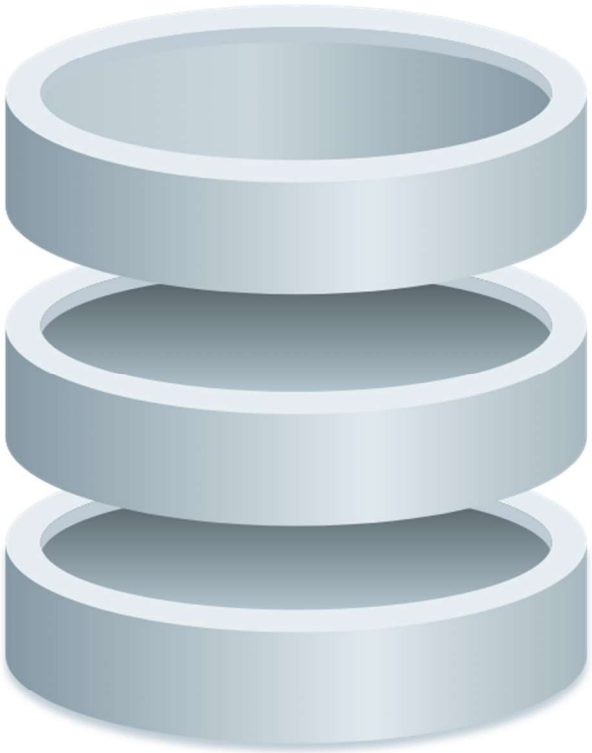
数据库上机实验报告



王英泰
1120210964
北京理工大学 计算机学院
2023 年 6 月 24 日

目录

第一章 MySQL 基础和实验说明

1.1MySQL 介绍

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL 是最好的 RDBMS（Relational Database Management System，关系数据库管理系统）应用软件之一。MySQL 是一种关系型数据库管理系统，关系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 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型和大型网站的开发都选择 MySQL 作为网站数据库。

1.2 实验环境

	操作系统	DBMS	图形界面软件
服务器环境	Ubuntu 18.04.1	MySQL Community 5.7.42	无
本地环境	Windows 11	MySQL Community 8.0.33	MySQL Workbench 8.0 CE

1.3MySQL 基础操作

- 登录 MySQL

```
$ mysql -h host -u user -p
```

  - Host：表示主机名，当 DBMS 位于本机上时，-h host 可以省略
  - User：表示用户名

- 登出 MySQL

```
mysql> quit
```

- 取消当前输入

```
mysql> …\c
```

- 执行 sql 文件

```
mysql > source <sql 文件路径和文件名>.sql;
```

1.4 实验要求内容与本实验报告对应关系

关系数据库系统环境和数据库建立	**第二章**
标准 SQL 语言和简单查询	第三章和第四章 4.1
复杂查询，触发器和存储过程	第四章 4.1、4.6 和第七章 7.5、7.3
备份、恢复数据库和数据库权限管理	第五章 5.3 和第五章 5.1

1.5 数据类型

- 整型：M 表示显示宽度，在 0 填充时才起作用

  BOOL | BOOLEAN：与 TINTINT(1)同义
  TINYINT[M] [UNSIGNED] [ZEROFILL] 1 字节
  SAMLLINT[M] [UNSIGNED] [ZEROFILL] 2 字节
  MEDIUMINT[M] [UNSIGNED] [ZEROFILL] 3 字节
  INT[M] | INTEGER[M] [UNSIGNED] [ZEROFILL] 4 字节
  BIGINT[M] [UNSIGNED] [ZEROFILL] 8 字节

- 定点型

  DECIMAL(M, D) | DEC(M, D) M+2 字节

- 浮点型

  FLOAT 4 字节

  DOUBLE 8 字节

- 日期和时间

YEAR 格式为 YYYY 1 字节

TIME 格式为 HH:MM:SS 3 字节

DATE 格式为 YYYY-MM-DD 3 字节

DATETIME 格式为 YYYY-MM-DD HH:MM:SS 8 字节

TIMESTAMP 格式为 YYYY-MM-DD HH:MM:SS 4 字节

- 字符串类型

  CHAR(M) 固定长度字符串 M 字节

  VARCHAR(M) 变长字符串，最长为 M 位 L+1 字节

  TINYTEXT L+1 字节，L<2^8

  TEXT L+1 字节，L<2^16

  MEDIUMTEXT L+1 字节，L<2^24

  LONGTEXT L+1 字节，L<2^32

- 二进制类型

  BIT(M) 位字段类型

  BINARY(M) 固定长度二进制串 M 字节

  VARBINARY(M) 可变长度二进制串 L+1 字节

  TINYBLOB(M) L+1 字节，L<2^8

  BLOB(M) L+1 字节，L<2^16

  MEDIUMBLOB(M) L+1 字节，L<2^24

  LONGBLOB(M) L+1 字节，L<2^32

1.6 表达式与运算符

- 算符运算符

  x1 + x2

  x1 – x2

  x1 * x2

  x1 / x2 或 x1 DIV x2

x1 % x2 或 MOD(x1, x2)

- 比较运算符

x1 = x2
x1 <> x2 或 x1 != x2
x1 > x2
x1 >= x2
x1 < x2
x1 <= x2
x1 IS [NOT] NULL
x1 BETWEEN m AND n
x1 [NOT] IN (value1, value2, ···)
x1 LIKE 表达式 模糊匹配
x1 REGEXP 正则表达式

- 逻辑运算符

&& 或 AND
! 或 NOT
|| 或 OR
XOR 异或

- 位运算符

1.7 系统函数

- 聚集函数

avg
count
max
min
sum

- 用于处理字符串的函数

合并字符串函数：concat(str1,str2,str3···)
比较字符串大小函数：strcmp(str1,str2)
获取字符串字节数函数：length(str)

获取字符串字符数函数：char_length(str)

字母大小写转换函数：大写 upper(x), ucase(x)；小写 lower(x), lcase(x)

- 用于处理数值的函数

  绝对值函数：abs(x)

  向上取整函数：ceil(x)

  向下取整函数：floor(x)

  取模函数：mod(x,y)

  随机数函数：rand()

  四舍五入函数：round(x,y)

  数值截取函数：truncate(x,y)

- 用于处理时间日期的函数

  获取当前日期：curdate()，current_date()

  获取当前时间：curtime()，current_time()

  获取当前日期时间：now()

  从日期中选择出月份数：month(date)，monthname(date)

  从日期中选择出周数：week(date)

  从日期中选择出周数：year(date)

  从时间中选择出小时数：hour(time)

  从时间中选择出分钟数：minute(time)

  从时间中选择出今天是周几：weekday(date)，dayname(date)

1.8 参考资料

- 课本
- MySQL 官方文档

第二章 MySQL 的安装和实验环境的搭建

2.1 MySQL 的安装

2.1.1 本地安装

访问 MySQL Community 8.0.33 的官方下载地址，打开软件，选择自定义（Custom）安装模式，安装 MySQL Server 和 MySQL Workbench。

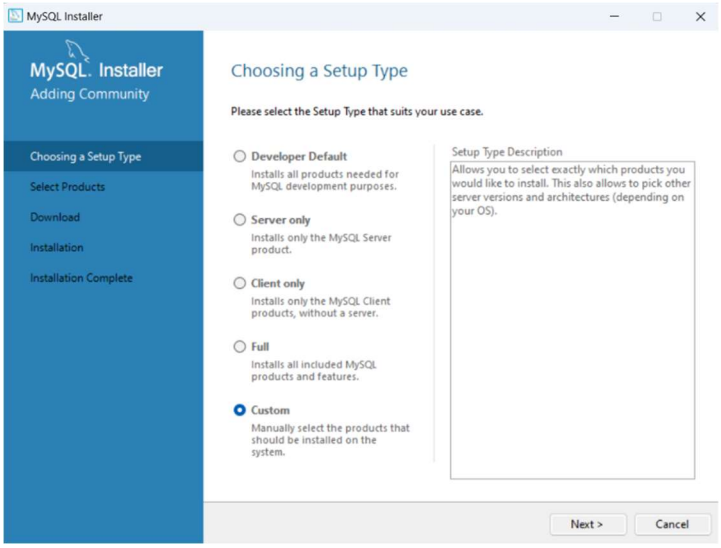


图 2-1 选择自定义（Custom）安装模式

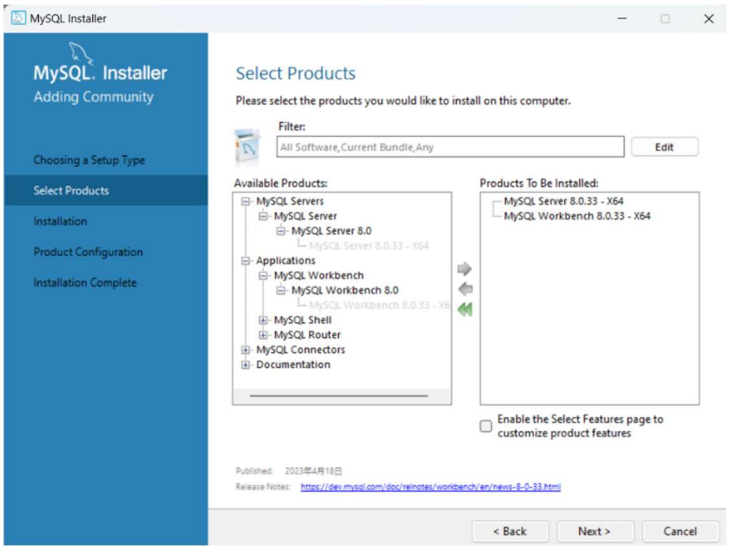


图 2-2 选择 MySQL Server 和 MySQL Workbench

配置 MySQL Server 的端口、认证方式、管理员账户密码、Windows 服务名

和数据文件权限。

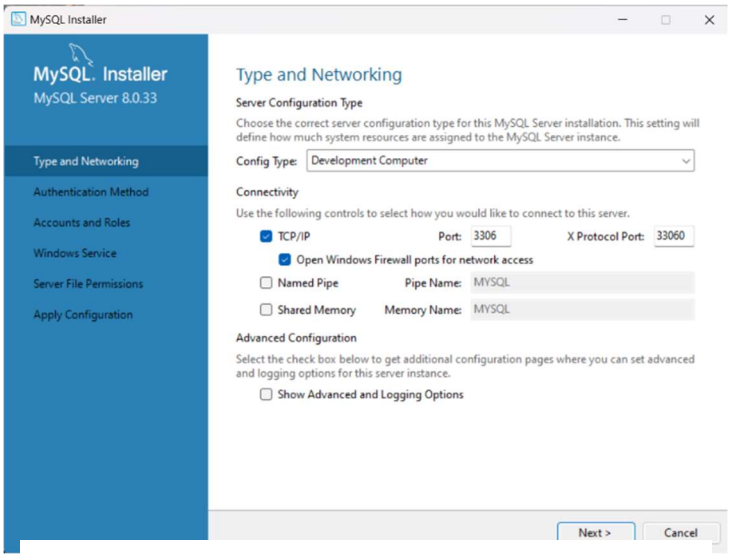


图 2-3 配置 MySQL 监听的端口号

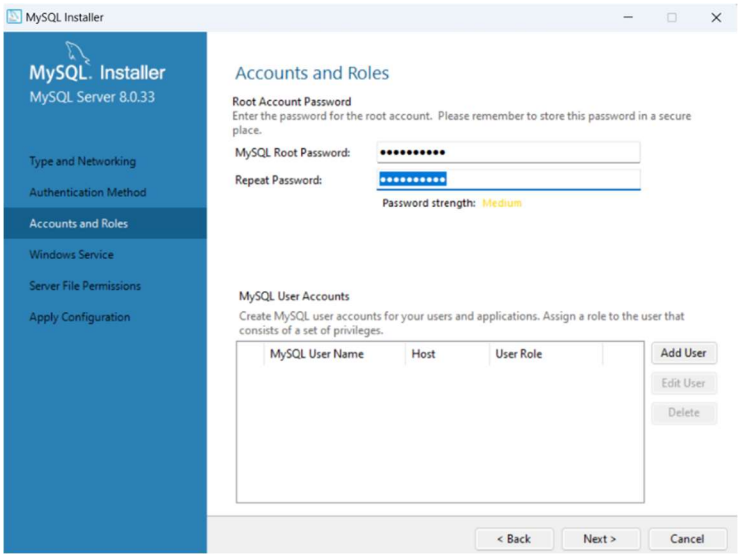


图 2-4 配置管理员账号 root 的密码

2.1.2 服务器安装

在 Ubuntu 终端中输入如下命令：

```
$ sudo apt-get update # 更新软件源
$ sudo apt-get install mysql-server # 安装 MySQL
```

输入以上命令后，MySQL 会自动安装。可以使用

```
$ mysql --version
```

来查看 MySQL 是否成功安装。

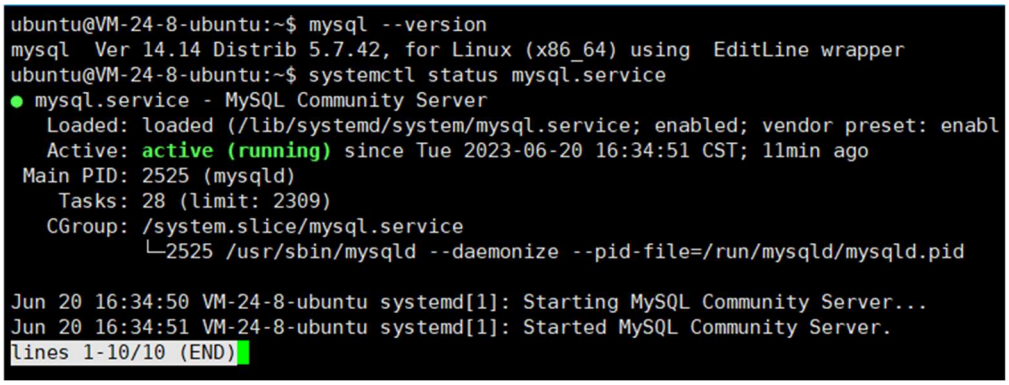
安装完成后，MySQL 服务器会自动启动，使用

```
$ systemctl status mysql.service
```

来检测 MySQL 是否正在运行；如果没有运行，使用命令

```
$ sudo systemctl start mysql
```

来启动 MySQL。



接下来对 MySQL 进行安全配置，输入如下命令开始。

```
$ sudo mysql_secure_installation
```

选择一种密码验证策略后，移除匿名用户，禁用 root 远程登录，移除 test 数据库，刷新权限表。

```
ubuntu@VM-24-8-ubuntu:~$ sudo mysql_secure_installation

Securing the MySQL server deployment.

Connecting to MySQL using a blank password.

VALIDATE PASSWORD PLUGIN can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD plugin?

Press y|Y for Yes, any other key for No: y

There are three levels of password validation policy:

LOW    Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and special characters
STRONG Length >= 8, numeric, mixed case, special characters and dictionary
         file

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 1
Please set the password for root here.

New password:

Re-enter new password:

Estimated strength of the password: 100
Do you wish to continue with the password provided?(Press y|Y for Yes, any other ke
y for No) : y
By default, a MySQL installation has an anonymous user,
allowing anyone to log into MySQL without having to have
a user account created for them. This is intended only for
testing, and to make the installation go a bit smoother.
You should remove them before moving into a production
environment.

Remove anonymous users? (Press y|Y for Yes, any other key for No) : y
Success.


Normally, root should only be allowed to connect from
'localhost'. This ensures that someone cannot guess at
the root password from the network.

Disallow root login remotely? (Press y|Y for Yes, any other key for No) : y
Success.

By default, MySQL comes with a database named 'test' that
anyone can access. This is also intended only for testing,
and should be removed before moving into a production
environment.


Remove test database and access to it? (Press y|Y for Yes, any other key for No) :
y
 - Dropping test database...
Success.

 - Removing privileges on test database...
Success.

Reloading the privilege tables will ensure that all changes
made so far will take effect immediately.

Reload privilege tables now? (Press y|Y for Yes, any other key for No) : y
Success.

All done!
```

2.2 实验环境的搭建

2.2.1 本地创建数据库

命令行方式：

```
$ mysql -u root -p # 登录数据库，之后输入密码
mysql> create database 数据库名;
```

```
PS C:\Users\wang_> mysql -u root -p
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

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

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

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

mysql> create database study;
Query OK, 1 row affected (0.01 sec)

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

图形化界面方式：

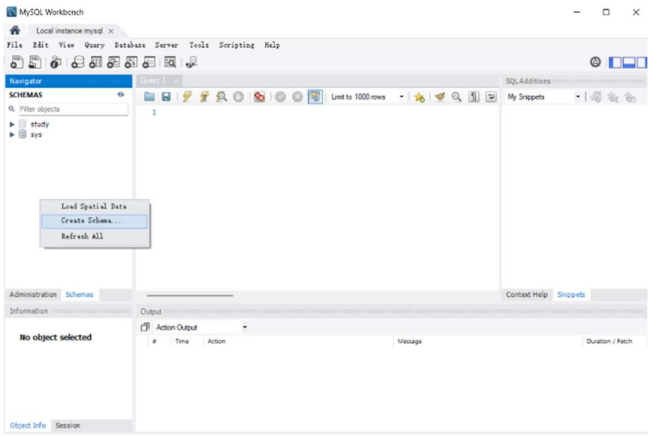


图 2-5 在 SCHEMA 处右键选择 Create Schema

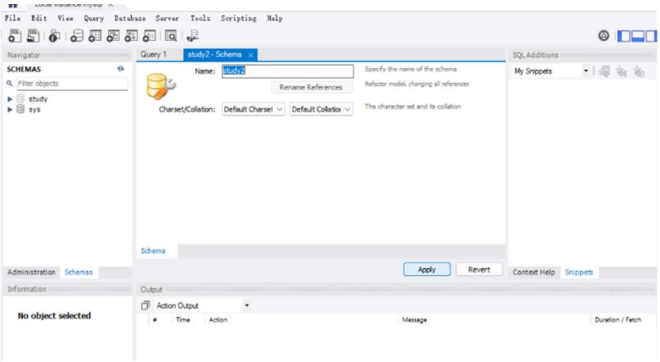


图 2-6 输入要创建的数据库名

2.2.2 服务器创建数据库

登录 MySQL：

```
$ sudo mysql -u root -p # 然后输入密码
```

```
ubuntu@VM-24-8-ubuntu:~$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.42-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

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

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

mysql>
```

创建用户：

```
mysql> create user '用户名'@'允许访问的地址' identified by '密码';
```

```
mysql> create user 'wyt'@'%' identified by 'wyt12345.';
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
mysql> create user 'wyt'@'%' identified by 'WYT12345a.';
Query OK, 0 rows affected (0.00 sec)

mysql> select user, plugin, host from mysql.user;
+------------------+-----------------------+-----------+
| user             | plugin                | host      |
+------------------+-----------------------+-----------+
| root             | auth_socket           | localhost |
| mysql.session    | mysql_native_password | localhost |
| mysql.sys        | mysql_native_password | localhost |
| debian-sys-maint | mysql_native_password | localhost |
| wyt              | mysql_native_password | %         |
+------------------+-----------------------+-----------+
5 rows in set (0.00 sec)
```

创建 study 数据库并授予用户在该数据库上的所有权限用于远程登录：

```
mysql> grant all privileges on study.* to ‘上面创建的用户名’@’%’ with grant option;
```

```
mysql> create database study;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| study              |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> grant all privileges on study.* to 'wyt'@'%' with grant option;
Query OK, 0 rows affected (0.00 sec)

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

修改 MySQL 配置文件来允许远程登录：

```
$ sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

将 bind-address = 127.0.0.1 注释，并重启 MySQL 服务。

```
user            = mysql
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
port            = 3306
basedir         = /usr
datadir         = /var/lib/mysql
tmpdir          = /tmp
lc-messages-dir = /usr/share/mysql
skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
# bind-address          = 127.0.0.1
#
# * Fine Tuning
#
key_buffer_size         = 16M
max_allowed_packet      = 16M
thread_stack            = 192K
thread_cache_size       = 8
```

最后重启 MySQL 服务：

```
$ sudo systemctl restart mysql,service
```

本地连接远程服务器测试：

```
PS C:\Users\wang_> mysql -h 82.156.175.138 -u wyt -p
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 80
Server version: 5.7.42-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

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

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

mysql>
```

图 2-7 本地命令行测试

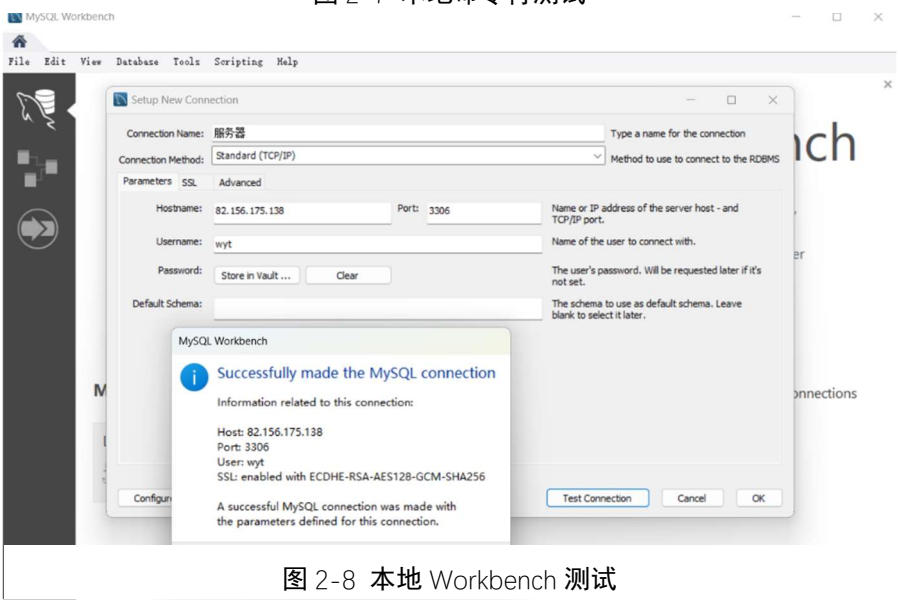


图 2-8 本地 Workbench 测试

第三章 数据定义（DDL）

3.1 与模式相关

一个 SQL 模式（schema）由模式名、权限标识符和模式中元素的描述符组成，权限标识符指明拥有该模式的用户或账号，模式元素包含一个数据库应用的表、视图和索引等，属于同一应用的表、视图和索引等可以定义在同一模式中，在定义模式时，可以先给出模式名和权限标识符，以后再定义其中的元素。

在 MySQL 中，模式（schema）与数据库（database）基本等价，语法中的 schema 可以和 database 互换。

3.1.1 语法

- 定义

```
CREATE SCHEMA | DATABASE <模式名>;
```

- 删除

```
DROP SCHEMA | DATABASE <模式名>;
```

- 查看

```
SHOW SCHEMAS | DATABASES;
```

- 使用

```
USE <模式名>;
```

3.1.2 使用示例

```
/*
* 模式
*/
-- mysql 中的模式与数据库基本上是等价的，命令中的 database 可以与 schema 互换
create database study; -- 定义数据库
show databases; -- 查看数据库
use study; -- 使用数据库
drop database study; -- 删除数据库
show databases; -- 查看数据库
```

结果如下：

```
mysql> source experiment.sql;
Query OK, 1 row affected (0.00 sec)

+--------------------+
| Database           |
+--------------------+
| information_schema |
| lexue              |
| mysql              |
| performance_schema |
| sinan              |
| study              |
| sys                |
+--------------------+
7 rows in set (0.00 sec)

Database changed
Query OK, 0 rows affected (0.00 sec)

+--------------------+
| Database           |
+--------------------+
| information_schema |
| lexue              |
| mysql              |
| performance_schema |
| sinan              |
| sys                |
+--------------------+
6 rows in set (0.00 sec)
```

3.2 与基本表相关

3.2.1 语法

- 定义

```
CREATE TABLE <表名>
   (<列名> <数据类型> [<列级完整性约束条件>]
   [, <列名> <数据类型> [<列级完整性约束条件>]]…
   [, <表级完整性约束条件>]);
```

在定义基本表时，表所属的数据库模式可以显示指定，格式为：<模式名>.<表名>。

- 删除

```
DROP TABLE <表名> [RESTRICT | CASCADE];
```

在 MySQL 中，RESTRICT 和 CASCADE 关键字没有任何作用。它们被允许使从其他数据库系统移植变得更容易。

- 修改

```
ALERT TABLE <表名>
   [ADD <列名> <数据类型> [<完整性约束>]]
   [DROP <列名>]
   [MODIFY <列名> <数据类型>];
```

  - add 用于增加新列
  - drop 用于删除指定列
  - modify 用于修改指定列的定义

在 MySQL 中，ALTER 可以用于修改列的默认值，不能修改列的数据类型。

- 查看：该语句提供基本表中有关列的信息

```
DESCRIBE <表名>;
```

3.2.2 列级或表级约束

```
CONSTRAINT <约束名> <约束>
```

- 列级约束

  NULL | NOT NULL：默认为 NULL
  DEFAULT value：指定默认值
  AUTO_INCREMENT：自增
  UNIQUE：唯一约束
  PRIMARY KEY：主键约束
  REFERENCES：外键，不过没有效果
  CHECK：检查约束，check (age >= 0 and age <= 100)
  CONSTRAINT 语句

- 表级约束

CONSTRAINT 语句，可以省略 CONSTRAINT <约束名>，直接写约束

PRIMARY KEY(<列名>)

FOREIGN KEY(<列名>) REFERENCES <表名>(<列名>) [ON DELETE | UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT]

UNIQUE(<列名>)

CHECK(约束)

CONSTRAINT 语句

从 MySQL 8.0.16 开始，才支持 check 语句。

3.2.3 使用示例

```
/*
* 基本表
*/

create database study; -- 定义数据库
-- 创建
create table study.student -- 在 study 数据库中创建
student 基本表
    (sid char(10) not null unique, -- 列级约束
    sname char(20),
    sgender char(6) check(sgender in ('male',
'female')), -- 列级约束，使用了 check 约束
    sage tinyint unsigned check(sage >= 0 and sage <=
100),
    constraint c1 primary key(sid)); -- 表级约束，如果主
键涉及多个属性，必须使用表级约束

create table study.course
    (cid char(9) not null unique primary key,
    cname char(20) not null);

create table study.sc
    (sid char(10) references study.student(sid), -- 列级
约束，使用了外键约束，不过不会生效
    cid char(9),
    score tinyint unsigned null,
```

```
    check(score >= 0 and score <= 100), -- 表级约束，可以省略 constraint <约束名>
     foreign key(sid) references study.student(sid), -- 表级约束，使用了列级约束
    foreign key(cid) references study.course(cid));

use study;
show tables;
describe student; -- 描述表的结构，describe 可以简写为 desc
describe course;
describe sc;
-- 修改
alter table study.student
    add smajor varchar(20),
    drop sgender,
    modify sname char(15);
describe student;
-- 删除
drop table study.sc;
show tables;
```

结果如下：

```
Query OK, 1 row affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

Database changed
+-----------------+
| Tables_in_study |
+-----------------+
| course          |
| sc              |
| student         |
+-----------------+
3 rows in set (0.00 sec)

+---------+------------------+------+-----+---------+-------+
| Field   | Type             | Null | Key | Default | Extra |
+---------+------------------+------+-----+---------+-------+
| sid     | char(10)         | NO   | PRI | NULL    |       |
| sname   | char(20)         | YES  |     | NULL    |       |
| sgender | char(2)          | YES  |     | NULL    |       |
| sage    | tinyint unsigned | YES  |     | NULL    |       |
+---------+------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| sid    | char(10)         | NO   | PRI | NULL    |       |
| sname  | char(15)         | YES  |     | NULL    |       |
| sage   | tinyint unsigned | YES  |     | NULL    |       |
| smajor | varchar(20)      | YES  |     | NULL    |       |
+--------+------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

Query OK, 0 rows affected (0.02 sec)

+-----------------+
| Tables_in_study |
+-----------------+
| course          |
| student         |
+-----------------+
2 rows in set (0.00 sec)
```

3.3 与视图相关

视图是建立在一个或多个基本表上的虚表。

3.3.1 语法

- 创建

```
CREATE VIEW <视图名> [(<列名>[, <列名>, …])]
   AS <SELECT 子查询语句>
   [WITH CHECK OPTION];
```

  - [(<列名>[, <列名>, …])]用来定义视图的列名
  - WITH CHECK OPTION：通过视图进行增删改操作时，需要满足子查询中的条件表达式

在数据库中，基表和视图共享相同的名称空间，因此基表和视图不能具有相同的名称

- 删除

```
DROP VIEW <视图名>;
```

- 修改

```
ALTER VIEW <视图名> [(<列名>[, <列名>, …])]
   AS <SELECT 子查询语句>
   [WITH CHECK OPTION];
```

- 查询

```
DESCRIBE <视图名>;
```

3.3.2 使用示例

```
/*
* 视图
*/

create view male_age (sid, sname, sage) -- 创建视图
    as select sid, sname, sage from student
```

```
    with check option;

desc male_age; -- 查看视图定义

alter view male_age (sname, sage) -- 修改视图，注意
select 中的列要与视图中的列对应
    as select sname, sage from student
    with check option;

desc male_age;

drop view male_age; -- 删除视图
```

结果如下：

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

+-------+-----------------+------+-----+---------+-------+
| Field | Type            | Null | Key | Default | Extra |
+-------+-----------------+------+-----+---------+-------+
| sid   | char(10)        | NO   |     | NULL    |       |
| sname | char(20)        | YES  |     | NULL    |       |
| sage  | tinyint unsigned | YES  |     | NULL    |       |
+-------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

+-------+-----------------+------+-----+---------+-------+
| Field | Type            | Null | Key | Default | Extra |
+-------+-----------------+------+-----+---------+-------+
| sname | char(20)        | YES  |     | NULL    |       |
| sage  | tinyint unsigned | YES  |     | NULL    |       |
+-------+-----------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

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

3.4 与索引相关

3.4.1 语法

- 建立

```
CREATE [UNIQUE] INDEX <索引名>
   ON <表名>(<列名>[, <次序>][<列名>[<次序>], …]);
```

- 次序有：升序 ASC（默认值），降序 DESC
- UNIQUE 表明此索引的每一个索引值只对应唯一的数据记录，在插入或更新元组时，系统将检查该列值的唯一性；CLUSTER 表示要建立的索引是聚簇索引，索引次序与表中元组的物理次序一致的索引；一个表只能包含一个聚簇索引，但该索引可以包含多个列（组合索引），适用于经常查询的基本表。

只有 InnoDB 引擎支持聚簇索引，MyISAM 不支持聚簇索引

- 删除

```
DROP INDEX <索引名> ON <表名>;
```

- 查看

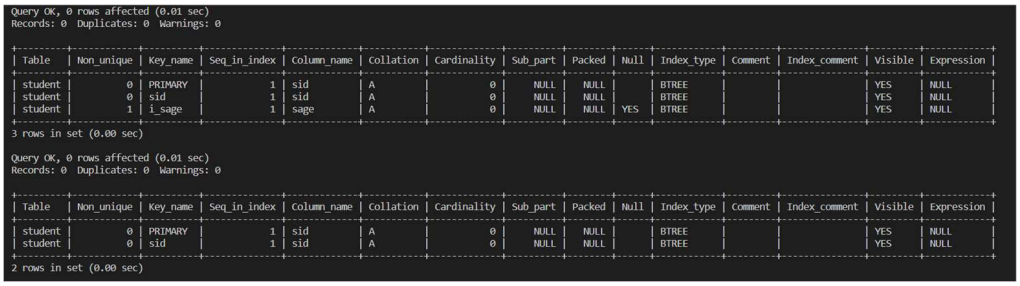
```
SHOW INDEX FROM <表名>;
```

3.4.2 使用示例

```
/*
* 索引
*/

create index i_sage -- 创建索引
    on study.student(sage asc);
show index from study.student; -- 查看索引
drop index i_sage on study.student; -- 删除索引
show index from study.student;
```

结果如下：



第四章 数据操纵（DML）

4.1 查询

4.1.1 语法

```
SELECT [ALL | DISTINCT] <目标列表达式>[, <目标列表达式>…]
   FROM <表名或视图名>[, <表名或视图名>…]
   [WHERE <条件表达式>]
   [GROUP BY <列名 1> [HAVING <条件表达式>]]
   [ORDER BY <列名 2> [ASC | DESC]];
```

- ALL 表示查询所有元组（默认），DISTINCT 表示去掉重复元组
- GROUP BY：按照列名的值进行分组查询，值相同的为一组，对应一个结果元组，通常会在每组中作用集函数；HAVING 筛选出只有满足指定条件的组
- ORDER BY：对查询结果按指定列值的升序（默认）或降序排序
- 目标列表达式：可以是字段名，也可以是字符串常量、由字段和常量组成的算术表达式、函数、列别名、* 等，其中列别名的格式为“<列名> [as] <列别名>”
- WHERE 子句作用域基表或视图，从中选择满足条件的元组，不能使用聚集函数；HAVING 作用于组，从中选择满足条件的组，其中可以使用聚集函数
- 使用 GROUP BY 子句后，SELECT 子句的列名列表中只能出现分组属性和集函数
- WHERE 子句的查询条件

比较表达式	<列名1> 比较算符 <列名2（或常量）> 比较算符：=、>、>=、<、<=、<>（或!=）
逻辑表达式	<条件表达式1> 逻辑算符 <条件表达式2> 逻辑算符：AND、OR、NOT
BETWEEN	<列名1>（NOT）BETWEEN <常量1或列名2> AND <常量2或列名3>
IN	<列名>（NOT）IN（常量表列 或 SELECT语句）
LIKE	<列名>（NOT）LIKE ‘匹配字符串’ 匹配符：“_”表示匹配一个字符，“%”表示匹配任意字符串
NULL	<列名> IS（NOT） NULL
EXISTS	（NOT）EXISTS（SELECT语句）

4.1.2 连接查询

- 内连接

```
select <目标列表达式> from <表 1> inner join <表 2>[on 表 1.col_name1 = 表 2.col_name2];
```

- 左外连接

```
select <目标列表达式> from <表 1> left join <表 2>[on 表 1.col_name1 = 表 2.col_name2];
```

- 右外连接

```
select <目标列表达式> from <表 1> right join <表 2>[on 表 1.col_name1 = 表 2.col_name2];
```

4.1.3 嵌套查询

SELECT 语句出现在查询条件（WHERE 或 HAVING）中，称为嵌套查询或子查询，分为一般子查询和相关子查询

- 一般子查询：子查询中的条件不依赖于父查询，执行时先执行子查询，后执行父查询，使用 **in 谓词**
- 相关子查询：子查询的查询条件依赖于父查询；取外层查询中表的一个元组，根据它与内层查询相关的属性值处理内层查询，若 WHERE 子句返回值为真，则取此元组放入结果表，使用 **exists 谓词**

4.1.4 集合查询

```
<SELECT 子句 1> <集合连接词> [ALL | DISTINCT] <SELECT 子句 2> [<集合连接词> [ALL | DISTINCT] <SELECT 子句 3> …]
```

MySQL 支持三种集合查询，分别为：

- UNION：并集
- INTERSECT：交集
- EXCEPT：差集

4.1.5 使用示例

使用 Node.js 构造数据，向 student 表中插入 1000 条数据，向 course 表中插入 100 条数据，向 sc 表中插入 10000 条数据。程序文件名为 insert.js，代码如

下：

```
const mysql = require('mysql2');
const db = mysql.createPool({
    host: 'localhost',
    port: '3306',
    user: 'root',
    password: 'WYT12345a.',
    database: 'study'
});

db.query('select 1;', (err, res) => { // 测试连接数据库是否成功
    if(err)
        return console.log(err.message);
    else
        return console.log(res, '数据库连接成功！');
});

// 向 student 表中插入数据
const sqlStr = `insert into student
(sid, sname, sgender, sage)
values ?;`

let data = [];

for(let i = 0; i < 1000; i++) { // 构造要插入的数据
    data.push([addZerosBeforeNum(i, 10), '学生' + getRandomNum(0, 1000000), getRandomNum(0,1) ? 'male' : 'female', getRandomNum(0, 100)]);
}


db.query(sqlStr, [data], (err, res) => {
    if(err)
        return console.log(err.message);
    else
        return console.log('插入数据成功！插入', res.affectedRows, '行！');
});

// 向 course 表中插入数据
const sqlStr1 = `insert into course
(cid, cname)
```

```
values ?;`

let data1 = [];

for(let i = 0; i < 100; i++) { // 构造要插入的数据
    data1.push([addZerosBeforeNum(i, 9), '课程' +
getRandomNum(0, 100)]);
}


db.query(sqlStr1, [data1], (err, res) => {
    if(err)
        return console.log(err.message);
    else
        return console.log('插入数据成功! 插入',
res.affectedRows, '行! ');
});

// 向 sc 表中插入数据
const sqlStr2 = `insert into sc
(sid, cid, score)
values ?;`

let data2 = [];

for(let i = 0; i < 10000; i++) { // 构造要插入的数据
    data2.push([getRandomNum(0, 999, 10),
getRandomNum(0, 99, 9), Math.random()*100]);
}


db.query(sqlStr2, [data2], (err, res) => {
    if(err)
        return console.log(err.message);
    else
        return console.log('插入数据成功! 插入',
res.affectedRows, '行! ');
});



function addZerosBeforeNum(i, digitNum) { // 数字前补 0
    let nowDigit = 0;
```

```
    let num = i;
    if(num === 0) {
        nowDigit = 1;
    }
    else {
        while(num > 0) {
            num  = Math.floor(num / 10);
            nowDigit++;
        }
    }
    let temp = "";
    for(let i = 0; i < digitNum - nowDigit; i++) {
        temp += "0";
    }
    return temp + i;
}

function getRandomNum(min, max, digitNum) { // 返回[min,
max]范围内的随机数
    let res = Math.floor(Math.random()*(max-
min+1))+min;
    if(digitNum) {
        res = addZerosBeforeNum(res, digitNum);
    }
    return res;
}
```

单表查询:

```
/*
* 单表查询
*/

-- 查询年龄小于 10 岁的男学生信息
select sid, sname, sage from student where sage < 5 and
sgender = 'male';
-- 查询名字的倒数第二位为 6 且年龄小于 10 岁的学生信息
select * from student where sname like "%6_" and sage <
10;
-- 查询年龄为 18 或 19 的男学生信息
select * from student where sage in (18, 19) and
sgender = 'male';
-- 查询年龄在 20~30 岁之间的学生人数
select sage, count(*) as "num_of_students" from student
```

```
group by sage having sage <= 30 and sage >= 20
order by sage asc;
```

结果如下：

```
+------------+------------+-----+
| sid        | sname      | sage |
+------------+------------+-----+
| 0000000017 | 学生194120 |    2 |
| 0000000082 | 学生463973 |    1 |
| 0000000096 | 学生202370 |    1 |
| 0000000115 | 学生863072 |    2 |
| 0000000277 | 学生556355 |    2 |
| 0000000350 | 学生40954  |    2 |
| 0000000410 | 学生60523  |    1 |
| 0000000473 | 学生255975 |    0 |
| 0000000502 | 学生117262 |    0 |
| 0000000560 | 学生170230 |    1 |
| 0000000583 | 学生184227 |    1 |
| 0000000585 | 学生355315 |    0 |
| 0000000587 | 学生830158 |    2 |
| 0000000654 | 学生372118 |    2 |
| 0000000749 | 学生317880 |    0 |
| 0000000761 | 学生783569 |    1 |
| 0000000787 | 学生656830 |    4 |
| 0000000810 | 学生476764 |    2 |
| 0000000886 | 学生326050 |    4 |
| 0000000941 | 学生276865 |    4 |
| 0000000961 | 学生648118 |    0 |
| 0000000979 | 学生334234 |    2 |
+------------+------------+-----+
22 rows in set (0.00 sec)

+------------+------------+---------+-----+
| sid        | sname      | sgender | sage |
+------------+------------+---------+-----+
| 0000000133 | 学生412768 | female  |    9 |
| 0000000308 | 学生504160 | male    |    5 |
| 0000000339 | 学生181767 | male    |    7 |
| 0000000401 | 学生643165 | female  |    4 |
| 0000000502 | 学生117262 | male    |    0 |
| 0000000650 | 学生834966 | female  |    4 |
| 0000000761 | 学生783569 | male    |    1 |
| 0000000790 | 学生407860 | female  |    7 |
| 0000000810 | 学生476764 | male    |    2 |
| 0000000941 | 学生276865 | male    |    4 |
+------------+------------+---------+-----+
10 rows in set (0.00 sec)

+------------+------------+---------+-----+
| sid        | sname      | sgender | sage |
+------------+------------+---------+-----+
| 0000000057 | 学生560581 | male    |   19 |
| 0000000098 | 学生388286 | male    |   19 |
| 0000000113 | 学生482295 | male    |   18 |
| 0000000114 | 学生70769  | male    |   18 |
| 0000000135 | 学生697507 | male    |   18 |
| 0000000360 | 学生601072 | male    |   18 |
| 0000000426 | 学生936554 | male    |   18 |
| 0000000456 | 学生276822 | male    |   19 |
| 0000000521 | 学生645589 | male    |   18 |
| 0000000826 | 学生792359 | male    |   19 |
| 0000000939 | 学生551281 | male    |   18 |
+------------+------------+---------+-----+
11 rows in set (0.00 sec)

+------+-----------------+
| sage | num_of_students |
+------+-----------------+
|   20 |               4 |
|   21 |               6 |
|   22 |              15 |
|   23 |               9 |
|   24 |              12 |
|   25 |               9 |
|   26 |              12 |
|   27 |              12 |
|   28 |              15 |
|   29 |               7 |
|   30 |              11 |
+------+-----------------+
11 rows in set (0.00 sec)
```

连接查询:

```
/*
* 连接查询
*/
-- 查询所有课程中成绩为 95~100、年龄在 18~30 岁之间的学生姓名，课程名和成绩
select sname, cname, score from student, course, sc
where student.sid = sc.sid and course.cid = sc.cid and
score between 98 and 100
and sage between 18 and 30
order by score desc;
```

结果如下:

```
+-----------+--------+-------+
| sname     | cname  | score |
+-----------+--------+-------+
| 学生388286 | 课程58 |   100 |
| 学生471450 | 课程78 |   100 |
| 学生629696 | 课程77 |   100 |
| 学生792359 | 课程6  |   100 |
| 学生608602 | 课程24 |   100 |
| 学生87542  | 课程89 |   100 |
| 学生936734 | 课程58 |    99 |
| 学生11350  | 课程63 |    99 |
| 学生482295 | 课程90 |    99 |
| 学生639144 | 课程88 |    99 |
| 学生208227 | 课程63 |    99 |
| 学生783416 | 课程85 |    99 |
| 学生973064 | 课程85 |    99 |
| 学生608602 | 课程16 |    99 |
| 学生936734 | 课程89 |    98 |
| 学生70769  | 课程63 |    98 |
| 学生63631  | 课程77 |    98 |
| 学生459550 | 课程65 |    98 |
| 学生17927  | 课程11 |    98 |
| 学生17927  | 课程28 |    98 |
| 学生711357 | 课程63 |    98 |
| 学生778999 | 课程17 |    98 |
| 学生186564 | 课程6  |    98 |
| 学生208227 | 课程80 |    98 |
| 学生236113 | 课程59 |    98 |
| 学生406783 | 课程40 |    98 |
+-----------+--------+-------+
26 rows in set (0.00 sec)
```

嵌套查询:

```
/*
* 嵌套查询
*/

-- 查询选修了 20 门课以上的学习信息
select sid, sname, sage, sgender from student
where sid in (select sid from sc group by sid having
count(*) > 20);
-- 查询成绩为 100 分的学生信息
select sname from student
where exists (select * from sc where student.sid =
sc.sid and score = 100);
```

结果如下：

```
mysql> source 4.1.sql;
+------------+-----------+------+---------+
| sid        | sname     | sage | sgender |
+------------+-----------+------+---------+
| 0000000788 | 学生66176 |   42 | female  |
| 0000000973 | 学生608602 |   22 | male    |
+------------+-----------+------+---------+
2 rows in set (0.01 sec)

+------------+
| sname      |
+------------+
| 学生884512 |
| 学生119471 |
| 学生450153 |
| 学生163001 |
| 学生857178 |
| 学生509817 |
| 学生388286 |
| 学生894475 |
| 学生124122 |
| 学生951586 |
| 学生24111  |
| 学生809324 |
| 学生500502 |
| 学生389199 |
| 学生455403 |
| 学生871010 |
| 学生444840 |
| 学生274066 |
| 学生439546 |
| 学生463140 |
| 学生792359 |
| 学生30111  |
| 学生398031 |
| 学生43943  |
| 学生283689 |
| 学生930610 |
| 学生847700 |
| 学生555983 |
| 学生608602 |
| 学生361004 |
| 学生87542  |
| 学生464745 |
+------------+
60 rows in set (0.01 sec)
```

集合查询：

```
/*
* 集合查询
*/
-- 查询学号为 0000000001 和 0000000002 的选课信息
select * from sc where sid = "0000000001"
union
select * from sc where sid = "0000000002";
```

结果如下：

```
+------------+-----------+-------+
| sid        | cid       | score |
+------------+-----------+-------+
| 0000000001 | 000000057 |    72 |
| 0000000001 | 000000074 |    50 |
| 0000000001 | 000000057 |    81 |
| 0000000001 | 000000050 |     6 |
| 0000000001 | 000000018 |    45 |
| 0000000001 | 000000023 |    32 |
| 0000000001 | 000000022 |    84 |
| 0000000001 | 000000071 |    37 |
| 0000000001 | 000000038 |     9 |
| 0000000001 | 000000071 |    30 |
| 0000000001 | 000000043 |    22 |
| 0000000001 | 000000003 |    95 |
| 0000000002 | 000000003 |    90 |
| 0000000002 | 000000060 |    96 |
| 0000000002 | 000000047 |   100 |
| 0000000002 | 000000092 |     3 |
| 0000000002 | 000000040 |    11 |
| 0000000002 | 000000015 |    40 |
| 0000000002 | 000000088 |     7 |
| 0000000002 | 000000039 |    52 |
| 0000000002 | 000000060 |    86 |
| 0000000002 | 000000082 |    58 |
| 0000000002 | 000000050 |    44 |
+------------+-----------+-------+
23 rows in set (0.00 sec)
```

4.2 插入

4.2.1 插入单个元组

- 语法

```
INSERT INTO <表名>
[(<列名 1>[, <列名 2>…])] VALUES
(<常量 11>[, <常量 12>…])[, (<常量 21>[, <常量 22>…])…];
```

  - 没有指定列名：则新插入的元组必须按表中定义的列顺序给出每个列的值

4.2.2 插入子查询结果

- 语法

```
INSERT INTO <表名>
[(<列名 1>[, <列名 2>…])]
<SELECT 语句>;
```

  - select 子句目标列必须与 into 子句匹配：值的类型和个数

4.3 删除

```
DELETE FROM <表名> [[AS] <表的别名>] [WHERE <条件>];
```

4.4 修改

```
UPDATE <表名> SET <列名 1> = <表达式 1>[, <列名 2> = <表达式 2>…]
[WHERE <条件表达式>];
```

  - set 的表达式可以是常量或带列名的算术表达式

4.5 视图上的操作

将视图当做基本表进行操作即可。

4.6 使用示例

```
/*
* 插入、删除、更新
*/
-- 插入两条课程记录
insert into course (cid, cname)
```

```
values ("123456789", "C++"), ("987654321", "Java");
-- 删除 30 分以下的选课记录
delete from sc
where sc.score < 30;
-- 修改 60 分以下的成绩为 60
update sc set score = 60
where score < 60;

/*
* 视图
*/

create view best (sid, sname, score) -- 创建视图，该视图存放获得过 99 分和 100 分成绩的学生信息，该视图无法更新
    as select distinct student.sid, sname, score from student, sc
    where student.sid = sc.sid and score > 98;

select * from best;
```

结果如下：

```
Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
Rows matched: 0  Changed: 0  Warnings: 0

Query OK, 0 rows affected (0.01 sec)

+------------+------------+-------+
| sid        | sname      | score |
+------------+------------+-------+
| 0000000663 | 学生589788 |    99 |
| 0000000130 | 学生780201 |    99 |
| 0000000984 | 学生87542  |   100 |
| 0000000418 | 学生53283  |   100 |
| 0000000364 | 学生980890 |    99 |
| 0000000116 | 学生951586 |   100 |
| 0000000746 | 学生453231 |    99 |
| 0000000068 | 学生11350  |    99 |
| 0000000374 | 学生289785 |    99 |
| 0000000858 | 学生60102  |    99 |
| 0000000773 | 学生918398 |    99 |
| 0000000555 | 学生471039 |   100 |
| 0000000008 | 学生119471 |   100 |
| 0000000756 | 学生339006 |    99 |
| 0000000912 | 学生347164 |    99 |
| 0000000145 | 学生164910 |    99 |
| 0000000483 | 学生850479 |   100 |
```

第五章 数据控制（DCL）

5.1 数据安全性

5.1.1 访问控制

MySQL 将**帐户存储在 mysql 系统数据库的 user 表**中。帐户是根据用户名和用户可以连接到服务器的一个或多个客户端主机来定义的，即账户的格式为**'user_name'@'host_name'**。其中@'host_name' 部分是可选的，仅由用户名组成的帐户名相当于 'user_name'@'%'，%表示任意客户端主机。角色名也是由**'role_name'@'host_name'**组成的。

相关语法如下：

- 查看当前账户

```
SELECT CURRENT_USER();
```

- 创建账户

```
CREATE USER <账户名> [IDENTIFIED BY '<密码>'][, <账户名>
[IDENTIFIED BY '<密码>']…];
```

- 删除账户

```
DROP USER <账户名>[, <账户名>…];
```

5.1.2 自主存取控制（DAC）

相关语法如下：

- 权限授予

```
GRANT <权限 1>[, <权限 2>, …]
   ON TABLE <权限作用对象>
   TO <账户名或角色名>[, <账户名或角色名>…]
   [WITH GRANT OPTION];
```

  - 当有 WITH GRANT OPTION 短语时，被授权的用户还可以把获得的权限再授予其它用户
  - 权限作用对象：

    .：全局权限

<数据库名>.*：数据库权限

*：默认数据库

<数据库名>.<基本表名>：表权限

<基本表名>：默认数据库上的表权限

- 常见的权限

ALL [PRIVILEGES]：所有权限

ALTER：修改表

CREATE：创建数据库和表

CREATE ROLE：创建角色

CREATE USER：启用 CREATE USER、DROP USER、RENAME USER 和 REVOKE ALL PRIVILEGES 的使用

CREATE VIEW：创建或修改视图

DELETE：启用 DELETE 的使用

DROP：删除数据库、表和视图

DROP ROLE：删除角色

GRANT OPTION：允许向其他账户授予权限或从其他账户删除权限

INDEX：允许创建或删除索引

INSERT：启用 INSERT 的使用

SELETE：启用 SELECT 的使用

SHOW DATABASES：启用 SHOW DATABASES 的使用

SUPER：启用其他管理操作，如 CHANGE REPLICATION SOURCE TO 、CHANGE MASTER TO 、 KILL 、 PURGE BINARY LOGS 、 SET GLOBAL 和 mysqladmin 调试命令

TRIGGER：启用触发器操作

UPDATE：启用 UPDATE 的使用

SYSTEM USER：指定账户为系统账户

SELECT(col 1, col2 …)：列权限

- 权限回收

```
REVOKE <权限 1>[, <权限 2>, …]
   ON TABLE <权限作用对象>
   FROM <账户名或角色名>[, <账户名或角色名>…];
```

- 查询权限

```
SHOW GRANTS[ FOR <账户名或角色名> [USING <角色名>[, <角色
名>…]]];
```

  - 当 FOR <账户名或角色名>省略时，表示查询当前登录用户的权限。
  - [USING <角色名>[, <角色名>…]]表示账户与角色相关联的角色相关联的权限。

5.1.3 数据库角色

数据库角色是被命名的一组与数据库操作相关的权限，是权限的集合。用户和角色存在**多对多**的关系。

相关语法如下：

- 为用户授予角色

```
GRANT <角色名 1>[, <角色 2>…] TO <账户名 1>[, <账户名 2>…];
```

- 在角色中删除用户

```
REVOKE <角色名 1>[, <角色名 2>…] FROM <账户名 1>[,
<账户名 2>…];
```

- 创建角色

```
CREATE ROLE <角色名>[, <角色名>];
```

- 删除角色

```
DROP ROLE <角色名>[, <角色名>];
```

- 查看当前账户的角色

```
SELECT CURRENT_ROLE();
```

5.1.4 视图机制

可以为不同用户定义不同的视图，通过视图机制可以对无权存取的用户把数据隐藏起来，从而自动对数据提供一定程度的安全保护。

视图机制主要的功能在于提供数据独立性，其安全保护功能并不精细。实际中通常是**视图机制与授权机制配合使用**，即先用视图机制屏蔽掉一部分保密数据，然后在视图上面再进一步定义存取权限。

有关视图的使用方法请查看 3.3 节和 4.5 节。

5.2 数据完整性

5.3 数据库备份与恢复

在 MySQL 中，备份方法有物理备份和逻辑备份两种。物理备份是对数据库系统的物理文件备份，逻辑备份是备份 SQL 语句。逻辑备份又分为完全备份和增量备份。

5.3.1 物理备份和恢复

先关闭 mysql 服务，然后备份 mysql 根目录文件。恢复时将备份文件复制到 mysql 文件夹下即可。

```
$ systemctl stop mysqld # Ubuntu 系统关闭 mysql 服务
```

5.3.2 二进制日志

每次重新启动时，MySQL 服务器都会使用序列中的下一个数字创建一个新的二进制日志文件。当服务器运行时，您还可以告诉它关闭当前的二进制日志文件，并通过发出 FLUSH LOGS SQL 语句或使用 mysqladmin flush logs 命令手动开始新的日志文件。mysqldump 还有一个刷新日志的选项。data 目录中的 .index 文件包含该目录中所有 MySQL 二进制日志的列表。

5.3.3 完全备份

```
$ mysqldump -u <用户名> -p <备份对象> > <备份路径或备份文件名>.sql # 然后输入密码
```

- 备份对象有
  - --databases <数据库名 1>[<数据库名 2>…]：完全备份一个或多个完整的数据库，包括其中所有的表
  - --all-databases：完全备份 MySQL 服务器中所有的数据库
  - <数据库名> <基本表名 1>[<基本表名 2>…]：完全备份指定库中的部分表

- 其他选项
  - -d：只保存数据库的表结构不保存表中的数据
  - --delete-master-logs：删除不再需要的二进制日志文件
  - --single-transaction：执行不锁定表的在线备份

mysqldump 生成的.sql 文件包含一组 SQL INSERT 语句，这些语句可用于以后重新加载转储的表。此备份操作在转储开始时获取所有表的全局读取锁定（使用 FLUSH TABLES WITH READ LOCK）。一旦获取了该锁，就会读取二进制日志坐标并释放该锁。如果在发出 FLUSH 语句时正在运行长更新语句，则备份操作可能会暂停，直到这些语句完成。之后，转储将成为无锁的，并且不会干扰表上的读写操作。

5.3.4 完全备份的恢复

有两种方法可以进行恢复：

- 登录 mysql 后执行备份文件

```
mysql > source <备份路径或备份文件名>.sql;
```

- 不登录 mysql 恢复

```
$ mysql -u <用户名> -p <备份对象> < <备份路径或备份文件名>.sql;
```

5.3.5 增量备份

要进行增量备份，需要保存增量更改。在 MySQL 中，这些更改在二进制日志中表示，因此 MySQL 服务器应**启用二进制日志**。启用二进制日志记录后，服务器在更新数据的同时将每个数据更改写入文件。

增量备份要先做完全备份，完全备份时要确保刷新日志，完全备份的语法如下：

```
$ mysqldump --single-transaction --flush-logs --master-data --all-databases > <备份路径或备份文件名>.sql
```

- --flush-logs 执行备份前切断当前 binlog 和数据库的联系，在备份期间写入的数据都会写入新的 binlog 文件中，方便后面通过 binlog 来恢复数据
- --master-data 将新创建的二进制日志信息写入 sql 文件中，之后创建的二进制日志将包含自备份以来所做的所有数据的更改。

增量备份通过完全刷新+刷新日志来完成。

5.3.6 增量备份的恢复

要进行增量备份的恢复，需要先进行完全恢复：

- 登录 mysql 后执行备份文件

```
mysql > source <备份路径或备份文件名>.sql;
```

- 不登录 mysql 恢复

```
$ mysql -u <用户名> -p <备份对象> < <备份路径或备份文件名>.sql;
```

然后还原增量备份：

```
$ mysqlbinlog <日志文件名>;
```

第六章 嵌入式 SQL

SQL 语句是面向集合的，一次查询可能会产生多个记录，而应用程序是面向记录的，一次只能处理一行，游标可以协调这两种不同的处理方式。游标是系统为用户开设的一个数据缓冲区，存放 SQL 语句的执行结果，可以通过移动指针每次获取结果集中的一行记录给主变量，由应用程序进行进一步处理。

6.1 在 Node.js 中嵌入 SQL 语句

在 Node.js 中使用 mysql2 模块来进行 MySQL 与 Node.js 的通信。

在项目根目录下打开终端，输入 npm install mysql2 来安装 mysql2 模块。

使用方法如下：

- 建立 Node.js 与 MySQL 的连接

```
const mysql = require('mysql2');
const db = mysql.createPool({
    host: '主机地址，本地即 localhost',
    port: '端口号',
    user: '账户名',
    password: '账户密码',
    database: '使用的数据库名'
});

db.query('select 1;', (err, res) => { // 测试数据库连接是否成功
    if(err)
        return console.log(err.message);
    else
        return console.log(res, '数据库连接成功！');
});
```

- 插入数据

```
// 插入
const data = [['1120210000', 'wyt', '男', 20],
['1120210964', '王英泰', '男', 18]];
const sqlStr = `insert into student
(sid, sname, sgender, sage)
values ?;`;
db.query(sqlStr, [data], (err, res) => {
    if(err)
```

```
        return console.log(err.message);
    console.log('插入数据成功！插入', res.affectedRows, '
行！');
});
```

- 查询数据

```
// 查询
db.query('select * from student;', (err, res) => {
    if(err)
        return console.log(err.message);
    console.log(res);
});
```

- 更新数据

```
// 修改
const newData= {sid: '1120210000',sname: 'wangyingtai',
sgender: '男', sage: 19};
const sqlStr1 = `update student set
sname = ?, sage = ?
where sid = ?; `;
db.query(sqlStr1, [newData.sname, newData.sage,
newData.sid], (err, res) => {
    if(err)
        return console.log(err.message);
    console.log('更新数据成功！更新', res.affectedRows, '
行！');
});
```

- 删除数据

```
// 删除
const sqlStr2 = `delete from student
where sid = ?`;
db.query(sqlStr2, ["1120210964"], (err, res) => {
    if(err)
        return console.log(err.message);
    console.log('删除数据成功！删除', res.affectedRows, '
行！');
});
```

在使用中要注意**异步 JavaScript**，程序代码在附录文件 dmbededSQL.js 中。

6.2 在 Python 中使用 SQL 语句

- 安装 pymysql 模块：在终端中输入 pip install pymysql
- 连接 MySQL

```
conn=pymysql.connect(host="127.0.0.1", port=3306,
user='root', passwd='', charset='utf8', db='数据库名称')
cursor=conn.cursor(cursor=pumysql.cursors.DictCursor)
```

- 发送指令

```
cursor.execute('sql 语句')
conn.commit()
```

  - **不要用字符串格式化去做 SQL 的拼接，否则有 SQL 注入的安全隐患**
  - cursor.execute("select * from admin where id>%s", [2,])，无论数字还是字符串，均使用%s 作为占位符
  - cursor.execute("select * from admin where id>%(n1)s", {"n1": 2})

- 获取数据

```
cursor.execute("select * from admin where id>%s", [2,])
data_list=cursor.fatchall()
```

  - cursor.fatchall()会取回所有数据，以列表形式，元素是字典
  - cursor.fatchone()会取回满足条件的第一个数据，是字典

- 关闭

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

第七章 数据库编程

MySQL 支持存储例程（Stored Rountines），包括存储过程（PROCEDURE）和函数（FUNCTION）。存储例程是存储在服务器中的一组 SQL 语句，这样客户端就可以直接使用存储例程而不用自己再重新定义。**触发器是一种特殊类型的存储过程**。

存储过程和函数的区别如下：

- 调用方式不同：存储过程只能使用 CALL <存储过程名称>来调用，函数可以像其他函数一样在语句内部调用。
- 传递返回值方式不同：存储过程需要在程序中使用变量来传递返回值，而函数可以直接返回标量值。

7.1 变量

7.1.1 系统变量和会话变量

系统变量（GLOBAL）针对于所有会话，会话变量（SESSION）针对于单个会话，在另外一个会话窗口就不会生效。

相关语法如下：在 mysql 服务重新启动后，设置的全局参数会失效，要想不失效，可以在 /etc/my.cnf 中进行配置。

- 查看所有系统变量

```
SHOW [SESSION | GLOBAL] VARIABLES;
```

- 通过 LIKE 模糊匹配方式查找系统变量

```
SHOW [SESSION | GLOBAL] VARIABLES LIKE <模糊匹配>;
```

- 查看指定系统变量的值

```
SELECT @@[SESSION | GLOBAL] <系统变量名>;
```

- 设置系统变量

```
SET [@@] [SESSION | GLAOBAL] <系统变量名> = <值>;
```

以上语法如果没有指定 SESSION | GLOBAL，则默认为 SESSION。

7.1.2 用户变量

用户变量的格式为'@var_name'，用户定义的变量是特定于会话的，一个客户端定义的用户变量不能被其他客户端看到或使用，当给定客户端会话退出时，该客户端会话的所有变量都会自动释放；用户变量不区分大小写。

相关语法如下：

- 设置用户定义变量方法

```
SET @var_name = <表达式或值>[, @var_name = <表达式或值>];
```

7.2 流程控制

流程控制语句需要在 BEGIN 和 END 之间使用。

7.2.1 选择

- CASE 语句
  - 等值判断：case 后面直接跟需要被做等值判断的表达式或表字段

```
case 表达式或字段
   when 用来比较的值 1 then 返回的值 1 或语句 1;
   when 用来比较的值 2 then 返回的值 2 或语句 2;
   ...
   else 返回的值 n 或语句 n;
end case;
```

  - 区间判断：case 后面不跟任何值或表达式，在 when 后通过条件表达式来判断所属情况

```
case
   when 判断条件 1 then 返回的值 1 或语句 1;
   when 判断条件 2 then 返回的值 2 或语句 2;
   ...
   else 返回的值 n 或语句 n;
end case;
```

- IF 语句

```
if 判断条件 1
then 语句 1;
elseif 判断条件 2
then 语句 2;
...
else then 语句 n;
end if;
```

7.2.2 循环

- WHILE 循环

```
WHILE 条件 DO
   循环体;
END WHILE;
```

- REPEAT 循环

```
REPEAT
   循环体;
UNTIL 条件
END REPEAT;
```

- LOOP 循环

```
LOOP
   循环体;
END LOOP;
```

- 在循环中使用 ITERATE 来结束本次循环，执行下一次循环；LEAVE 结束整个循环。

7.3 存储过程

7.3.1 语法

- 定义存储过程

```
CREATE PROCEDURE <存储过程名称> ([参数列表])
BEGIN
<SQL 语句>
END;
```

  - 参数列表格式为：参数模式 参数名 参数类型
  - 参数模式分为三种：IN（仅用来像存储过程输入），OUT（仅用来向存储过程外部输出），INOUT（即可以用来输入也可以用来输出）

- 调用存储过程

```
CALL <存储过程名称> ([参数]);
```

- 查询存储过程的定义

```
SHOW CREATE PROCEDURE <存储过程名称>;
```

- 删除存储过程

```
DROP PROCEDURE <存储过程名称>;
```

7.3.2 使用示例

因为存储过程定义中包含";"，直接输入会产生问题，所以需要**重新定义换行符**。

使用 delimiter 命令重新定义分隔符。

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

7.4 函数

7.4.1 语法

- 定义函数

```
CREATE FUNCTION <函数名称> ([参数列表])
RETRUNS <返回数据类型>
BEGIN
   <函数体>;
   RETRUN <返回值>;
END;
```

- 查询函数的定义

```
SHOW CREATE FUNCTION <函数名称>;
```

- 删除函数

```
DROP FUNTCTION <函数名称>;
```

7.4.2 使用示例

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
       RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

7.5 触发器

触发器是一个与表关联的命名数据库对象，当表发生特定事件时，该对象将激活。触发器的一些用途是对要插入到表中的值执行检查，或者对更新中涉及的值执行计算。触发器信息存储在 INFORMATION_SCHEMA.TRIGGERS 表中

7.5.1 语法

- 创建触发器

```
CREATE TRIGGER <触发器名>
BEFORE | AFTER INSERT | UPDATE | DELETE
ON <基本表名> FOR EACH ROW
[[FOLLOWS | PRECEDES] <另一触发器名>]
<触发器体>;
```

  - 可以为具有相同触发事件和操作事件的给定表上定义多个触发器，而[FOLLOWS | PRECEDES] <另一触发器名>可以影响触发器执行顺序

- 在触发器主体中，**OLD 和 NEW** 关键字可以访问触发器影响的行。**INSERT 触发器中只能使用 NEW.col_name，在 DELETE 触发器中只能使用 OLD.col_name，在 UPDATE 触发器中使用 OLD 表示修改前的行，NEW 表示修改后的行。**
- 通过使用 BEGIN ... END 构造，您可以定义一个执行多个语句的触发器，也需要重新定义 mysql 语句分隔符，以便在触发器中使用";"分割符。
- **单独定义存储过程，然后使用简单的 CALL 语句从触发器中调用它存储过程**，这样可以简化触发器的书写，也可以实现代码重用。

➢ 删除触发器

```
DROP TRIGGER [<数据库名>.]<触发器名>;
```

➢ 查看触发器信息

```
SHOW CREATE TRIGGER <触发器名>;
```

➢ 查询数据库中的触发器

```
SHOW TRIGGERS FEOM | IN <数据库名>;
```

7.5.2 使用示例

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
       FOR EACH ROW
       BEGIN
           IF NEW.amount < 0 THEN
               SET NEW.amount = 0;
           ELSEIF NEW.amount > 100 THEN
               SET NEW.amount = 100;
           END IF;
       END;//
mysql> delimiter ;
```

第八章 补充内容

8.1 delete 语句和 truncate 语句清空基本表速度对比

语法：

```
TRUNCATE [TABLE] <表名>;
DELETE FROM <表名>;
```

截断操作的原理是删除并重新创建表，比 DELETE 逐个删除行快的多，尤其是对于大型表。

当删除基本表中所有元组时，为了实现高性能，TRUNCATE 绕过了删除数据的 DML 方法。因此，它不会导致 DELETE 触发器触发，不能对具有父子外键关系的 InnoDB 表执行，也不能像 DML 操作那样回滚。

截断操作会导致隐式提交，因此无法回滚。如果操作表上有锁，则无法执行截断操作。

使用 Node.js 向 study 数据库中的 student 表插入一百万条数据，程序如下：

```
const mysql = require('mysql2');
const db = mysql.createPool({
    host: 'localhost',
    port: '3306',
    user: 'root',
    password: 'WYT12345a.',
    database: 'study'
});

db.query('select 1;', (err, res) => { // 测试连接数据库是否成功
    if(err)
        return console.log(err.message);
    else
        return console.log(res, '数据库连接成功！');
});

const sqlStr = `insert into student
(sid, sname, sgender, sage)
values ?;`

let data = [];

for(let i = 0; i < 1000000; i++) { // 构造要插入的数据
    data.push([addZerosBeforeNum(i, 10), '学生' + getRandomNum(0, 1000000), getRandomNum(0,1) ? '男' : '女', getRandomNum(0, 100)]);
}
```

```
db.query(sqlStr, [data], (err, res) => {
    if(err)
        return console.log(err.message);
    else
        return console.log('插入数据成功! 插入',
res.affectedRows, '行! ');
});

function addZerosBeforeNum(i, digitNum) { // 数字前补0
    let nowDigit = 0;
    let num = i;
    if(num === 0) {
        nowDigit = 1;
    }
    else {
        while(num > 0) {
            num  = Math.floor(num / 10);
            nowDigit++;
        }
    }
    let temp = "";
    for(let i = 0; i < digitNum - nowDigit; i++) {
        temp += "0";
    }
    return temp + i;
}

function getRandomNum(min, max, digitNum) { // 返回[min,
max]范围内的随机数
    let res = Math.floor(Math.random()*(max-min+1))+min;
    if(digitNum) {
        res = addZerosBeforeNum(res, digitNum);
    }
    return res;
}
```

使用 delete 清空表用时 22.17s，结果如下：

```
mysql> select count(*) from student;
+----------+
| count(*) |
+----------+
|  1000000 |
+----------+
1 row in set (0.05 sec)

mysql> delete from student;
Query OK, 1000000 rows affected (22.17 sec)
```

使用 truncate 清空表用时 0.07s，结果如下：

```
mysql> select count(*) from student;
+----------+
| count(*) |
+----------+
|  1000000 |
+----------+
1 row in set (0.05 sec)

mysql> truncate table student;
Query OK, 0 rows affected (0.07 sec)

mysql> select * from student;
Empty set (0.02 sec)
```