# 注意事项：

## 1.使用pymysql时：

sql = **'insert into qiubai values("%s","%s")'**%(item[**'author'**],item[**'content'**])

注意语句中，%s是加引号的

## 2.将字符创转成datetime类型进行存贮：

sql = **'insert into qiubai values(str\_to\_date("%s"，"%%Y-%%m-%%d %%H:%%i:%%S**

**"))'**%(item[**'datetime'**])

使用双%是因为一个%会被转义掉。

str\_to\_date第二个参数是按照你输入的字符串的时间格式来确定，不是固定的。

## 3.使用pymysql出现Latin-1 error时，

self.conn = pymysql.Connect(host=**'127.0.0.1'**, port=3306, user=**'root'**, password=**'111'**, db=**'proxy\_ip'**, charset=**'utf8'**)

设定字符编码格式为charset=’utf8’

# 一、mysql的安装及初始化

mysql：解压即安装。将文件夹下bin文件夹添加入系统环境变量。我的电脑🡪高级🡪环境变量。

## 将数据库设置为系统服务

在cmd中 输入mysql --install 将mysql设置为系统服务

mysql --remove 解除mysql系统服务。

tasklist |findstr mysql 查找数据库pid

taskkill /F /PID 进程号 结束该进程

查看当前用户

select user(); 查看当前用户

exit 客户端退出

mysql –u用户名 –p密码 启动mysql 用户名 密码。.

mysql –uroot –p111

mysqladmin –u用户名 –p密码 password ”新密码”

修改密码

本次修改root密码为111

net start MySQL 启动服务端

**破解密码：**

net stop MySQL 关闭服务端，此项因为数据库已加入windows服务列表。

mysqld --skip–grant-tables 在cmd中跳过mysql授权表启动数据库服务端

mysql –uroot –p 然后登陆root账号,此时不需密码。

update mysql.user set password=password(“111”) where user=”root” and host =”localhost”;



然后杀死进程，重启mysql。

\s 查看字符编码和一系列信息

## 数据库配置

#在mysql的解压目录下，新建my.ini,然后配置

#1. 修改配置文件

[mysqld]

default-character-set=utf8

[client]

default-character-set=utf8

#解压的目录

basedir=E:\mysql-5.7.19-winx64

#data目录

datadir=E:\my\_data #在mysqld --initialize时，就会将初始数据存入此处指定的目录，在初始化之后，启动mysql时，就会去这个目录里找数据

[mysql]

default-character-set=utf8

#mysql5.5以上：修改方式有所改动

[mysqld] #针对mysqld设置的配置。即服务端

character-set-server=utf8

collation-server=utf8\_general\_ci

[client] #针对所有客户端设置默认配置，如果客户端没有专门的设置，则自动加载本设置。

default-character-set=utf8

[mysql] 针对mysql设置的默认设置。客户端的一个

port=3306

default-character-set=utf8

user=root

password=123456 #设置登录默认客户。

#2. 重启服务

#3. 查看修改结果：

\s

show variables like '%char%'

# 二、初识sql语句

SQL语言主要用于存取数据、查询数据、更新数据和管理关系数据库系统,SQL语言由IBM开发。SQL语言分为3种类型：

1、DDL语句 数据库定义语言： 数据库、表、视图、索引、存储过程，例如CREATE DROP ALTER

查看当前用户：select user（）；

2、DML语句 数据库操纵语言： 插入数据INSERT、删除数据DELETE、更新数据UPDATE、查询数据SELECT

3、DCL语句 数据库控制语言： 例如控制用户的访问权限GRANT、REVOKE

## 1.操作文件夹（库）

增： create database db1 charset utf8; #db1 为库的名字。 charset后面跟编码格式。

查：**show create** database db1; 查看刚才建的数据库db1

show databases; 查看所有数据库

**改：**数据库不能改名。

alter database db1 charset gbk; 修改字符编码

**删：**drop database db1; 删除数据库db1

## 2.操作文件（表）

切换文件夹（数据库）：use db1; 切换到指定数据库

查看当前数据库：select database();

**增：**create table t1(id int,name char); 创建表t1，并指定字段和类型。id 整型，name 字符型

**查：**show create table t1; 查看表t1的所有信息。

show tables;查看当前数据库的所有表。

desc t1 ; 查看表t1的结构信息。

**改：**alter table t1 modify char(6); 将t1 name字段改为6字符。

alter table t1 change name NAME char(7); 将t1 name字段改为NAME,7字符。

**删：**drop table t1

## 3.操作文件内容（记录）

**增：**insert t1 (id,name) values(1001,’zhangsan’),(2,’lisi’); 按照指定顺序增加记录。

insert t1 values(1001,’wangwu’) 按照默认顺序增加目录。

**查：**select id,name from db1.t1; 查看db1数据库下表t1中id name 两个字段的所有内容。如果目录本身就在db1数据库，则可以省略db1.

select \* from db1.t1; \*代表所有字段

**改：**update db1.t1 set name=’SB’; 将name下所有内容改为SB

update db1.t1 set name=’zhaoliu’ where id=2 将id为2的name改为zhaoliu

**删：**delete from t1; 删除t1中所有内容

delete from t1 where id=2; 删出id为2的记录。

# 三、库/表操作

## 1.系统数据：

information\_schema： 虚拟库，不占用磁盘空间，存储的是数据库启动后的一些参数，如用户表信息、列信息、权限信息、字符信息等  
performance\_schema： MySQL 5.5开始新增一个数据库：主要用于收集数据库服务器性能参数，记录处理查询请求时发生的各种事件、锁等现象  
mysql： 授权库，主要存储系统用户的权限信息  
test： MySQL数据库系统自动创建的测试数据库

## 2. 创建数据库

1 语法(help create database)

CREATE DATABASE 数据库名 charset utf8;

2 数据库命名规则：

可以由字母、数字、下划线、＠、＃、＄

区分大小写

唯一性

不能使用关键字如 create select

不能单独使用数字

最长128位

查看数据库

show databases;

show create database db1;

select database();

选择数据库

USE 数据库名 切换至数据库。

删除数据库

DROP DATABASE 数据库名;

修改数据库

alter database db1 charset utf8;

## 3.存储引擎介绍：

存储引擎就是指表的类型。

mysql支持的存储引擎：show engines;

默认、常用的 InnoDB存储引擎。

create table t1(id int)engine=innodb; 创建引擎为innodb的表。即使不指定，也默认是innodb。

## 4.表的增删改查

语法

create table 表名(

字段名1 类型[(宽度) 约束条件], #[]为可有可无

字段名2 类型[(宽度) 约束条件],

字段名3 类型[(宽度) 约束条件]

);

#注意：

1. 在同一张表中，字段名是不能相同

2. 宽度和约束条件可选

3. 字段名和类型是必须的

show create table t1\G :表的内容按行显示

### 修改表结构

语法：

1. 修改表名

ALTER TABLE 表名

RENAME 新表名;

2. 增加字段

ALTER TABLE 表名

ADD 字段名 数据类型 [完整性约束条件…],

ADD 字段名 数据类型 [完整性约束条件…];

ALTER TABLE 表名

ADD 字段名 数据类型 [完整性约束条件…] **FIRST**; #指定字段位置

ALTER TABLE 表名

ADD 字段名 数据类型 [完整性约束条件…] **AFTER 字段名**;

3. 删除字段

ALTER TABLE 表名

DROP 字段名;

4. 修改字段

ALTER TABLE 表名

MODIFY 字段名 数据类型 [完整性约束条件…];

ALTER TABLE 表名

CHANGE 旧字段名 新字段名 旧数据类型 [完整性约束条件…];

ALTER TABLE 表名

CHANGE 旧字段名 新字段名 新数据类型 [完整性约束条件…];

示范

示例：

1. 修改存储引擎

mysql> alter table service

-> engine=innodb;

2. 添加字段

mysql> alter table student10

-> add name varchar(20) not null,

-> add age int(3) not null default 22;

mysql> alter table student10

-> add stu\_num varchar(10) not null after name; //添加name字段之后

mysql> alter table student10

-> add sex enum('male','female') default 'male' first; //添加到最前面

3. 删除字段

mysql> alter table student10

-> drop sex;

mysql> alter table service

-> drop mac;

4. 修改字段类型modify

mysql> alter table student10

-> modify age int(3);

mysql> alter table student10

-> modify id int(11) not null primary key auto\_increment; //修改为主键

5. 增加约束（针对已有的主键增加auto\_increment）

mysql> alter table student10 modify id int(11) not null primary key auto\_increment;

ERROR 1068 (42000): Multiple primary key defined

mysql> alter table student10 modify id int(11) not null auto\_increment;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

6. 对已经存在的表增加复合主键

mysql> alter table service2

-> add primary key(host\_ip,port);

7. 增加主键

mysql> alter table student1

-> modify name varchar(10) not null primary key;

8. 增加主键和自动增长

mysql> alter table student1

-> modify id int not null primary key auto\_increment;

9. 删除主键

a. 删除自增约束

mysql> alter table student10 modify id int(11) not null;

b. 删除主键

mysql> alter table student10

-> drop primary key;

### 复制表

复制表结构＋记录 （key不会复制: 主键、外键和索引）

mysql> create table new\_service select \* from service;

只复制表结构

mysql> select \* from service where 1=2; //条件为假，查不到任何记录

Empty set (0.00 sec)

mysql> create table new1\_service select \* from service where 1=2;

Query OK, 0 rows affected (0.00 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create table t4 like employees;

create table t1 select host, user from mysql.user;

在当前数据库下创建表t1，并将mysql库中user表中的host、user字段的所有内容复制至表t1

create table t1 select host, user from mysql.user where 1>5;

设置查询的条件，此条件为False时，就查询不出内容，t1中就只有host,user的字段名，而没有其他内容，这样就只复制的表结构。

create table t3 like mysql.user; 复制mysql.user完成表结构，而不复制数据

### 删除表

mysql.user

## 5.数据类型

### （1）整型

整数类型：TINYINT SMALLINT MEDIUMINT INT BIGINT

默认为有符号。若存入的数值超过范围，则按最大范围存储。

create table t1 (id tinyint unsigned) 设置为无符号。

create table t1 (id tinyint（5） unsigned) 设置显示宽度，不足5位，补空格，超过5位，全部显示。

create table t1 (id int(5) unsigned zerofill) 置显示宽度，不足5位，补0，超过5位，全部显示。

默认位数是该类型的最大值位数，不足默认补空格。



### （2）浮点型

定点数类型 DEC等同于DECIMAL

浮点类型：FLOAT DOUBLE

作用：存储薪资、身高、体重、体质参数等

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

#FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]

定义：

单精度浮点数（非准确小数值），m是数字总个数，d是小数点后个数。m最大值为255，d最大值为30

有符号：

-3.402823466E+38 to -1.175494351E-38,

1.175494351E-38 to 3.402823466E+38

无符号：

1.175494351E-38 to 3.402823466E+38

精确度：

\*\*\*\* 随着小数的增多，精度变得不准确 \*\*\*\*

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

#DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]

定义：

双精度浮点数（非准确小数值），m是数字总个数，d是小数点后个数。m最大值为255，d最大值为30

有符号：

-1.7976931348623157E+308 to -2.2250738585072014E-308

2.2250738585072014E-308 to 1.7976931348623157E+308

无符号：

2.2250738585072014E-308 to 1.7976931348623157E+308

精确度：

\*\*\*\*随着小数的增多，精度比float要高，但也会变得不准确 \*\*\*\*

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

decimal[(m[,d])] [unsigned] [zerofill]

定义：

准确的小数值，m是数字总个数（负号不算），d是小数点后个数。 m最大值为65，d最大值为30。

精确度：

\*\*\*\* 随着小数的增多，精度始终准确 \*\*\*\*

对于精确数值计算时需要用此类型

decaimal能够存储精确值的原因在于其内部按照字符串存储。

验证

mysql> create table t1(x float(256,31));

ERROR 1425 (42000): Too big scale 31 specified for column 'x'. Maximum is 30.

mysql> create table t1(x float(256,30));

ERROR 1439 (42000): Display width out of range for column 'x' (max = 255)

mysql> create table t1(x float(255,30)); #建表成功

Query OK, 0 rows affected (0.02 sec)

mysql> create table t2(x double(255,30)); #建表成功

Query OK, 0 rows affected (0.02 sec)

mysql> create table t3(x decimal(66,31));

ERROR 1425 (42000): Too big scale 31 specified for column 'x'. Maximum is 30.

mysql> create table t3(x decimal(66,30));

ERROR 1426 (42000): Too-big precision 66 specified for 'x'. Maximum is 65.

mysql> create table t3(x decimal(65,30)); #建表成功

Query OK, 0 rows affected (0.02 sec)

mysql> show tables;

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

| Tables\_in\_db1 |

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

| t1 |

| t2 |

| t3 |

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

rows in set (0.00 sec)

mysql> insert into t1 values(1.1111111111111111111111111111111); #小数点后31个1

Query OK, 1 row affected (0.01 sec)

mysql> insert into t2 values(1.1111111111111111111111111111111);

Query OK, 1 row affected (0.00 sec)

mysql> insert into t3 values(1.1111111111111111111111111111111);

Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> select \* from t1; #随着小数的增多，精度开始不准确

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

| x |

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

| 1.111111164093017600000000000000 |

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

row in set (0.00 sec)

mysql> select \* from t2; #精度比float要准确点，但随着小数的增多，同样变得不准确

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

| x |

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

| 1.111111111111111200000000000000 |

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

row in set (0.00 sec)

mysql> select \* from t3; #精度始终准确,d为30，于是只留了30位小数

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

| x |

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

| 1.111111111111111111111111111111 |

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

row in set (0.00 sec)

### （3）时间类型

**DATE TIME DATETIME TIMESTAMP YEAR**

year：年 ‘2017’

date：月、日 ’10-14’

time：时分秒 ’14:25:21’

datetime：年月日时分秒 ‘2017-10-14 13:14:22’

函数now() :获取当前年月日时分秒。

YEAR

YYYY（1901/2155）

DATE

YYYY-MM-DD（1000-01-01/9999-12-31）

TIME

HH:MM:SS（'-838:59:59'/'838:59:59'）

DATETIME

YYYY-MM-DD HH:MM:SS（1000-01-01 00:00:00/9999-12-31 23:59:59 Y）

TIMESTAMP

YYYYMMDD HHMMSS（1970-01-01 00:00:00/2037 年某时）

在实际应用的很多场景中，MySQL的这两种日期类型都能够满足我们的需要，存储精度都为秒，但在某些情况下，会展现出他们各自的优劣。

下面就来总结一下两种日期类型的区别。

1.DATETIME的日期范围是1001——9999年，TIMESTAMP的时间范围是1970——2038年。

2.DATETIME存储时间与时区无关，TIMESTAMP存储时间与时区有关，显示的值也依赖于时区。在mysql服务器，

操作系统以及客户端连接都有时区的设置。

3.DATETIME使用8字节的存储空间，TIMESTAMP的存储空间为4字节。因此，TIMESTAMP比DATETIME的空间利用率更高。

4.DATETIME的默认值为null；TIMESTAMP的字段默认不为空（not null）,默认值为当前时间（CURRENT\_TIMESTAMP），

如果不做特殊处理，并且update语句中没有指定该列的更新值，则默认更新为当前时间。

### （4）字符类型

char：定长

varchar：变长

char（5）：定长5字符，不够5个，自动补空格，但是在取的时候，会将数据末尾空格去掉。包括在存入数据末尾有空格时，也默认将去掉。储存速度快，读取速度快。浪费空间。

varchar（5）：保存数据所有内容，包括数据末尾输入的空格。而此格式将保存数据末尾的空格。精确、节省空间，读取、存储速度慢。

两者的区别在设置精确搜索条件时，是否能需要忽略末尾空格。char就会自动忽略，varchar不会忽略。

同一个表，最好不要char varchar混合使用。

length() 查看字节数

char\_length() 查看字符数

语句：select chalength(name) from t3 返回t3表name字段的长度。

SET sql\_mode = ‘PAD\_CHAR\_TO\_FILL\_LENGTH’ :填充字符到原长度。

### （5）枚举类型和集合类型

sex enum(‘男',’女’,’其他’) 在建立记录的时候，只能在三个中选一个。如果填了其他值，则不会报错，但是也不会被保存。该字段为空。

hobbies set(‘play’,’music,’read’); 在建立记录时，可以从中选取多个

## 6.约束语句

### （1）not null default ….

create table t1(

id int(11) unsigned zerofill

);

create table t1 (

id int,

name char(6),

sex enum(‘male’,’female’) not null default ‘male’ 不能为空，设置默认值

);

### （2）unique key

内容唯一、不重复

1.单列唯一。

方式1：

create table department(

id int unique,

name char(10) unique

);

方式2：

create table department(

id int,

name char(10) ,

unique(int),

unique(name)

);

2.联合唯一

create table service (

id int,

ip char(15),

port int,

unique(ip,port), #不要求单列唯一，组合内容是在联合字段中是

唯一的就行。

unique(int) #单列唯一。内容在本字段中不能是重复的。

);

重复存储，会报错。

### （3）primary key ：主键

不能为空，且唯一

innodb引擎，一张表必须有一个主键。如果未设置，则mysql会扫描全表，自动找到不为空且唯一的字段设为主键。如果没有，则会设置一个隐藏列主键。

**单列主键**：

create table t1(

id int primary key,

name char(15)

);

如果未传值，则默认为null。null也不能重复。

**复合主键**

create table t1(

ip char(15),

port int,

primary(ip,port)

);

### （4）auto\_increment 自增长

自增长的字段必须是一个key,可以使unique key 也可以是primary key

默认从1开始，步长1.

create table t1 (

id int primary key auto\_increment

);

show variables like ‘auto\_inc%’ 模糊搜索，%代表任意个数任意字符。

auto\_increment\_increment 步长为1

auto\_increment\_offset 起始 默认为1

设置步长：

set session auto\_increment\_increment=5 ； 只在本次链接有效，下次就无效了

set global auto\_increment\_increment=5 全局有效

设置初始值：

set global auto\_increment\_offset=3 起始数值必须小于等于步长，否则失效。

修改后需要重启客户端。

**清空表：**

delete from t1; 将t1全部清空。

在清空自增长的表后，不会重置自增长的初始值，下次插入记录时，继续沿着原序号继续自增。

truncate t1;清空表，并重置。

### （5）foreign key 外键

用于建立表之间的关系。

外键必须是被关联表的主键或者unique key。

注意先后顺序。

#1、建立表关系：  
 #先建被关联的表,并且**保证被关联的字段唯一**  
 create table dep(  
 id int primary key,  
 name char(16),  
 comment char(50)  
 );  
  
  
 #再建立关联的表  
 create table emp(  
 id int primary key,  
 name char(10),  
 sex enum('male','female'),  
 dep\_id int,  
 foreign key(dep\_id) references dep(id)   
 **on delete cascade   
 on update cascade** #删除和更新同步， 操作被关联表，关联表自动删除和更新。  
 );

#2、插入数据

**插入记录时，外键值必须是被关联表存在的。**  
insert into dep values  
(1,"IT","技术能力有限部门"),  
(2,"销售","销售能力不足部门"),  
(3,"财务","花钱特别多部门");  
  
#再往关联表插入记录  
insert into emp values  
(1,'egon','male',1);  
  
insert into emp values  
(2,'alex','male',1),  
(3,'wupeiqi','female',2),  
(4,'yuanhao','male',3),  
(5,'jinximn','male',2);  
  
  
  
  
delete from emp where dep\_id=1;  
delete from dep where id=1;  
  
  
  
delete from dep where id=3; 在没有删除同步的情况下，在删除该数据前，需要先删除关联表的数据。

## 7、表之间关系

两张表之间的关系

### （1）多对一

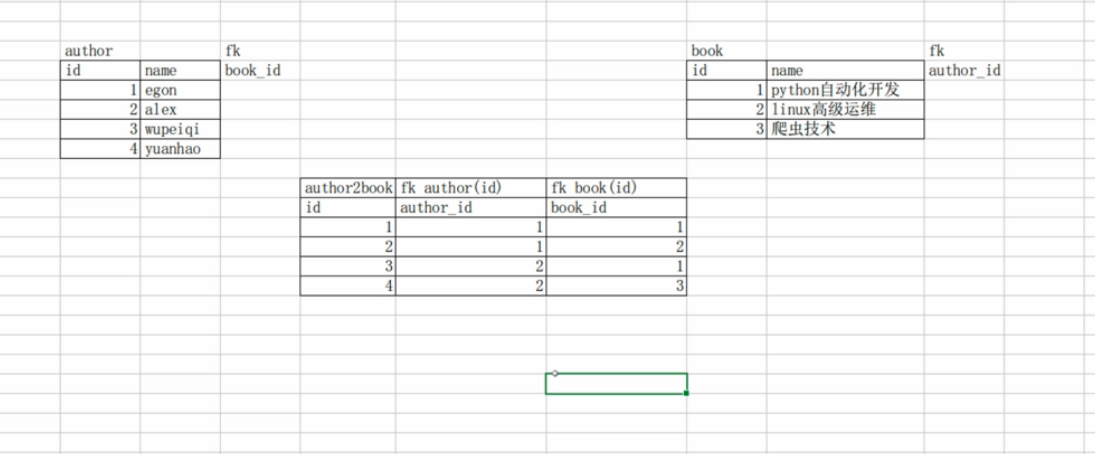
一个表的多条记录对应另一个表的一条记录。

外键就是多对一的关系。关联表多对被关联表一。因为外键需关联被关联表的unique字段。

### （2）多对多

其实就是双向的多对一关系。

再建立一个表，将两张表的关系独立出来。新建的表关联有多对多关系的两个表。



### （3）一对一

可以通过外键和unique来实现。

# 四、数据记录操作

## 1.记录的增删改查

### 插入

1. 插入完整数据（顺序插入）

语法一：

INSERT INTO 表名(字段1,字段2,字段3…字段n) VALUES(值1,值2,值3…值n);

语法二：

INSERT INTO 表名 VALUES (值1,值2,值3…值n);

2. 指定字段插入数据

语法：

INSERT INTO 表名(字段1,字段2,字段3…) VALUES (值1,值2,值3…);

3. 插入多条记录

语法：

INSERT INTO 表名 VALUES

(值1,值2,值3…值n),

(值1,值2,值3…值n),

(值1,值2,值3…值n);

#### 4. 插入查询结果 从另外一个表复制过来。

语法：

INSERT INTO 表名(字段1,字段2,字段3…字段n)

SELECT (字段1,字段2,字段3…字段n) FROM 表2

WHERE …;

### 更新、修改

语法：

UPDATE 表名 SET

字段1=值1,

字段2=值2,

WHERE CONDITION;

示例：

UPDATE mysql.user SET password=password(‘123’)

where user=’root’ and host=’localhost’;

### 删除

语法：

DELETE FROM 表名

WHERE CONITION;

示例：

DELETE FROM mysql.user

WHERE password=’’;

练习：

更新MySQL root用户密码为mysql123

删除除从本地登录的root用户以外的所有用户

### 查询select

#### select的嵌套

(select name from t1 where ..) as t2 将筛选出来的内容作为虚拟表t2，可以在嵌套中使用。在本语句中即可作为一个真实存在的表t2使用。常在多级连表中使用。

select t3.name,tt.id from t3,(select t1.\*,t2.id from t1,t2 where t1.name = t2.name) as tt where t3.id = tt.id;

#### 单表查询

##### 基本语法

select distinct 字段1，字段2 #distinct 表示去重。

from 库.表

where 条件

group by 分组条件

having 过滤条件

order by 排序条件

limit start,n; 起始行、显示条数。

执行顺序：

from 🡪where 🡪group by🡪having🡪select、distinct🡪order by 🡪 limit

##### 1.查询记录

select\*from 表名 [where 条件];  
  
*eg:select \* from students;//查询 students 表中所有记录，所有字段的值都显示出来*  
  
select field1,field2,...fieldn... from 表名 [where 条件];  
  
*eg:select id,name,age from students;//查询 students 表中所有记录, 只显示出 id，name，age三个字段的值*

1.“\*”表示将所有的字段都显示出来

2.用逗号分割，列出需要显示的字段

##### 2.查询不重复的记录

select distinct 字段 from 表名;  
  
eg: select distinct name from students;//查询名字不相同的学生;  
　　select distinct name,age from students;//查询名字和年龄**同时**不同的学生  
　　  
**1.distinct必须放在最开头**  
**2.distinct只能使用需要去重的字段进行操作。 ----**也就是说我sidtinct了name,age两个字段，我后面想根据id进行排序，是不可以的，因为只能name,age两个字段进行操作.  
**3.distinct去重多个字段时，含义是：几个字段 同时重复 时才会被 过滤。**

##### 3.条件查询where

可以使用：

1.比较运算符：><>= <= <> !=

2.between 80 and 100 值在80到100之间

3.in(80,90,100) 值是80或90或1000

4.like 'egon%'，pattern可以是%或\_，  
%表示任意多字符  
\_表示一个字符

5.逻辑运算符：在多个条件直接可以使用逻辑运算符 and or not

6.is null

is not null

select 字段 from 表名 where 条件;

eg:select \* from student where sex='男' and age>**20**; //查询性别是男，并且年龄大于20岁的人。  
  
where后面的条件可以用>、<、>=、<=、!=等多种比较运算符，多个条件之间可以用or、and等逻辑运算符

##### 4.排序order by和限制limit

**排序**  
select \* from 表名 [where 条件] [ order by field1 [desc/asc],field2 [desc/asc]... ];  
  
eg:select \*from student order by age desc;//查询学生表并按年龄降序排列。  
  
1.desc 降序排列，asc 升序排列  
2.order by 后面可以跟多个不同的排序字段，每个排序字段都可以有不同的排序顺序。  
3.如果排序字段的值一样，则相同的字段按照第二个排序字段进行排序。  
4.如果只有一个排序字段，则字段相同的记录将会无序排列。

**限制**

select ... [limit 起始偏移量,行数];  
  
eg:select \* from student order by mark desc limit 5;//取出成绩前五名的学生(省略了起始偏移量，此时默认为0)  
  
1.默认情况下，起始偏移量为0，只写记录行数就可以。

##### 5.聚合

max

min

avg

sum

count

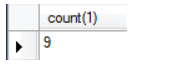
select 字段 fun\_name from 表名 [where 条件] [group by field1,field2...] [with rollup] [having 条件];  
  
eg:  
  
1.fun\_name 表示要做的聚合操作，也就是说聚合函数，常用的有 : sum(求和)、count(\*)(记录数)、max(最大值)、min(最小值)、avg。  
2.group by关键字 表示要进行分类聚合的字段。比如要按照部门分类统计员工数量，部门就应该写在group by 后面。  
3.with rollup 是可选语法，表明是否对分类聚合后的结果进行再汇总  
4.having 关键字表示对分类后的结果再进行条件过滤。对前面的聚合函数的结果加以筛选

公司员工表A如下 (编号，姓，名，薪水) ：



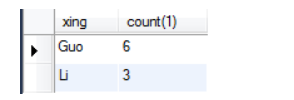
统计总人数

select count(1) from A; #count(1)是对第一列进行统计



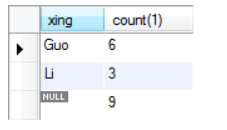
统计各个姓的人数

select xing,count(**1**) from A group by xing;



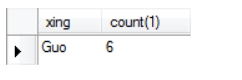
既要统计各个姓的人数，又统计总人数

select xing,count(**1**) from A group by xing with rollup;



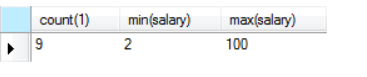
统计人数大4的姓

select xing,count(**1**) from A group by xing having count(**1**)>**4**;



统计薪水总额，最低薪资，最高薪资

select count(**1**),min(salary),max(salary) from A;



##### 查询的四则运算

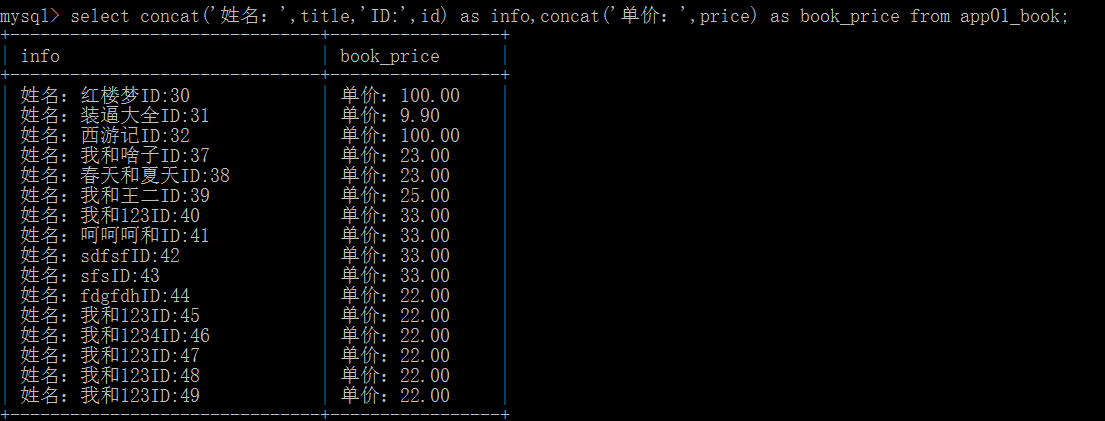
select name,salary\*12 from t1 ;

select name,salart\*12 as year\_salary from t1; as可以省略，起对应表头。

##### concat as：拼接字符串

select concat(‘姓名：’,name,’性别：’,sex) as info,concat (‘年薪：”’,salary\*12) as year\_salary from t1.

将name、sex字段拼接到一个名叫info的字段中。将年薪拼接到一个叫year\_salary的字段中。



select contact\_ws(‘:’,name,sex,age) from t1, #设置：为字段间的分割符。即每个字段间都插入：

##### group\_concat

select post,group\_concat(name) from t1 group by post,

将组内的所有名字组合成一个字段。

##### having 过滤

是在分组之后过滤。而where是分组之前的过滤。

对使用聚可以用聚合函数。

select post,group\_concat(name),count(id) from employee group by post having count(id) < 2;  
select post,avg(salary) from employee group by post having avg(salary) > 10000;

##### 正则表达式查询

select \* from employee where name like 'jin%';  
select \* from employee where name regexp '^jin';  
select \* from employee where name regexp '^jin.\*(g|n)$';

正则表达式用regexp关键字

基本形式   
  
属性名 regexp ‘匹配方式'   
  
正则表达式的模式字符   
  
^ 匹配字符开始的部分   
  
eg1: 从info表name字段中查询以L开头的记录   
  
select \* from info where name regexp '^L';   
  
eg2: 从info表name字段中查询以aaa开头的记录   
  
select \* from info where name regexp '^aaa';   
  
$ 匹配字符结束的部分   
  
eg1: 从info表name字段中查询以c结尾的记录   
  
select \* from info where name regexp 'c$';   
  
eg2: 从info表name字段中查询以aaa结尾的记录   
  
select \* from info where name regexp 'aaa$';   
  
. 匹配字符串中的任意**一个**字符，包括回车和换行   
  
eg1: 从info表name字段中查询以L开头y结尾中间有两个任意字符的记录   
  
select \* from info where name regexp '^L..y$';   
  
[字符集合]匹配字符集合中的任意一个字符   
  
eg1: 从info表name字段中查询包含c、e、o三个字母中任意一个的记录   
  
select \* from info where name regexp '[ceo]';   
  
eg2: 从info表name字段中查询包含数字的记录   
  
select \* from info where name regexp '[0-9]';   
  
eg3: 从info表name字段中查询包含数字或a、b、c三个字母中任意一个的记录   
  
select \* from info where name regexp '[0-9a-c]';   
  
[^字符集合]匹配除了字符集合外的任意字符   
  
eg1: 从info表name字段中查询包含a-w字母和数字以外字符的记录   
  
select \* from info where name regexp '[^a-w0-9]';   
  
s1|s2|s3 匹配s1s2s3中的任意一个   
  
eg1: 从info表name字段中查询包含'ic'的记录   
  
select \* from info where name regexp 'ic';   
  
eg2: 从info表name字段中查询包含ic、uc、ab三个字符串中任意一个的记录   
  
select \* from info where name regexp 'ic|uc|ab';   
  
\* 代表多个该字符前的字符，包括0个或1个   
  
eg1: 从info表name字段中查询c之前出现过a的记录   
  
select \* from info where name regexp 'a\*c';   
  
+ 代表多个该字符前的字符，包括1个   
  
eg1: 从info表name字段中查询c之前出现过a的记录   
  
select \* from info where name regexp 'a+c';(注意比较结果！)   
  
字符串{N} 字符串出现N次   
  
eg1: 从info表name字段中查询出现过a3次的记录   
  
select \* from　info where name regexp 'a{3}';   
  
字符串{M，N}字符串最少出现M次，最多出现N次   
  
eg1: 从info表name字段中查询ab出现最少1次最多3次的记录   
  
select \* from info where name regexp 'ab{1,3}';   
  
MYSQL中自带通配符（LIKE关键词）   
  
%可以表示任意长度的字符（包括0）   
  
-可以表示单个字符   
   
#正则表达式  
select \* from employee where name like 'jin%';  
select \* from employee where name regexp '^jin';  
select \* from employee where name regexp '^jin.\*(g|n)$';

#### 连表查询

连表时选择字段可以使用 \* 表示选择两个表的所有字段

t1.\*表示选择t1表的所有字段。

concat(t2.name) as t2\_name 将t2的name字段命名为t2\_name

常用查询时，可以用括号来嵌套。

(select id，name，sex from t1) as tt;

把查出的虚拟表命名，后续可以引用。

##### 内连接：

直接用 , 将两个表隔开即可。from t1,t2 where ….

或者使用from t1 inner join t2 on 条件…

select \* from t1 inner join t2 on t1.dep=t2.department

select t1.\*,t2.id from t1,t2 where t1.dep = t2.department

##### 左连接

from t1 left join t2 on 条件…

select t1.\*,t2.id from t1 left join t2 on t1.dep=t2.depar

在内连接的基础上保留左表的记录。

##### 右连接：

**from t1 right join t2 on 条件**

在内连接的基础上保留右表的记录

##### 全连接

**from t1 full join t2 on……**

在内连接的基础上，保留两个表没有对应关系的部分

内连接：只取两张表的共同部分  
select \* from employee inner join department on employee.dep\_id = department.id ;  
  
左连接:在内连接的基础上保留左表的记录  
select \* from employee left join department on employee.dep\_id = department.id ;  
  
右连接：在内连接的基础上保留右表的记录  
select \* from employee right join department on employee.dep\_id = department.id ;  
  
全外连接：在内连接的基础上保留左右两表没有对应关系的记录  
select \* from employee full join department on employee.dep\_id = department.id ;  
  
  
select \* from employee left join department on employee.dep\_id = department.id  
union  
select \* from employee right join department on employee.dep\_id = department.id ;

#### 联合查询union/union all

**记录联合可以把两个或多个select查询的结果合并一起显示出来。**

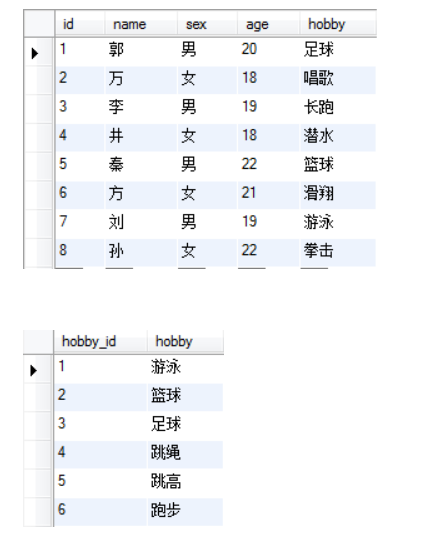
select \* from t1 union/union all select \* from t2...union/union allselect \* from tn;  
  
union 和 union all的区别是union all是把结果集直接合并在一起，而union会将union后的结果进行一次distinct去重。

#### 子查询

当进行查询的时候，需要的条件是另外一个select语句的结果，这时候就要用到子查询

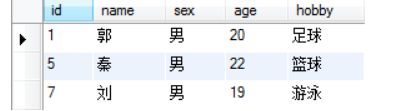
用于子查询的主要关键字有：in,not in,=,!=,exists,not exists等。

以下两张表学生表，爱好表



**从[student表]中查出爱好在[hobby表]中的学生**

select\*from student where hobby in (select hobby from hobby);



**如果子查询只有1条记录，可以用=代替in**

select\*from student where hobby = (select hobby from hobby limit **1**);

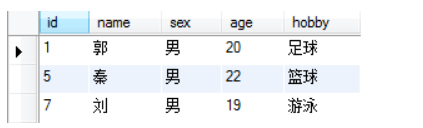


**子查询语法上可以用表连接进行替代  
  
子查询：**

select\*from student where hobby in (select hobby from hobby);

**用表连接进行替代**

select student.\* from student,hobby where student.hobby = hobby.hobby;  
  
**结果都是相同的：**



**表连接替代子查询的意义：**

**1.MySQL4.1以前的版本不支持子查询，需要用表连接来实现子查询的内容**

**2.表连接在很多情况下用于优化子查询**

常用查询时，可以用括号来嵌套。

(select id，name，sex from t1) as tt;

把查出的虚拟表命名，后续可以引用。

# 五、权限管理

## 1、创建账号

使用root创建：

本地账号

create user’s1’@’localhost’ identified by ‘123’

远程账号

create user’s1’@’192.168.1.3’ identified by ‘123’

运行某一网段登录

create user’s1’@’192.168.1.%’ identified by ‘123’

create user’s1’@’ %’ identified by ‘123’

## 2、授权

### 对用户授权：

grant 权限 on 范围 to 用户

授权操作只能用root账户操作

权限：

all :所有权限（不含授权权限）

select :选择权限

select（某字段名，..） on db1.t1 : db1.t1表的某（几个）字段的选择权。

update：更新权限

范围：

\*.\* 所有库下的所有表

db1.\* 对某一个库下的所有表

db1.t1 某一库下的一个表

columns\_priv :对某库某表下的一个字段操作。

示例：

grant select on \*.\* to ‘ohaha’@’localhost’; 对该账号授权select权限，对所有库操作。

grant all on \*.\* to ‘ohaha’@’localhost’;对该账号授权所有权限，对所有库所有表。但是不包括授权权限。

grant select(id,name),update(age) on db1.t2 to 'egon1'@'localhost';对字段放开的权限。

## 查看用户权限

mysql.user 是一个存放账号的数据库。

select \* from mysql.user\G 查看user表中用户权限。

mysql.db

select \* from mysql.db \G 查看账户的db权限，是否对某一个数据库有操作权限。

## 回收权限：

revoke 权限 on 范围 from 用户

revoke：

revoke select on \*.\* from ‘ohaha’@’localhost’;回收该账户的对所有库的select权限。

权限管理  
 1、创建账号  
 # 本地账号  
 create user 'egon1'@'localhost' identified by '123'; # mysql -uegon1 -p123  
 # 远程帐号  
 create user 'egon2'@'192.168.31.10' identified by '123'; # mysql -uegon2 -p123 -h 服务端ip  
 create user 'egon3'@'192.168.31.%' identified by '123'; # mysql -uegon3 -p123 -h 服务端ip  
 create user 'egon3'@'%' identified by '123'; # mysql -uegon3 -p123 -h 服务端ip  
   
 2、授权  
 user:\*.\*  
 db:db1.\*  
 tables\_priv:db1.t1  
 columns\_priv:id,name  
   
 grant all on \*.\* to 'egon1'@'localhost';  
 grant select on \*.\* to 'egon1'@'localhost';  
 revoke select on \*.\* from 'egon1'@'localhost';  
   
 grant select on db1.\* to 'egon1'@'localhost';  
 revoke select on db1.\* from 'egon1'@'localhost';  
   
   
 grant select on db1.t2 to 'egon1'@'localhost';  
 revoke select on db1.t2 from 'egon1'@'localhost';  
   
 grant select(id,name),update(age) on db1.t2 to 'egon1'@'localhost';

# 六、Navicat工具的使用

# 七、pymysql

*#pip3 install pymysql***import** pymysql  
  
user=input(**'user>>: '**).strip()  
pwd=input(**'password>>: '**).strip()  
  
*# 建立链接*conn=pymysql.connect(  
 host=**'192.168.10.15'**,  
 port=3306,  
 user=**'root'**,  
 password=**'123'**,  
 db=**'db9'**,  
 charset=**'utf8'**)  
  
*# 拿到游标*cursor=conn.cursor()  
  
*# 执行sql语句  
  
# sql='select \* from userinfo where user = "%s" and pwd="%s"' %(user,pwd)  
# print(sql)*sql=**'select** *\** **from userinfo where** *user* **= %s and pwd=%s'**rows=cursor.execute(sql,(user,pwd))  
  
cursor.close()  
conn.close()  
  
*# 进行判断***if** rows:  
 print(**'登录成功'**)  
**else**:  
 print(**'登录失败'**)

## 增删改

*#1、增删改***import** pymysql  
  
*# 建立链接*conn=pymysql.connect(  
 host=**'192.168.10.15'**,  
 port=3306,  
 user=**'root'**,  
 password=**'123'**,  
 db=**'db9'**,  
 charset=**'utf8'**)  
  
*# 拿游标*cursor=conn.cursor()  
  
*# 执行sql  
# 增、删、改*sql=**'insert into userinfo(user,pwd) values(%s,%s)'***# rows=cursor.execute(sql,('wxx','123'))  
# print(rows)  
# rows=cursor.executemany(sql,[('yxx','123'),('egon1','111'),('egon2','2222')])  
# print(rows)*rows=cursor.executemany(sql,[(**'egon3'**,**'123'**),(**'egon4'**,**'111'**),(**'egon5'**,**'2222'**)])  
print(cursor.lastrowid) #返回该表的最后一行的行数。  
  
conn.commit()  
*# 关闭*cursor.close()  
conn.close()  
  
  
  
*#2、查询  
# import pymysql  
#  
# # 建立链接  
# conn=pymysql.connect(  
# host='192.168.10.15',  
# port=3306,  
# user='root',  
# password='123',  
# db='db9',  
# charset='utf8'  
# )  
  
# 拿游标  
# cursor=conn.cursor(pymysql.cursors.DictCursor)  
  
# 执行sql  
# 查询  
# rows=cursor.execute('select \* from userinfo;')  
# print(rows)  
# print(cursor.fetchone())  
# print(cursor.fetchone())  
# print(cursor.fetchone())  
# print(cursor.fetchone())  
# print(cursor.fetchone())  
# print(cursor.fetchone())  
# print(cursor.fetchone())  
  
  
# print(cursor.fetchmany(2))  
  
# print(cursor.fetchall())  
# print(cursor.fetchall())  
  
  
  
# cursor.scroll(3,mode='absolute') # 相对绝对位置移动  
# print(cursor.fetchone())  
# cursor.scroll(2,mode='relative') # 相对当前位置移动  
# print(cursor.fetchone())  
  
#  
  
# 关闭  
# cursor.close()  
# conn.close()*

# **八、 mysql内置功能**

## 1.视图

create view tablename as ……语句

建立一个虚拟表tablename，在硬盘上只保存表结构，没有表数据。可以直接引用该表名。实际上是每次都会触发该语句。该表会根据其他关联表数据的变化而变化。

## 2.触发器

创建触发器：

格式：

# 插入前

CREATE TRIGGER tri\_before\_insert\_tb1 BEFORE INSERT ON tb1 FOR EACH ROW

BEGIN

...

END

# 插入后

CREATE TRIGGER tri\_after\_insert\_tb1 AFTER INSERT ON tb1 FOR EACH ROW

BEGIN

...

END

# 删除前

CREATE TRIGGER tri\_before\_delete\_tb1 BEFORE DELETE ON tb1 FOR EACH ROW

BEGIN

...

END

# 删除后

CREATE TRIGGER tri\_after\_delete\_tb1 AFTER DELETE ON tb1 FOR EACH ROW

BEGIN

...

END

# 更新前

CREATE TRIGGER tri\_before\_update\_tb1 BEFORE UPDATE ON tb1 FOR EACH ROW

BEGIN

...

END

# 更新后

CREATE TRIGGER tri\_after\_update\_tb1 AFTER UPDATE ON tb1 FOR EACH ROW

BEGIN

...

END

例子：

#准备表

CREATE TABLE cmd (

id INT PRIMARY KEY auto\_increment,

USER CHAR (32),

priv CHAR (10),

cmd CHAR (64),

sub\_time datetime, #提交时间

success enum ('yes', 'no') #0代表执行失败

);

CREATE TABLE errlog (

id INT PRIMARY KEY auto\_increment,

err\_cmd CHAR (64),

err\_time datetime

);

#创建触发器

delimiter // # delimiter 是设置结束符号为//

CREATE TRIGGER tri\_after\_insert\_cmd AFTER INSERT ON cmd FOR EACH ROW

BEGIN

IF NEW.success = 'no' THEN #等值判断只有一个等号 new针对新插入的记录，old针对删除前的记录。

INSERT INTO errlog(err\_cmd, err\_time) VALUES(NEW.cmd, NEW.sub\_time) ; #必须加分号

END IF ; #必须加分号

END//

delimiter ; #将结束符号还原成；

#往表cmd中插入记录，触发触发器，根据IF的条件决定是否插入错误日志

INSERT INTO cmd (

USER,

priv,

cmd,

sub\_time,

success

)

VALUES

('egon','0755','ls -l /etc',NOW(),'yes'),

('egon','0755','cat /etc/passwd',NOW(),'no'),

('egon','0755','useradd xxx',NOW(),'no'),

('egon','0755','ps aux',NOW(),'yes');

#查询错误日志，发现有两条

mysql> select \* from errlog;

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

| id | err\_cmd | err\_time |

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

| 1 | cat /etc/passwd | 2017-09-14 22:18:48 |

| 2 | useradd xxx | 2017-09-14 22:18:48 |

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

rows in set (0.00 sec)

插入后触发触发器

## 3.存储过程

#无参存储过程

delimiter //

create procedure p1()

BEGIN

select \* from db.teacher;

END //

delimiter ;

用哪个数据库，就在哪个数据库下创建存储过程。

#MySQL调用

call p1();

类似于函数，执行函数中的语句。

#python中调用

cursor.callproc(‘p1’)

#有参存储过程

delimiter //

create procedure p1(in n1 int,in n2 int,out res int) #in 是输入，out输出，inout类型 输入输出类型。

BEGIN

select \* from db.teacher where tid>n1 and tid<n2;

set res=1; 设置返回值

END //

delimiter ;

#MySQL调用

set @x=0

call p1(2,4,@x); 将值传给res。执行后，res将值赋给@x

执行完毕后，

select @x;可以看x值是否改变。

类似于函数，执行函数中的语句。

#python中调用

cursor.callproc(‘p1’,(2,4,0)) ;python 中自动创建变量，@\_p2\_0=2,@\_p2\_1=4,@\_p2\_2=0

# 九、应用程序与数据库

方式一：Python 调用存储过程，MySQL：编写存储过程

方式二：编写纯生SQL MySQL：

方式一的效率高，但是在扩展功能时，需要数据库和应用程序同时改，需要跨部门协作，比较麻烦

方式二在应用程序扩展时，会更方便，但是编写时更麻烦。

方式三：

Python ：基于ORM—>自动转为纯生SQL，本质还是由应用程序来控制SQL

MySQL:

# 十、函数与流程控制、索引

## date\_format

sub\_time 字段：2017-03-01 15:32:25

select \* from t1 group by date\_format(sub\_time,’%Y-%m’)

按照sub\_time 字段 年-月 格式分组。也就是同年同月的为一组。

SELECT DATE\_FORMAT(sub\_time,'%Y-%m'),COUNT(1) FROM blog GROUP BY DATE\_FORMAT(sub\_time,'%Y-%m');

其他函数：

一、数学函数

ROUND(x,y)

返回参数x的四舍五入的有y位小数的值

RAND()

返回０到１内的随机值,可以通过提供一个参数(种子)使RAND()随机数生成器生成一个指定的值。

二、聚合函数(常用于GROUP BY从句的SELECT查询中)

AVG(col)返回指定列的平均值

COUNT(col)返回指定列中非NULL值的个数

MIN(col)返回指定列的最小值

MAX(col)返回指定列的最大值

SUM(col)返回指定列的所有值之和

GROUP\_CONCAT(col) 返回由属于一组的列值连接组合而成的结果

三、字符串函数

CHAR\_LENGTH(str)

返回值为字符串str 的长度，长度的单位为字符。一个多字节字符算作一个单字符。

CONCAT(str1,str2,...)

字符串拼接

如有任何一个参数为NULL ，则返回值为 NULL。

CONCAT\_WS(separator,str1,str2,...)

字符串拼接（自定义连接符）

CONCAT\_WS()不会忽略任何空字符串。 (然而会忽略所有的 NULL）。

CONV(N,from\_base,to\_base)

进制转换

例如：

SELECT CONV('a',16,2); 表示将 a 由16进制转换为2进制字符串表示

FORMAT(X,D)

将数字X 的格式写为'#,###,###.##',以四舍五入的方式保留小数点后 D 位， 并将结果以字符串的形式返回。若 D 为 0, 则返回结果不带有小数点，或不含小数部分。

例如：

SELECT FORMAT(12332.1,4); 结果为： '12,332.1000'

INSERT(str,pos,len,newstr)

在str的指定位置插入字符串

pos：要替换位置其实位置

len：替换的长度

newstr：新字符串

特别的：

如果pos超过原字符串长度，则返回原字符串

如果len超过原字符串长度，则由新字符串完全替换

INSTR(str,substr)

返回字符串 str 中子字符串的第一个出现位置。

LEFT(str,len)

返回字符串str 从开始的len位置的子序列字符。

LOWER(str)

变小写

UPPER(str)

变大写

REVERSE(str)

返回字符串 str ，顺序和字符顺序相反。

SUBSTRING(str,pos) , SUBSTRING(str FROM pos) SUBSTRING(str,pos,len) , SUBSTRING(str FROM pos FOR len)

不带有len 参数的格式从字符串str返回一个子字符串，起始于位置 pos。带有len参数的格式从字符串str返回一个长度同len字符相同的子字符串，起始于位置 pos。 使用 FROM的格式为标准 SQL 语法。也可能对pos使用一个负值。假若这样，则子字符串的位置起始于字符串结尾的pos 字符，而不是字符串的开头位置。在以下格式的函数中可以对pos 使用一个负值。

mysql> SELECT SUBSTRING('Quadratically',5);

-> 'ratically'

mysql> SELECT SUBSTRING('foobarbar' FROM 4);

-> 'barbar'

mysql> SELECT SUBSTRING('Quadratically',5,6);

-> 'ratica'

mysql> SELECT SUBSTRING('Sakila', -3);

-> 'ila'

mysql> SELECT SUBSTRING('Sakila', -5, 3);

-> 'aki'

mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);

-> 'ki'

四、日期和时间函数

CURDATE()或CURRENT\_DATE() 返回当前的日期

CURTIME()或CURRENT\_TIME() 返回当前的时间

DAYOFWEEK(date) 返回date所代表的一星期中的第几天(1~7)

DAYOFMONTH(date) 返回date是一个月的第几天(1~31)

DAYOFYEAR(date) 返回date是一年的第几天(1~366)

DAYNAME(date) 返回date的星期名，如：SELECT DAYNAME(CURRENT\_DATE);

FROM\_UNIXTIME(ts,fmt) 根据指定的fmt格式，格式化UNIX时间戳ts

HOUR(time) 返回time的小时值(0~23)

MINUTE(time) 返回time的分钟值(0~59)

MONTH(date) 返回date的月份值(1~12)

MONTHNAME(date) 返回date的月份名，如：SELECT MONTHNAME(CURRENT\_DATE);

NOW() 返回当前的日期和时间

QUARTER(date) 返回date在一年中的季度(1~4)，如SELECT QUARTER(CURRENT\_DATE);

WEEK(date) 返回日期date为一年中第几周(0~53)

YEAR(date) 返回日期date的年份(1000~9999)

重点:

DATE\_FORMAT(date,format) 根据format字符串格式化date值

mysql> SELECT DATE\_FORMAT('2009-10-04 22:23:00', '%W %M %Y');

-> 'Sunday October 2009'

mysql> SELECT DATE\_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');

-> '22:23:00'

mysql> SELECT DATE\_FORMAT('1900-10-04 22:23:00',

-> '%D %y %a %d %m %b %j');

-> '4th 00 Thu 04 10 Oct 277'

mysql> SELECT DATE\_FORMAT('1997-10-04 22:23:00',

-> '%H %k %I %r %T %S %w');

-> '22 22 10 10:23:00 PM 22:23:00 00 6'

mysql> SELECT DATE\_FORMAT('1999-01-01', '%X %V');

-> '1998 52'

mysql> SELECT DATE\_FORMAT('2006-06-00', '%d');

-> '00'

五、加密函数

MD5()

计算字符串str的MD5校验和

PASSWORD(str)

返回字符串str的加密版本，这个加密过程是不可逆转的，和UNIX密码加密过程使用不同的算法。

六、控制流函数

CASE WHEN[test1] THEN [result1]...ELSE [default] END

如果testN是真，则返回resultN，否则返回default

CASE [test] WHEN[val1] THEN [result]...ELSE [default]END

如果test和valN相等，则返回resultN，否则返回default

IF(test,t,f)

如果test是真，返回t；否则返回f

IFNULL(arg1,arg2)

如果arg1不是空，返回arg1，否则返回arg2

NULLIF(arg1,arg2)

如果arg1=arg2返回NULL；否则返回arg1

七、控制流函数小练习

#7.1、准备表

/\*

Navicat MySQL Data Transfer

Source Server : localhost\_3306

Source Server Version : 50720

Source Host : localhost:3306

Source Database : student

Target Server Type : MYSQL

Target Server Version : 50720

File Encoding : 65001

Date: 2018-01-02 12:05:30

\*/

SET FOREIGN\_KEY\_CHECKS=0;

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

-- Table structure for course

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

DROP TABLE IF EXISTS `course`;

CREATE TABLE `course` (

`c\_id` int(11) NOT NULL,

`c\_name` varchar(255) DEFAULT NULL,

`t\_id` int(11) DEFAULT NULL,

PRIMARY KEY (`c\_id`),

KEY `t\_id` (`t\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

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

-- Records of course

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

INSERT INTO `course` VALUES ('1', 'python', '1');

INSERT INTO `course` VALUES ('2', 'java', '2');

INSERT INTO `course` VALUES ('3', 'linux', '3');

INSERT INTO `course` VALUES ('4', 'web', '2');

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

-- Table structure for score

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

DROP TABLE IF EXISTS `score`;

CREATE TABLE `score` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`s\_id` int(10) DEFAULT NULL,

`c\_id` int(11) DEFAULT NULL,

`num` double DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=12 DEFAULT CHARSET=utf8;

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

-- Records of score

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

INSERT INTO `score` VALUES ('1', '1', '1', '79');

INSERT INTO `score` VALUES ('2', '1', '2', '78');

INSERT INTO `score` VALUES ('3', '1', '3', '35');

INSERT INTO `score` VALUES ('4', '2', '2', '32');

INSERT INTO `score` VALUES ('5', '3', '1', '66');

INSERT INTO `score` VALUES ('6', '4', '2', '77');

INSERT INTO `score` VALUES ('7', '4', '1', '68');

INSERT INTO `score` VALUES ('8', '5', '1', '66');

INSERT INTO `score` VALUES ('9', '2', '1', '69');

INSERT INTO `score` VALUES ('10', '4', '4', '75');

INSERT INTO `score` VALUES ('11', '5', '4', '66.7');

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

-- Table structure for student

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

DROP TABLE IF EXISTS `student`;

CREATE TABLE `student` (

`s\_id` varchar(20) NOT NULL,

`s\_name` varchar(255) DEFAULT NULL,

`s\_age` int(10) DEFAULT NULL,

`s\_sex` char(1) DEFAULT NULL,

PRIMARY KEY (`s\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

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

-- Records of student

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

INSERT INTO `student` VALUES ('1', '鲁班', '12', '男');

INSERT INTO `student` VALUES ('2', '貂蝉', '20', '女');

INSERT INTO `student` VALUES ('3', '刘备', '35', '男');

INSERT INTO `student` VALUES ('4', '关羽', '34', '男');

INSERT INTO `student` VALUES ('5', '张飞', '33', '女');

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

-- Table structure for teacher

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

DROP TABLE IF EXISTS `teacher`;

CREATE TABLE `teacher` (

`t\_id` int(10) NOT NULL,

`t\_name` varchar(50) DEFAULT NULL,

PRIMARY KEY (`t\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

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

-- Records of teacher

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

INSERT INTO `teacher` VALUES ('1', '大王');

INSERT INTO `teacher` VALUES ('2', 'alex');

INSERT INTO `teacher` VALUES ('3', 'egon');

INSERT INTO `teacher` VALUES ('4', 'peiqi');

#7.2、统计各科各分数段人数.显示格式:课程ID,课程名称,[100-85],[85-70],[70-60],[ <60]

select score.c\_id,

course.c\_name,

sum(CASE WHEN num BETWEEN 85 and 100 THEN 1 ELSE 0 END) as '[100-85]',

sum(CASE WHEN num BETWEEN 70 and 85 THEN 1 ELSE 0 END) as '[85-70]',

sum(CASE WHEN num BETWEEN 60 and 70 THEN 1 ELSE 0 END) as '[70-60]',

sum(CASE WHEN num < 60 THEN 1 ELSE 0 END) as '[ <60]'

from score,course where score.c\_id=course.c\_id GROUP BY score.c\_id;

#1 基本使用

mysql> SELECT DATE\_FORMAT('2009-10-04 22:23:00', '%W %M %Y');

-> 'Sunday October 2009'

mysql> SELECT DATE\_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');

-> '22:23:00'

mysql> SELECT DATE\_FORMAT('1900-10-04 22:23:00',

-> '%D %y %a %d %m %b %j');

-> '4th 00 Thu 04 10 Oct 277'

mysql> SELECT DATE\_FORMAT('1997-10-04 22:23:00',

-> '%H %k %I %r %T %S %w');

-> '22 22 10 10:23:00 PM 22:23:00 00 6'

mysql> SELECT DATE\_FORMAT('1999-01-01', '%X %V');

-> '1998 52'

mysql> SELECT DATE\_FORMAT('2006-06-00', '%d');

-> '00'

#2 准备表和记录

CREATE TABLE blog (

id INT PRIMARY KEY auto\_increment,

NAME CHAR (32),

sub\_time datetime

);

INSERT INTO blog (NAME, sub\_time)

VALUES

('第1篇','2015-03-01 11:31:21'),

('第2篇','2015-03-11 16:31:21'),

('第3篇','2016-07-01 10:21:31'),

('第4篇','2016-07-22 09:23:21'),

('第5篇','2016-07-23 10:11:11'),

('第6篇','2016-07-25 11:21:31'),

('第7篇','2017-03-01 15:33:21'),

('第8篇','2017-03-01 17:32:21'),

('第9篇','2017-03-01 18:31:21');

#3. 提取sub\_time字段的值，按照格式后的结果即"年月"来分组

SELECT DATE\_FORMAT(sub\_time,'%Y-%m'),COUNT(1) FROM blog GROUP BY DATE\_FORMAT(sub\_time,'%Y-%m');

#结果

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

| DATE\_FORMAT(sub\_time,'%Y-%m') | COUNT(1) |

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

| 2015-03 | 2 |

| 2016-07 | 4 |

| 2017-03 | 3 |

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

rows in set (0.00 sec)

需要掌握函数：date\_format

更多函数：[中文猛击这里](http://doc.mysql.cn/mysql5/refman-5.1-zh.html-chapter/functions.html#encryption-functions) OR [官方猛击这里](https://dev.mysql.com/doc/refman/5.7/en/functions.html)

## ****一 自定义函数****

#！！！注意！！！

#函数中不要写sql语句（否则会报错），函数仅仅只是一个功能，是一个在sql中被应用的功能

#若要想在begin...end...中写sql，请用存储过程

delimiter //

create function f1(

i1 int,

i2 int)

returns int

BEGIN

declare num int;

set num = i1 + i2;

return(num);

END //

delimiter ;

delimiter //

create function f5(

i int

)

returns int

begin

declare res int default 0;

if i = 10 then

set res=100;

elseif i = 20 then

set res=200;

elseif i = 30 then

set res=300;

else

set res=400;

end if;

return res;

end //

delimiter ;

## ****二 删除函数****

drop function func\_name;

## ****三 执行函数****

# 获取返回值

select UPPER('egon') into @res;

SELECT @res;

# 在查询中使用

select f1(11,nid) ,name from tb2;

# 十一、索引

索引相当于书的目录，方便查询，但是每次新写入数据，都需要重新更新目录，所以索引会降低写的速度，索引并不是越多越好。

辅助索引：辅助索引值存放辅助字段的值和所对应的主键的值，就是通过辅助索引找到聚集索引。

#### 一 功能

#1. 索引的功能就是加速查找

#2. mysql中的primary key，unique，联合唯一也都是索引，这些索引除了加速查找以外，还有约束的功能

#### 二 MySQL常用的索引

普通索引INDEX：加速查找

唯一索引：

-主键索引PRIMARY KEY：加速查找+约束（不为空、不能重复）

-唯一索引UNIQUE:加速查找+约束（不能重复）

联合索引：

-PRIMARY KEY(id,name):联合主键索引

-UNIQUE(id,name):联合唯一索引

-INDEX(id,name):联合普通索引

各个索引的应用场景

举个例子来说，比如你在为某商场做一个会员卡的系统。

这个系统有一个会员表

有下列字段：

会员编号 INT

会员姓名 VARCHAR(10)

会员身份证号码 VARCHAR(18)

会员电话 VARCHAR(10)

会员住址 VARCHAR(50)

会员备注信息 TEXT

那么这个 会员编号，作为主键，使用 PRIMARY

会员姓名 如果要建索引的话，那么就是普通的 INDEX

会员身份证号码 如果要建索引的话，那么可以选择 UNIQUE （唯一的，不允许重复）

#除此之外还有全文索引，即FULLTEXT

会员备注信息 ， 如果需要建索引的话，可以选择全文搜索。

用于搜索很长一篇文章的时候，效果最好。

用在比较短的文本，如果就一两行字的，普通的 INDEX 也可以。

但其实对于全文搜索，我们并不会使用MySQL自带的该索引，而是会选择第三方软件如Sphinx，专门来做全文搜索。

#其他的如空间索引SPATIAL，了解即可，几乎不用

#### 三 索引的两大类型hash与btree

#我们可以在创建上述索引的时候，为其指定索引类型，分两类

hash类型的索引：查询单条快，范围查询慢

btree类型的索引：b+树，层数越多，数据量指数级增长（我们就用它，因为innodb默认支持它）

#不同的存储引擎支持的索引类型也不一样

InnoDB 支持事务，支持行级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；

MyISAM 不支持事务，支持表级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；

Memory 不支持事务，支持表级别锁定，支持 B-tree、Hash 等索引，不支持 Full-text 索引；

NDB 支持事务，支持行级别锁定，支持 Hash 索引，不支持 B-tree、Full-text 等索引；

Archive 不支持事务，支持表级别锁定，不支持 B-tree、Hash、Full-text 等索引；

#### 四 创建/删除索引的语法

#方法一：创建表时

　　CREATE TABLE 表名 (

字段名1 数据类型 [完整性约束条件…],

字段名2 数据类型 [完整性约束条件…],

[UNIQUE | FULLTEXT | SPATIAL ] INDEX | KEY

[索引名] (字段名[(长度)] [ASC |DESC])

);

#方法二：CREATE在已存在的表上创建索引

CREATE [UNIQUE | FULLTEXT | SPATIAL ] INDEX 索引名

ON 表名 (字段名[(长度)] [ASC |DESC]) ;

#方法三：ALTER TABLE在已存在的表上创建索引

ALTER TABLE 表名 ADD [UNIQUE | FULLTEXT | SPATIAL ] INDEX

索引名 (字段名[(长度)] [ASC |DESC]) ;

#删除索引：DROP INDEX 索引名 ON 表名字;

示范

#方式一

create table t1(

id int,

name char,

age int,

sex enum('male','female'),

unique key uni\_id(id),

index ix\_name(name) #index没有key

);

#方式二

create index ix\_age on t1(age);

#方式三

alter table t1 add index ix\_sex(sex);

#查看

mysql> show create table t1;

| t1 | CREATE TABLE `t1` (

`id` int(11) DEFAULT NULL,

`name` char(1) DEFAULT NULL,

`age` int(11) DEFAULT NULL,

`sex` enum('male','female') DEFAULT NULL,

UNIQUE KEY `uni\_id` (`id`),

KEY `ix\_name` (`name`),

KEY `ix\_age` (`age`),

KEY `ix\_sex` (`sex`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1

### 六 测试索引

#### 一 准备

#1. 准备表

create table s1(

id int,

name varchar(20),

gender char(6),

email varchar(50)

);

#2. 创建存储过程，实现批量插入记录

delimiter $$ #声明存储过程的结束符号为$$

create procedure auto\_insert1()

BEGIN

declare i int default 1;

while(i<3000000)do

insert into s1 values(i,'egon','male',concat('egon',i,'@oldboy'));

set i=i+1;

end while;

END$$ #$$结束

delimiter ; #重新声明分号为结束符号

#3. 查看存储过程

show create procedure auto\_insert1\G

#4. 调用存储过程

call auto\_insert1();

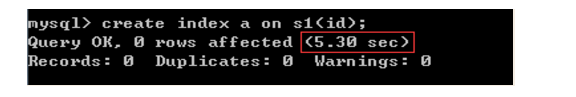
#### 二 在没有索引的前提下测试查询速度

#无索引：mysql根本就不知道到底是否存在id等于333333333的记录，只能把数据表从头到尾扫描一遍，此时有多少个磁盘块就需要进行多少IO操作，所以查询速度很慢

mysql> select \* from s1 where id=333333333;

Empty set (0.33 sec)

#### 三 在表中已经存在大量数据的前提下，为某个字段段建立索引，建立速度会很慢



#### 四 在索引建立完毕后，以该字段为查询条件时，查询速度提升明显

PS：

1. mysql先去索引表里根据b+树的搜索原理很快搜索到id等于333333333的记录不存在，IO大大降低，因而速度明显提升
2. 我们可以去mysql的data目录下找到该表，可以看到占用的硬盘空间多了
3. 需要注意，如下图



#### 五 总结

#1. 一定是为搜索条件的字段创建索引，比如select \* from s1 where id = 333;就需要为id加上索引

#2. 在表中已经有大量数据的情况下，建索引会很慢，且占用硬盘空间，建完后查询速度加快

比如create index idx on s1(id);会扫描表中所有的数据，然后以id为数据项，创建索引结构，存放于硬盘的表中。

建完以后，再查询就会很快了。

#3. 需要注意的是：innodb表的索引会存放于s1.ibd文件中，而myisam表的索引则会有单独的索引文件table1.MYI

MySAM索引文件和数据文件是分离的，索引文件仅保存数据记录的地址。而在innodb中，表数据文件本身就是按照B+Tree（BTree即Balance True）组织的一个索引结构，这棵树的叶节点data域保存了完整的数据记录。这个索引的key是数据表的主键，因此innodb表数据文件本身就是主索引。

因为inndob的数据文件要按照主键聚集，所以innodb要求表必须要有主键（Myisam可以没有），如果没有显式定义，则mysql系统会自动选择一个可以唯一标识数据记录的列作为主键，如果不存在这种列，则mysql会自动为innodb表生成一个隐含字段作为主键，这字段的长度为6个字节，类型为长整型.











分析原因

我们编写存储过程为表s1批量添加记录，name字段的值均为egon，也就是说name这个字段的区分度很低（gender字段也是一样的，我们稍后再搭理它）

回忆b+树的结构，查询的速度与树的高度成反比，要想将树的高低控制的很低，需要保证：在某一层内数据项均是按照从左到右，从小到大的顺序依次排开，即左1<左2<左3<...

而对于区分度低的字段，无法找到大小关系，因为值都是相等的，毫无疑问，还想要用b+树存放这些等值的数据，只能增加树的高度，字段的区分度越低，则树的高度越高。极端的情况，索引字段的值都一样，那么b+树几乎成了一根棍。本例中就是这种极端的情况，name字段所有的值均为'egon'

#现在我们得出一个结论：为区分度低的字段建立索引，索引树的高度会很高，然而这具体会带来什么影响呢？？？

#1：如果条件是name='xxxx',那么肯定是可以第一时间判断出'xxxx'是不在索引树中的（因为树中所有的值均为'egon’），所以查询速度很快

#2：如果条件正好是name='egon',查询时，我们永远无法从树的某个位置得到一个明确的范围，只能往下找，往下找，往下找。。。这与全表扫描的IO次数没有多大区别，所以速度很慢

**3 =和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式**

**4 索引列不能参与计算，保持列“干净”，比如from\_unixtime(create\_time) = ’2014-05-29’就不能使用到索引，原因很简单，b+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，显然成本太大。所以语句应该写成create\_time = unix\_timestamp(’2014-05-29’)**

# 十二、常用命令

## 对端/账户/系统操作

### 1. 将数据库设置为系统服务

在cmd中 输入mysql --install 将mysql设置为系统服务

前提是已经将bin文件夹写入系统环境变量

### 解除系统服务

mysql --remove 解除mysql系统服务。

### 3.查看当前用户

select user();

### 4.客户端退出

exit

### 启动服务端

net start MySQL

### 关闭服务端：

net stop MySQL

### 破解密码

如果忘记了用户密码，可以通过跳过mysql授权表的方式启动数据库服务端，修改用户密码表，修改该用户的密码。然后重启服务端登录即可。

net stop MySQL 关闭服务端，此项因为数据库已加入windows服务列表。

mysqld --skip–grant-tables 在cmd中跳过mysql授权表启动数据库服务端

mysql –uroot –p 然后登陆root账号,此时不需密码。

update mysql.user set password=password(“111”) where user=”root” and host =”localhost”;

然后杀死进程，重启mysql。

\s 查看字符编码和一系列信息

### 查看数据库进程/杀死进程

tasklist |findstr mysql 查找数据库pid

taskkill /F /PID 进程号 结束该进程

## 对库操作

### 查看所有数据库

show databases;

### 查看当前数据库

select database();

### 查看某一个数据库db1

**show create** database db1;

### 修改字符编码

alter database db1 charset gbk;

### 创建数据库

CREATE DATABASE 数据库名 charset utf8;

create table t1(id int)engine=innodb; 创建引擎为innodb的表。即使不指定，也默认是innodb。

### 查看引擎

show engines;

### 删除数据库db1

drop database db1;

### 切换数据库

use database db1;

## 对表操作

### 新建表

create table t1(id int,name char(5)); 创建表t1，并指定字段和类型。id 整型，name 字符型.char(5)是指定字符长度。若省略长度，默认为1.

指定引擎：

create table t1(id int)engine=innodb; 创建引擎为innodb的表。即使不指定，也默认是innodb。

### 查看当前数据库的所有表

show tables;

### 查看表t1所有信息

show create table t1; 查看表t1的所有信息。

### 查看表t1的结构信息

desc t1 ;

**改：**alter table t1 modify char(6); 将t1 name字段改为6字符。

alter table t1 change name NAME char(7); 将t1 name字段改为NAME,7字符。

**删：**drop table t1

### 修改表结构

#### 1. 修改表名

ALTER TABLE 表名 RENAME 新表名;

#### 2. 增加字段

#添加多个字段可以使用多个add，并用逗号隔开。

ALTER TABLE 表名 ADD 字段名 数据类型 [完整性约束条件…], ADD 字段名 数据类型 [完整性约束条件…];

指定字段位置：

ALTER TABLE 表名 ADD 字段名 数据类型 [完整性约束条件…] **FIRST**; #将字段放在首位

ALTER TABLE 表名 ADD 字段名 数据类型 [完整性约束条件…] **AFTER 字段名**; #指定新增字段在什么字段后面。

#### 3. 删除字段

ALTER TABLE 表名 DROP 字段名;

#### 4. 修改字段

ALTER TABLE 表名 MODIFY 字段名 数据类型 [完整性约束条件…];#只修改字段的约束性条件。将所有的约束性条件都要写上去。modify

ALTER TABLE 表名 CHANGE 旧字段名 新字段名 旧数据类型 [完整性约束条件…];#修改字段的名称及约束性条件。change

ALTER TABLE 表名 CHANGE 旧字段名 新字段名 新数据类型 [完整性约束条件…];#修改字段名称、数据类型及约束性条件。change

示范

示例：添加字段

mysql> alter table student10

-> add name varchar(20) not null,

-> add age int(3) not null default 22;

mysql> alter table student10

-> add stu\_num varchar(10) not null after name; //添加name字段之后

mysql> alter table student10

-> add sex enum('male','female') default 'male' first; //添加到最前面

3. 删除字段

mysql> alter table student10

-> drop sex;

mysql> alter table service

-> drop mac;

4. 修改字段类型modify

mysql> alter table student10

-> modify age int(3);

mysql> alter table student10

-> modify id int(11) not null primary key auto\_increment; //修改为主键

### 修改表存储引擎

alter table service engine=innodb;

### 复制表

create table t1 select host, user from mysql.user;

在当前数据库下创建表t1，并将mysql库中user表中的host、user字段的所有内容复制至表t1

设置条件的复制

create table t1 select host, user from mysql.user where 1>5;

设置查询的条件，当条件不被满足时，就查询不出内容，t1中就只有host,user的字段名，而没有其他内容，这样就只复制的表结构。

create table t3 like mysql.user;

复制mysql.user完成表结构，而不复制数据

### 删除表

drop table t1

## 对记录操作（增删改查）