mysql 学习笔记

面试：

1. **char 和 varchar 数据类型的区别及存储方式？**
2. **pymysql 写 sql 语句，不要用 python 格式化方式写入字符串变量，用 pymysql 提供的%s+列表或字典的方法**

```
cursor.execute("insert into user_info([username, password) values(%s, %s)" , [username, password])
```

前言：

Web 服务器部署环境：

LAMP(Linux + Apache + MySQL/MariaDB + PHP)

LNMP(Linux +Nginx + MySQL/MariaDB + PHP)

什么是数据库？

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库。

每个数据库都有一个或多个不同的 API 用于创建，访问，管理，搜索和复制所保存的数据。

我们也可以将数据存储在文件中，但是在文件中读写数据速度相对较慢。

所以，现在我们使用关系型数据库管理系统（RDBMS）来存储和管理大数据量。所谓的关系型数据库，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。

RDBMS 即关系数据库管理系统(Relational Database Management System)的特点：

1.数据以表格的形式出现

2.每行为各种记录名称

3.每列为记录名称所对应的数据域

4.许多的行和列组成一张表单

5.若干的表单组成 database

RDBMS 术语

在我们开始学习 MySQL 数据库前，让我们先了解下 RDBMS 的一些术语：

数据库: 数据库是一些关联表的集合。

数据表: 表是数据的矩阵。在一个数据库中的表看起来像一个简单的电子表格。

列: 一列(数据元素) 包含了相同类型的数据, 例如邮政编码的数据。

行：一行（=元组，或记录）是一组相关的数据，例如一条用户订阅的数据。

冗余：存储两倍数据，冗余降低了性能，但提高了数据的安全性。

主键：主键是唯一的。一个数据表中只能包含一个主键。你可以使用主键来查询数据。

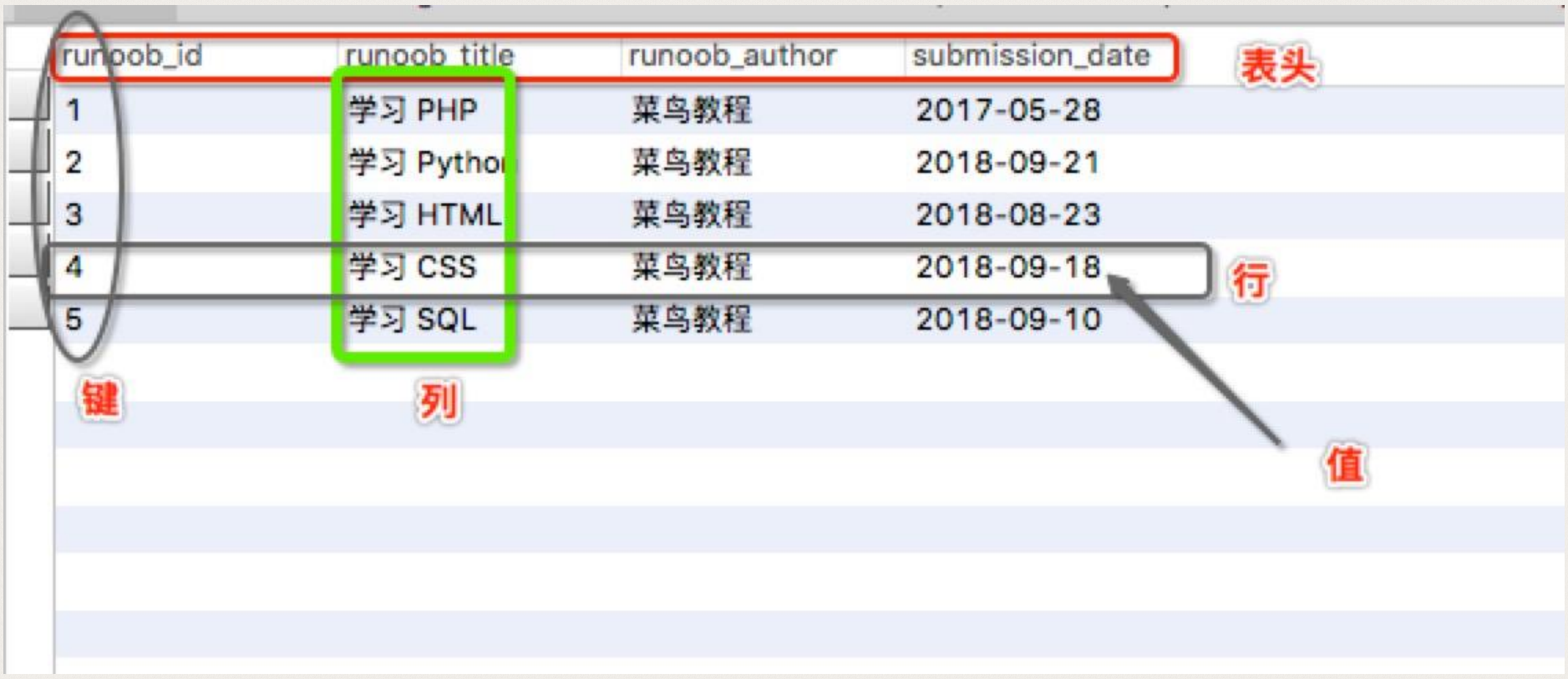
外键：外键用于关联两个表。

复合键：复合键（组合键）将多个列作为一个索引键，一般用于复合索引。

索引：使用索引可快速访问数据库表中的特定信息。索引是对数据库表中一列或多列的值进行排序的一种结构。类似于书籍的目录。

参照完整性: 参照的完整性要求关系中不允许引用不存在的实体。与实体完整性是关系模型必须满足的完整性约束条件，目的是保证数据的一致性。

MySQL 为关系型数据库(Relational Database Management System)，这种所谓的"关系型"可以理解为"表格"的概念，一个关系型数据库由一个或数个表格组成，如图所示的一个表格:



- 表头(header): 每一列的名称;
- 列(col): 具有相同数据类型的数据的集合;
- 行(row): 每一行用来描述某条记录的具体信息;
- 值(value): 行的具体信息, 每个值必须与该列的数据类型相同;
- 键(key): 键的值在当前列中具有唯一性。

MySQL 数据库

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 公司。MySQL 是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL 是开源的，目前隶属于 Oracle 旗下产品。

MySQL 支持大型的数据库。可以处理拥有上千万条记录的大型数据库。

MySQL 使用标准的 SQL 数据语言形式。

MySQL 可以运行于多个系统上，并且支持多种语言。这些编程语言包括 C、C++、Python、Java、Perl、PHP、Eiffel、Ruby 和 Tcl 等。

MySQL 对 PHP 有很好的支持，PHP 是目前最流行的 Web 开发语言。

MySQL 支持大型数据库，支持 5000 万条记录的数据仓库，32 位系统表文件最大可支持 4GB，64 位系统支持最大的表文件为 8TB。

MySQL 是可以定制的，采用了 GPL 协议，你可以修改源码来开发自己的 MySQL 系统。

1. 安装 Mysql

- **安装 mysql5.7.36 版本**

/更新 centeros 出错解决 :https://blog.csdn.net/sqlquan/article/details/123226913

- **//Vm 环境搭建 centos**

  学习链接：https://docs.qq.com/doc/DWHhFTGpyQ09xZkdo

- **//msyql 下载地址**：https://downloads.mysql.com/archives/community/

MySQL Community Downloads

MySQL Community Server

MySQL官方安装文档
点击这里进入

General Availability (GA) Releases | Archives

选择社区版的5.7.36，非社区版的要收费

MySQL Community Server 5.7.36

5.7是上一个版本，
较稳定的版本，
新版的是8.0，
可点击这里切换查看

Looking for the latest GA version?

Select Operating System:

Linux - Generic

选择Linux通用版本

Select OS Version:

All

Linux - Generic (glibc 2.12) (x86, 32-bit), Compressed TAR Archive (mysql-5.7.36-linux-glibc2.12-i686.tar.gz)	5.7.36	603.3M	Download
		MD5: a0852fc908eab72bde6d4d1e27a35ebf | Signature	
这里选64位，三进制压缩版本 **Linux - Generic (glibc 2.12) (x86, 64-bit), Compressed TAR Archive** (mysql-5.7.36-linux-glibc2.12-x86_64.tar.gz)	5.7.36	635.7M	Download
		MD5: 1748ec2c8b5ca1bcf8ba3b1f5e956139 | Signature	
Linux - Generic (glibc 2.12) (x86, 32-bit), Compressed TAR Archive **Test Suite** (mysql-test-5.7.36-linux-glibc2.12-i686.tar.gz)	5.7.36	32.2M	Download
		MD5: 22134fbe428880e274df258e0abbf66a | Signature	
Linux - Generic (glibc 2.12) (x86, 64-bit), Compressed TAR Archive **Test Suite** (mysql-test-5.7.36-linux-glibc2.12-x86_64.tar.gz)	5.7.36	32.5M	Download
		MD5: acb176c10d6dcca8e7a2d7d60e33cecc | Signature	
Linux - Generic (glibc 2.12) (x86, 32-bit), TAR (mysql-5.7.36-linux-glibc2.12-i686.tar)	5.7.36	635.5M	Download
		MD5: 86e38d0d59d28245146e5757ea596a62 | Signature	
Linux - Generic (glibc 2.12) (x86, 64-bit), TAR (mysql-5.7.36-linux-glibc2.12-x86_64.tar)	5.7.36	668.2M	Download
		MD5: caebde1c69440bd03754be217ed86954 | Signature	

//正式安装

```
# 检查是否存在 mysql
rpm    -qa | grep mysql


# 删除***包
rpm -e --nodeps ***


# 查看原版本的文件
whereis mysql
find / -name mysql

# 删除相关目录或文件
rm -rf /usr/bin/mysql /usr/include/mysql /data/mysql /data/mysql/mysql


# 安装从网上下载文件的 wget 命令
yum -y install wget
```

```
# 下载 mysql 通用版、压缩包
wget https://downloads.mysql.com/archives/get/p/23/file/mysql-5.7.36-linux-glibc2.12-x86_64.tar.gz

# 移动文件、解压
mv mysql-5.7.36-linux-glibc2.12-x86_64.tar.gz /usr/local/
tar -xvf mysql-5.7.36-linux-glibc2.12-x86_64.tar.gz

# 重命名
mv mysql-5.7.36-linux-glibc2.12-x86_64 mysql

# mysql 在不同版本会有一些不同的依赖，为了避免后面奇奇怪怪的错误，最好都安装上。
yum install libaio libaio-devel numactl-libs wget vim -y

# 创建用户和组
# 因为用户仅用于所有权目的，而不是登录目的，所以 useradd 命令使用 -r 和-s /bin/false 选项
groupadd mysql
useradd -r -g mysql -s /bin/false mysql


# 环境变量设置：方式一
# 创建软连接、为了避免在使用 MySQL 时总是输入客户端程序的路径名，将/usr/local/mysql/bin 目录添加到 PATH 变量中
vim /etc/bashrc
# 在文件头部加上如下代码后保存退出 export PATH=$PATH:/usr/local/mysql/bin # 刷新文件，使配置生效
source /etc/bashrc


# 环境变量设置：方式二
# 添加用户级环境变量 profile.d--好维护，不要直接删除.sh 文件即可。
vim /etc/profile.d/mysql.sh
# 写入内容保存
export PATH="$PATH:/usr/local/mysql/bin"
# 重载环境变量文件
source /etc/profile
# 查看当前环境变量是否存在
echo $PATH


# 创建相应目录并赋予合适权限
mkdir /usr/local/mysql/data
chown -R mysql:mysql /usr/local/mysql
chmod -R 755 /usr/local/mysql

# bin 目录下的工具 mysqld 查看配置文件优先级(我选/usr/local/mysql/etc/my.cnf，为了防止多个 mysql 版本安装相互影响)
/usr/local/mysql/bin/mysqld --verbose --help | grep my.cnf
```

```
# 编译安装并初始化
/usr/local/mysql/bin/mysqld  --initialize --user=mysql  --datadir=/usr/local/mysql/data  --basedir=/usr/local/mysql
# 注意保存密码，最后几行生成 A temporary password is generated for root@localhost: .eJm(RQyys7xR
# 没注意看，可以到日志目录查看

#创建配置文件并 vim 写入
touch /usr/local/mysql/etc/my.cnf
vim  /usr/local/mysql/etc/my.cnf

# 复制配置数据写入（初始化参数一定要填写正确！）  注意开头是 mysqld
[mysqld]
port = 3306
bind-address=0.0.0.0
socket=/tmp/mysql.sock
basedir=/usr/local/mysql
datadir=/usr/local/mysql/data

# 启动服务
service mysql start
# 重启服务
service mysql restart
# 假设报错可以关掉进程再重启，也可能是配置文件有问题！
ps -ef|grep mysql
ps -ef|grep mysqld
kill -9 PID
service mysqld restart
# 查看状态
systemctl status mysql

# 设置开启自动启动
# 将 mysql.server 放置到/etc/init.d/mysql 中
cp /usr/local/mysql/support-files/mysql.server /etc/init.d/mysqld
# 赋予可执行权限
chmod +x /etc/init.d/mysqld
# 添加服务
chkconfig --add mysqld
# 显示服务列表
chkconfig --list
# 如果看到 mysql 的服务，并且 3,4,5 都是开的话则成功，如果是关，则键入
chkconfig --level 345 mysqld on
```

修改密码

mysql -u root -p

复制粘贴输入之前生成的密码：eJm(RQyys7xR（比如我的）

mysql>set password = password('Cc158854@');

开放远程连接

mysql>use mysql;

msyql>update user set user.Host='%' where user.User='root';

mysql>flush privileges;

远程连接需要关闭防火墙

查看状态

systemctl status firewalld.service

打开防火墙

systemctl start firewalld.service

关闭防火墙

systemctl stop firewalld.service

开启防火墙

systemctl enable firewalld.service

禁用防火墙

systemctl disable firewalld.service

关闭 **selinux**

vi /etc/selinux/config

enforcing 强制模式

permissive 宽容模式

disable 关闭

OK !!!!!!!!!!!!!

现在可以使用 SQLyog 爽爽的连接了~~~~~~~~~~~~~~~~~~~~~~~~~~

//忘记密码操作

配置文件修改

[mysqld]

... ...

skig-grant-table=1

... ...

重启 mysql 服务

```
systemctl restart mysql


# 无密码进入

mysql -uroot -p


# 修改密码

mysql>use mysql;

mysql>update usr set authentication_string=password("新密码"), password_last_changed=now() where user="root";


# 配置文件删除：skig-grant-table=1

[mysqld]

... ...

skig-grant-table=1

... ...

# 再次重启即可新密码登录~~~~~~
```

//my.cnf 文件解释

```
###################### 解释版本 start ##########################
[mysqld]
# 设置能访问数据库的 ip 地址 0.0.0.0 代表无限制
bind-address=0.0.0.0
# mysqld 服务端监听端口
port=3306
# mysql 数据库运行所使用的用户
user=mysql
# mysql 安装位置
basedir=/usr/local/mysql
# 数据库存放位置
datadir=/usr/local/mysql/data
# 本地连接的 socket 套接字
socket=/tmp/mysql.sock
# 错误日志存放位置
log_error=/usr/local/mysql/log/mysql.err
# 错误日志等级
log_warnings = 2
# pid-file 文件位置 当 MySQL 实例启动时，会将自己的进程 ID 写入一个文件中——该文件即为 pid 文件
pid_file=/usr/local/mysql/mysql.pid
# 默认字符集设置，mysql 无脑 utf8mb4 就对了，别杠，杠就是你对，别问，问就是不知道
character_set_server=utf8mb4
# 数据库字符集对应排序规则，需要与字符集一一对应
# 对于 utf8 常用的排序规则可以看这个网址：https://blog.csdn.net/qq_17555933/article/details/101445526
collation_server = utf8mb4_unicode_ci
# 设置 client 连接 mysql 时的字符集，防止乱码
init_connect='SET NAMES utf8mb4'
# 是否对 sql 语句大小写敏感，1 表示不敏感
lower_case_table_names = 1
# 禁止 mysql 使用软连接
symbolic_links=0
# https://www.cnblogs.com/zqifa/p/mysql-3.html 自己去看
explicit_defaults_for_timestamp=true
###################### 解释版本 end ##########################
```

2. 数据库环境

登录数据库：

//格式：mysql -u 用户名 -P 端口号 -p
//输入密码
//默认端口号为 3306 的，无需加入参数大写：-P
mysql -u root -pcc158854；
//修改密码
mysqladmin -uroot -p
cc158854{d};
help;

参数说明：

（\c）清除当前输入语句。
（\r）重新连接到服务器。可选参数是 db 和 host。
（\d）设置语句分隔符。
（\e）用$EDITOR 编辑命令。
（\G）向 MariaDB 服务器发送命令，垂直显示结果。
（\q）退出 mysql。和退出一样。
（\g）向 MariaDB 服务器发送命令。
（\h）显示此帮助。
（\n）禁用寻呼机，打印到标准输出。
（\t）不写入 outfile。
（\P）设置寻呼机[到寻呼机]。通过寻呼机打印查询结果。
（\p）打印当前命令。
（\R）更改 mysql 提示符。
（\q）退出 mysql。
（\#）重建完成哈希。
（\）执行 SQL 脚本文件。以文件名作为参数。
（\s）从服务器获取状态信息。
(\!) 执行系统 shell 命令。
（\T）设置 outfile[到 outfile]。将所有内容附加到给定的 outfile 中。
（\u）使用其他数据库。以数据库名称为参数。
（\C）切换到另一个字符集。可能需要处理具有多字节字符集的 binlog。
（\W）在每条语句后显示警告。
（\w）不要在每条语句后显示警告。

//显示所有数据库
show databases;

//使用数据库 mysql
use mysql

///时间/连接的用户/查看当前连接的数据库/数据库服务版本
select now();
select user();
select database();
select version();

3. 数据库用户

//新增 mysql 数据库的用户

//进入数据库
use test;
//新增用户
insert into user (host, user, password, select_priv, insert_priv, update_priv) values ('localhost', 'guest', password('guest123'), 'Y', 'Y', 'Y');
//刷新权限
flush privileges;
//显示所有数据库用户信息

```
select * from mysql.user;
//指定显示所有数据库用户信息
select user,host from mysql.user;
//显示某个用户的指定信息
select password from mysql.user where user="root";
//显示所有不重复的用户
select distinct user mysql.user;
```

4. 数据值类型

4.1 数值类型

MySQL 支持所有标准 SQL 数值数据类型。
这些类型包括严格数值数据类型(INTEGER、SMALLINT、DECIMAL 和 NUMERIC)，以及近似数值数据类型(FLOAT、REAL 和 DOUBLE PRECISION)。
关键字 INT 是 INTEGER 的同义词，关键字 DEC 是 DECIMAL 的同义词。
BIT 数据类型保存位字段值，并且支持 MyISAM、MEMORY、InnoDB 和 BDB 表。
作为 SQL 标准的扩展，MySQL 也支持整数类型 TINYINT、MEDIUMINT 和 BIGINT。下面的表显示了需要的每个整数类型的存储和范围。

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 byte	(-128，127)	(0，255)	小整数值
SMALLINT	2 bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT 或 INTEGER	4 bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 bytes	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度 浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度
DECIMAL	对 DECIMAL(M,D) ，如果 M>D，为 M+2 否则为 D+2	依赖于 M 和 D 的值	依赖于 M 和 D 的值	小数值

4.2 日期和时间类型

表示时间值的日期和时间类型为 DATETIME、DATE、TIMESTAMP、TIME 和 YEAR。
每个时间类型有一个有效值范围和一个"零"值，当指定不合法的 MySQL 不能表示的值时使用"零"值。
TIMESTAMP 类型有专有的自动更新特性，将在后面描述。

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970/1/1 0:00	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

		结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038 年 1 月 19 日 凌晨 03:14:07	

4.3 字符串类型

字符串类型指 CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM 和 SET。该节描述了这些类型如何工作以及如何在查询中使用这些类型。

char(n) 和 varchar(n) 中括号中 n 代表字符的个数，并不代表字节个数，比如 CHAR(30) 就可以存储 30 个字符。

CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。

BINARY 和 VARBINARY 类似于 CHAR 和 VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。

BLOB 是一个二进制大对象，可以容纳可变数量的数据。有 4 种 BLOB 类型：TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。它们区别在于可容纳存储范围不同。

有 4 种 TEXT 类型：TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。对应的这 4 种 BLOB 类型，可存储的最大长度不同，可根据实际情况选择。

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

5. 语言分类

一、DQL、DML、DDL、DCL 的概念与区别

SQL(Structure Query Language)语言是数据库的核心语言。
SQL 的发展是从 1974 年开始的，其发展过程如下：
1974 年-----由 Boyce 和 Chamberlin 提出，当时称 SEQUEL。
1976 年-----IBM 公司的 Sanjase 研究所在研制 RDBMS SYSTEM R 时改为 SQL。
1979 年-----ORACLE 公司发表第一个基于 SQL 的商业化 RDBMS 产品。
1982 年-----IBM 公司出版第一个 RDBMS 语言 SQL/DS。
1985 年-----IBM 公司出版第一个 RDBMS 语言 DB2。
1986 年-----美国国家标准化组织 ANSI 宣布 SQL 作为数据库工业标准。
SQL 是一个标准的数据库语言，是面向集合的描述性非过程化语言。它功能强，效率高，简单易学易维护（迄今为止，我还没见过比它还好学的语言）。然而 SQL 语言由于以上优点，同时也出现了这样一个问题：它是非过程性语言，即大多数语句都是独立执行的，与上下文无关，而绝大部分应用都是一个完整的过程，显然用 SQL 完全实现这些功能是很困难的。所以大多数数据库公司为了解决此问题，作了如下两方面的工作：(1)扩充 SQL，在 SQL 中引入过程性结构；(2)把 SQL 嵌入到高级语言中，以便一起完成一个完整的应用。

二、SQL 语言的分类

SQL 语言共分为四大类：数据查询语言 DQL，数据操纵语言 DML，数据定义语言 DDL，数据控制语言 DCL。

1. 数据查询语言 DQL——Data Query Language

数据查询语言 DQL 基本结构是由 SELECT 子句，FROM 子句，WHERE 子句组成的查询块：

SELECT <字段名表>

FROM <表或视图名>

WHERE <查询条件>

2 .数据操纵语言 DML——Data Manipulate Language

数据操纵语言 DML 主要有三种形式：

1) 插入：INSERT

2) 更新：UPDATE

3) 删除：DELETE

3. 数据定义语言 DDL——Data Define Language

数据定义语言 DDL 用来创建数据库中的各种对象-----表、视图、索引、同义词、聚簇等如：

CREATE TABLE/VIEW/INDEX/SYN/CLUSTER

表 视图 索引 同义词 簇

DDL 操作是隐性提交的！不能 rollback

4. 数据控制语言 DCL——Data Control Language

数据控制语言 DCL 用来授予或回收访问数据库的某种特权，并控制数据库操纵事务发生的时间及效果，对数据库实行监视等。如：

1) GRANT：授权。

2) ROLLBACK [WORK] TO [SAVEPOINT]：回退到某一点。

回滚---ROLLBACK

回滚命令使数据库状态回到上次最后提交的状态。其格式为：

SQL>ROLLBACK;

3) COMMIT [WORK]：提交。

在数据库的插入、删除和修改操作时，只有当事务在提交到数据库时才算完成。在事务提交前，只有操作数据库的这个人才能有权看到所做的事情，别人只有在最后提交完成后才可以看到。

提交数据有三种类型：显式提交、隐式提交及自动提交。下面分别说明这三种类型。

(1) 显式提交

用 COMMIT 命令直接完成的提交为显式提交。其格式为：

SQL>COMMIT;

(2) 隐式提交

用 SQL 命令间接完成的提交为隐式提交。这些命令是：

ALTER，AUDIT，COMMENT，CONNECT，CREATE，DISCONNECT，DROP，EXIT，GRANT，NOAUDIT，QUIT，REVOKE，RENAME。

(3) 自动提交

若把 AUTOCOMMIT 设置为 ON，则在插入、修改、删除语句执行后，系统将自动进行提交，这就是自动提交。其格式为：

SQL>SET AUTOCOMMIT ON;

5.事务控制语言 TCL——Data Control Language

#TCL

/*

Transaction Control Language 事务控制语言

事务：

一个或一组 sql 语句组成一个执行单元，这个执行单元要么全部执行，要么全部不执行。

案例：转账

张三丰 1000

郭襄 1000

update 表 set 张三丰的余额=500 where name='张三丰'

意外

update 表 set 郭襄的余额=1500 where name='郭襄'

事务的特性：

ACID

原子性：一个事务不可再分割，要么都执行要么都不执行

一致性：一个事务执行会使数据从一个一致状态切换到另外一个一致状态

隔离性：一个事务的执行不受其他事务的干扰

持久性：一个事务一旦提交，则会永久的改变数据库的数据.

事务的创建

隐式事务：事务没有明显的开启和结束的标记

比如 insert、update、delete 语句

delete from 表 where id =1;
显式事务：事务具有明显的开启和结束的标记
前提：必须先设置自动提交功能为禁用
set autocommit=0;
步骤 1：开启事务
set autocommit=0;
start transaction;可选的
步骤 2：编写事务中的 sql 语句(select insert update delete)
语句 1;
语句 2;
...
步骤 3：结束事务
commit;提交事务
rollback;回滚事务

savepoint 节点名;设置保存点
事务的隔离级别：
脏读 不可重复读 幻读
read uncommitted：√ √ √
read committed： × √ √
repeatable read： × × √
serializable × × ×
mysql 中默认 第三个隔离级别 repeatable read
oracle 中默认第二个隔离级别 read committed
查看隔离级别
select @@tx_isolation;
设置隔离级别
set session|global transaction isolation level 隔离级别;

开启事务的语句;
update 表 set 张三丰的余额=500 where name='张三丰'
update 表 set 郭襄的余额=1500 where name='郭襄'
结束事务的语句;
*/
SHOW VARIABLES LIKE 'autocommit';
SHOW ENGINES;

#1.演示事务的使用步骤
#开启事务
SET autocommit=0;
START TRANSACTION;
#编写一组事务的语句
UPDATE account SET balance = 1000 WHERE username='张无忌';
UPDATE account SET balance = 1000 WHERE username='赵敏';
#结束事务
ROLLBACK;
#commit;
SELECT * FROM account;
*/

#2.演示事务对于 delete 和 truncate 的处理的区别
SET autocommit=0;
START TRANSACTION;
DELETE FROM account;
ROLLBACK;

#3.演示 savepoint 的使用
SET autocommit=0;
START TRANSACTION;
DELETE FROM account WHERE id=25;
SAVEPOINT a;#设置保存点
DELETE FROM account WHERE id=28;
ROLLBACK TO a;#回滚到保存点
SELECT * FROM account;

6. 数据库操作指令

- **基本命令**

use test;
show database;
show tables; //查看当前数据库表
show tables from mysql; //查看 mysql 里面的表

- **创建新表**

```
create table stuinfo(
    -> id int,
    -> name varchar(20));
desc stuinfo;

//查看数据表结构
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int         | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+

//查看数据表信息
select * from stuinfo;
Empty set (0.00 sec)
insert into stuinfo (id,name) values(1,'John')
insert into stuinfo (id,name) values(2,'Rose')
select * from stuinfo;
+------+------+
| id   | name |
+------+------+
|    1 | John |
|    2 | John |
+------+------+
2 rows in set (0.00 sec)

//修改数据表内容
update stuinfo set name='李雷' where id=1;
select * from stuinfo;
+------+--------+
| id   | name   |
+------+--------+
|    1 | 李雷   |
|    2 | John   |
+------+--------+
2 rows in set (0.00 sec)
```

总结语法规范：

不区大小写(默认。可以配置文件 mys.cnf 中修改)，建议关键字大写
分号结尾；
根据命令需要，可分行书写；
    注释格式
    单行注释：#注释文字 or --注释文字
    多行注释：/*注释文字*/

6.1 创建数据库

创建 SQL 语法格式

```
#  默认创建方式
create database db_name;
#  标准格式写法
CREATE DATABASE [IF NOT EXISTS] db_name [DEFAULT] CHARACTER SET [=] charset_name collate utf8_general_ci;
```

db_name: 数据库名称
charset_name: 字符编码（中文编码字符集名称：utf8 而不是 utf-8）

推荐创建方式（utf8_general_ci 大小写不敏感，..._cs 大小写敏感**）**

create database db_name default charset utf8 collate utf8_general_ci；

实验：

6.2 修改数据库

修改编码格式 SQL 语法格式：

//查看当前数据库编码格式
show variables like "character%"
//修改数据库语法
alter database db_name [default] character set [=] charset_name;

6.3 删除数据库

删除 SQL 语法格式：

DROP DATABASE [IF EXISTS] db_name;
关于删除知识：
1、drop table table_name : 删除表全部数据和表结构，立刻释放磁盘空间，不管是 Innodb 和 MyISAM;
实例，删除学生表：
drop table student;

2、truncate table table_name : 删除表全部数据，保留表结构，立刻释放磁盘空间 ，不管是 Innodb 和 MyISAM;
实例，删除学生表：
truncate table student;

3、delete from table_name : 删除表全部数据，表结构不变，对于 MyISAM 会立刻释放磁盘空间，InnoDB 不会释放磁盘空间;
实例，删除学生表：
delete from student;

4、delete from table_name where xxx : 带条件的删除，表结构不变，不管是 innodb 还是 MyISAM 都不会释放磁盘空间;
实例，删除学生表中姓名为 "张三" 的数据：
delete from student where T_name = "张三";

5、delete 操作以后，使用 optimize table table_name 会立刻释放磁盘空间，不管是 innodb 还是 myisam;
实例，删除学生表中姓名为 "张三" 的数据：
delete from student where T_name = "张三";
实例，释放学生表的表空间：
optimize table student;

6、delete from 表以后虽然未释放磁盘空间，但是下次插入数据的时候，仍然可以使用这部分空间。

总结：

1、当你不再需要该表时， 用 drop;

2、当你仍要保留该表，但要删除所有记录时， 用 truncate;

3、当你要删除部分记录时， 用 delete。

mysql truncate、delete 与 drop 区别

相同点：
1.truncate 和不带 where 子句的 delete、以及 drop 都会删除表内的数据。
2.drop、truncate 都是 DDL 语句(数据定义语言),执行后会自动提交。
不同点：
1. truncate 和 delete 只删除数据不删除表的结构(定义)
drop 语句将删除表的结构被依赖的约束(constrain)、触发器(trigger)、索引(index)；依赖于该表的存储过程/函数将保留,但是变为 invalid 状态。
2. delete 语句是数据库操作语言(dml)，这个操作会放到 rollback segement 中，事务提交之后才生效；如果有相应的 trigger，执行的时候将被触发。
truncate、drop 是数据库定义语言(ddl)，操作立即生效，原数据不放到 rollback segment 中，不能回滚，操作不触发 trigger。
3.delete 语句不影响表所占用的 extent，高水线(high watermark)保持原位置不动
drop 语句将表所占用的空间全部释放。

truncate 语句缺省情况下见空间释放到 minextents 个 extent，除非使用 reuse storage；truncate 会将高水线复位(回到最开始)。
4.速度，一般来说: drop> truncate > delete
5.安全性：小心使用 drop 和 truncate，尤其没有备份的时候.否则哭都来不及
使用上,想删除部分数据行用 delete，注意带上 where 子句. 回滚段要足够大.
想删除表,当然用 drop
想保留表而将所有数据删除，如果和事务无关，用 truncate 即可。如果和事务有关,或者想触发 trigger,还是用 delete。
如果是整理表内部的碎片，可以用 truncate 跟上 reuse stroage，再重新导入/插入数据。
6.delete 是 DML 语句,不会自动提交。drop/truncate 都是 DDL 语句,执行后会自动提交。
7、TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同：二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少。DELETE 语句每次删除一行，并在事务日志中为所删除的每行记录一项。TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。
8、TRUNCATE TABLE 删除表中的所有行，但表结构及其列、约束、索引等保持不变。新行标识所用的计数值重置为该列的种子。如果想保留标识计数值，请改用 DELETE。如果要删除表定义及其数据，请使用 DROP TABLE 语句。
9、对于由 FOREIGN KEY 约束引用的表，不能使用 TRUNCATE TABLE，而应使用不带 WHERE 子句的 DELETE 语句。由于 TRUNCATE TABLE 不记录在日志中，所以它不能激活触发器。
10、TRUNCATE TABLE 不能用于参与了索引视图的表。

7. 进阶一：表结构操作

7.1 数据库表的创建

id	name	email	age

```
--切换数据库
use dbname;
--查看存在表
show tables;

--创建数据表
create table tbname(
      列名 1 类型，
      列名 2 类型，
      列名 3 类型，
) default charset=utf8;

--一类：
create table tbA(
    id int，
    name varchar(16)，
) default charset=utf8;

--二类：
create table tbB(
    id int，
    name varchar(16) not null， --不允许为空
    email varchar(32) null,  --允许为空(默认)
    age int
) default charset=utf8;

--三类：
create table tbC(
    id int，
    name varchar(16) not null， --不允许为空
    email varchar(32) null,  --允许为空(默认)
    age int default 3,  --插入数据时，如果不设置值，默认值为 3
) default charset=utf8;

--四类：
create table tbD(
    id int primary key,  --设置主键（不允许为空且不能重复）
    name varchar(16) not null， --不允许为空
    email varchar(32) null,  --允许为空(默认)
    age int default 3,  --插入数据时，如果不设置值，默认值为 3
```

```
) default charset=utf8;

--五类：
create table tbE(
    id int primary key,   --设置主键（不允许为空且不能重复）
    name varchar(16) not null，  --不允许为空
    email varchar(32) null,   --允许为空(默认)
    age int default 3,   --插入数据时，如果不设置值，默认值为 3
) default charset=utf8;


--六类：enum-枚举
create table tbF(
    id int primary key,   --设置主键（不允许为空且不能重复）
    name varchar(16) not null，  --不允许为空
    email varchar(32) null,   --允许为空(默认)
    age int default 3,   --插入数据时，如果不设置值，默认值为 3
    gender enum("男", "女")   default "男" not null,   --枚举类型，男或女，默认为男
) default charset=utf8;
```

枚举和集合类型： https://ramble.3vshej.cn/mysql-enum-set-field-usage/

作业实例：

```
create table hwork(
        id int not null primary key auto_increment,
        name varchar(32) not null,
        password varchar(64) not null,
        gender enum('男','女') not null default '男',
        email varchar(64),
        amount decimal(10,2) not null default 0,
        ctime datetime
)default charset=utf8;


insert into hwork(
        name,
        password,
        gender,
        email,
        amount,
        ctime
)values(
        'test1',
        'test1',
        '女',
        '1@qq.com',
        123.89,
        '2022-02-27'
);
```

修改数据表的名称

alter table 旧表名 rename to 新表名；

7.2 创建主键自增表及主键添加删除

```
-- 创建表
create table tbE(
        id int not null auto_increment primary key,   --设置主键 & 不能为空 & 自增
        name varchar(16) not null，  --不允许为空
        email varchar(32) null,   --允许为空(默认)
        age int default 25,   --插入数据时，如果不设置值，默认值为 25
) default charset=utf8;

-- 添加主键
mysql> ALTER TABLE testalter_tbl MODIFY i INT NOT NULL;
```

```
mysql> ALTER TABLE testalter_tbl ADD PRIMARY KEY (i);

-- 删除主键
mysql> ALTER TABLE testalter_tbl DROP PRIMARY KEY;
```

7.3 查看数据表结构

desc 表名;

7.4 删除数据整张表

drop table 表名;

7.3 清空数据表内容

delete from 表名;
--或者
truncate table 表名; --速度快，无法撤回及恢复。

7.4 修改数据表结构

- **增加列**

alter table 表名 add 列名 类型;
alter table 表名 add 列名 类型 default 值; --设置默认值
alter table 表名 add 列名 类型 default 值 not null; --设置默认值 & 不能为空
alter table 表名 add 列名 类型 default 值 not null primary key; --设置默认值 & 不能为空 & 设为主键

- **删除列**

alter table 表名 drop column 列名

- **修改列：类型**

alter table 表名 modify column 列名 新类型；

- **修改列：类型+名称**

alter table 表名 change 原列名 新列名 新类型;

```
--例子
alter table tbname change id nid int not null;
alter table tbname change id nid int not null primary key;
alter table tbname change id nid int not null primary key auto_increment;
alter table tbname change id nid int;    --清除之前设置的类型
```

- **修改列：其他场景**

```
--列重新设置默认值
alter table 表名 alter 列名 set default 重设默认值;

--列去掉设置默认值
alter table 表名 alter 表名 drop default;

--列添加主键属性
alter table 表名 add primary key (列名)

--列去掉主键属性
alter table 表名 drop 表名 primary key;
```

7.3 极常用数据类型

- **int 类型**

int --整数
int unsigned --无符号正整数
int(5) zerofill --5 位显示，左边不满 5 用 0 补齐

实验创建表及插入数据

```
--创建数据表
create table L1 (
        id int not null primary key auto_increment,
        uid int unsigned,
        zid int(5)zerofill
) default charset=utf8;

--插入数据
insert into L1(uid, zid) value(1, 1)
insert into L1(uid, zid) value(2, 2)
+----+------+-------+
| id | uid  | zid   |
+----+------+-------+
|  1 |    1 | 00001 |
|  2 |    2 | 00002 |
+----+------+-------+
2 rows in set (0.00 sec)

--数据类型不匹配会报错
insert into L1(uid, zid) values (214748364100, 4294967294, 3000000)
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

- **decimal 类型**

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

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

实例：工资表

```
create table L2(
        id int auto_increment primary key not null,
        salary decimal(8,2)
)default charset=utf8;

--插入数据 decimal 保存两位小数，支持四舍五入存入
insert into L2(salary)values(1200.00);
insert into L2(salary)values(1200.8444);
insert into L2(salary)values(1200.8555);

mysql> select * from L2;
+----+---------+
| id | salary  |
+----+---------+
|  1 | 1200.00 |
|  2 | 1200.84 |
|  3 | 1200.86 |
+----+---------+3 rows in set (0.00 sec)
```

- **char 类型**

char(m)

定长字符串,m 代表字符串的长度，最多可容纳 255 个字符。
定长的体现:即使内容长度小于 m，也会占用 m 长度。
例如: char(5)，数据是: yes，底层也会占用 5 个字符;
如果超出 m 长度限制（默认 MysQL 是严格模式，所以会报错)。
如果在配置文件中加入如下配置，
sql-mode="NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION",
保存并重启服务，此时 MySQz 则是非严格模式，此时超过长度则自动截断（不报错）... ...
注意:默认底层存储是固定的长度（不够则用空格补齐)，但是查询数据时，会自动将空白去除体现出来还是那么多字符。
如果想要保留空白，在 sql-mode 中加入 PAD_CHAR_TO_FULL_LENGTH 即可。
例如：

```
create table L3(
        -> id int not null primary key auto_increment,
        -> name varchar(5),
        -> depart char(3)
        -> )default charset=utf8;
mysql> select id,name,depart,length(name),length(depart) from L3;
+----+------+--------+--------------+---------------+
```

| id | name | depart | length(name) | length(depart) |
+----+------+--------+--------------+----------------+
| 1 | chc | ds | 3 | 2 |
+----+------+--------+--------------+----------------+
在 sql-mode 中加入 PAD_CHAR_TO_FULL_LENGTH 后再次查询
mysql> select id,name,depart,length(name),length(depart) from L3;
+----+------+--------+--------------+----------------+
| id | name | depart | length(name) | length(depart) |
+----+------+--------+--------------+----------------+
| 1 | chc | ds | 3 | 3 |
+----+------+--------+--------------+----------------+

查看模式 sql-mode，执行命令:
show variables like 'sql_mode';

- **varchar 类型**

变长字符串，m 代表字符串的长度,最多可容纳 65535 个字节。
变长的体现:内容小于 m 时，会按照真实数据长度存储;如果超出 m 长度限制（(默认 MysQL 是严格模式，所以会报错)。
如果在配置文件中加入如下配置:
sql-mode="NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION"
保存并重启，此时 MySQL 则是非严格模式，此时超过长度则自动截断（不报错)。

8. 进阶二：表内容操作

8.1 新增数据

insert into 表名(字段 1,字段 2,字段 3) values(值 1,值 2,值 3)

--顺序插入（插入值的数量必须和字段数相等，从左到右填入）
insert into values(值 1,值 2,值 3)

8.2 删除数据

delete from 表名; --删除整表
delete from 表名 where 条件; --删除符合条件的行
delete from L3; --删除整表
delete from L3 where name="chc"; --删除对应列
delete from L3 where id>3; --清空 id 大于 3 的行
delete from L3 where name="chc" and id=1; --删除对应单元格的行

8.3 修改数据

update 表名 set 列名=值； --修改列的值
update 表名 set 列名=值 where 条件； --修改符合条件的列
update L3 set phone="12345667788"; --修改列数据
update L3 set phone=139 where id=12 and name='mmy'; --修改符合条件的所有数据，id 不唯一情况也要考虑！
--数值类型做运算
update L3 set age=age+1; --数值类型 age 字段全部加 1
update L3 set age=age+1 and id=2;
--连接字符串方法
update L3 set phone=concat(name, "db") --字段列 phone 和字符串 db 连接
update L3 set phone=concat(name, phone) --字段列 phone=字段列 name 和原 phone 列字符串连接
update L3 set phone=concat(name, "db") where id=12 --符合条件的 phone 列，id 为 12 的数据和字符串 db 连接

8.4 查询数据

select * from 表名; --查询全部
select 列名 1，列名 2，列名 3 from 表名; --查询指定列
select 列名 1，列名 2 as N from 表名; --查询指定列,并设置显示指定的表头 N
select 列名 1，列名 2,111 as N from 表名; --查询指定列,并额外增加一值为 111 的列，且设置新增列显示指定的表头为 N
select 列名 1，列名 2，列名 3 from 表名 where 条件; --条件查询
select * from L3;
select id,name from l3; --查询指定列
select id,name as N from L3; --查询指定列,并设置显示指定的表头 N
select id,name,111 as N from L3; --查询指定列,并额外增加一值为 111 的列，且设置新增列显示指定的表头为 N
select id,name,111 as N from L3 where id=9;

-- 反转 reverse 函数
select * from student where reverse(name) = "cenhongchang"

实验：pymysql 连接数据库

```
# _*_ Anaconda3-Python3.8 _*_
import pymysql


def create_table():
    cursor.execute("create table user_info("
                   "id int primary key auto_increment not null,"
                   "username varchar(32),"
                   "password varchar(64)) "
                   "default charset=utf8;"
                   )
    conn.commit()
    cursor.close()
    conn.close()


def login():
    print("----用户登录----")
    usr = input("用户名：")
    pwd = input("密码：")
    # 此写法有 sql 注入的风险比如：用户名（or 1=1 --),密码（123）,在 sql 语句中，--代表注释，下面变为
    # select * from user_info where username='or 1=1 --' and password='123'
    # --注释掉后边内容，返回不为空！可以登录~~~，逻辑错误！
    # cursor.execute("select * from user_info where username='{}' and password='{}'".format(usr, pwd))
    # 正确写法：使用 mysql 的语句 s%,不加引号。
    cursor.execute("select * from user_info where username=%s and password=%s", [usr, pwd])
    # 或 cursor.execute("select * from user_info where username=%(usr)s and password=%(pwd)s", {"usr":usr, "pwd":pwd})
    tf = cursor.fetchone()  # 如果不存在，则为空，存在则获取当前登录信息
    if tf:
        print("登录成功！")
    else:
        print("用户不存在，请先注册！")


def register():
    print("----用户注册----")
    username = input("用户名：")
    password = input("密码：")
    # 注入风险写法
    # cursor.execute("insert into user_info(username,password) values('{}','{}')".format(username, password))
    # 正确写法
    cursor.execute("insert into user_info([username, password) values(%s, %s)", [username, password])
    conn.commit()
    print("注册成功！")


if __name__ == '__main__':
    # 游标创建
    conn = pymysql.connect(
        host='192.168.127.140',
        port=3306,
        user='root',
        password='Cc158854@',
        charset='utf8',
        db='userdb'
    )
    cursor = conn.cursor()

    # 创建表，执行一次即可
    # create_table()

    while True:
        print("-----------------")
```

```
        print(" 1.登录", "2.注册")
        print("-----------------")
        choice = input("请选择：")
        if choice == "1":
            login()
        elif choice == "2":
            register()
        else:
            continue
cursor.close()
conn.close()
```

9. 进阶三：查询语句深入

表：depart

id	title
1	开发
2	运营
3	销售

表：info

id	name	email	age	depart_id
1	武沛齐	wupeiqi@live.com	19	1
2	于超	pyyu@live.com	49	1
3	alex	alex@live.com	9	2
4	tony	tony@live.com	29	1
5	kelly	kelly@live.com	99	3
6	james	james@live.com	49	1
7	李杰	lijie@live.com	49	1

根据设计创建表结构：

```
create table info(
    id int auto_increment primary key not null,
    name varchar(16) not null,
    email varchar(32) not null,
    age int not null,
    depart_id int not null
)default charset=utf8;


select * from info where age > 30;
select * from info where id > 1;
select * from info where id = 1;
select * from info where id >= 1;
select * from info where id != 1;
select * from info where id between 2 and 4; -- id 大于等于 2、且小于等于 4
select * from info where name = '武沛齐' and age = 19 ;
select * from info where name = 'alex' or age = 49;
select * from info where (name = '李杰' or email="pyyu@live.com" ) and age=49;
select * from info where id in (1,4,6);
select * from info where id not in (1,4,6);
select * from info where id in (select id from depart);
# select * from info where id in (1,2,3);
# exists select * from depart where id=5，去查数据是否存在，如果存在，如果不存在。
select * from info where exists (select * from depart where id=5);
select * from info where not exists (select * from depart where id=5);
select * from (select * from info where id>2) as T where age > 10;   --从两次查询写法
select * from info where id > 10;
select * from info where info.id > 10;   --单表查询写法
```

10. 进阶四：模糊搜索-通配符(%代表任意字符，_代表 1 个字符)

```
select * from info where name like "%w%";
select * from info where name like "%n";
select * from info where name like "k%y";
select * from info where email like "_cctv_live.co_";
```

一般数据量大的时候，在企业内，不会采用此方法搜索数据，而用专门的组件实现。

11.进阶五：映射

表：depart

id	title
1	开发
2	运营
3	销售

表：info

id	name	email	age	depart_id
1	武沛齐	wupeiqi@live.com	19	1
2	于超	pyyu@live.com	49	1
3	alex	alex@live.com	9	2
4	tony	tony@live.com	29	1
5	kelly	kelly@live.com	99	3
6	james	james@live.com	49	1
7	李杰	lijie@live.com	49	1

```
select * from info;
select id, name from info;
select id, name as NM from info;    --显示 name 列名为 NM
select id, name as NM,123 from info;    --更改显示列名和值
注意:少些 select * ，自己需求。

select
    id,name,
    666 as num,
    ( select max(id) from depart ) as mid, -- max/min/sum    --从另外的表中获取最大 id 并展示为 mid 列
    ( select min(id) from depart) as nid, -- max/min/sum    --从另外的表中获取最小 id 并展示为 nid 列
    age
from info;
select
    id,
    name,
    (select title from depart where depart.id=info.depart_id) as x1    --匹配部门
from info;


select
    id,
    name,
    (select title from depart where depart.id=info.depart_id) as x1,
    ( select title from depart where depart.id=info.id) as x2
from info;

# 注意:此查询方法，效率很低，一般不采用！
```

mysql 里面的 case 语句：如果... 那么...

判断值相等则有：(case 列名 when 值 then "设置值" end 新增列)   --否则为空

判断值范围则有：(case when 列名 > 值 then "设置值" end 新增列)   --否则为空

判断值范围则有：(case when 列名 > 值 then "设置值 A" else "设置值 B" end 新增列)   --满足条件填设置值 A，否则填设置值 B。

表：depart

id	title
1	开发
2	运营
3	销售

表：info

id	name	email	age	depart_id
1	武沛齐	wupeiqi@live.com	19	1
2	于超	pyyu@live.com	49	1
3	alex	alex@live.com	9	2
4	tony	tony@live.com	29	1
5	kelly	kelly@live.com	99	3
6	james	james@live.com	49	1
7	李杰	lijie@live.com	49	1

```
select
    id,
    name,
    case depart_id when 1 then "第 1 部门" end v1,  --满足条件的，创建第三列 v1，值为第 1 部门，否则为空（不写为空）。
    case depart_id when 1 then "第 1 部门" else "其他" end v2,
    case depart_id when 1 then "第 1 部门" when 2 then "第 2 部门" else "其他" end v3,
    case when age<18 then "少年"end v4,
    case when age<18 then "少年" else "油腻男" end v5,
    case when age<18 then "少年" when age<30 then "青年" else "油腻男" end v6
from info;
```

12.进阶六：排序

```
select * from info order by age desc; --倒序
select * from info order by age asc;--顺序

select * from info order by id desc;
select * from info order by id asc;

select * from info order by age asc,id desc; --  优先按照 age 从小到大;如果 age 相同则按照 id 从大到小。
select * from info where id>10 order by age asc,id desc;
select * from info where id>6 or name like "&y" order by age asc,id desc;
```

13.进阶七：limit-获取前几条数据

表：depart

id	title
1	开发
2	运营
3	销售

表：info

id	name	email	age	depart_id
1	武沛齐	wupeiqi@live.com	19	1
2	于超	pyyu@live.com	49	1
3	alex	alex@live.com	9	2
4	tony	tony@live.com	29	1
5	kelly	kelly@live.com	99	3
6	james	james@live.com	49	1
7	李杰	lijie@live.com	49	1

```
select * from info limit 5;   --获取前 5 条数据
select * from info order by id desc limit 3;   --先排序，再获取前 3 条数据
select * from info where id > 10 order by id desc limit 3 ;   --先排序，再获取前 3 条数据

select * from info limit 3 offset 2; --从位第 2 行后边取（不含），向后获取 3 条数据。
```

数据库表中:1000 条数据，实现分页显示思路：

- 第一页:select * from info limit 10 offset 0;
- 第二页:select * from info limit 10 offset 10;
- 第三页:select * from info limit 10 offset 20;
- 第四页:select * from info limit 10 offset 30;

14.进阶八：分组-聚合（类似 Excel 透视操作）

表：depart

id	title
1	开发
2	运营
3	销售

表：info

id	name	email	age	depart_id
1	武沛齐	wupeiqi@live.com	19	1
2	于超	pyyu@live.com	49	1
3	alex	alex@live.com	9	2
4	tony	tony@live.com	29	1
5	kelly	kelly@live.com	99	3
6	james	james@live.com	49	1
7	李杰	lijie@live.com	49	1

聚合函数

select age, max (id), min (id), sum(id), avg(id), count(id) from info group by age;
--此处 count(id) <=> count(name) ...或其他，也可以写 count(1) 计数列，用到 age 为该组出现的行次数.

分组之后继续实现筛选显示:不能用 where,只能用 having

select age, count(1) from info group by age; --类似透视显示 age 列和它的计数列
select age, count(id) from info group by age having count(id) > 3; --age 列和他的计数列,筛出计数大于 3 的行数据

截至目前学习,mysql 执行顺序为(高到低排序)

where > having > order by > limit
select name, age from info group by age; --不建议使用
select * from info where id in (select max (id) from info group by age);
select age , count(id) from info group by age having count(id) > 2;
select age,count(id) from info where id > 4 group by age having count(id) > 2;--聚合条件放在 having 后面

理清顺序-语法格式

select age,count(id) from info where id > 2 group by age having count(id) > 1 order by age desc limit 1;
-要查询的表 info-条件 id>2
-根据 age 分组
-对分组后的数据再根据聚合条件过滤 count (id) >1-根据 age 从大到小排序
-获取第 1 条

15.进阶九：左右连表（类似 Excel-Vlookup 操作）

left/right outer join：主表匹配从表时，主表中未匹配到的依然显示出来为 null

主表 left outer join 从表 on 主表.x=从表.y
--可以简写
主表 left join 从表 on 主表.x=从表.y

select * from info left outer join depart on info.depart_id = depart.id;
select info.id,info.name,info.email,depart.title from info left outer join depart on info.depart_id=depart.id;

从表 right outer join 主表 on 主表.x=从表.y
select info.id,info.name,info.email,depart.title from info right outer join depart on info.depart_id=depart.id;

inner join：主表匹配从表时，只显示两者都能匹配到的数据(其实不区分主从表关系)

select * from info inner join depart on info.depart_id=depart.id;

16.进阶十：上下联合（union）

- **列数相同才能连接，默认当行数据完全相同，自动去重**

select id,title from depart
union
select name,email from info;

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

```
| id      | title        |
+-------+-----------+
| 1       | 网络部      |
| 2       | 人事部      |
| 3       | 企划部      |
| test1 | 1@qq.com   |
| test2 | 2@qq.com   |
| test3 | 3@qq.com   |
+-------+-----------+
```

- **若想完全显示，则可以取消自动去重 union all**

```
mysql> select id from depart
    -> union all
    -> select id from info;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
|  6 |
|  7 |
|  8 |
|  9 |
+----+
```

17.表关系设计思考

- **单表**

表：area

id	name	code
1	北京	010
2	上海	021
3	深圳	0755
4	天津	022

- **一对多**

表：depart

id	title
1	开发
2	运营
3	销售
4	运维

表：info

id	name	email	age	depart_id
1	武沛齐	wupeiqi@live.com	19	1
2	于超	pyyu@live.com	49	1
3	alex	alex@live.com	9	2
4	tony	tony@live.com	29	1
5	kelly	kelly@live.com	99	3
6	james	james@live.com	49	1
7	李杰	lijie@live.com	49	1

```
create table depart(
    id int not null auto_increment primary key,
    title varchar(16) not null
)default charset=utf8;

create table info(
```

```
    id int not null auto_increment primary key,
    name varchar (16) not null,
    email varchar (32) not null,age int,
    depart_id int not null,
    constraint fk_info_depart foreign key info(depart_id) references depart(id)    --外键约束
)default charset=utf8;
```

外键约束：constraint fk_表 1_表 2 foreign key 表 1(关联列) references 表 2(关联列)

假设一对多表已经创建好了，则用 alter 增加外键【中文理解：给表增加约束【然后名称】，外键以谁的哪一列作为参考】

```
alter table info add constraint fk_into_depart foreign key info(depart_id) references depart(id);
```

删除一对多外键约束

```
alter table info drop foreign key fk_info_depart;
```

- **多对多**

表：boy

id	name
1	李杰
2	alex
3	于超
4	日天

表：girl

id	name
1	苍老师
2	波多
3	凤姐
4	石榴

关系表：boy_girl

id	boy_id	girl_id
1	1	1
2	1	2
3	2	1
4	2	3

```
create table boy(
    id int not null auto_increment primary key,
    name varchar ( 16 ) not null
)default charset=utf8;

create table girl(
    id int not null auto_increment primary key,
    name varchar ( 16) not null
)default charset=utf8;

create table boy_girl(
    id int not null auto_increment primary key,
    boy_id int not null,
    girl_id int not null,
    constraint fk_boy_girl_boy foreign key boy_girl (boy_id) references boy( id)，
    constraint fk_boy_girl_girl foreign key boyl_girl(girl_id) references girl(id)
)default charset=utf8;
```

假设多对多表已经创建好了，则用 alter 增加外键

```
alter table info add constraint fk_boy_girl_boy foreign key boy_girl (boy_id) references boy( id);
alter table info add constraint fk_boy_girl_girl foreign key boyl_girl(girl_id) references girl(id);
```

删除多对多外键约束

```
alter table info drop foreign key fk_boy_girl_boy;
alter table info drop foreign key fk_boy_girl_girl;
```

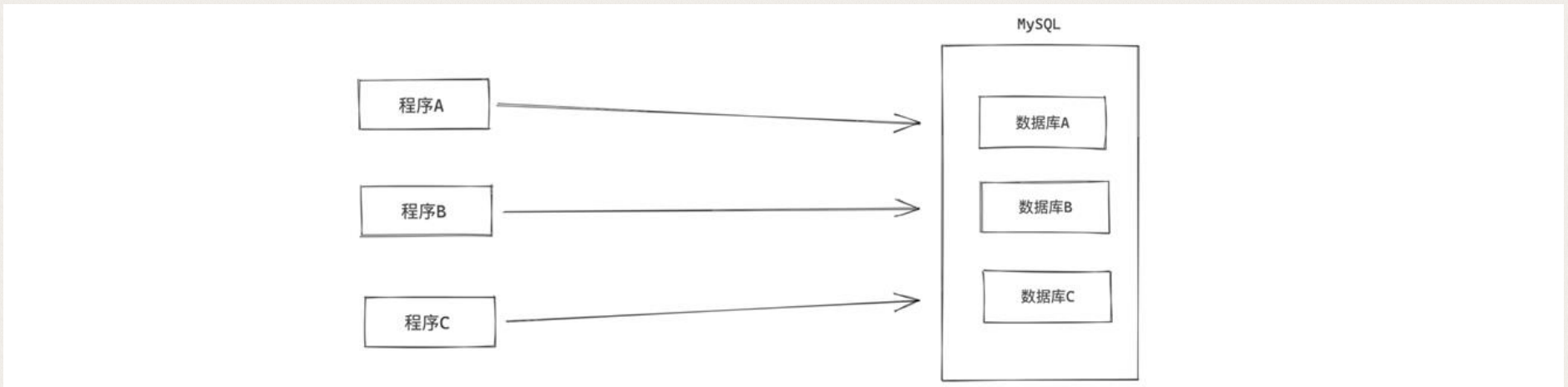
- **表设计的重要性**

  在以后项目开发时，设计表结构及其关系的是一个非常重要的技能。一般项目开始开发的步骤:

  - 需求调研
  - 设计数据库表结构(根据需求)
  - 项目开发（写代码)

  大量的工作应该放在前 2 个步骤，前期的设计完成之后，后续的功能代码开发就比较简单了。

18.用户权限管理



18.1 用户管理

- **查看已创建用户信息（存于 mysql 数据库的 user 表中）**

```
mysql> select user,host from mysql.user;    --
+---------------+-----------------------------------------+-----------+
| user          | authentication_string                   | host      |
+---------------+-----------------------------------------+-----------+
| root          | *5787FB44FFC363C874409B2403719469738D17D6 | %         |    --root 账户，%代表所有 ip 都可以连接
| mysql.session | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost |
| mysql.sys     | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost |
+---------------+-----------------------------------------+-----------+
```

- **创建和删除用户**

```
create user '用户名'@'连接者的 IP 地址' identified by '密码';

create user cenhongchang1@127.0.0.1 identified by 'root123';
drop user cenhongchang1@127.0.0.1;

create user cenhongchang2@'127.0.0.%' identified by 'root123';
drop user cenhongchang2@'127.0.0.%';

create user cenhongchang3@'%' identified by 'root123';
drop user cenhongchang3@'%';

create user 'cenhongchang4'@'%' identified by 'root123';
drop user 'cenhongchang4'@'%';
```

- **修改用户**

```
rename user '用户名'@'IP 地址' to '新用户名'@'IP 地址';
rename user cenhongchang1@127.0.0.1 to cenhongchang1@localhost;
rename user 'cenhongchang1'@'127.0.0.1' to 'cenhongchang1'@'localhost ';
```

- **修改密码**

```
set password for '用户名'@'IP 地址’= Password( '新密码')
```

set password for 'cenhongchangx'@'%' = Password ('123123');

- ## 18.2 授权管理

创建好用户之后，就可以做授权了

- ### 给出授权

grant all privileges on *.* to 'cenhongchang'@'localhost'; --给出全部数据库及全部表的所有权限
grant all privileges on mytest.* to 'cenhongchang'@'localhost'; --给出 mytest 数据库及其全部表的所有权限

grant all privileges on mytest.info to 'cenhongchang'@'localhost'; --给出 mytest 数据库及其 info 表的所有权限
grant select on mytest.info to 'cenhongchang'@'localhost'; --给出 mytest 数据库及其 info 表的 select 权限
grant select,insert on mytest.info to 'cenhongchang'@'localhost'; --给出 mytest 数据库及其 info 表的 select 权限

注意：刷新权限
flush privileges; --将数据读取到内存中，立即生效。

关于权限

```
all privileges   除grant外的所有权限
select           仅查权限
select,insert    查和插入权限
...
usage                    无访问权限
alter                    使用alter table
alter routine            使用alter procedure和drop procedure
create                   使用create table
create routine           使用create procedure
create temporary tables 使用create temporary tables
create user              使用create user、drop user、rename user和revoke  all privileges
create view              使用create view
delete                   使用delete
drop                     使用drop table
execute                  使用call和存储过程
file                     使用select into outfile 和 load data infile
grant option             使用grant 和 revoke
index                    使用index
insert                   使用insert
lock tables              使用lock table
process                  使用show full processlist
select                   使用select
show databases           使用show databases
show view                使用show view
update                   使用update
reload                   使用flush
shutdown                 使用mysqladmin shutdown(关闭MySQL)
super                    使用change master、kill、logs、purge、master和set global。还允许mysqladmin调试登陆
replication client       服务器位置的访问
replication slave        由复制从属使用
```

关于数据库和表

```
数据库名.*           数据库中的所有
数据库名.表名         指定数据库中的某张表
数据库名.存储过程名    指定数据库中的存储过程
*.*                  所有数据库
```

- ### 查看授权

show grants for '用户名'@’ip 地址‘
show grants for 'cenhongchang'@"localhost‘"

show grants for 'cenhongchang'@"%"

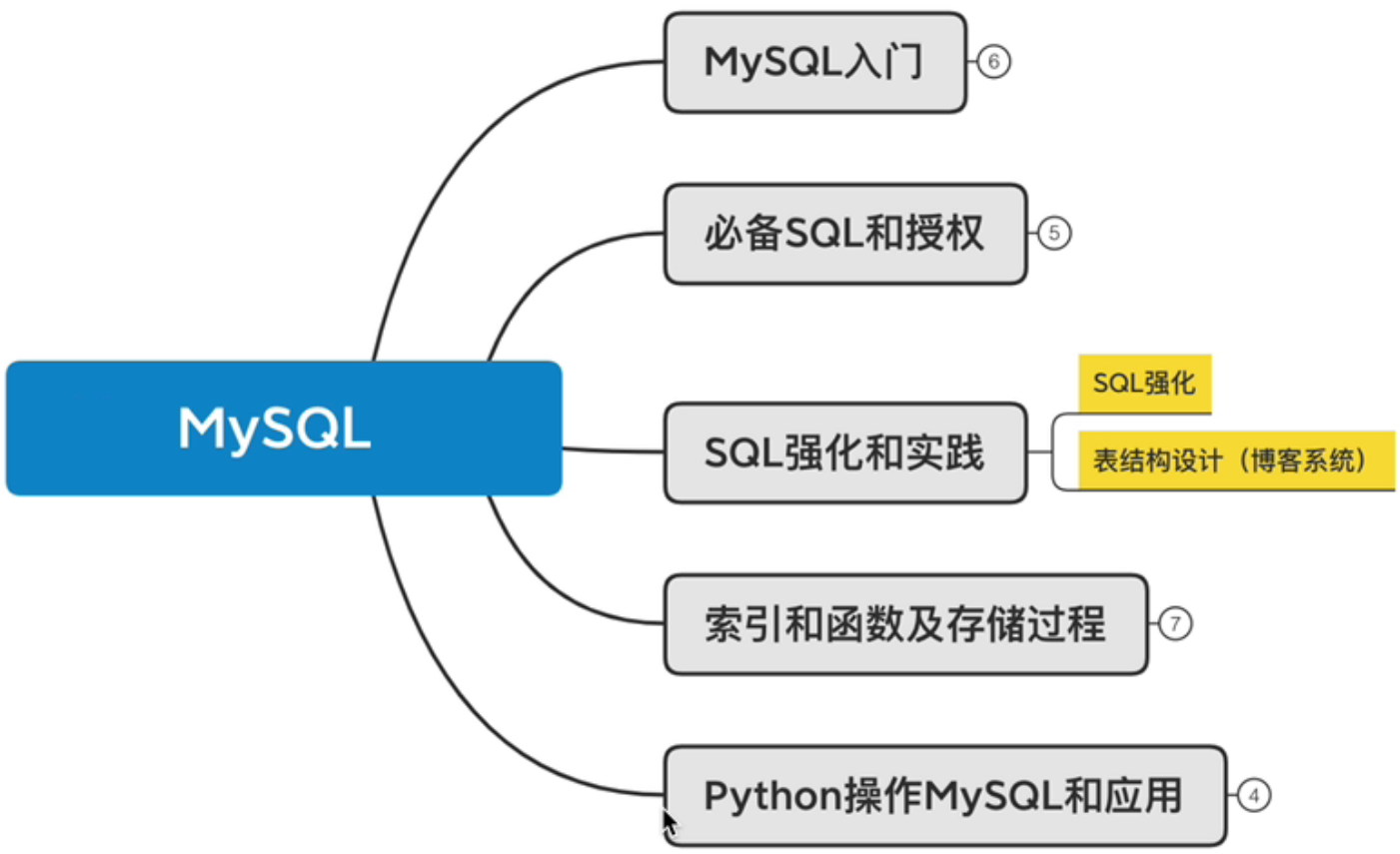
- **取消授权**

revoke 权限 on 数据库名.表名 from '用户名'@'ip 地址'
revoke all privileges on *.* from 'cenhongchang'@"localhost‘"
revoke all privileges on info.* from 'cenhongchang'@"%"

flush privileges;

一般情况下，很多正规公司，数据库都是有 DBA 统一来管理的，DBA 为每个数据库创建用户，并赋予相关权限。

19.强化 mysql 练习



- **表结构设计（博客系统）**

班级表：class

cid	caption
1	三年二班
2	一年三班
3	三年一班

学生表：student

sid	sname	gender	class_id
1	钢蛋	女	1
2	铁锤	女	1
3	山炮	男	2

老师表：teacher

tid	tname
1	波多
2	苍空
3	饭岛

课程表：course

cid	cname	teacher_id
1	生物	1
2	体育	1
3	物理	2

成绩表：score

sid	student_id	course_id	number
1	1	1	60
2	1	2	59
3	2	2	100

1-创建表代码

```
--班级表
create table class(
	cid int not null primary key auto_increment,
	caption varchar(32)
)default charset=utf8;

--学生表
create table student(
	tid int not null primary key auto_increment,
	sname varchar(32),
	gender enum("男","女") not null,
	class_id int not null,
	constraint fk_student_class foreign key student(class_id) references class(cid)
)default charset=utf8;

--教师表
create table teacher(
	tid int not null primary key auto_increment,
	tname varchar(32)
)default charset=utf8;

--课程表
create table course(
	cid int not null primary key auto_increment,
	cname varchar(32),
	teacher_id int not null,
	constraint fk_course_teacher foreign key course(teacher_id) references teacher(tid)
)default charset=utf8;

--成绩表
create table score(
	sid int not null primary key auto_increment,
	student_id int not null,
	corse_id int not null,
	number int not null,
	constraint fk_score_student foreign key score(student_id) references student(tid),
	constraint fk_score_course foreign key score(corse_id) references course(cid)
)default charset=utf8;
```

假设因为外键约束，删除表出现错误

```
SET foreign_key_checks = 0;  // 先设置外键约束检查关闭
drop table mytable;  // 删除数据，表或者视图
SET foreign_key_checks = 1; // 开启外键约束检查，以保持表结构完整性
```

2.插入数据

```
insert into class(caption)values("三年一班");
insert into class(caption)values("一年二班");
insert into class(caption)values("二年三班");

insert into student(sname,gender,class_id)values("钢弹",'男',1);
insert into student(sname,gender,class_id)values("铁锤",'女',2);
insert into student(sname,gender,class_id)values("三炮",'女',2);
insert into student(sname,gender,class_id)values("李湘",'男',3);
insert into student(sname,gender,class_id)values("李明",'女',2);
insert into student(sname,gender,class_id)values("张三",'男',3);

insert into teacher(tname)values("波多");
insert into teacher(tname)values("苍空");
insert into teacher(tname)values("麻希");
insert into teacher(tname)values("明步");
insert into teacher(tname)values("饭岛");
insert into teacher(tname)values("金肢");

insert into course(cname,teacher_id)values("语文",2);
insert into course(cname,teacher_id)values("数学",2);
insert into course(cname,teacher_id)values("英语",3);
insert into course(cname,teacher_id)values("物理",4);
insert into course(cname,teacher_id)values("生物",1);
insert into course(cname,teacher_id)values("化学",2);
insert into course(cname,teacher_id)values("历史",1);
insert into course(cname,teacher_id)values("政治",5);
insert into course(cname,teacher_id)values("地理",6);

insert into score(student_id,course_id,number)values(1,2,58);
insert into score(student_id,course_id,number)values(3,4,75);
insert into score(student_id,course_id,number)values(4,5,89);
insert into score(student_id,course_id,number)values(2,1,68);
```

3. 导入 sql 脚本

mysql -uroot - p 数据库名 < ~/test.sql --运行家目录下的 mysql 脚本

4. 导出 sql 脚本（备份）

导出结构+数据

musqldump -uroot -p 数据库名 > ~/out.sql --导出结构+数据到家目录 0

只导出结构（-d 参数）

musqldump -uroot -p -d 数据库名 > ~/out.sql --导出结构到家目录

5. 课后作业题目

1. 从下面表中查询各科成绩平均分和及格率，要求显示出课程 id，课程名称，平均分，及格率。

```
mysql> select * from course;
+-----+--------+------------+
| cid | cname  | teacher_id |
+-----+--------+------------+
|   1 | 语文   |          2 |
|   2 | 数学   |          2 |
|   3 | 英语   |          3 |
|   4 | 物理   |          4 |
|   5 | 生物   |          1 |
|   6 | 化学   |          2 |
|   7 | 历史   |          1 |
|   8 | 政治   |          5 |
|   9 | 地理   |          6 |
+-----+--------+------------+
```

```
mysql> select * from student;
+-----+--------+--------+----------+
| sid | sname  | gender | class_id |
+-----+--------+--------+----------+
|   1 | 钢弹   | 男     |        1 |
|   2 | 铁锤   | 女     |        2 |
|   3 | 三炮   | 女     |        2 |
|   4 | 李湘   | 男     |        3 |
|   5 | 李明   | 女     |        2 |
|   6 | 张三   | 男     |        3 |
+-----+--------+--------+----------+

mysql> select * from score;
+-----+------------+-----------+--------+
| sid | student_id | course_id | number |
+-----+------------+-----------+--------+
|   1 |          1 |         2 |     58 |
|   2 |          3 |         4 |     75 |
|   3 |          4 |         5 |     89 |
|   4 |          2 |         1 |     68 |
|   5 |          5 |         2 |     99 |
+-----+------------+-----------+--------+

mysql>  select
    -> score.course_id,course.cname,avg(score.number)
    ->  from
    -> score left outer join course on score.course_id=course.cid group by score.course_id;
+-----------+--------+------------------+
| course_id | cname  | avg(score.number) |
+-----------+--------+------------------+
|         1 | 语文   |          68.0000 |
|         2 | 数学   |          78.5000 |
|         4 | 物理   |          75.0000 |
|         5 | 生物   |          89.0000 |
+-----------+--------+------------------+

--查看是否及格
mysql> select *,case when number > 59 then 1 else 0 end "是否及格" from score;
+-----+------------+-----------+--------+--------------+
| sid | student_id | course_id | number | 是否及格     |
+-----+------------+-----------+--------+--------------+
|   1 |          1 |         2 |     58 |            0 |
|   2 |          3 |         4 |     75 |            1 |
|   3 |          4 |         5 |     89 |            1 |
|   4 |          2 |         1 |     68 |            1 |
|   5 |          5 |         2 |     99 |            1 |
|   6 |          5 |         1 |     39 |            0 |
|   7 |          6 |         1 |     59 |            0 |
|   8 |          2 |         1 |     59 |            0 |
|   9 |          2 |         1 |     48 |            0 |
|  10 |          2 |         3 |     88 |            1 |
+-----+------------+-----------+--------+--------------+

 select
      score.course_id,
      course.cname as "课程名称",
      avg(score.number) as "平均分",
      count(1) as "选课人数",
      sum(case when number > 59 then 1 else 0 end) "及格人数"
      sum(case when number > 59 then 1 else 0 end)/count(1) * 100 as "及格率"
 from
      score left outer join course on score.course_id=course.cid
group by score.course_id;
```

2. 查询没学过“波多”老师课的同学的学号、姓名（思路为先找到学过的学生，再取课程 id 不在里面的）

```
SELECT * FROM student WHERE sid NOT IN(
```

```
        SELECT
            student.sid
        FROM
            score
            LEFT JOIN student ON score.student_id = student.sid
            LEFT JOIN course ON score.course_id = course.cid
            LEFT JOIN teacher ON course.teacher_id = teacher.tid
        WHERE
            teacher.tname ="波多";
)
```

3. 查询选修苍空老师课程的最高分学生信息（不考虑并列最高，只显示 1 条）

```
select
    student.sname,
    score.number
from
    score
    left outer join student on score.student_id=student.sid
    left outer join course on score.course_id=course.cid
    left outer join teacher on course.teacher_id=teacher.tid
where
    teacher.tname="苍空"
order by
    score.number desc
    limit 1;
SELECT
    student.sname,
    score.number
FROM
    score
    LEFT OUTER JOIN student ON score.student_id = student.sid
    LEFT OUTER JOIN course ON score.course_id = course.cid
    LEFT OUTER JOIN teacher ON course.teacher_id = teacher.tid
WHERE
    teacher.tname = "苍空"
    AND score.number =(
        SELECT
            MAX(number)
        FROM
            score
            LEFT OUTER JOIN course ON score.course_id = course.cid
            LEFT OUTER JOIN teacher ON course.teacher_id = teacher.tid
            )
```

4. 查询选修苍空老师课程的最高分学生信息（考虑并列最高，全部显示）=>重点理解 min(), max()等都是返回值，可以自己利用！

5. 查询只选修了一门课程的全部学生的学号、姓名.

```
SELECT
    student.sid,
    student.sname
FROM
    score
    LEFT JOIN student ON score.student_id = student.sid
GROUP BY
    student_id
HAVING
    count( 1 )=1
```

6. 查询同时选修生物和物理的学生显示学号和姓名（筛选选了其中之一的学生，再通过计数=2；再次筛选）

```
SELECT
    student.sid,
    student.sname
FROM
    score
    LEFT JOIN course ON score.course_id = course.cid
```

```
    LEFT JOIN student ON score.student_id = student.sid
WHERE
    course.cname IN ( "生物","物理")
GROUP BY
    student_id
HAVING
    count( 1 ) = 2;
```

7. 查询“生物"课程比“物理"课程成绩高的所有学生的学号

```
SELECT
    score.student_id,
    MAX( CASE course.cname WHEN "生物" THEN score.number ELSE 0 END ) AS "sw",   --注意 as 写在括号外边
    MAX( CASE course.cname WHEN "物理" THEN score.number ELSE NULL END ) AS "wl"   --注意 as 写在括号外边
FROM
    score
    LEFT OUTER JOIN student ON score.student_id = student.sid
    LEFT OUTER JOIN course ON score.course_id = course.cid
WHERE
    course.cname IN ( "生物", "物理" )
GROUP BY
    score.student_id
HAVING
    sw > wl;
```

8. 展示各科成绩前三名（考虑并列第一第二第三的分数）

```
SELECT
    cid AS  序号,
    cname  课程,
    (select number from score where course_id = course.cid GROUP BY number order by number desc limit 1 offset 0) as "第一名",
    (select number from score where course_id = course.cid GROUP BY number order by number desc limit 1 offset 1) as "第二名",
    (select number from score where course_id = course.cid GROUP BY number order by number desc limit 1 offset 2) as "第三名"
FROM
    course;
SELECT
    *
FROM
    score
    LEFT JOIN (
    SELECT
        cid,
        cname,
        ( SELECT number FROM score WHERE course_id = course.cid GROUP BY number ORDER BY number DESC LIMIT 1 OFFSET 0 ) AS "第一名",
        ( SELECT number FROM score WHERE course_id = course.cid GROUP BY number ORDER BY number DESC LIMIT 1 OFFSET 1 ) AS "第二名",
        ( SELECT number FROM score WHERE course_id = course.cid GROUP BY number ORDER BY number DESC LIMIT 1 OFFSET 2 ) AS "第三名"
    FROM
    course
    ) AS C ON score.course_id = C.cid
WHERE
    score.number >= C.第三名;   --  只展示最少大于第三名的学生
```

```
+-----+------------+-----------+--------+------+--------+-----------+-----------+-----------+
| sid | student_id | course_id | number | cid  | cname  |  第一名     |  第二名     |  第三名     |
+-----+------------+-----------+--------+------+--------+-----------+-----------+-----------+
|   2 |          3 |         4 |     75 |    4 | 物理   |        99 |        75 |        46 |
|  18 |          6 |         4 |     99 |    4 | 物理   |        99 |        75 |        46 |
|  19 |          4 |         4 |     99 |    4 | 物理   |        99 |        75 |        46 |
|  20 |          5 |         4 |     99 |    4 | 物理   |        99 |        75 |        46 |
|  21 |          2 |         4 |     46 |    4 | 物理   |        99 |        75 |        46 |
```

```
|    3 |            4 |           5 |      89 |      5 |  生物     |            89 |           86 |           67 |
|   17 |            3 |           5 |      86 |      5 |  生物     |            89 |           86 |           67 |
|   22 |            2 |           5 |      67 |      5 |  生物     |            89 |           86 |           67 |
+-----+------------+-----------+--------+------+--------+-----------+-----------+-----------+
```

9. 当两=张表列名相同时，可以直接将其中一张表 1 数据插入另一张表 2 中

INSERT INTO 表名 2 SELECT * FROM 表名 1

20. Mysql 索引

数据库的索引是基于上述 B+Tree 的数据结构实现，但在创建数据库表时，如果指定不同的引擎，底层使用的 B+Tree 结构的原理有些不同。

索引的引擎

- **myisam 引擎（非聚簇索引-——数据和索引结构分开存储）**
- **innodb 引擎（聚簇索引———数据和索引一起存储）**

索引创建表：

```
-- myisam
create table isam (
	id int auto_increment not null primary key,
	name varchar ( 16 ) not null,
	age int not null
)engine=myisam charset=utf8

-- innodb
create table isam (
	id int auto_increment not null primary key,
	name varchar ( 16 ) not null,
	age int not null
)engine=innodb charset=utf8
```

企业开发一般使用 innodb 引擎，它支持内部事务、行级锁、外键等特点，mysql5.5 之后的版本默认都是 innodb 引擎，默认主键为索引。

查看创建表的引擎

```
mysql> show create table class;
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table | Create Table
                                                                                  |
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------+
| class | CREATE TABLE `class` (
  `cid` int(11) NOT NULL AUTO_INCREMENT,
  `caption` varchar(32) DEFAULT NULL,
  PRIMARY KEY (`cid`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 |
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

引擎不同，表结构创建出来的文件也不同，innodb 两个文件，myisam 三个文件

```
root@192 userdb # pwd
/usr/local/mysql/data/userdb
root@192 userdb # ls -l
total 1412928
-rw-r-----  1 _mysql  _mysql       8684 May 15 22:51 big.frm, 表结构。
-rw-r-----  1 _mysql  _mysql  717225984 May 15 22:51 big.ibd, 数据和索引结构。
-rw-r-----  1 _mysql  _mysql       8588 May 16 11:38 goods.frm
-rw-r-----  1 _mysql  _mysql      98304 May 16 11:39 goods.ibd
-rw-r-----  1 _mysql  _mysql       8586 May 26 10:57 t2.frm, 表结构
-rw-r-----  1 _mysql  _mysql          0 May 26 10:57 t2.MYD, 数据
-rw-r-----  1 _mysql  _mysql       1024 May 26 10:57 t2.MYI, 索引结构
```

索引的弊端

不要盲目的创建索引，只为查询操作频繁的列创建索引，创建索引会使查询操作变得更加快速，但是会降低增加、删除、更新操作的速度，因为执行这些操作的同时会对索引文件进行重新排序或更新。
但是，在互联网应用中，查询的语句远远大于 DML 的语句，甚至可以占到 80%~90%，所以也不要太在意，只是在大数据导入时，可以先删除索引，再批量插入数据，最后再添加索引。

常见索引

主键索引：
　　加速查找，不能为空，不能重复。 +联合主键索引；

唯一索引：
　　加速查找，不能重复。 +联合唯一索引；

普通索引：
　　加速查找。 +联合索引。

主键和联合主键索引

```
create table 表名(
    id int not null auto_increment primary key,--主键
    name varchar (32) not null
)charset=utf8;

create table 表名(
    id int not null auto_increment,
    name varchar ( 32 ) not null,
    primary key (id)
)charset=utf8;

create table 表名(
    id int not null auto_increment,
    name varchar ( 32 ) not null,
    primary key(列 1, 列 2)  --如果有多列,称为联合主键(不常用且 myisam 引擎支持)
)charset=utf8;
```

唯一和联合唯一索引

```
-- 唯一索引（单列不能重复）
create table 表名(
    id int not null auto_increment primary key,--主键
    name varchar ( 32 ) not null,
    email varchar ( 32 ) not null,
    unique ix_name(name),
    unique ix_email(email),
)charset=utf8;

-- 联合唯一索引（多列联合不能重复）
create table 表名(
```

```
    id int not null auto_increment,
    name varchar ( 32 ) not null,
    primary key(列 1，列 2)   --如果有多列,称为联合主键(不常用且 myisam 引擎支持)
)charset=utf8;
```

无法命中索引的几种情况

- 类型不一致

```
-- 假设 name 列为字符型，且已创建索引
select * from big where name=123;   --能查到，很慢
select * from big where name="123";   --很快查到

--特殊情况：当是主键的时候，此情况可以不考虑
select * from big where name="123";
```

- 最左前缀，如果是联合索引，要遵循最左前缀原则

```
-- 如果联合索引为:(name ,password)
name and password   --命中
name   --命中
password   --未命中
name or password   --未命中
```

21. Explain 执行计划

explain + sql 语句

```
mysql> explain select * from class;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | class | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    2 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.01 sec)
```

其中比较重要的是 type，他 SQL 性能比较重要的标志，性能从低到高依次:
all < index < range < index_merge < ref_or_null < ref < eq_ref < system < const

22. mysql 存储过程

学习链接：https://www.runoob.com/w3cnote/mysql-stored-procedure.html

23. mysql 视图

视图其实是一个虚拟表（非真实存在)，其本质是【根据 SQL 语句获取动态的数据集，并为其命名】，用户使用时只需使用【名称】即可获取结果集，并可以将其当作表来使用。

- 实例：

```
SELECT
    *
FROM
    (SELECT nid , name FROM tb1 WHERE nid > 2) AS A
WHERE
    A.name > 'alex ' ;
```

- 创建视图(相当于作为一个单独的表)：

```
create view v1 as select id,name from d1 where id > l;
```

- 使用视图

```
select * from v1;
```

-- select * from (select id , name from d1 where id > 1) as v1;

- 删除视图

drop view v1

- 修改视图

alter view v1 as SQL 语句

24.mysql 事务与应用

- **事务的四大特性：原子性、一致性、隔离性、持久性。**

原子性：指所有操作必须不可分割，要么都成功，要么都失败！

一致性：执行前后数据完整性保持一致

隔离性：一个事务执行过程中，不应该受到其他事务的干扰

持久性：事务一旦结束，数据就持久到数据库。

- **Begin　commit or rollback**

mysql> begin;　-- 开启事务 或 start transaction;
Query oK,0 rows affected (o.0o sec)

mysql> update users set amount=amount-2 where id=1;　-- 执行操作
ouery OK, 1 row affected (o.o0 sec)
Rows matched: 1　changed: 1 warnings : o

mysql> update users set amount=amount+2 where id=2;　-- 执行操作
Query oK,1 row affected (o.o0 sec)
Rows matched: 1　changed: 1　warnings: o

mysql> commit;　-- 提交事务　[回滚：rollback;]
Query oK,0 rows affected (o.oo sec)

Python 代码

```
import pymysql


conn = pymysql.connect (host= '127.0.0.1',
                        port=3306，user='root ',
                        passwd='root123 ',
                        charset="utf8",
                        db= 'userdb'
                    )
cursor = conn.cursor()
conn.begin()

try:
    cursor.execute("update users set amount=1 where id=1")
    int ('asdf')
    cursor.execute("update tran set amount=2 where id=2")
except Exception as e:
    #  回滚
    print ( "回滚")
    conn.rollback()
else:
    #提交
    print( "提交")
    conn.commit()
```

```
cursor.close()
conn.close()
```

25. mysql 排他锁和共享锁

在 innodb 引擎中，update、insert、delete 的行为内部都会先申请锁（排它锁)，申请到之后才执行相关操作，最后再释放锁。所以，当多人同时数据库执行: insert、update、delete 等操作时，内部加锁后会排队逐一执行。而 select 则默认不会申请锁。

- **排他锁 for update 锁住后不能读写**

mysql 内部默认在修改数据的时候，会加上索，直到释放才执行下一个操作；要想人为控制锁可以：**事务+排他锁实现**

id 为索引，行锁：

```
begin
    select * from tbname where id = 1 for update    -- 锁住 id 为 1 的行,直到 commit 才能读写
commit;
```

name 不是索引，表锁：

```
begin
    select * from tbname where name="cenhongchang" for update    -- 锁住 id 为 1 的行,直到 commit 才能读写
```

- **共享锁 lock in share mod 锁住后不能写，可以读**

```
begin
    select * from tbname where id = 1 LOCK IN SHARE MODE    -- 锁住 id 为 1 的行,直到 commit 才能写入，但可以读取

begin
    select * from tbname where name="cenhongchang" LOCK IN SHARE MODE    -- 锁住 id 为 1 的行,直到 commit 才能写入，但可以读取
commit;
```

- **python 应用排他锁应用场景**

```
import pymysql
import threading


def task():
    conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='root123', charset="utf8",
db='userdb')
    cursor = conn.cursor(pymysql.cursors.DictCursor)
    # cursor = conn.cursor()

    # 开启事务
    conn.begin()

    cursor.execute("select id,age from tran where id=2 for update")
    # fetchall      ( {"id":1,"age":10},{"id":2,"age":10}, )   ((1,10),(2,10))
    # {"id":1,"age":10}   (1,10)
    result = cursor.fetchone()
    current_age = result['age']

    if current_age > 0:
        cursor.execute("update tran set age=age-1 where id=2")
    else:
        print("已售罄")

    conn.commit()

    cursor.close()
    conn.close()


def run():
    for i in range(5):
        t = threading.Thread(target=task)
        t.start()
```

26. mysql 数据库连接池

必须安装

```
pip install pymysql
pip install dbutils
```

代码实例

```
# _*_ Anaconda3-Python3.8 _*_
import pymysql
from pymysql import cursors
from dbutils.pooled_db import PooledDB

MYSQL_DB_POOL = PooledDB(
    creator=pymysql,   # 使用链接数据库的模块
    maxconnections=50,   # 连接池允许的最大连接数, o 和 None 表示不限制连接数
    mincached=2,   # 初始化时，链接池中至少创建的空闲的链接，0 表示不创建
    maxcached=3,   # 链接池中最多闲置的链接，o 和 None 不限制
    blocking=True,   # 连接池中如果没有可用连接后，是否阻塞等待。True，等待;False，不等待然后报错
    setsession=[],   # 开始会话前执行的命令列表。如:[ "set datestyle to ..." ,"set time zone ..."]
    ping=0,   # ping MySQL 服务端,检查是否服务可用。
    # 如: 0 = None = never, 1 = default = whenever it is requested,
```

```
        # 2 = when a cursor is created, 4 = when a query is executed, 7 = always
        host='192.168.127.140',
        port=3306,
        user='root',
        password='Cc158854@',
        database='mywork',
        charset='utf8'
)


def task():
        #  去连接池中去一个连接
        conn = MYSQL_DB_POOL.connection()
        cursor = conn.cursor(cursors.DictCursor)
        cursor.execute('select sleep(2)')
        result = cursor.fetchall()
        print(result)

        #  关闭游标，将连接交给连接池
        cursor.close()
        conn.close()

if __name__ == '__main__':
        task()

#  结果列表里边套字典：    [{'sleep(2)': 0}]
```