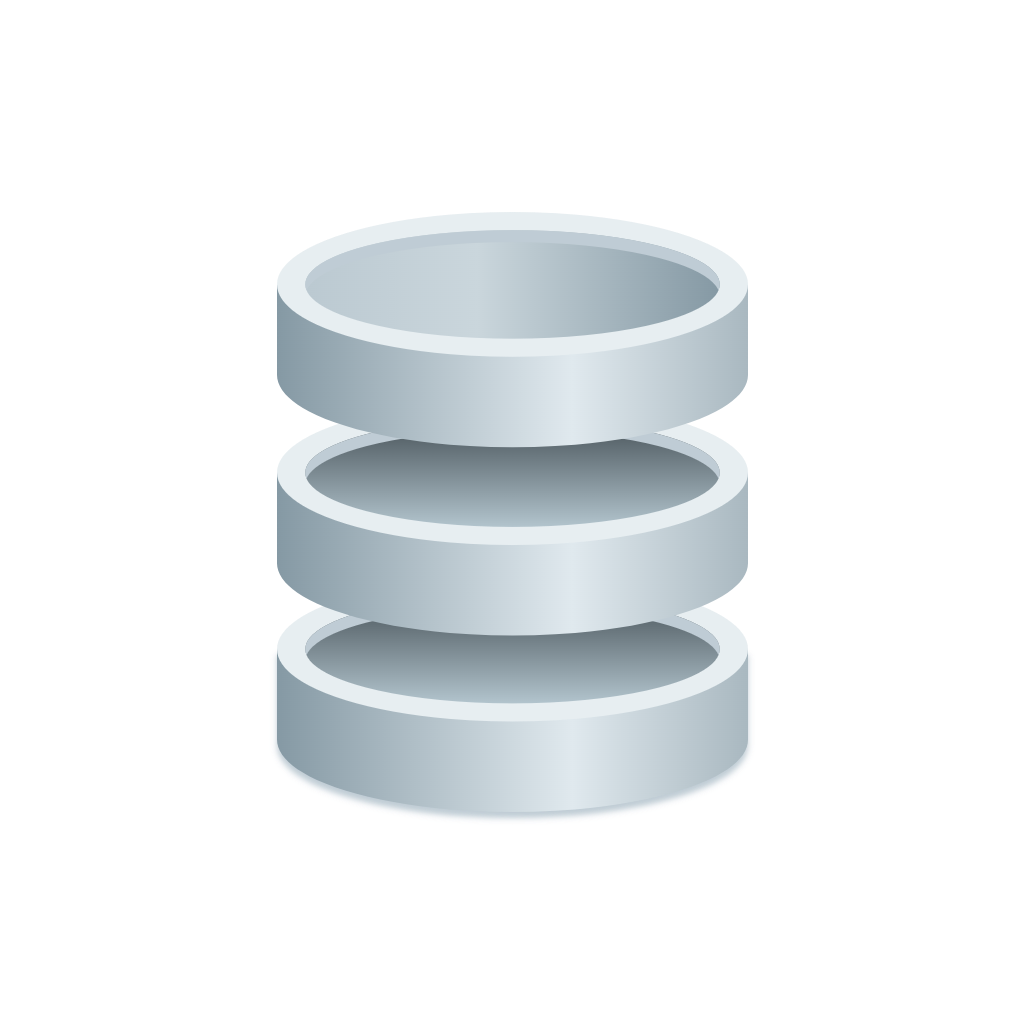
**数据库上机实验报告**



**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 4](#_Toc138612541)

[1.1MySQL介绍 4](#_Toc138612542)

[1.2实验环境 4](#_Toc138612543)

[1.3MySQL基础操作 4](#_Toc138612544)

[1.4实验要求内容与本实验报告对应关系 5](#_Toc138612545)

[1.5数据类型 5](#_Toc138612546)

[1.6表达式与运算符 6](#_Toc138612547)

[1.7系统函数 7](#_Toc138612548)

[1.8参考资料 8](#_Toc138612549)

[第二章 MySQL的安装和实验环境的搭建 9](#_Toc138612550)

[2.1 MySQL的安装 9](#_Toc138612551)

[2.1.1本地安装 9](#_Toc138612552)

[2.1.2服务器安装 10](#_Toc138612553)

[2.2 实验环境的搭建 12](#_Toc138612554)

[2.2.1本地创建数据库 12](#_Toc138612555)

[2.2.2服务器创建数据库 14](#_Toc138612556)

[第三章 数据定义（DDL） 16](#_Toc138612557)

[3.1与模式相关 16](#_Toc138612558)

[3.1.1语法 16](#_Toc138612559)

[3.1.2使用示例 16](#_Toc138612560)

[3.2与基本表相关 17](#_Toc138612561)

[3.2.1语法 17](#_Toc138612562)

[3.2.2列级或表级约束 18](#_Toc138612563)

[3.2.3使用示例 19](#_Toc138612564)

[3.3与视图相关 21](#_Toc138612565)

[3.3.1语法 21](#_Toc138612566)

[3.3.2使用示例 21](#_Toc138612567)

[3.4与索引相关 22](#_Toc138612568)

[3.4.1语法 22](#_Toc138612569)

[3.4.2使用示例 23](#_Toc138612570)

[第四章 数据操纵（DML） 24](#_Toc138612571)

[4.1查询 24](#_Toc138612572)

[4.1.1语法 24](#_Toc138612573)

[4.1.2连接查询 25](#_Toc138612574)

[4.1.3嵌套查询 25](#_Toc138612575)

[4.1.4集合查询 25](#_Toc138612576)

[4.1.5使用示例 25](#_Toc138612577)

[4.2插入 31](#_Toc138612578)

[4.2.1插入单个元组 32](#_Toc138612579)

[4.2.2插入子查询结果 32](#_Toc138612580)

[4.3删除 32](#_Toc138612581)

[4.4修改 32](#_Toc138612582)

[4.5视图上的操作 32](#_Toc138612583)

[4.6使用示例 32](#_Toc138612584)

[第五章 数据控制（DCL） 34](#_Toc138612585)

[5.1数据安全性 34](#_Toc138612586)

[5.1.1访问控制 34](#_Toc138612587)

[5.1.2自主存取控制（DAC） 34](#_Toc138612588)

[5.1.3数据库角色 36](#_Toc138612589)

[5.1.4视图机制 36](#_Toc138612590)

[5.2数据完整性 37](#_Toc138612591)

[5.2.1实体完整性 38](#_Toc138612592)

[5.2.2参照完整性 38](#_Toc138612593)

[5.2.3用户定义完整性 38](#_Toc138612594)

[5.3数据库备份与恢复 37](#_Toc138612595)

[5.3.1物理备份和恢复 37](#_Toc138612596)

[5.3.2二进制日志 37](#_Toc138612597)

[5.3.3完全备份 37](#_Toc138612598)

[5.3.4完全备份的恢复 38](#_Toc138612599)

[5.3.5增量备份 38](#_Toc138612600)

[5.3.6增量备份的恢复 39](#_Toc138612601)

[第六章 嵌入式SQL 40](#_Toc138612602)

[6.1在Node.js中嵌入SQL语句 40](#_Toc138612603)

[6.2在Python中使用SQL语句 41](#_Toc138612604)

[第七章 数据库编程 43](#_Toc138612605)

[7.1变量 43](#_Toc138612606)

[7.1.1系统变量和会话变量 43](#_Toc138612607)

[7.1.2用户变量 43](#_Toc138612608)

[7.2流程控制 44](#_Toc138612609)

[7.2.1选择 44](#_Toc138612610)

[7.2.2循环 45](#_Toc138612611)

[7.3存储过程 45](#_Toc138612612)

[7.3.1语法 45](#_Toc138612613)

[7.3.2使用示例 46](#_Toc138612614)

[7.4函数 46](#_Toc138612615)

[7.4.1语法 46](#_Toc138612616)

[7.4.2使用示例 47](#_Toc138612617)

[7.5触发器 47](#_Toc138612618)

[7.5.1语法 47](#_Toc138612619)

[7.5.2使用示例 48](#_Toc138612620)

[第八章 补充内容 48](#_Toc138612621)

[8.1 delete语句和truncate语句清空基本表速度对比 48](#_Toc138612622)

1. ****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](https://dev.mysql.com/doc/refman/8.0/en/)

1. MySQL的安装和实验环境的搭建
   1. MySQL的安装

2.1.1本地安装

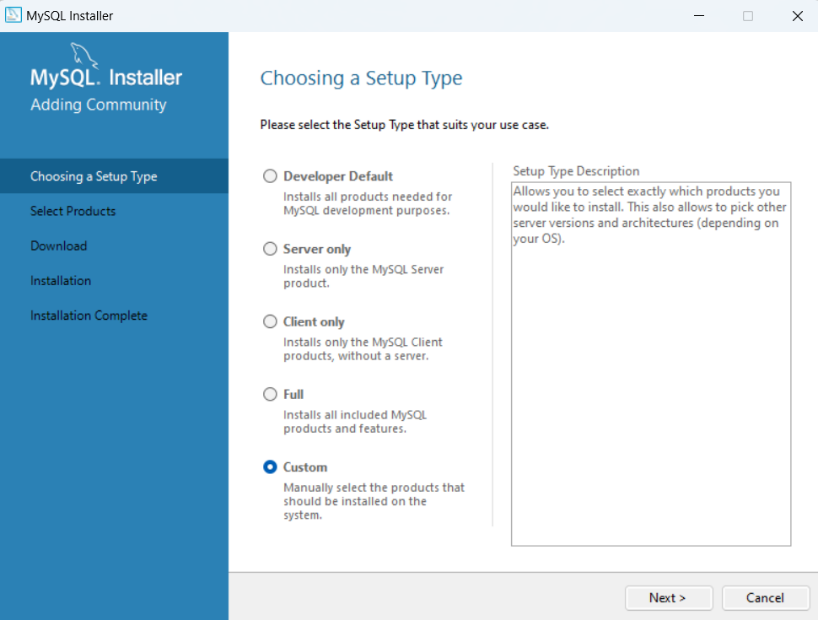
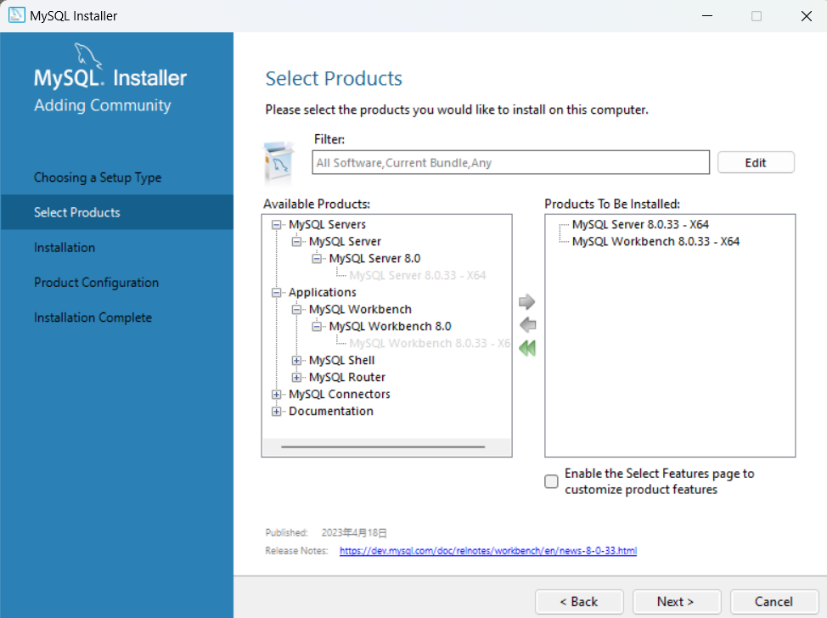
访问MySQL Community 8.0.33的[官方下载地址](https://dev.mysql.com/get/Downloads/MySQLInstaller/mysql-installer-community-8.0.33.0.msi)，打开软件，选择自定义（Custom）安装模式，安装MySQL Server和MySQL Workbench。

图2-2 选择MySQL Server和MySQL Workbench

图2-1 选择自定义（Custom）安装模式

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。

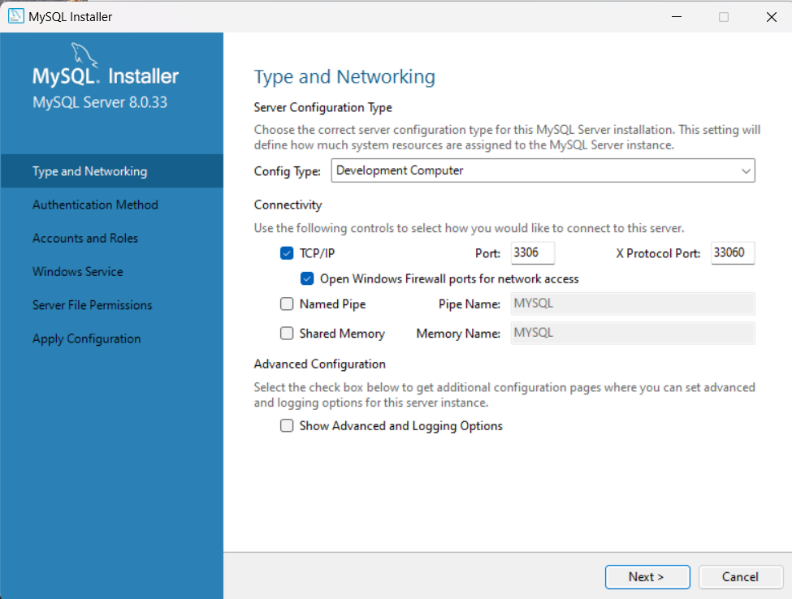
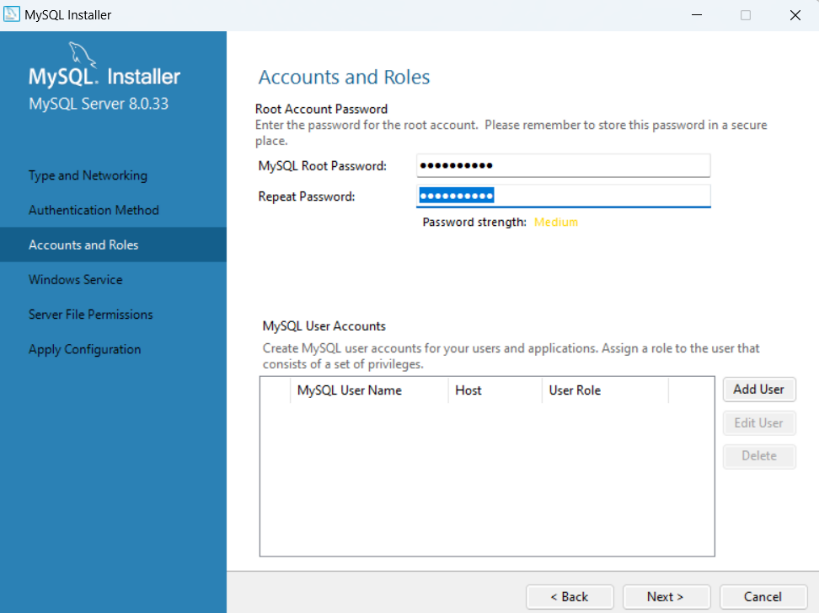


图2-4 配置管理员账号root的密码

图2-3 配置MySQL监听的端口号

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

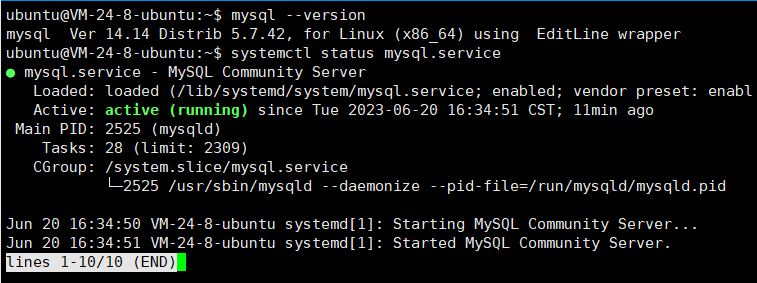
来查看MySQL是否成功安装。

安装完成后，MySQL服务器会自动启动，使用

$ systemctl status mysql.service

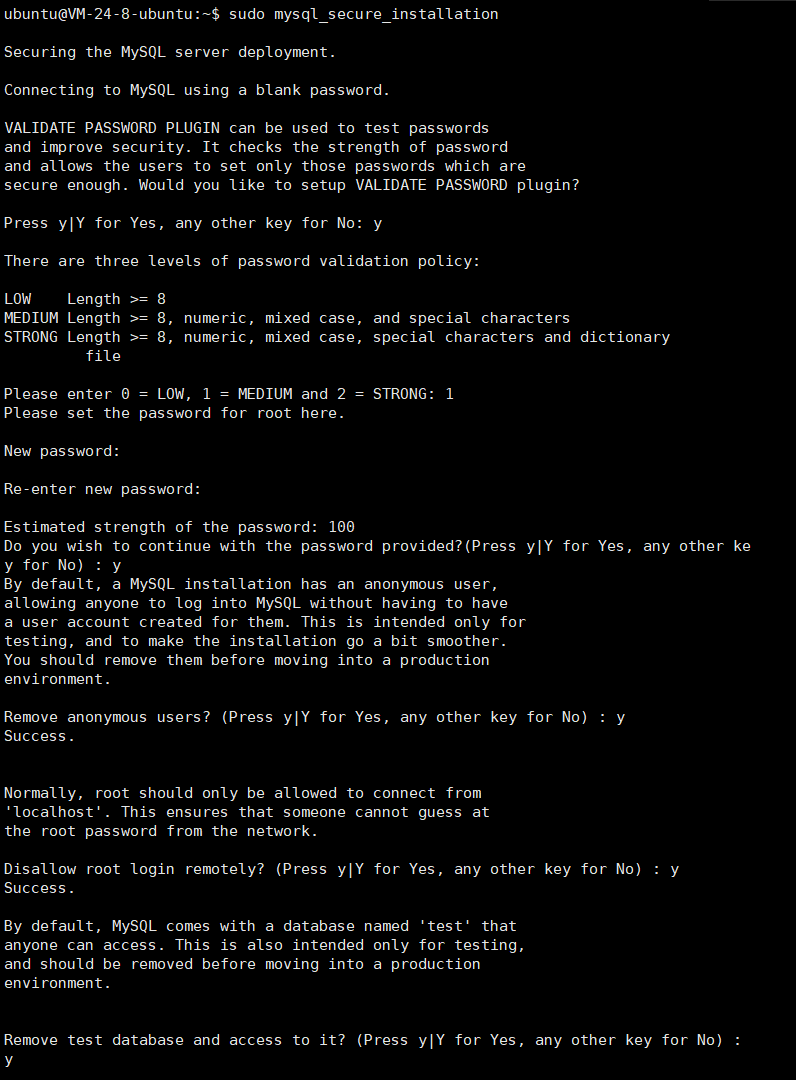
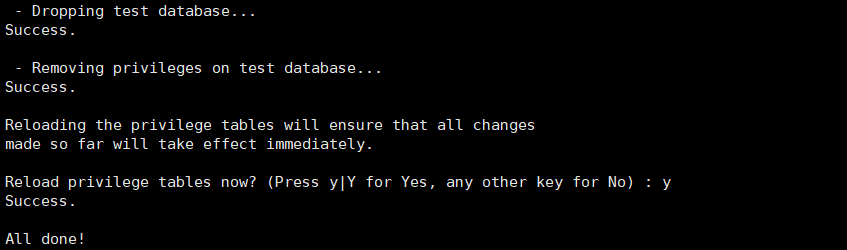
来检测MySQL是否正在运行；如果没有运行，使用命令

$ 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

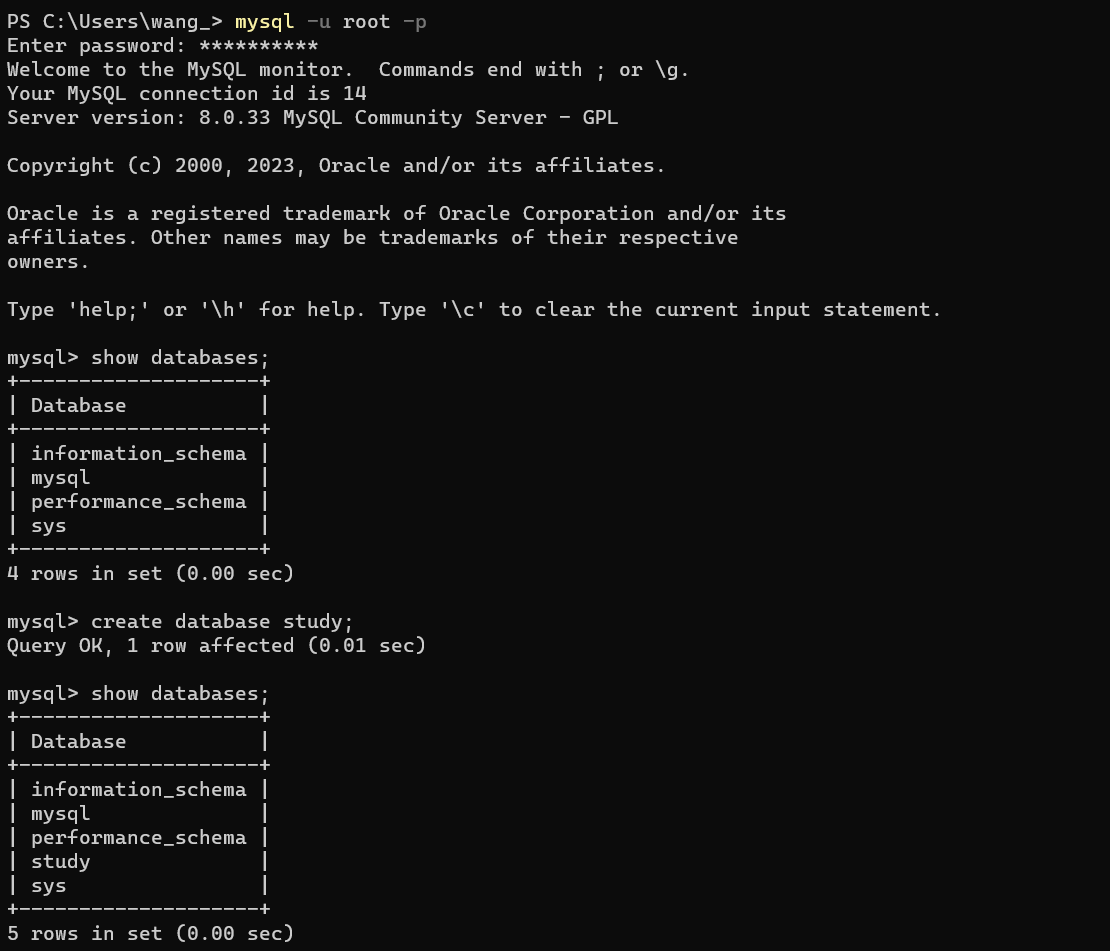
选择一种密码验证策略后，移除匿名用户，禁用root远程登录，移除test数据库，刷新权限表。

* 1. 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;

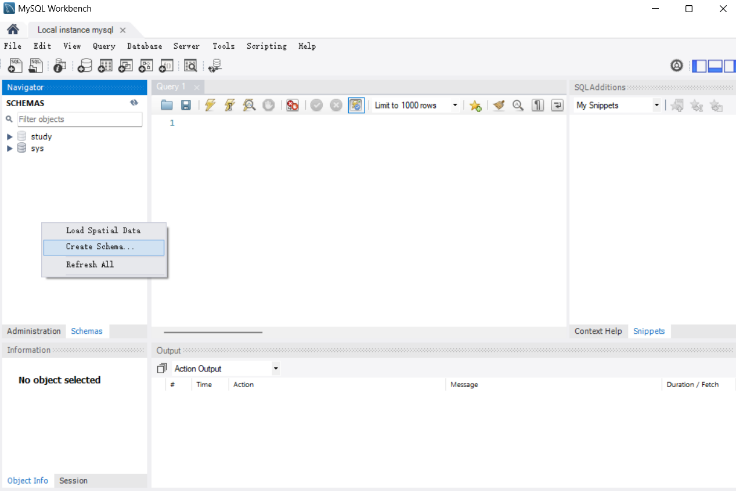
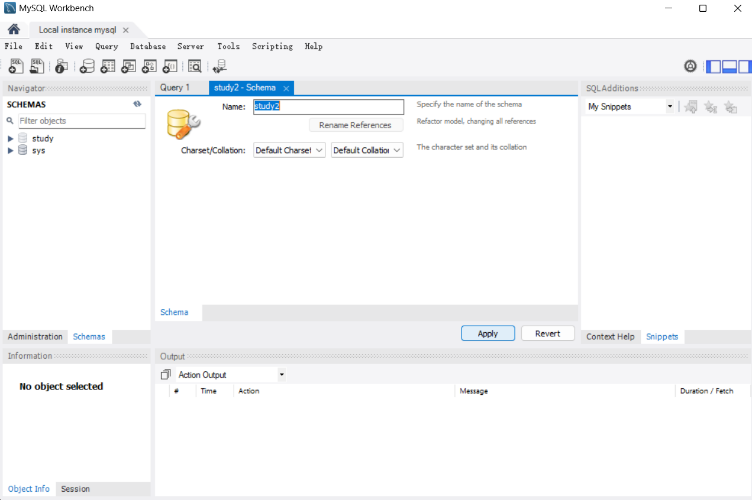
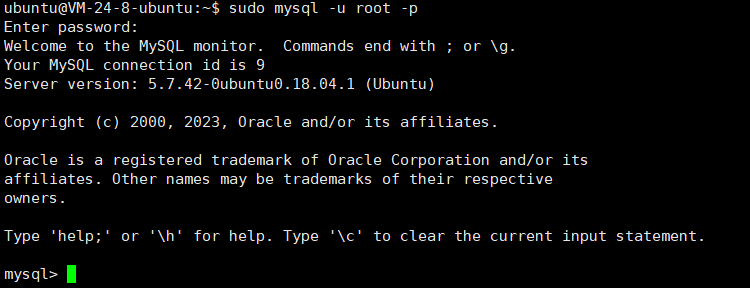
图形化界面方式：

图2-6 输入要创建的数据库名

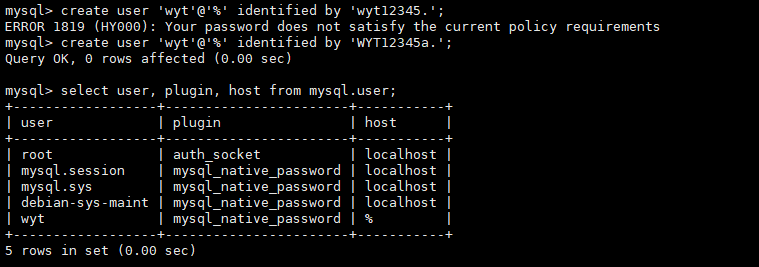
图2-5 在SCHEMA处右键选择Create Schema

2.2.2服务器创建数据库

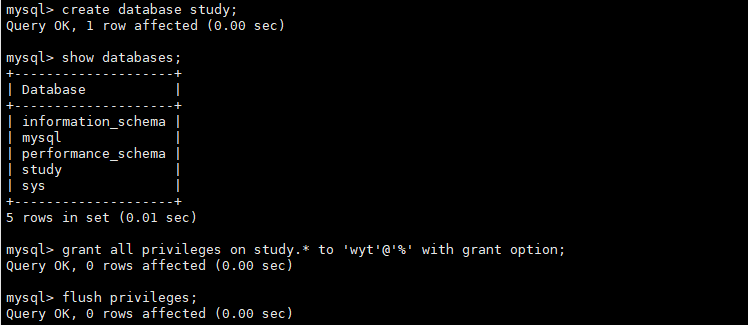
登录MySQL：

$ sudo mysql -u root -p # 然后输入密码

创建用户：

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

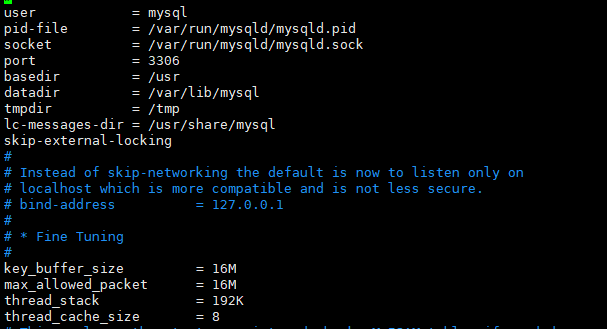
创建study数据库并授予用户在该数据库上的所有权限用于远程登录：

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

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。

最后重启MySQL服务：

$ sudo systemctl restart mysql,service

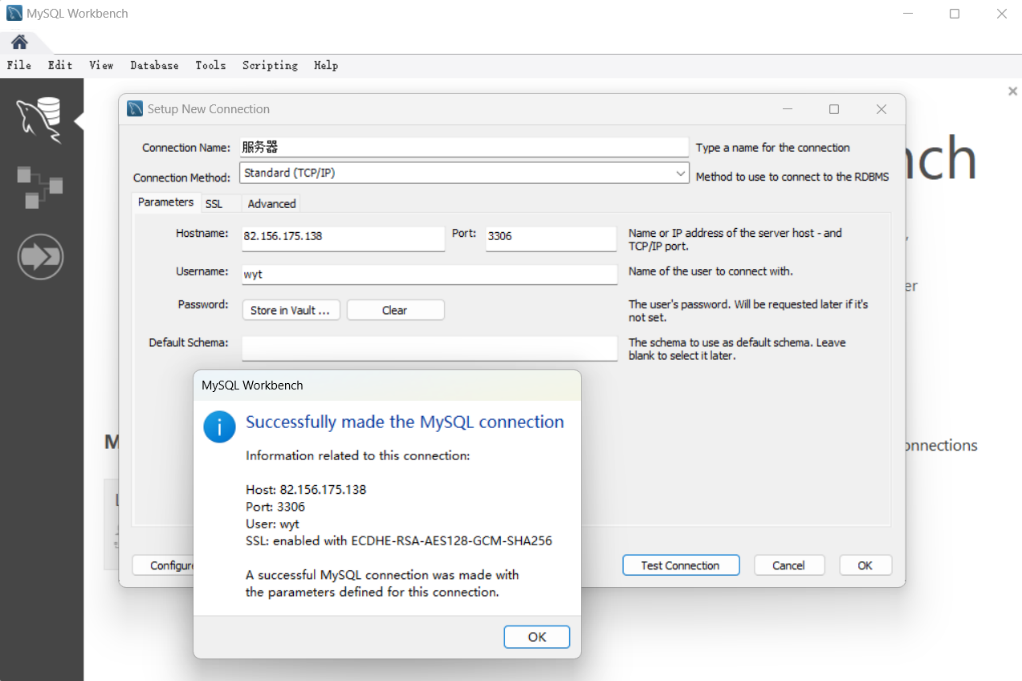
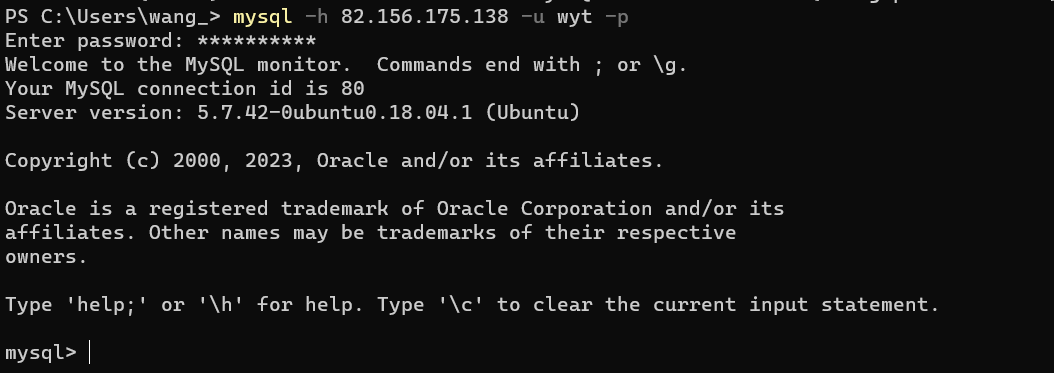
本地连接远程服务器测试：

图2-8 本地Workbench测试

图2-7 本地命令行测试

1. 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

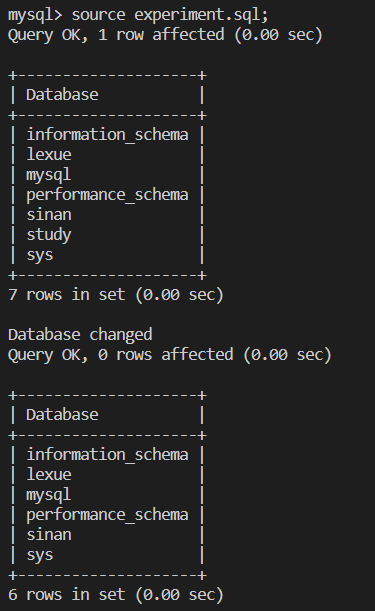
create database study; -- 定义数据库

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：

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,

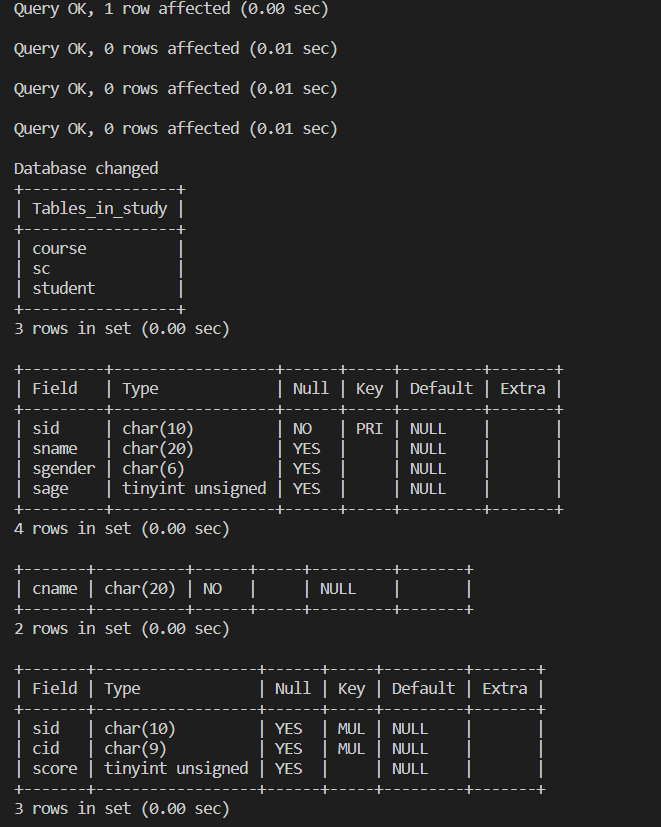
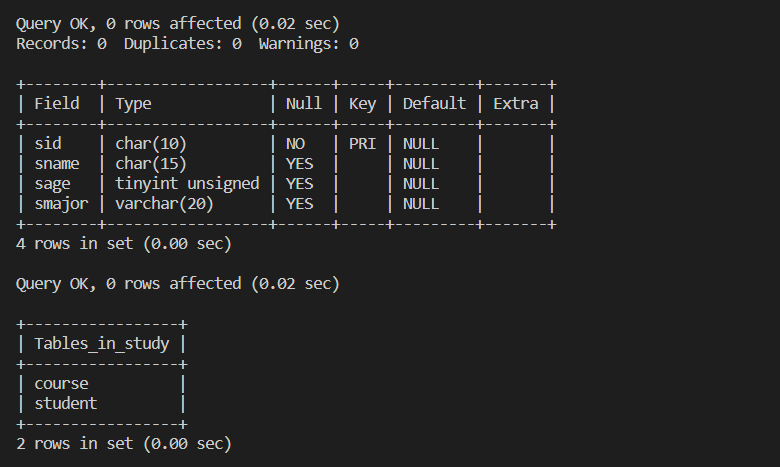
modify sname char(15);

describe student;

-- 删除

drop table study.sc;

show tables;

结果如下：

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

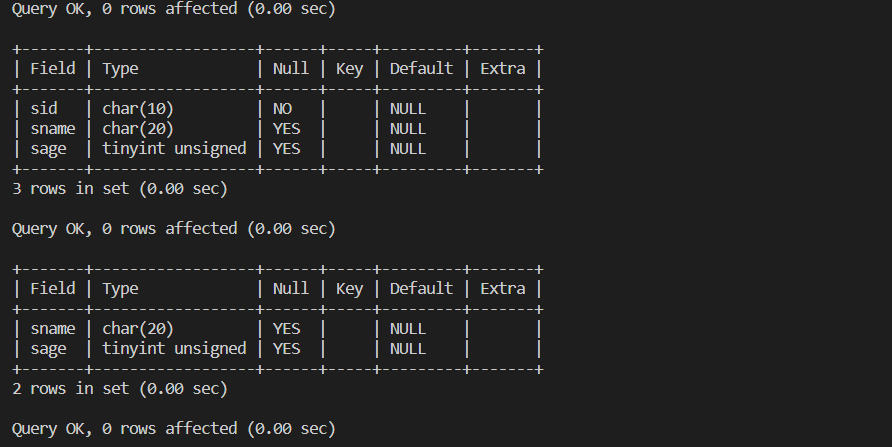
alter view male\_age (sname, sage) -- 修改视图，注意select中的列要与视图中的列对应

as select sname, sage from student

with check option;

desc male\_age;

drop view male\_age; -- 删除视图

结果如下：

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

\*/

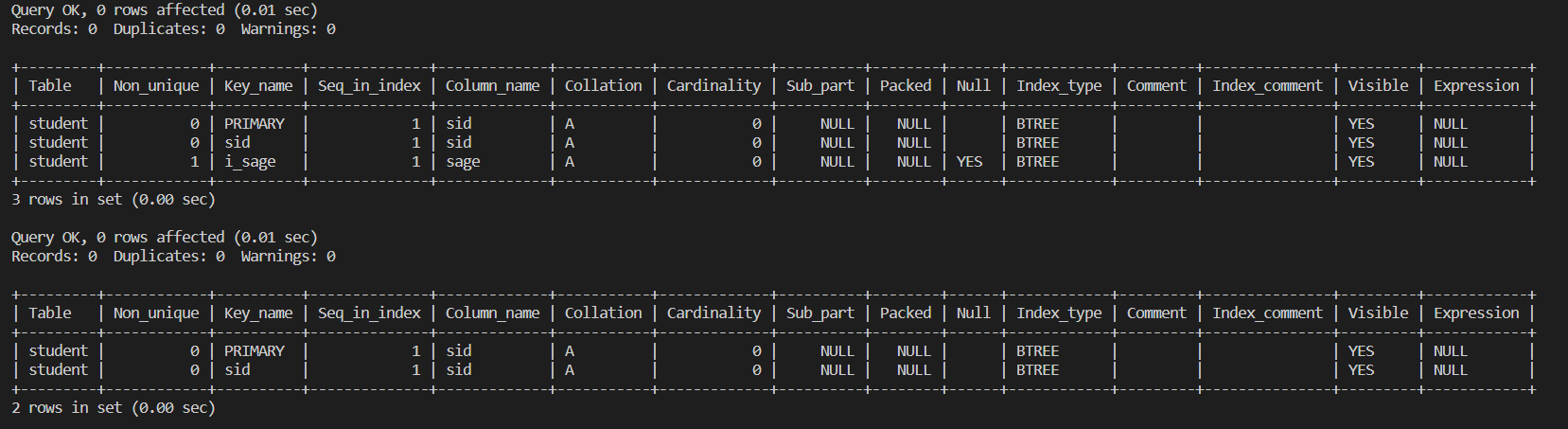
create index i\_sage -- 创建索引

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：

1. 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

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

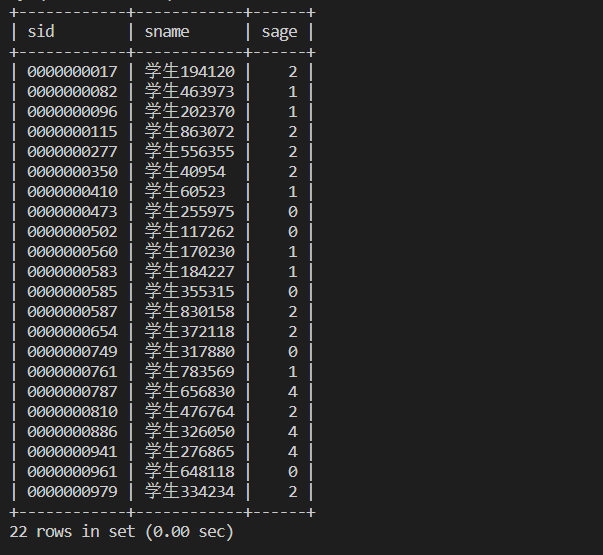
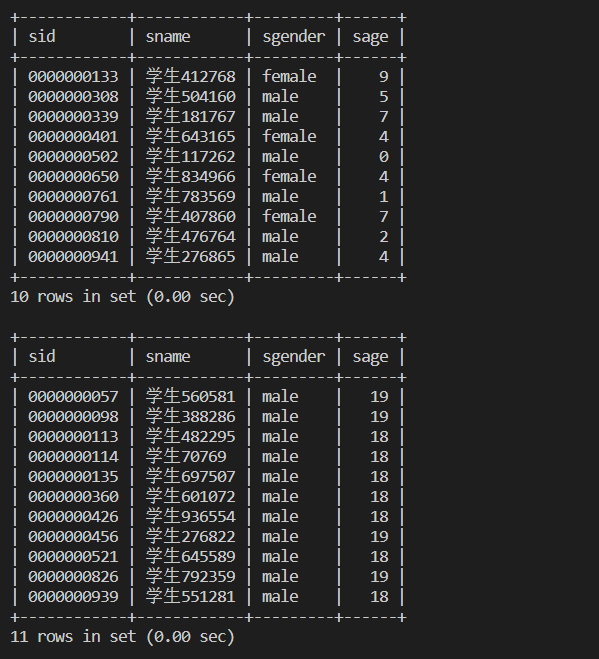
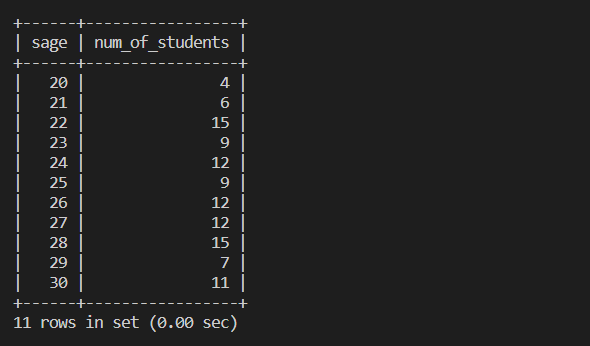
select \* from student where sage in (18, 19) and sgender = 'male';

-- 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：

连接查询：

/\*

\* 连接查询

\*/

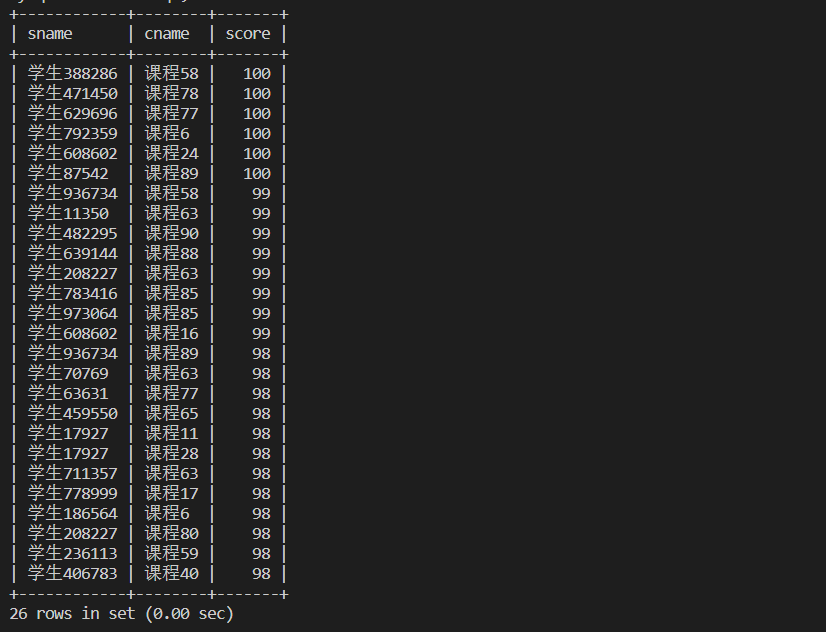
-- 查询所有课程中成绩为95~100、年龄在18~30岁之间的学生姓名，课程名和成绩

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：

嵌套查询：

/\*

\* 嵌套查询

\*/

-- 查询选修了20门课以上的学习信息

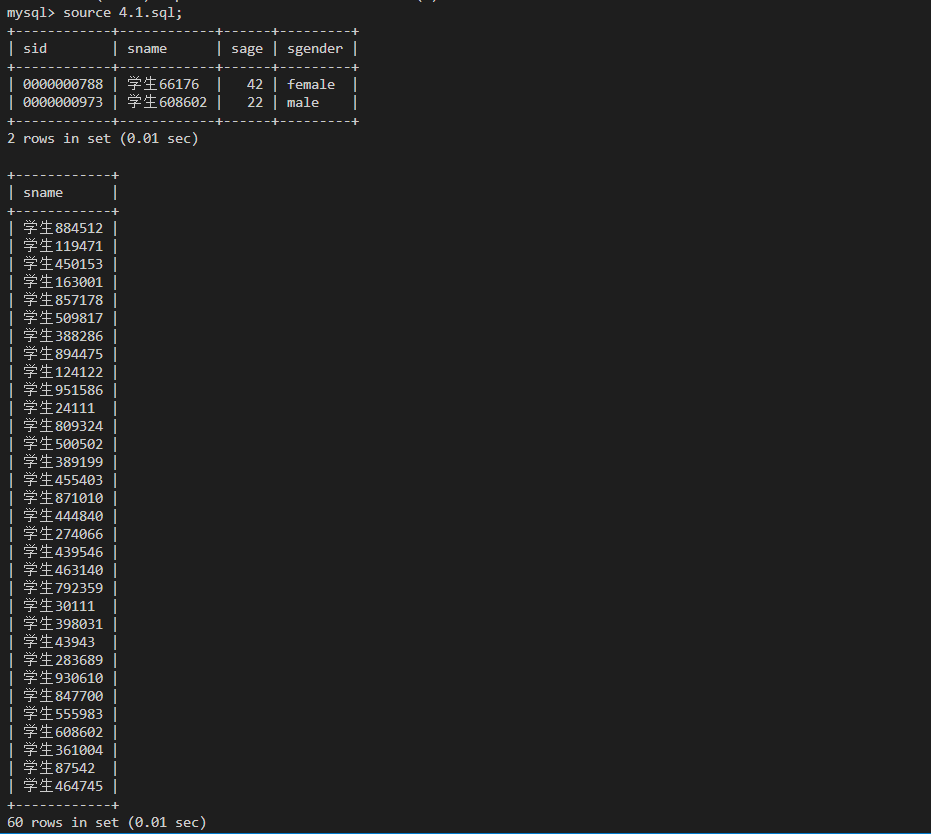
select sid, sname, sage, sgender from student

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：

集合查询：

/\*

\* 集合查询

\*/

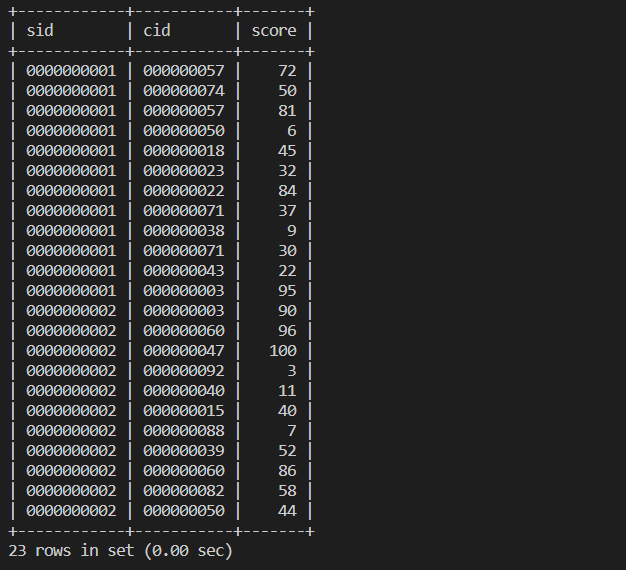
-- 查询学号为0000000001和0000000002的选课信息

select \* from sc where sid = "0000000001"

union

select \* from sc where sid = "0000000002";

结果如下：

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

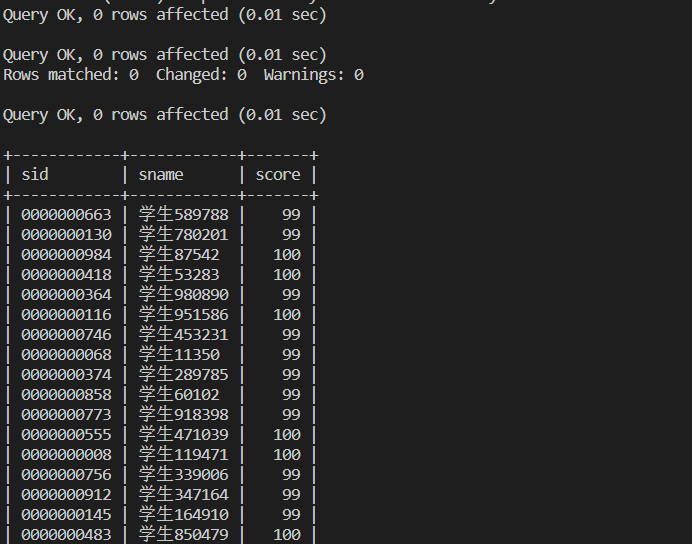
\*/

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：

1. 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

5.2.1实体完整性

实体完整性是指主键唯一且不为空。通过对主键值的约束实现实体完整性；一般系统会在主键属性上自动创建唯一索引来强制唯一性约束。

在创建表时可以定义实体完整性：

* 在列级定义

create table study.course

(cid char(9) not null unique primary key,

cname char(20) not null);

* 在表级定义（**多个属性构成的主键只能使用该方式**）

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(2) check(sgender in ('男', '女')), -- 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

5.2.2参照完整性

参照完整性是指外键要么为空值，要么是被引用关系中元组中的对应值。参照完整性不仅存在于关系之间，**同一个关系内部属性之间也会有参照完整性约束**。

在定义基本表时可以指定参照完整性，**MySQL的外键参照只能定义在表级约束，列级约束不会生效**：

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，使用了外键约束

foreign key(cid) references study.course(cid));

在定义参照完整性时，可以指定删除或更新时参照关系或被参照关系更改的策略。需要在表级约束后面指定，格式为：

FOREIGN KEY(<外键>) REFERENCES <基本表名>(<被参照列名>) ON DELETE | ON UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

|  |  |
| --- | --- |
| 参数 | 意义 |
| RESTRICT | 限制参照关系中外键的改动（默认值） |
| CASCADE | 跟随外键改动 |
| SET NULL | 设为空值 |
| NO ACTION | 无动作 |
| SET DEFAULT | 设为默认值 |

5.2.3用户定义完整性

* 列级约束：UNIQUE、NOT NULL、CHECK约束、DEFAULT、CONSTRAINT约束等
* 表级约束：CONSTRAINT约束
* 触发器：触发器是用户定义在关系数据表上的一类由事件驱动的特殊类型的存储过程。它用编程的方法实现复杂的业务规则，在对表或视图发出UPDATE、INSERT或DELETE语句时自动执行。
  + 触发器使用场景

1. 实现复杂的业务规则
2. 实现比CHECK约束更复杂的数据完整性
3. 比较数据修改前后的状态
4. 定义用户定制的错误信息
5. 维护非规范化数据
6. 如果应用程序要求根据另一个表中的列验证列值，则必须使用触发器

* 具体使用方法请查看第七章。

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>;

1. 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()

1. 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。

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

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

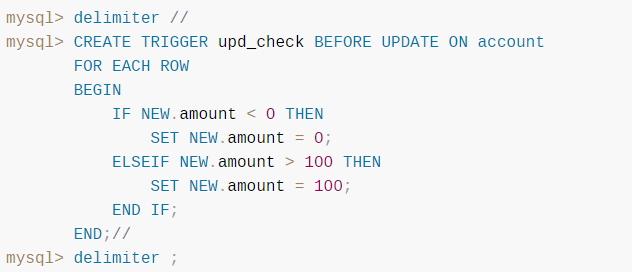
DROP TRIGGER [<数据库名>.]<触发器名>;

* 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

1. 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;

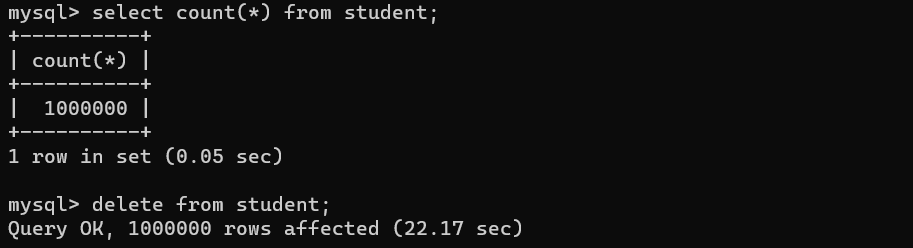
if(digitNum) {

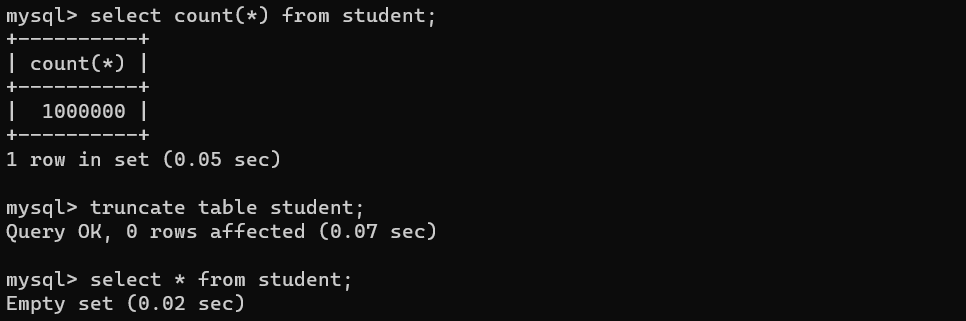
res = addZerosBeforeNum(res, digitNum);

}

return res;

}

使用delete清空表用时22.17s，结果如下：

使用truncate清空表用时0.07s，结果如下：