 获取每个班上最高身高的学生(一个)
-- select * from my_stu group by class_id having stu_height = max(stu_height); -- 不可行
-- group by 只保存第一个数据

-- 1、将每个班最高的学生排在最前面: order by
-- 2、再针对结果进行group by: 保留每组第一个
select * from (select * from my_stu order by stu_height desc) as temp group by class_id;

```

别名必须，后面要 提取字段
必须从表中提取

exists 子查询

exists 子查询：返回的结果 1 或者 0（类似布尔操作，mysql 中没有布尔类型），1 代表成立，0 代表不成立。

基本语法：where exists (查询语句); //exists 就是根据查询得到的结果进行判断，如果结果存在，那么返回 1，否则返回 0

where 1：永远为真

```
-- 求出，有学生在的所有班级
select * from my_class as c where exists(select stu_id from my_stu as s where s.class_id = c.id);
```

特定关键字

In

主查询 where 条件 in (列子查询);

Any

任意一个

= any (列子查询)：条件在查询结果中有任意一个匹配即可，等价于 in
<>any (列子查询)：条件在查询结果中不等于任意一个

```
1 = any (1,2,3) =====true
1<>any(1,2,3) =====true
```

Some

与 any 完全一样

All

= all (列子查询): 等于里面所有

<>all (列子查询): 不等于里面所有

```
-- in,any,some,all
select * from my_stu where class_id in (select id from my_class);
select * from my_stu where class_id = any (select id from my_class);          结果一样
select * from my_stu where class_id <> any (select id from my_class);

select * from my_stu where class_id = all (select id from my_class);
select * from my_stu where class_id <> all (select id from my_class);          查询结果为空

select * from my_calss where id = all (select class_id from my_stu);
select * from my_calss where id <> all (select class_id from my_stu);          该表对应记录为1, 2, 3。3的确不等于1也不等于2,          查询出来的结果只有1和2
```

如果对应的匹配字段有 NULL, 那么不参与匹配。

数据备份与还原

意义

- 1、提高系统的可用性和灾难可恢复性，在数据库系统崩溃的时候，没有数据库备份就无法找到数据。
- 2、使用数据库备份还原数据库是数据库系统崩溃时提供数据恢复最小代价的最优方案，如果让客户重新填报数据，代价那就太大了。
- 3、没有数据就没有一切，数据库备份就是一种防范灾难于未然的强力手段，没有了数据，应用再花哨也是镜中花水中月。

整库备份与还原

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

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（数据库被删除），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性的数据库备份

应用方案

SQL 备份

SQL 备份用到的是专门的备份客户端，因此还没有与数据库服务器进行连接。

基本语法：mysqldump/ mysqldump.exe -hPup 数据库名字 [表 1 [表 2 ...]] > 备份文件地址

-hPup:ip 地址 端口 用户 密码

备份可以有三种形式：

1、整库备份（只需要提供数据库名字）

```
-- 利用mysqldump进行SQL备份
-- 整库备份
mysqldump.exe -hlocalhost -P3306 -uroot -p mydatabase > d:/Server/temp.sql
```

2、单表备份：数据库后面跟一张表

3、多表备份：数据库后面跟多张表

```
-- 多表备份
mysqldump -uroot -p mydatabase my_stu my_class > d:/Server/student.sql
```

查看备份文件

```
DROP TABLE IF EXISTS `my_stu`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
SET character_set_client = utf8mb4 ;
CREATE TABLE `my_stu` (           ← 创建表结构
    `stu_id` varchar(10) NOT NULL COMMENT '瀛︽敤 ID',
    `stu_name` varchar(10) NOT NULL,
    `class_id` int(11) DEFAULT NULL,
    `stu_age` tinyint(3) unsigned DEFAULT NULL,
    `stu_height` tinyint(3) unsigned DEFAULT NULL,
    `gender` enum('鑾','濂','淇漬病') DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

-- 
-- Dumping data for table `my_stu`
-- 

LOCK TABLES `my_stu` WRITE;
/*!40000 ALTER TABLE `my_stu` DISABLE KEYS */;
INSERT INTO `my_stu` VALUES ('stu001','澶忓礽',2,18,185,'鑾'),('stu002','
/*!40000 ALTER TABLE `my_stu` ENABLE KEYS */;
UNLOCK TABLES;
```

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

1、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

```
C:\Users\YZQ>
C:\Users\YZQ>mysql -uroot -p mydatabase < d:/Server/temp.sql
Enter password: *****

C:\cmd.exe - 快捷方式 - mysql -uroot -p
mysql> create database mydatabase;
Query OK, 1 row affected (0.23 sec)

mysql> show tables;
ERROR 1046 (3D000): No database selected
mysql> use mydatabase;
Database changed
mysql> show tables;
Empty set (0.00 sec)

mysql> show tables;
+-----+
| Tables_in_mydatabase |
+-----+
| my_auto
| my_class
| my_comment
| my_date
| my_decimal
| my_default
| my_enum
| my_float
| my_friend
| my_int
| my_primary
| my_primary2
| my_primary3
| my_score
| my_set
| my_simple |
+-----+
```

2、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

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

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

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 中所有用户的信息都保存在 mysql 数据库下的 user 表中。

默认的，在安装 mysql 的时候，如果没有创建匿名用户，那么意味着所有的用户只有一个；root 超级用户

PS：修改 root 用户密码： alter user'root'@'localhost' identified by '新密码'；

```
mysql> select * from mysql.user\G
*****1 row*****
      Host: localhost
      User: mysql.infoschema
      Select_priv: Y
      Insert_priv: N
      Update_priv: N
      Delete_priv: N
      Create_priv: N
      Drop_priv: N
      Reload_priv: N
      Shutdown_priv: N
      Process_priv: N
```



```
mysql> desc mysql.user;
+-----+-----+-----+-----+-----+
| Field | Type | 复合主键 | Null | Key |
+-----+-----+-----+-----+-----+
| Host | char(60) | NO | PRI |
| User | char(32) | NO | PRI |
| Select_priv | enum('N','Y') | NO |
| Insert_priv | enum('N','Y') | NO |
| Update_priv | enum('N','Y') | NO |
```

在 mysql 中，对用户的用户管理中，是由对应的 host 和 user 共同组成主键来区分用户；

User：代表用户的用户名

Host：代表本机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）都可以进行访问

创建用户

理论上讲，可采用两种方案创建用户：

1、直接使用 root 用户在 mysql.user 表中插入记录（不推荐）

2、专门用于创建用户的 SQL 指令

基本语法：create user 'username'@'host' identified by 'password';

username：你将创建的用户名

host：指定该用户在哪个主机上可以登陆，如果是本地用户可用 localhost，如果想让该用户可以从任意远程主机登陆，可以使用通配符%。通配符%可以用作整个主机名，或者用作主机名的一部分。如果主机名为空，表示是一个通配符，但比%通配符通配范围低。

password：该用户的登陆密码，密码可以为空，如果为空则该用户可以不需要密码登陆服务器

```
mysql> -- 创建用户
mysql> create user 'user1'@'%' identified by '123456';
Query OK, 0 rows affected (0.40 sec)

mysql> select * from mysql.user;
```

查看 mysql.user 是否存在新增的用户

%	user1	N	N	N	N	N	N
N	N	N	N	N	N	N	N

简化创建用户

```
mysql> -- 简化创建用户
mysql> create user user2;
Query OK, 0 rows affected (0.49 sec)
```

- 1、不限定客户端IP
- 2、没有密码的用户
- 3、谁都可以访问，不需要密码

```
C:\Windows\System32>mysql -uuser2
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 20
Server version: 8.0.11 MySQL Community Server - GPL

Copyright (c) 2000, 2018, 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>
```

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 'username'@'host';

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 'username'@'host' = ‘newpassword’ ;

如果是当前用户： set password = ‘newpassword’ ;

```
'For the right syntax to use near password(123456'
'mysql> set password for 'user1'@'%' = '654311';
Query OK, 0 rows affected (0.39 sec)
```

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

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 [with grant option];

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.*），也可以是整库（*.*）

mysql 中只有 with grant option，对 A 用户进行的授权，A 可以授予给其他用户，当收回对 A 的授权时，A 授予给其他用户的权限不会被级联收回。注意 with grant option 也可以被授予给其他用户。

```

mysql> create user 'user1'@'%' identified by '123456';
Query OK, 0 rows affected (0.45 sec)

mysql> grant select on mydatabase.my_stu to 'user1'@'%';
Query OK, 0 rows affected (0.46 sec)

mysql>

用户被分配权限后不需要退出就可以看到效果

```

Database
information_schema

1 row in set (0.00 sec)

Database
information_schema
mydatabase

2 rows in set (0.00 sec)

Tables_in_mydatabase
my_stu

1 row in set (0.00 sec)

具体权限查看：单表权限只能看到数据库中的一张表

Show 语句的一个变体用来查看用户被授予的权限：

show grants for user;

```

mysql> show grants for 'user1'@'localhost';
+-----+
| Grants for user1@localhost |
+-----+
| GRANT USAGE ON *.* TO `user1`@`localhost` |
| GRANT SELECT ON `news`.`n_news` TO `user1`@`localhost` |
+-----+
2 rows in set (0.00 sec)

```

取消权限：revoke

权限回收：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/*表名/* from 用户；

授予权限时如果有 with grant option 必须再执行

revoke grant option from 'test1'@'localhost';

完整才能回收其权限。

```

mysql> -- 回收权限
mysql> revoke all privileges on mydatabase.my_stu from 'user1'@'%';
Query OK, 0 rows affected (0.50 sec)

```

权限回收，同样不需要刷新，用户马上就会感受到

```

mysql> show tables;
ERROR 1044 (42000): Access denied for user 'user1'@'%' to database 'mydatabase'
mysql>

```

刷新权限：flush

(1) 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。此操作需要在 mysql 命令提示符下执行（需要以管理员身份登录）。

基本语法：flush privileges;

(2) mysqladmin flush-privileges

(3) mysqladmin reload

(2) 和 (3) 需要在操作系统环境下运行。

最后，mysql 服务器将在重新启动时重新载入授权表。

当用户下次再连接时，全局级别权限将再次被检查。当下一个 use 语句触发时，数据库权限将被检查，而表级别和列级别权限将在用户下次请求时检查。

密码丢失找回

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

<https://blog.csdn.net/gupao123456/article/details/80766154>

外键

外键概念

如果公共关键字在一个关系中是主关键字，那么这个公共关键字被称为另一个关系的外键。由此可见，外键表示了两个关系之间的相关联系。以另一个关系的外键作为主关键字的表被称为为主表，具有此外键的表被称为从表，外键又称之为外关键字

外键：foreign key ， 一张表（A）中有一个字段保存的值指向另外一张表（B）的主键

B： 主表

A： 从表

外键的操作

增加外键

提供了两种方式增加外键：

1、在创建表的时候增加外键（类似主键）

基本语法：在字段之后增加一条语句

[constraint `外键名`] foreign key (外键字段) references 主表 (主键);

```

ERROR 1075 (42000): Incorrect table definition; there can be only one
mysql> create table my_foreign(
    -> age int primary key auto_increment,
    -> name varchar(10) not null,
    -> -- 关联班级my_class
    -> id int(11) not null,
    -> 增加外键
    -> foreign key(id) references my_class(id)
    -> )charset utf8;
Query OK, 0 rows affected, 1 warning (0.65 sec)

mysql> desc my_foreign;
+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra   |
+-----+-----+-----+-----+-----+
| age   | int(11)| NO   | PRI | NULL    | auto_increment |
| name  | varchar(10)| NO  |     | NULL    |                |
| id    | int(11) | NO   | MUL | NULL    |                |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)          多索引
mysql>

```

MUL:多索引，外键本身是一个索引，还要求外键字段本身也是一种普通索引

```

mysql> show create table my_foreign;
+-----+-----+
| Table      | Create Table           |
+-----+-----+
| my_foreign | CREATE TABLE `my_foreign` (
    -> `age` int(11) NOT NULL AUTO_INCREMENT,          创建外键时自动增加的普通索引
    -> `name` varchar(10) NOT NULL,
    -> `id` int(11) NOT NULL,
    -> PRIMARY KEY (`age`),
    -> KEY `id` (`id`),
    -> CONSTRAINT `my_foreign_ibfk_1` FOREIGN KEY (`id`) REFERENCES `my_class` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+-----+

```

2、在创建表后增加外键

基本语法: alter table 从表 add [constraint`外键名`] foreign key(外键字段) references 主表 (主键);

```

2
3 -- 修改my_stu表, 将class_id 设为外键字段          反引号
4 alter table my_stu add constraint `student_class_ibfk_1` foreign key(class_id) references my_class(id);

```

外键名字可以指定

修改&删除外键

外键不允许修改，只能先删除后增加。

基本语法: alter table 从表 drop foreign key 外键名字;

```

mysql> -- 删除外键
mysql> alter table my_stu drop foreign key student_class_ibfk_1;
Query OK, 0 rows affected (0.46 sec)
Records: 0  Duplicates: 0  Warnings: 0

```

外键不能删除产生的普通索引，只会删除自己。如果想要删除对应的索引：alter table 表名 drop index 索引名字；

my_stu CREATE TABLE `my_stu` (stu_id varchar(10) NOT NULL COMMENT '瀛︽敂 ID',	stu_name varchar(10) NOT NULL,	class_id int(11) DEFAULT NULL,	stu_age tinyint(3) unsigned DEFAULT NULL,	stu_height tinyint(3) unsigned DEFAULT NULL,	gender enum('鑾', '濂', '淇濶瘣') DEFAULT NULL,	KEY student_class_ibfk_1 (class_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8							外键创建会自动增加一个索引：但是外键删除只会删除自己

外键的基本要求

- 1、外键字段需要保证与关联的主表的主关键字字段类型完全一致
- 2、基本属性也要相同
- 3、如果在表后增加外键，对数据还有一定的要求（从表数据与主表的关联关系）
- 4、外键只能使用 innodb 存储引擎

外键的约束

外键约束：通过建立外键关系之后，对主表和从表都会有一定的数据约束效率。

约束的基本概念

- 1、当一个外键产生时：外键所在的表(从表)会受制于主表数据的存在从而导致数据不能进行某些不符合规范的操作（不能插入主表不存在的数据）；

主表	
id	name
1	1
2	2
3	3

3 rows in set (0.00 sec)

```

mysql> select * from my_class;

```

向从表插入数据

```

mysql> insert into my_foreign values(null,'小妹'1); -- 正确
Query OK, 1 row affected (0.43 sec)

mysql> insert into my_foreign values(null,'小妹'4); -- 不正确 (主表没有该记录)
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails ('mydatabase`.`my_foreign`, CONSTRAINT `my_foreign_ibfk_1` FOREIGN KEY (`id`) REFERENCES `my_class` (`id`))

```

- 2、如果一张表被其他表外键引入，那么该表的数据操作就不能随意，必须保证从表数据的有效性（不能随便删除一个被从表引入的记录）

```

mysql> select * from my_foreign;
+---+---+---+
| age | name | id |
+---+---+---+
| 1   | 小妹 | 1  |
+---+---+---+
1 row in set (0.00 sec)

mysql> delete from my_class where id = 1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails ('mydatabase`.`my_foreign`, CONSTRAINT `my_foreign_ibfk_1` FOREIGN KEY (`id`) REFERENCES `my_class` (`id`))
mysql>

```

外键约束的概念

可以在创建外键的时候，对外键约束进行选择性的操作。

基本语法：add foreign key (外键字段) references 主表（主键） on 约束模式；

约束模式有三种：

- 1、restrict: 严格模式（默认），不允许操作
- 2、cascade: 级联模式，一起操作，主表变化，从表数据跟着变化
- 3、set null: 置空模式，主表变化（删除），从表对应记录设置为空：前提是表中对应的外键字段允许为空。

外键约束的主要约束对象是主表操作：从表就是不能插入主表不存在的数据。

通常在进行约束使用的时候，需要指定操作：update 和 delete

常用的约束模式：on update cascade, on delete set null, 更新级联，删除置空

```

alter table my_stu add foreign key(class_id)
references my_class(id)

-- 约束
on update cascade          ← 更新模式
on delete set null;         ← 删除模式

-- 更新主表
update my_class set id = 4 where id = 2;

```

如果子表试图创建一个在父表中不存在的外键值，InnoDB会拒绝任何INSERT或UPDATE操作。如果父表试图UPDATE或者DELETE任何子表中存在或匹配的外键值，最终动作取决于外键约束定义中的ON UPDATE和ON DELETE选项。InnoDB支持5种不同的动作，如果没有指定ON DELETE或者ON UPDATE，默认的动作是RESTRICT：

1. CASCADE: 从父表中删除或更新对应的行，同时自动的删除或更新自表中匹配的行。ON DELETE CASCADE和ON UPDATE CASCADE都被InnoDB所支持。

2. SET NULL: 从父表中删除或更新对应的行，同时将子表中的外键列设为空。注意，这些在外键列没有被设为NOT NULL时才有效。ON DELETE SET NULL和ON UPDATE SET NULL都被InnoDB所支持。

3. NO ACTION: InnoDB拒绝删除或者更新父表。

4. RESTRICT: 拒绝删除或者更新父表。指定RESTRICT（或者NO ACTION）和忽略ON DELETE或者ON UPDATE选项的效果是一样的。

5. SET DEFAULT: InnoDB目前不支持。

mysql> select * from my_stu;

stu_id	stu_name	class_id	stu_age	stu_height	gender
stu001	澧邑碘	2	18	185	鑾
stu002	寮豺洓	2	28	175	濂
stu003	寮狖簾	1	20	165	鑾
stu004	鎔嶽	1	25	195	濂
stu005	灝快尗	1	30	172	濂

5 rows in set (0.00 sec)

mysql> **更新主表**
 mysql> update my_class set id = 4 where id = 2;
 Query OK, 1 row affected (0.39 sec)
 Rows matched: 1 Changed: 1 Warnings: 0

约束:
on update cascade

mysql> select * from my_stu;

stu_id	stu_name	class_id	stu_age	stu_height	gender
stu001	澧邑碘	4	18	185	鑾
stu002	寮豺洓	4	28	175	濂
stu003	寮狖簾	1	20	165	鑾
stu004	鎔嶽	1	25	195	濂
stu005	灝快尗	1	30	172	濂

mysql> select * from my_stu;

stu_id	stu_name	class_id	stu_age	stu_height	gender
stu001	澧邑碘	4	18	185	鑾
stu002	寮豺洓	4	28	175	濂
stu003	寮狖簾	1	20	165	鑾
stu004	鎔嶽	1	25	195	濂
stu005	灝快尗	1	30	172	濂

5 rows in set (0.00 sec)

mysql> delete from my_class where id = 4;
 Query OK, 1 row affected (0.39 sec)

约束
on delete set null

mysql> select * from my_stu;

stu_id	stu_name	class_id	stu_age	stu_height	gender
stu001	澧邑碘	NULL	18	185	鑾
stu002	寮豺洓	NULL	28	175	濂
stu003	寮狖簾	1	20	165	鑾
stu004	鎔嶽	1	25	195	濂
stu005	灝快尗	1	30	172	濂

5 rows in set (0.00 sec)

mysql>

约束作用

保证数据的完整性：主表与从表的数据要一致。

正是因为外键有非常强大的数据约束作用，而且可能导致数据在后台变化的不可控，导致程序在进行设计开发逻辑的时候，没有办法很好的把握数据（业务），所以外键比较少使用。

视图

视图概念

视图是指计算机数据库中的视图，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。行和列数据来自由定义视图的查询所引用的表，并且在引用视图时动态生成。

关系型数据库中的数据是由一张一张的二维关系表所组成，简单的单表查询只需要遍历一个表，而复杂的多表查询需要将多个表连接起来进行查询任务。对于复杂的查询事件，每次查询都需要编写 MySQL 代码效率低下。为了解决这个问题，数据库提供了视图（view）功能。

0 视图相关的MySQL指令

操作指令	代码
创建视图	<code>CREATE VIEW 视图名(列1, 列2...) AS SELECT (列1, 列2...) FROM ...;</code>
使用视图	当成表使用就好
修改视图	<code>CREATE OR REPLACE VIEW 视图名 AS SELECT [...] FROM [...];</code>
查看数据库已有视图	<code>>SHOW TABLES [like...];</code> (可以使用模糊查找)
查看视图详情	<code>DESC 视图名或者SHOW FIELDS FROM 视图名</code>
视图条件限制	<code>[WITH CHECK OPTION]</code>

视图是虚拟表，本身不存储数据，而是按照指定的方式进行查询。

视图基本操作

创建视图

视图的本质是 SQL 指令（select 语句）

基本语法:create[or repalce] view 视图名字 as select 指令 [with check option]; //可以是单表数据，也可以是连接查询，联合查询或者子查询

```
5 rows in set (0.00 sec)

mysql> -- 创建视图
mysql> create view stu_class_v as
-> select * from my_stu as s left join s.* , c.name
-> my_class c
-> on
-> s.class_id = c.id;
Query OK, 0 rows affected (0.42 sec)

mysql> -
```

表里面不能出现同名字段所以会显示重复。

查看视图：视图本身是虚拟表，所有关于表的一些操作都适合视图。

```
mysql> show create view stu_class_v\G
***** 1. row *****
      View: stu_class_v
      Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER
      select `s`.`stu_id` AS `stu_id`, `s`.`stu_name` AS `stu_name`, `s`.`class_id` AS `class_id`, `s`.`stu_height` AS `stu_height`, `s`.`gender` AS `gender`, `c`.`id` AS `id`, `c`.`name` AS `name` from `my_class` `c` on((`s`.`class_id` = `c`.`id`))
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (0.00 sec)
```

使用视图

视图是一张虚拟表，可以直接把视图当做“表”操作，但是视图本身没有数据，是临时执行 select 语句得到对应的结果。视图主要用于 **查询操作**。

基本语法：select 字段列表 from 视图名字[各种子句];

```
-- 使用视图
mysql> select * from stu_class_v;
+-----+-----+-----+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender | id | name |
+-----+-----+-----+-----+-----+-----+-----+-----+
| stu001 | 潘忆碘 | NULL | 18 | 185 | 鑫 | NULL | NULL |
| stu002 | 翟狩涑 | NULL | 28 | 175 | 澄 | NULL | NULL |
| stu003 | 翟狄簾 | 1 | 20 | 165 | 鑫 | 1 | 1 |
| stu004 | 鎏嶽 | 1 | 25 | 195 | 澄 | 1 | 1 |
| stu005 | 瀛快尙 | 1 | 30 | 172 | 澄 | 1 | 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

修改视图

修改视图：本质是修改视图对应的查询语句

基本语法：alter view 视图名字 as 新 select 指令；

```
-- 修改视图
mysql> alter view stu_class_v as
-> select * from my_stu as s left join
-> my_class c
-> on
-> s.class_id = c.id;
Query OK, 0 rows affected (0.47 sec)
```

删除视图

基本语法：drop view 视图名字；

```
mysql> -- 删除视图  
mysql> drop view stu_class_v;  
Query OK, 0 rows affected (0.39 sec)
```

DML 操作视图

因为视图本身没有数据，因此对视图进行的 dml 操作最终都体现在基表中。视图的 DML 操作，不是所有的视图都可以做 DML 操作。

有下列内容之一，视图不能做 DML 操作：

- ①select 子句中包含 distinct
- ②select 子句中包含组函数
- ③select 语句中包含 group by 子句
- ④select 语句中包含 order by 子句
- ⑤select 语句中包含 union 、 union all 等集合运算符
- ⑥where 子句中包含相关子查询
- ⑦from 子句中包含多个表
- ⑧如果视图中有计算列，则不能更新
- ⑨如果基表中有某个具有非空约束的列未出现在视图定义中，则不能做 insert 操作

with check option

含义：对视图所做的 DML 操作的结果，不能违反视图的 WHERE 条件的限制

嵌套视图：定义在另一个视图的上面的视图

```
mysql> create view v_ear_veterans  
-> as  
-> select * from v_veterans  
-> where JOINED < 1980;
```

使用 WITH CHECK OPTION 约束时，(不指定选项则默认是 CASCDED)

可以使用 CASCDED 或者 LOCAL 选项指定检查的程度：

- ①WITH CASCDED CHECK OPTION：检查所有的视图

例如：嵌套视图及其底层的视图

- ②WITH LOCAL CHECK OPTION：只检查将要更新的视图本身

对嵌套视图不检查其底层的视图

事务安全

事务概念

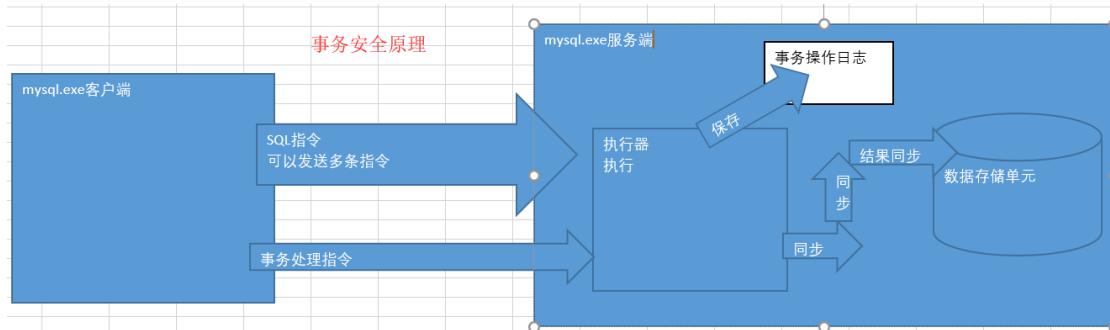
- Transaction
- 事务：一个最小的不可再分的工作单元；通常一个事务对应一个完整的业务(例如银行账户转账业务，该业务就是一个最小的工作单元)
- 一个完整的业务需要批量的 DML(insert、update、delete)语句共同联合完成
- 事务只和 DML 语句有关，或者说 DML 语句才有事务。这个和业务逻辑有关，业务逻辑不同，DML 语句的个数不同

事务由事务开始到事务结束之间执行的全体操作组成。

- 开启事务：Start Transaction
- 事务结束：End Transaction
- 提交事务：Commit Transaction
- 回滚事务：Rollback Transaction

事务基本原理

基本原理：mysql 允许将事务统一进行管理（存储引擎 innodb），将用户所做的操作暂时保存起来，不直接放到数据表（更新），等到用户确认结果之后再进行操作。



事务在 mysql 中通常是自动提交的，但是也是可以使用手动事务。

自动事务

自动事务：autocommit，当客户端发送一条 SQL 指令（写操作：增删改）给服务器的时候，服务器在执行之后，不用等待用户反馈结果，会自动将结果同步到数据表。

证明：打开两个客户端，一个客户端执行 SQL 指令，另一个客户端查看执行结果。

The screenshot shows two MySQL command-line interfaces. The left client (mysql1) has the following session:

```
mysql> select * from my_class;
+----+----+
| id | name |
+----+----+
| 1  | 1   |
| 3  | 3   |
+----+----+
2 rows in set (0.00 sec)

mysql> insert into my_class values(4, '4');
Query OK, 1 row affected (0.54 sec)

mysql>
```

The right client (mysql2) has the following session:

```
Database changed
mysql> select * from my_class;
+----+----+
| id | name |
+----+----+
| 1  | 1   |
| 3  | 3   |
+----+----+
2 rows in set (0.00 sec)

mysql> select * from my_class;
+----+----+
| id | name |
+----+----+
| 1  | 1   |
| 3  | 3   |
| 4  | 4   |
+----+----+
3 rows in set (0.00 sec)
```

A red arrow points from the '4' inserted in mysql1 to the fourth row in mysql2's result set. Red annotations include '系统在执行SQL指令之后，自动将结果同步到数据表' (The results are automatically synchronized to the data table after executing the SQL instruction) and '证明：打开' (Prove: Open).

自动事务：系统做了额外的步骤来帮助用户操作，系统是通过变量来控制的。autocommit

The screenshot shows the MySQL command-line interface with the following session:

```
mysql> show variables like 'autocommit';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | ON   |
+-----+-----+
```

A red box highlights the 'ON' value, with the annotation '自动提交开启' (Autocommit is enabled).

关闭自动事务：关闭之后系统就不再帮助用户提交结果了。

set autocommit = off;

The screenshot shows the MySQL command-line interface with the following session:

```
1 row in set, 1 warning (0.00 sec)

mysql> -- 关闭自动事务
mysql> set autocommit = off;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'autocommit';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | OFF  |
+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

查看执行结果

The screenshot shows the MySQL command-line interface with the following session:

```
mysql> show variables like 'autocommit';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | OFF  |
+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> insert into my_class values(4, '4');
Query OK, 1 row affected (0.54 sec)

mysql> select * from my_class;
+----+----+
| id | name |
+----+----+
| 1  | 1   |
| 3  | 3   |
| 4  | 4   |
+----+----+
3 rows in set (0.00 sec)

mysql>
```

A red box highlights the 'OFF' value of autocommit. A red annotation at the bottom says '虽然执行成功，但是数据没有同步' (Execution was successful, but the data was not synchronized). Another red annotation on the right says '另外客户端看不到数据' (The other client cannot see the data).

一旦自动事务关闭，那么需要用户提供是否同步的命令

Commit: 提交（同步到数据表，事务会被清空）

Rollback: 回滚（清空之前的操作，不要了）

事务没有提交的对比查看：在执行事务端的客户端中，系统在进行数据查看的时候会利用事务日志进行数据加工、

```
mysql> insert into my_class values(5,'5');
Query OK, 1 row affected (0.00 sec)

mysql> select * from my_class;
+----+----+
| id | name |
+----+----+
| 1  | 1   |
| 3  | 3   |
| 4  | 4   |
| 5  | 5   |
+----+----+
4 rows in set (0.00 sec)

mysql>
```

执行成功，虽然没有同步到
数据表，
但是当前客户端查询的时候，
系统会根据日志表中的操作
进行数据加工

```
mysql> insert into my_class values(4,'4');
Query OK, 1 row affected (0.54 sec)

mysql> select * from my_class;
+----+----+
| id | name |
+----+----+
| 1  | 1   |
| 3  | 3   |
| 4  | 4   |
| 5  | 5   |
+----+----+
3 rows in set (0.00 sec)

mysql>
```

事务提交

```
4 rows in set (0.00 sec)

mysql> -- 提交
mysql> commit
->;
Query OK, 0 rows affected (0.43 sec)

mysql>
```

```
mysql> select * from my_class;
+----+----+
| id | name |
+----+----+
| 1  | 1   |
| 3  | 3   |
| 4  | 4   |
| 5  | 5   |
+----+----+
```

通常，我们不会关闭自动事务，这样操作麻烦。因此只会在需要使用事务处理的时候，才会进行操作（手动事务）

手动事务

手动事务：不管是开始还是过程还是结束，都需要用户（程序员）手动发送对应的事务操作指令来实现。

手动事务的命令：

1、start transaction; 开启事务：从这条语句开始，后面的所有语句都不会直接写入到数据表（保存在事务日志中）

2、事务处理：多个指令构成

3、事务提交：commit/rollback，到这个时候所有的事务才算结束

开始事务

```
mysql> -- 开启事务
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

执行事务

将多个连续的但是是一个整体的 SQL 指令，逐一执行。

提交事务

确认提交：commit，数据写到数据表，清空事务日志

回滚操作： rollback，所有数据无效并清空

回滚点

回滚点： savepoint，当有一系列的事务操作时，而其中的步骤如果成功了没有必要重新来过，那么可以在某个成功点设置一个记号（回滚点），如果后面有失败，那么可以回到这个记号位置。

回滚点在自动事务关闭后才能使用。set autocommit = off;

1、增加回滚点： savepoint 回滚点名字； //字母数字和下划线构成

2、回到回滚点： rollback to 回滚点名字； //那个记号（回滚点）之后的所有操作没有了

注意：如果在一个事务处理汇总，如果有很多步骤，那么可以设置多个回滚点。但是如果回到了前面的回滚点，后面的回滚点就失效。

事务特点

事务具有 4 个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为 ACID 特性

原子性 (atomicity)：一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做

事务从事务开始起到事务提交，要么所有操作都成功，要么所有操作都失败。

一致性 (consistency): 事务必须使数据库从一致性状态变到另一个一致性状态，一致性和原子性是密切相关的。

数据表中的数据修改，要么是所有操作一次性都修改，要么根本不动。

隔离性 (isolation):

一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰

一个客户端在使用事务操作一个数据（可能是一行或者整表）的时候，另外一个客户端不能对该数据进行操作。

```
for the right syntax to use near 'update my_stu set stu_age' | mysql> update my_stu set stu_age = 50 where stu_id = 'stu001';    mysql> update my_stu set stu_age = 50 where stu_id = 'stu001';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> update my_stu set stu_age = 50 where stu_id = 'stu001';
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

另外一个客户端操作同一个记录的时候，发生等待（隔离性）

什么时候是行被隔离？什么时候是整表被隔离？

说明：如果条件中使用了索引（主键），那么系统是根据主键直接找到某条记录，这个时候与其他记录无关，那么只隔离某条记录；反之，如果说系统是通过全表检索（每一条记录都去检查，没有索引），被检索的所有数据都会被锁定（整表）

持久性 (durability): 持久性也称永久性 (permanence)，指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

变量

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

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**

查看系统所有变量

```
show variables;
```

```
+-----+
547 rows in set, 1 warning (0.00 sec)
```

Mysql 允许用户使用 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 (0.35 sec)
```

修改系统变量

分为两种修改方式:

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;

```
mysql> -- 会话修改系统变量
mysql> set autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

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

mysql> update my_stu set stu_age =
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warni

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

mysql> -
```

2、全局修改: 针对所有客户端, “所有时刻”都有效

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

全局修改后, 所有连接的客户端并没有发现改变? 全局修改只针对新客户端生效 (正在连着的无效)

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;)

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。**

定义用户变量： set @变量名 = 值；

```
mysql> -- 定义用户变量
mysql> set @name = 'hello world';
Query OK, 0 rows affected (0.36 sec)
```

在 mysql 只因为没有比较符号 ==，所以使用=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： := 赋值： set @变量名 := 值；

```
mysql> -- 使用专用赋值符号
mysql> set @age := 50;
Query OK, 0 rows affected (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：查询得到的数据只能是一行数据（一个变量对应一个字段值）：mysql 中没有数组

1、赋值且查看赋值过程： select @变量 1 := 字段 1, @变量 2 := 字段 2 from 数据表 where 条件；

错误语法：就是因为使用=，系统会当作比较符号来处理

```
mysql> -- 通过查询数据为变量赋值
mysql> select @name = stu_name, @age = stu_age from my_stu limit 1;
+-----+-----+
| @name = stu_name | @age = stu_age |
+-----+-----+
|          0          |          0          |
+-----+-----+
1 row in set (0.35 sec)
```

正确使用： :=

```
mysql> select @name := stu_name, @age := stu_age from my_stu;
+-----+-----+
| @name := stu_name | @age := stu_age |
+-----+-----+
| 夏洛           |      18 |
| 张四           |      28 |
| 张五           |      20 |
| 晓婷           |      25 |
| 小猪           |      30 |
+-----+-----+
5 rows in set (0.00 sec)
```

2、只赋值不看过程： select 字段 1, 字段 2...from 数据源 where 条件 into @变量 1, @变量 2...；

```
mysql> select stu_name,stu_age from my_stu order by stu_height desc limit 1 into @name,@age;
Query OK, 1 row affected (0.34 sec)
```

```
mysql> select @name := stu_name,@age := stu_age from my_stu;
+-----+-----+
| @name := stu_name | @age := stu_age |
+-----+-----+
| 夏洛 | 18 |
| 张四 | 28 |
| 张五 | 20 |
| 晓婷 | 25 |
| 小猪 | 30 |
+-----+-----+
5 rows in set (0.00 sec)

mysql> select @name,@age;
+-----+-----+
| @name | @age |
+-----+-----+
| 小猪 | 30 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select @name := stu_name,@age := stu_age from my_stu limit 3;
+-----+-----+
| @name := stu_name | @age := stu_age |
+-----+-----+
| 夏洛 | 18 |
| 张四 | 28 |
| 张五 | 20 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> select @name,@age;
+-----+-----+
| @name | @age |
+-----+-----+
| 张五 | 20 |
+-----+-----+
```

查看变量的值：只返回了最后一个

=

只有在 set 和 update 时才是和:=一样，赋值的作用，其它都是等于的作用。

:=

不只在 set 和 update 时时赋值的作用，在 select 也是赋值的作用。

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（begin end 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[属性];

```
declare res int default 0;
```

```
declare res int = 0;
```

流程结构

流程结构：代码的执行顺序

If 分支

基本语法

If 在 mysql 中有两种基本用法：

1、用在 select 查询当中，当作一种条件来判断

基本语法：if (条件, 为真结果, 为假结果)

最好取别名 if (条件, 为真结果, 为假结果) as 别名

```
mysql> select *, if(stu_age > 20, '符合', '不符合') as judge from my_stu;
+-----+-----+-----+-----+-----+-----+-----+
| stu_id | stu_name | class_id | stu_age | stu_height | gender | judge |
+-----+-----+-----+-----+-----+-----+-----+
| stu001 | 夏洛     | NULL      | 18       | 185        | 男     | 不符合 |
| stu002 | 张四     | NULL      | 28       | 175        | 女     | 符合   |
| stu003 | 张五     | 1         | 20       | 165        | 男     | 不符合 |
| stu004 | 晓婷     | 1         | 25       | 195        | 女     | 符合   |
| stu005 | 小猪     | 1         | 30       | 172        | 女     | 符合   |
+-----+-----+-----+-----+-----+-----+-----+
5 rows in set, 1 warning (0.35 sec)
```

2、用在复杂的语句块中（函数/存储过程/触发器）

基本语法：

if 条件表达式 then

 满足条件要执行的语句；

end if;

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

 满足条件要执行的语句；

```
    end if;  
end if;
```

while 循环

循环体都是需要在大型代码块中使用。

基本语法

```
while 条件 do  
    循环体;  
end while;
```

结构标识符

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**）

基本语法：

```
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];
```

case 循环

基本语法：

```
case [条件]
when 条件 1 then 语句 1
when 条件 2 then 语句 2
....
else 语句 n
end case
```

case 可以用在 select 语句中，但不能用在 where 语句中。

1、判断的同时改变其值

```
select OperatorAccount,
       case
           when CreateTime>'2016-02-14 16:24:42' then 'after'
           when CreateTime<'2016-02-14 16:24:42' then 'before'
           else 'now' end stage
  from log_login order by CreateTime DESC
```

log_login (2×1,075)	
OperatorAccount	stage
dumini	after
admin	after
liyanqiu	after
admin	after
zhengchunlei	after
zhengchunlei	now
admin	before
admin	before
zhuwenkai	before
admin	before

2、拆分一行为多列

函数

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）

内置函数

字符串函数

Mysql8 字符串函数

https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_instr

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）

concat():连接字符串

instr():判断字符在目标字符串中是否存在，其开始位置从 1 开始，存在返回其位置，不存在返回 0

lcase():全部小写

left():从左侧开始截取到指定位置(位置如果超过长度，截取所有)

ltrim():消除左边对应的空格

mid():从中间位置开始截取，如果不指定截取长度，直接到最后

时间函数

Mysql8 时间函数

<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>

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

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（引号包裹）

date_add(日期, interval 时间数字 type):进行时间的增加

type: day/hour/minute/second

unix_timestamp():获取时间戳

from_unixtime():将指定时间戳转换成对应的日期时间格式

```

mysql> SELECT FROM_UNIXTIME(1447430881);
+-----+
| FROM_UNIXTIME(1447430881) |
+-----+
| 2015-11-14 00:08:01 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT FROM_UNIXTIME(1447430881) + 0;
+-----+
| FROM_UNIXTIME(1447430881) + 0 |
+-----+
| 20151114000801 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x');
+-----+
| FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x') |
+-----+
| 2018 10th August 08:20:21 2018 |
+-----+
1 row in set (0.00 sec)

```

数学函数

abs():绝对值

ceiling():向上取整

floor():向下取整

pow():求指数

rand():获取一个随机数(0-1)之间

round():四舍五入函数

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():显示当前所在的数据库

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 | 8.0.11 | mydatabase | 15085ef6-9c35-11e8-966e-80fa5b4ce580 |
+-----+-----+-----+-----+

```

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、函数内部的每一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，意味着整体不能中断。

3、mysql 一旦见到语句结束符分号，就会自动开始执行（问题）

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；

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 ;

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

函数定义基本语法：

修改语句结束符

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

语句结束符

修改语句结束符（改回来）

报：ERROR 1418 (HY000)

```
mysql> -- 创建自定义函数
mysql>
mysql> -- 修改语句结束符
mysql> delimiter $$ 
mysql> create function my_func1() returns int
-> begin
-> return 10;
-> end
-> -- 结束
-> $$
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
mysql>
```

解决方法: <https://www.cnblogs.com/xihong2014/p/5566383.html>

设置方法有三种:

- 1.在客户端上执行`SET GLOBAL log_bin_trust_function_creators = 1;`
- 2.MySQL启动时, 加上`--log-bin-trust-function-creators`选贤, 参数设置为1
- 3.在MySQL配置文件my.ini或my.cnf中的[mysqld]段上加`log-bin-trust-function-creators=1`

会话级别

参考: MySQL pho

```
mysql> delimiter ;  
mysql> -- 创建自定义函数  
mysql>  
mysql> -- 修改语句结束符  
mysql> delimiter $$  
mysql> create function my_func2() returns int  
    -> begin  
    -> return 10;  
    -> end  
    -> -- 结束  
    -> $$  
Query OK, 0 rows affected (0.41 sec)  
  
mysql>  
mysql> -- 修改语句结束符  
mysql> delimiter ;  
mysql>
```

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。

```
mysql> delimiter ;  
mysql> -- 最简函数  
mysql> create function my_func3() returns int  
    -> return 100;  
Query OK, 0 rows affected (0.40 sec)
```

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 my_func4(int_1 int, int_2 int) returns int  
    -> return int_1 + int_2;  
Query OK, 0 rows affected (0.40 sec)
```

查看函数

1、可以通过查看 function 状态，查看所有的函数

```
show function status [like 'pattern'];
```

```
mysql> -- 查看所有函数
mysql> show function status\G
***** 1. row *****
    Db: mydatabase
    Name: my_func1
    Type: FUNCTION
    Definer: root@localhost
    Modified: 2018-08-10 08:52:31
    Created: 2018-08-10 08:52:31
    Security_type: DEFINER
    Comment:
character_set_client: gbk
collation_connection: gbk_chinese_ci
Database Collation: utf8_general_ci
***** 2. row *****
```

函数所属数据库：只有在这个数据库中才能使用该函数

2、可以查看这个 函数的创建语句

```
show create function 函数名;
```

```
mysql> show create function my_func1\G
***** 1. row *****
    Function: my_func1
    sql_mode: ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO,
O_ENGINE_SUBSTITUTION
    Create Function: CREATE DEFINER='root'@'localhost' FUNCTION `my_func1`() 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)
```

调用函数

自定义函数的调用和内置函数的调用是一样的： select 函数名 (实参列表);

```
mysql> -- 调用函数
mysql> select my_func1(), my_func3(), my_func4(100, 1000);
+-----+-----+-----+
| my_func1() | my_func3() | my_func4(100, 1000) |
+-----+-----+-----+
|      10   |     100  |       1100  |
+-----+-----+-----+
1 row in set (0.00 sec)
```

修改函数

```
alter function 函数名
```

删除函数

删除自函数: drop function 函数名;

```
on for the right syntax to use near ,my_
mysql> drop function my_func2;
Query OK, 0 rows affected (0.43 sec)

mysql> drop function my_func1;
Query OK, 0 rows affected (0.12 sec)
```

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
- 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): select 字段 into @变量；（唯一可用）

在 mysql 的 trigger 和 function 中不能出现 select * from table 形式的查询，因为其会返回一个结果集；而这在 mysql 的 trigger 和 function 中是不可接受的，但是在存储过程中可以。在 trigger 和 function 中可以使用 select ... into ...形式的查询。

流程结构案例

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

设计：

- 1、创建函数
 - 2、需要一个形参，确定要累加到什么为止
 - 3、在函数体内部需要定义一个变量保存对应的结果 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]

```
-- 创建一个自动求和的函数
-- 修改语句结束符
delimiter $$

-- 创建函数
create function my_sum(end_value int) returns int
begin
    -- 声明变量（局部变量）：如果使用declare 声明变量，必须在函数体其他语句之前
    declare res int default 0;

end
|
-- 结束
$$

-- 修改语句结束符
delimiter ;
```

4、内部需要一个循环来实现迭代累加

```
-- 声明变量（局部变量）：如果使用declare 声明变量，必须在函数体其他语句之前
declare res int default 0;
declare i int default 1;
-- 循环处理
while i <= end_value do
    -- 修改变量 累加结果
    set res = res + i; -- mysql中没有++
    set i = i + 1;
end while;
```

5、循环内部需要进行条件判断控制：5 的倍数

```
-- 循环处理
mywhile:while i <= end_value do
    -- 判断当前数据是否合理
    if i % 5 = 0 then
        -- 5的倍数不要
        set i = i + 1;
    iterate mywhile;
end if;
-- 修改变量 累加结果
set res = res + i; -- mysql中没有++
set i = i + 1;
end while mywhile;
```

6、函数必须有返回值

```
        set i = i + 1
    end while mywhile;
    -- 返回值
    return res;
```

定义函数结构完成

-- 创建一个自动求和的函数

```

-- 修改语句结束符
delimiter $$

-- 创建函数
create function my_sum(end_value int) returns int
begin
    -- 声明变量（局部变量）：如果使用 declare 声明变量，必须在函数体其他语句之前
    declare res int default 0;
    declare i int default 1;
    -- 循环处理
    mywhile:while i <= end_value do
        -- 判断当前数据是否合理
        if i % 5 = 0 then
            -- 5 的倍数不要
            set i = i + 1;
        iterate mywhile;
    end if;
    -- 改变量 累加结果
    set res = res + i; -- mysql 中没有++
    set i = i + 1;
end while mywhile;
-- 返回值
return res;
end

```

-- 结束
\$\$

```
-- 修改语句结束符
delimiter ;
```

调用函数： select 函数名 (实参);

```

mysql> delimiter ;
mysql> -- 调用函数
mysql> select my_sum(100), my_sum(-100);
+-----+-----+
| my_sum(100) | my_sum(-100) |
+-----+-----+
|      4000 |          0 |
+-----+-----+
1 row in set (0.00 sec)

```

变量作用域

变量作用域：变量能够使用的区域范围

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。

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

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 = 'jack';
Query OK, 0 rows affected (0.00 sec)
mysql> create function my_func1() returns char(5)
rt -> return @name;
Query OK, 0 rows affected (0.41 sec)
mysql> select my_func1();
+-----+
| my_func1() |
+-----+
| jack       |
1 row in set, 1 warning (0.00 sec)
```

会话变量可以跨库

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

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。

存储过程

存储过程的概念

存储过程（stored procedure）是在大型数据库系统中，一组为了完成特定功能的 SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译，（效率比较高）用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象（针对 SQL 编程而言）。

存储过程：简称过程

与函数的区别

相同点

- 1、存储过程和函数的目的都是为了可重复地执行操作数据库的 SQL 语句的集合(代码复用)
- 2、存储过程、函数都是一次编译，后续执行

不同点

- 1、标识符不同。函数的标识符为 function，过程为 procedure
- 2、函数中有返回值，且必须返回，而过程没有返回值
- 3、过程无返回值类型，不能将结果直接赋值给变量；函数有返回值类型，调用时，除在 select 中，必须将返回值赋给变量
- 4、函数可以在 select 语句中直接使用，而过程不能。函数是使用 select 调用，而过程不是

存储过程操作

创建过程

基本语法：

```
create procedure 过程名字([参数列表])
begin
    过程体
end
结束符号
```

如果过程体中只有一条指令，那么可以省略 begin 和 end

```
-- 创建过程

create procedure my_pro1()
select * from my_stu;
```

过程基本上也可以完成函数对应的所有功能

```
delimiter $$ 
create procedure my_pro2()
begin
    -- 求1到100之间累加和
    declare i int default 1;
    -- declare sum int default 0; -- 局部变量
    set @sum = 0; -- 会话变量

    -- 开始循环获取结果
    while i <= 100 do
        -- 求和
        set @sum = @sum + i;
        set i = i + 1;
    end while;

    -- 显示结果
    select @sum;

end
$$
delimiter ;
```

查看过程

查看过程和查看函数完全一样，除了关键字。

查看全部存储过程： show procedure status [like 'pattern'];

查看过程的创建语句： show create procedure 过程名字；

调用过程

过程，没有返回值，select 不可能调用

调用过程有专门的语法：call 过程名 ([实参列表]) ;

```
mysql> call my_pro2();
+-----+
| @sum |
+-----+
| 5050 |
+-----+
1 row in set (0.01 sec)
```

删除过程

基本语法：drop procedure 过程名字；

存储过程的形参类型

存储过程也允许提供参数（形参和实参），存储过程的参数也和函数一样，需要指定其类型。

但是存储过程对参数还有额外的要求：自己的参数分类

in

表示参数从外部传入到里面使用（过程内部使用）：可以是直接数据也可以是保存数据的变量

out

表示参数是从过程里面把数据保存到变量中，交给外部使用：传入的必须是变量

如果说传入的 out 变量本身在外部有值，那么在进入过程之后第一件事就是被清空设为 NULL

inout

数据可以从外部传入到过程内部使用，同时内部操作之后，又会将数据返还给外部。

参数使用基本语法:

过程类型 变量名 数据类型; //in int_1 int;

```
-- 创建三个外部变量
set @n1 = 1;
set @n2 = 2;
set @n3 = 3;

-- 开始创建过程
delimiter $$ 
create procedure my_pro3(in int_1 int,out int_2 int,inout int_3 int)
begin
    -- 查看三个传入进来的数据的值
    select int_1,int_2,int_3;

    -- 修改三个变量的值
    set int_1 = 10;
    set int_2 = 100;
    set int_3 = 1000;

    select int_1,int_2,int_3;
    -- 查看会话变量
    select @n1,@n2,@n3;

    -- 修改会话变量
    set @n1 = 'a';
    set @n2 = 'b';
    set @n3 = 'c';

    select @n1,@n2,@n3;
end
$$
delimiter ;

-- 调用过程

-- 调用过程
call my_pro3(@n1,@n2,@n3);
```

```
-- 开始创建过程
delimiter $$
create procedure my_pro3(in int_1 int,out int_2 int,inout int_3 int)
begin
    -- 查看三个传入进来的数据的值
    select int_1,int_2,int_3;

    -- 修改三个变量的值
    set int_1 = 10;
    set int_2 = 100;
    set int_3 = 1000;

    select int_1,int_2,int_3;
    -- 查看会话变量
    select @n1,@n2,@n3;

    -- 修改会话变量
    set @n1 = 'a';
    set @n2 = 'b';
    set @n3 = 'c';

    select @n1,@n2,@n3;
end
$$
delimiter ;
```

mysql> -- 调用过程
mysql> call my_pro3(@n1,@n2,@n3);

int_1	int_2	int_3
1	NULL	3

1 row in set (0.00 sec)

int_1	int_2	int_3
10	100	1000

1 row in set (0.00 sec)

@n1	@n2	@n3
1	2	3

1 row in set (0.01 sec)

@n1	@n2	@n3
a	b	c

1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql>

在执行之后，再次查看会话变量。

```
mysql> select @n1,@n2,@n3;
+-----+
| @n1 | @n2 | @n3 |
+-----+
| a   | 100 | 1000 |
+-----+
1 row in set (0.00 sec)

mysql>
```

```
-- 开始创建过程
delimiter $$
create procedure my_pro3(in int_1 int,out int_2 int,inout int_3 int)
begin
    -- 查看三个传入进来的数据的值
    select int_1,int_2,int_3;

    -- 修改三个变量的值
    set int_1 = 10;
    set int_2 = 100;
    set int_3 = 1000;

    select int_1,int_2,int_3;
    -- 查看会话变量
    select @n1,@n2,@n3;

    -- 修改会话变量
    set @n1 = 'a';
    set @n2 = 'b';
    set @n3 = 'c';

    select @n1,@n2,@n3;
end
```

~~走到end代表过程结束：开始工作，判断变量是否是out或者inout类型；如果是，将内部代替out和inout变量的对应形参的值重新赋值给外部变量；会将外部变量本身的值给覆盖~~

~~当out类型和inout类型的数据传入之后，实际上没有改变外部变量的值，而是把值给了形参，但是形参会将out类型的值清除为NULL~~

mysql> -- 调用过程
mysql> call my_pro3(@n1,@n2,@n3);

@n1	@n2	@n3
1	2	3

1 row in set (0.01 sec)

@n1	@n2	@n3
a	b	c

1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> select @n1,@n2,@n3;

@n1	@n2	@n3
a	100	1000

1 row in set (0.00 sec)

mysql>

触发器

触发器概念

基本概念

触发器是一种特殊类型的存储过程，它不同于存储过程。触发器主要是通过事件进行触发而被执行的，而存储过程可以通够存储过程名字而被直接调用。

触发器：trigger，是一种非常接近于js中的事件的知识，提前给某张表的所有记录绑定一段代码，如果该行的操作满足条件（触发），这段提前准备好的代码就会自动执行。

作用

- 1、可在写入数据表之前，强制检验或者转换数据（保证数据安全）
- 2、触发器发生错误时，异动的结果会被撤销（如果触发器执行错误，那么前面用户已经执行成功的操作也会被撤销：事务安全）
- 3、部分数据库管理系统可以针对数据定义语言（DDL）使用触发器，称为DDL触发器
- 4、可依照特定的情况，替换异动的指令（instead of）。（mysql不支持）

触发器优缺点

优点

- 1、触发器可通过数据库中的相关表实现级联更改。（如果某张表的数据改变，可以利用触发器来实现其他表的无痕操作（用户不知道））
- 2、保证数据安全，进行安全校验

缺点

- 1、对触发器过分的依赖，势必影响数据库的结构，同时增加了维护的复杂程度
- 2、造成数据在程序层面不可控。（PHP层）

触发器基本语法

创建触发器

基本语法

```
create trigger 触发器名字 触发时机 触发时间 on 表 for each row  
begin  
end
```

触发对象: on 表 for each row , 触发器绑定实质是表中的所有行, 因此当每一行发生指定的改变的时候, 就会触发触发器。

触发时机

触发时机: 每张表中对应的行都会有不同的状态, 当 SQL 指令发生的时候, 都会令行中数据发生改变, 每一行总会有两种状态, 数据操作前和操作后

before: 在表中数据发生改变前的状态

after: 在表中数据已经发生改变后的状态

触发事件

触发事件: mysql 中触发器针对的目标是数据发生改变, 对应的操作只有写操作 (增删改)

insert: 插入操作

update: 更新操作

delete: 删除操作

注意事项:

一张表中, 每一个触发时机绑定的触发事件对应的触发器类型只能有一个: 一张表中只能有一个对应 after insert 触发器

因此, 一张表中最多的触发器只能有 6 个: before insert, before update, before delete, after insert, after update, after delete

查看触发器

1、查看全部触发器

```
show triggers;
```

2、查看触发器创建语句

```
show create trigger 触发器名字;
```

触发触发器

让触发器指定的表中，对应的时机发生对应的操作即可。

删除触发器

基本语法： drop trigger 触发器名字;

触发器应用

记录关键字： new 、 old

触发器针对的是数据表中的每条记录（每行），每行在数据操作前后都有一个对应的状态，触发器在执行之前就将对应的状态获取到了，将没有操作之前的状态（数据）都保存到 old 关键字中，而操作后的状态都放到 new 中

在触发器中，可以通过 old 和 new 关键字来获取绑定表中对应的记录数据

基本语法： **关键字.字段名**

old 和 new 并不是所有的触发器都有

insert: 插入之前为空，没有 old

delete: 清空数据，没有 new

注意：

触发器（TRIGGER）不能使用 select 来设置变量结果，这样会返回结果集，如果在触发器（TRIGGER）中要写成 into 的方式。

SELECT '没有这个生的信息，不可选课！';

改为：

select '没有这个生的信息，不可选课！' into @args;

商品自动扣除库存

需求：有两张表，一张是商品表，一张是订单表（订单中会保留商品 id），每次订单生成，商品表中对应的库存就应该发生变化。

1、创建两张表：商品表，订单表

2、创建触发器：如果订单表发生数据插入，对应的商品就应该减少库存

```
create trigger 名字 after insert on my_orders for each row
```

3、触发触发器

代码：

-- 创建两张表

```
create table my_goods(
id int primary key auto_increment,
name varchar(20) not null,
inv int
)charset utf8;
```

```
create table my_orders(
id int primary key auto_increment,
goods_id int not null,
goods_num int not null
)charset utf8;
```

```
insert into my_goods values(null,'手机',1000),(null,'电脑',5000),(null,'平板',100);
```

-- 自动扣除商品库存的触发器

delimiter \$\$

```
create trigger a_i_o_t after insert on my_orders for each row
```

begin

-- 更新商品库存：new 代表新增的订单

```
update my_goods set inv = inv - new.goods_num where id = new.goods_id;
```

end

\$\$

delimiter ;

-- 触发触发器

```
insert into my_orders values(null,3,5);
```

```

mysql> select * from my_goods;
+----+-----+-----+
| id | name | inv |
+----+-----+-----+
| 1  | 手机 | 1000 |
| 2  | 电脑 | 5000 |
| 3  | 平板 | 95   |
+----+-----+-----+
3 rows in set (0.00 sec)

```

完善：如果库存数量没有商品订单多怎么办？

操作目标：订单表，操作时机：下单前；操作事件：插入事件

```

-- 判断库存
delimiter $$

create trigger b_i_o_t before insert on my_orders for each row
begin
    -- 取出库存数据进行判断
    select inv from my_goods where id = new.goods_id into @inv;

    -- 判断
    if @inv < new.goods_num then
        -- 中断操作：暴力解决，主动出错
        insert into XXX values('XXX');
    end if;
end
$$
delimiter ;

```

insert into my_orders values(null,3,100);

触发器一旦出错，就会撤销之前操作。

```

mysql> insert into my_orders values(null,3,100);
ERROR 1146 (42S02): Table 'mydatabase.XXX' doesn't exist
mysql> select * from my_orders;
+----+-----+-----+
| id | goods_id | goods_num |
+----+-----+-----+
| 1  | 3       | 5       |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from mu_goods;
ERROR 1146 (42S02): Table 'mydatabase.mu_goods' doesn't exist
mysql> select * from my_goods;
+----+-----+-----+
| id | name | inv |
+----+-----+-----+
| 1  | 手机 | 1000 |
| 2  | 电脑 | 5000 |
| 3  | 平板 | 95   |
+----+-----+-----+
3 rows in set (0.00 sec)

```

游标

游标（Cursor）是处理数据的一种方法，为了查看或者处理结果集中的数据，游标提供了在结果集中一次一行或者多行前进或向后浏览数据的能力。可以把游标当作一个指针，它可以指定结果中的任何位置，然后允许用户对指定位置的数据进行处理。

有数据缓冲的思想：游标的*设计*是一种数据缓冲区的思想，用来存放 SQL 语句执行的结果。**先有数据基础：**游标是在先从数据表中检索出数据之后才能继续灵活操作的技术。**类似于指针：**游标类似于指向数据结构堆栈中的指针，用来 *pop* 出所指向的数据，并且只能**每次取一个**。

主用在循环处理、存储过程、函数中使用。

游标的使用一般分为 5 个步骤，主要是：**定义游标->打开游标->使用游标->关闭游标->释放游标。**

```
-- 定义变量
declare testrangeid BIGINT;
declare versionid BIGINT;
declare done int;
-- 创建游标，并存储数据
declare cur_test CURSOR for
    select id as testrangeid,version_id as versionid from tp_testrange;
-- 游标中的内容执行完后将done设置为1
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;
-- 打开游标
open cur_test;
-- 执行循环
posLoop:LOOP
-- 判断是否结束循环
IF done=1 THEN
    LEAVE posLoop;
END IF;
-- 取游标中的值
FETCH cur_test into testrangeid,versionid;
-- 执行更新操作
    update tp_data_execute set version_id=versionid where testrange_id = testrangeid;
END LOOP posLoop;
-- 释放游标
CLOSE cur_test;
```

jdbc

<http://www.runoob.com/java/java-mysql-connect.html>