華南師範大學
SOUTH CHINA NORMAL UNIVERSITY

实验报告

课程名称：_____

实验题目：_____

指导老师：_____

学生姓名：_____

学　　号：_____

学　　院：_____

专　　业：_____

班　　级：_____

时　　间：_____

目录

1. 实验目标

- 搭建服务器端的数据库系统软件，使之可以运行，并开放一定的访问权限。
- 编写简单的客户端程序，通过网络，访问各种类型的数据库。
- 基于 C/S 访问模式开发（客户端和服务器端包含独立 IP）。
- 服务器端要求全部在 Linux 环境下进行部署。

2. 实验内容及要求

- 在 Linux 下搭建服务器端，并部署运行传统数据库，包括但不限于：MySQL、PostgreSQL、SQL Server、DB2。
- 采用 C/C++、Java 或 Web 的方式来开发客户端。
- 实现数据库的基本功能，如：创建数据表、插入数据、删除数据、修改数据、删除数据表、跨数据表操作。
- 实现额外功能，如：事务支持、索引支持、多用户性能访问。
- 数据库包含至少 5 个表格共 100000 条数据。

3. 基本设计

实验的主体任务分为两个，第一个为在 Linux 平台上搭建一个传统的数据库，并且部署好后端代码控制数据库，响应来自于客户端的请求。第二个任务为搭建一个客户端来访问服务端，并且显示各数据表的信息。在服务端方面，我选择的是阿里云服务器，并在其上部署了 Python + Flask + Gunicorn + MySQL，Python 语言简单便捷，配合上微框架 Flask，大大提高了开发效率；在客户端方面，我选择了 Web 的方式进行开发，用到的语言和库有 JavaScript、HTML、CSS、jQuery。

下面给出基本设计的示意图，下面的章节将详细介绍各个部分的情况。

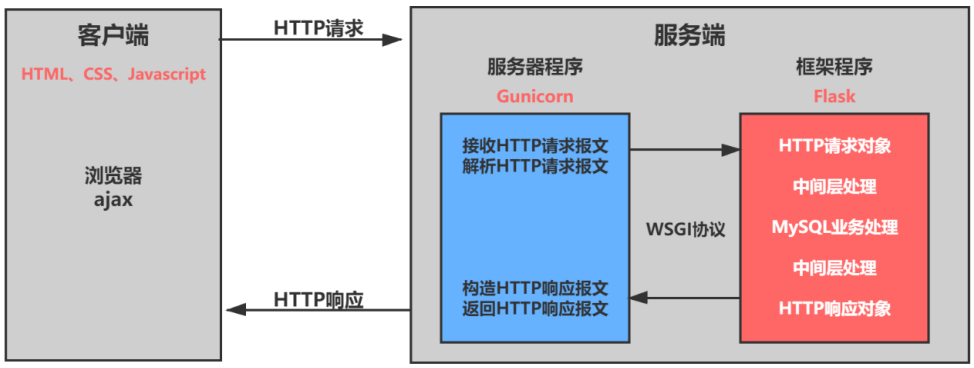

图 1　项目架构图

4. 服务端的基本搭建部署

为了能够让客户端在 24 小时内随时访问到服务端，我们采用了阿里云服务器来部署我们的服务端。选择阿里云服务器来开发有众多优点：（1）服务端具有真实的独立 IP，同时可以随时响应，而使用虚拟机则需要每次都花一段时间来启动。(2)最大程度释放主机的性能，让我们可以用自己的电脑专注于代码开发上。（3）接近真实的工作场景。在以后的工作中，公司的服务端程序也多部署在云服务器上，我们通过 SSH 与服务端进行多方面的交互。如今采用阿里云服务器来开发则可以帮助我们提前体验以后工作的模式。

在阿里云 CentOS8 服务器上配置好 SSH 客户端，并生成公私钥文件进行部署。部署完成后，便可以通过 Windows Terminal 的 SSH 客户端对其进行连接。连接成功的截图如下所示：

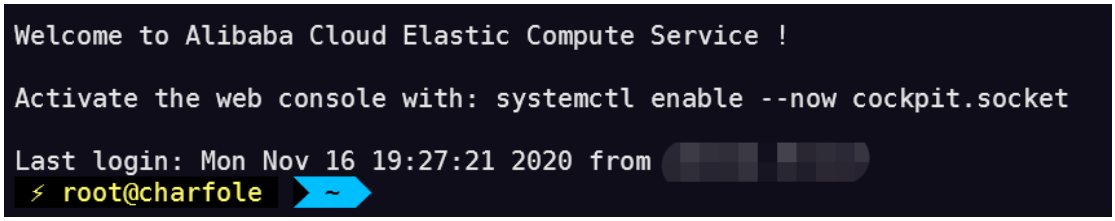

图 2　SSH 连接阿里云服务器

同样地，为了能够快捷高效地在服务器上进行后端代码的开发，我同时也在 VSCode 上连接了阿里云服务器，实现在 VSCode 中远程编辑服务器中的文件。

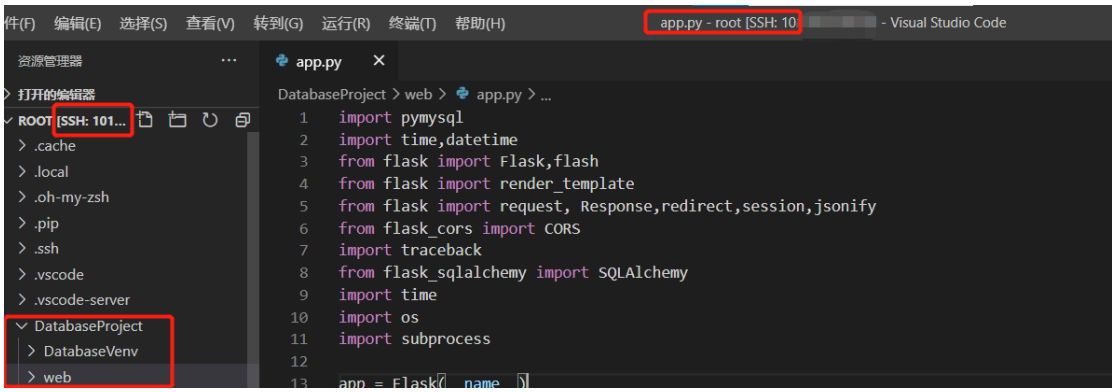

图 3　VSCode 远程连接服务器

经过这两步后，便可以开始准备搭建数据库和后端程序了。在传统数据库这一方面，我选择了 MySQL，主要是因为它具有以下这几个优点：（1）性能卓越、服务稳定，很少出现异常宕机。（2）开源无版权、使用成本低。（3）历史悠久、社区活跃。(4)软件体积小且安装使用简单，易于维护。(5)支持多种操作系统、

提供多种 API 接口，支持与多种主流的开发语言进行交互。经过简单的几步操作后，便在服务器中部署好了 MySQL，其版本信息如下所示：

```
⚡ root@charfole ~ mysql -v
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2981
Server version: 8.0.21 Source distribution

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
```

图 4　查看 MySQL 版本信息

在部署好数据库后，还需要有一个 DBMS 来对其进行更好的操作与管理。我选用了功能强大且较为流行的 Navicat 作为 DBMS，远程连接服务器，便可以在 Navicat 查看到部署好的数据库。

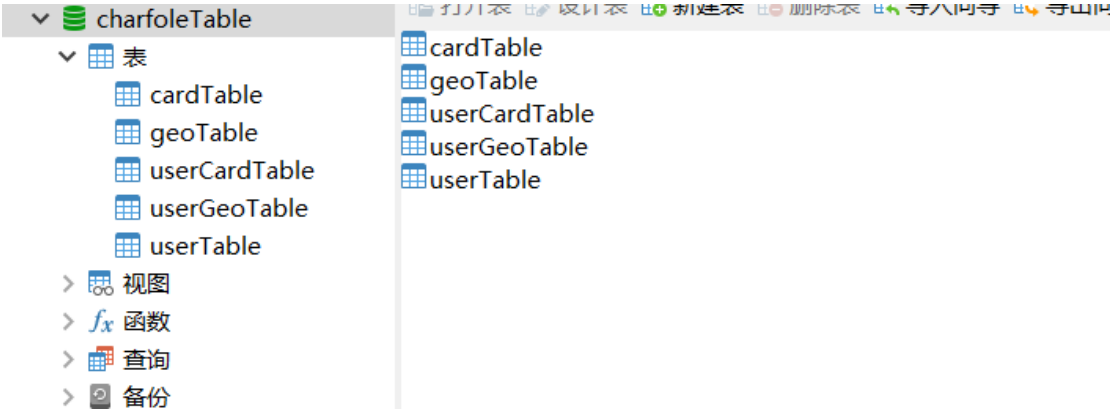

图 5　Navicat 远程管理 MySQL

在后端开发的框架方面，我选择了 Python 的 Flask 框架。Flask 框架的优点如下：（1）Flask 是一个微框架，十分轻巧简洁，易于学习。（2）Flask 可扩展性强，Python 语言的加持让它能够随时引入各类强大的库。（3）Flask 内置开发服务器和调试器。在阿里云服务器上按教程配置好 Flask 后，其版本信息如下所示：

```
>>> import flask
>>> flask.__version__
'1.1.2'
>>>
```

图 6　查看 Flask 版本信息

此外，配合 Gunicorn 服务器，便可以能够快速地将 Flask 程序部署到服务器中。

经过上述的工作后，如今的服务端已成功部署好了 MySQL 和 Flask，并实现了利用 Windows Terminal、VSCode 和 Navicat 对其进行远程管理。

5. MySQL 数据表的构造

根据实验的要求，数据库至少需要包含五个表格与 100000 条以上的数据。为了使数据库中的数据更贴切实际，我决定采用 Mock 的方法对数据表进行构造。至于 Mock 的工具选择，我采用了 Mockaroo 对数据表进行构造，Mockaroo 能够支持数据以不同的形式导出，包括 JSON、CSV、SQL 等；此外，其数据的种类非常贴近生活实际，而不是简单且无意义的数据。唯一美中不足的是，其免费的 Mock 数据上限为 1000 行，与要求的 100000 行相距甚远，因此，我将 Mock 出三个独立的基本数据表——userTable、geoTable、cardTable，其规模为数百行，其后对三个数据表进行两两的笛卡尔积操作，使其数据量翻倍，同时使不同维度的数据进行组合，形成两个新表——userGeoTable 和 userCardTable。这样，便解决了数据量不足的问题。

下面，我将展示五个表的规模。（有一部分数据在测试中改变，因此数据量不是十的倍数）

信息 | 结果 1 | 剖析 | 状态

TABLE_NAME	TABLE_ROWS
userCardTable	60041
userGeoTable	39421
cardTable	287
geoTable	200
userTable	200

图 7　查看五个表的规模

之后，我将展示三个基本表的概览信息。

userId	name	gender	email
1	Stevena Sutherns	Female	ssuthernsrr@goodreads.com
2	Evered Brigg	Male	ebrigg0@bloglovin.com
3	Giacobo Alchin	Male	galchin1@pinterest.com
4	Crystie Hakonsen	Female	chakonsen2@icq.com
5	Natasha Maharey	Female	nmaharey3@cafepress.com
6	Conroy Beattie	Male	cbeattie4@deliciousdays.com
7	Rey Scurman	Female	rscurman5@smh.com.au
8	Sinclare Kesterton	Male	skesterton6@narod.ru
9	Dieter Pibworth	Male	dpibworth7@bbc.co.uk
10	Amberly Marcum	Female	amarcum8@boston.com

图 8　userTable 概览

country	postalCode	college
China	(Null)	Université de Fianarantsoa
Argentina	2624	Universidad Nacional de San Luis
Jordan	(Null)	Jordan Academy of Music / Higher Institute of Music
Czech Republic	788 32	University of Veterinary and Pharmaceutical Science
Tajikistan	(Null)	Tajik Agrarian University
China	(Null)	China University of Geosciences Wuhan
Japan	349-0218	Nippon Dental University
Belarus	(Null)	Minsk State Linguistic University
Indonesia	(Null)	Universitas Merdeka Madiun
Philippines	3322	Adamson University
Albania	(Null)	Academy of Sports and Physical Training
Russia	363621	Kazan State Music Conservatory
Croatia	31214	University of Dubrovnik
Portugal	2705-750	Universidade Atlântica

图 9　cardTable 概览

cardType	cardNum	currencyType
visa	4017953432200	PHP
diners-club-carte-bl	30078784706079	EUR
bankcard	5602255434059995	RUB
instapayment	6378069475942377	JPY
jcb	3541999386950631	USD
mastercard	5100172838436926	IDR
mastercard	5048370199446311	RUB
jcb	3563586832616776	CNY
jcb	3582381184078388	JPY
jcb	3569147478926971	CNY
bankcard	5602254728028030	RWF
diners-club-carte-bl	30446483360458	RUB
jcb	3541706926978760	PLN

图 10　geoTable 概览

综上，通过 Mockaroo 和 Navicat 来对数据库进行操作，我们已经顺利创建了数据库，并构造了五个表共十万余行数据。

6. Flask 操纵 MySQL 数据库

Python 的生态十分强大，操纵各个主流数据库的工具也必不可少。如果想要利用 Python 操纵 MySQL，则需要借助一个名为 pymysql 的库，它操作简便，可以直接接收以字符串形式表示的 MySQL 语句，并很好地支持事务特性。利用 Flask 配合 pymysql，我们可以非常快速地实现增删改查、事务等功能。

由于 pymysql 可以直接执行字符串形式的 MySQL 语句，而数据库的基本操作——创建数据表、插入数据、删除数据、修改数据、删除数据表、跨数据表执行等操作均可以通过 MySQL 语句来实现。因此，对于这些基本功能，我们将其统一到一个函数中，这个函数接收到了前端 POST 的 SQL 语句后，便可以对数据库进行相应的操作，而接收到 GET 方法，则默认返回 userTable 中的数据。下面给出函数的核心代码：

代码 1　Flask 实现 MySQL 的基本操作

输入：请求的方法

输出：SQL 的执行情况

```
@app.route('/basic',methods=['POST','GET'])
def basic():
    conn = pymysql.connect(host='127.0.0.1', user='root', password='', db='charfoleTable', charset='utf8')
    cur = conn.cursor()

    if request.method=='GET':
        sql = "SELECT * FROM userTable"
        cur.execute(sql)
        cols = cur.description #获取列名
        col = []
        for i in cols:
            col.append(i[0])
        u = list(cur.fetchall())
        u = jsonify(col,u)
        return u

    if request.method=='POST':
        sql = request.form.get('SQL')
        judgeString = sql.split(' ',1)[0]
        selectString={'SELECT','select'}
        if judgeString in selectString:
            cur.execute(sql)
            cols = cur.description #获取列名
```

```
            col = []
            for i in cols:
                col.append(i[0])
            u = list(cur.fetchall())
            u = jsonify(col,u)
            return u
        else:
            try:
                # 提交到数据库执行
                cur.execute(sql)
                return "SUCCESS"
            except:
                return "ERROR"
```

在事务功能方面，客户端需要具备三个相关功能——执行事务、提交事务、事务回滚。因此在处理事务的后端函数方面，我们用三个 action 参数来区分这三种不同的操作，分别是 execute、commit、rollback，分别对应执行、提交和回滚这三个操作。下面给出事务函数的核心代码：

代码 2　Flask 实现 MySQL 的事务功能

输入：请求的方法

输出：事务的执行情况

```
@app.route('/Transaction',methods=['POST'])
def Transaction(myList = SqlList):
    conn = pymysql.connect(host='127.0.0.1', user='root', password='', db='charfoleTable', charset='utf8')
    cur = conn.cursor() #建立游标
    sql = request.form.get('SQL')
    action = request.form.get('action')
    DDL={'CREATE','create','DROP','drop','ALTER','alter','SELECT','select'}

    judgeString = sql.split(' ',1)[0]
    if judgeString in DDL and action=='execute':
        try:
            cur.execute(sql)
            return "SUCCESS"
        except:
            return "ERROR"
    else:
```

```
        if(action == 'execute'):
            try:
                cur.execute(sql)
                myList.append(sql)
                return "SUCCESS"
            except:
                return "ERROR"

        elif(action == 'commit'):
            for i in myList:
                cur.execute(i)
            conn.commit()
            myList.clear()
            return "SUCCESS"

        elif(action == 'rollback'):
            conn.rollback()
            myList.clear()
            return "SUCCESS"
```

在多用户性能查询方面，MySQL 提供了命令（mysqladmin -uusername variables status）供我们查询该数据库下所有用户的性能情况。由于 Flask 框架不能够直接运行命令行代码，需要引入一个名为 subprocess 的库来执行命令行。为了实现该功能，我为数据库额外创建了四个用户，加上原有的 root 用户，则一共有五个用户。下面给出该函数的核心代码：

代码 3　Flask 实现多用户性能查询功能

输入：请求的方法

输出：各个用户的性能情况

```
@app.route('/MultipleUser',methods=['GET'])
def MultipleUser():
    cmdLine = [["mysqladmin","-uroot","variables","status"],["mysqladmin","-uadmin1","variables","status"],  ["mysqladmin","-uadmin2","variables","status"],["mysqladmin","-uadmin3","variables","status"],  ["mysqladmin","-uadmin4","variables","status"]]
    result = []
    userList = ["root","admin1","admin2","admin3","admin4"]
```

```
    for i in range(len(cmdLine)):
        print(cmdLine[i])
        tempShell = subprocess.Popen(cmdLine[i], stdout = subprocess.PIPE,
                                     stderr=subprocess.PIPE,
                                     stdin=subprocess.PIPE)
        out,err = tempShell.communicate()
        out = str(out)
        result.append(out.split('+\\n')[-1])  #对返回的信息进行截取

    return jsonify(userList,result)
```

7. 基于 Web 的数据库客户端开发

在客户端方面，我选择了 Web 开发来实现，借助 HTML、CSS、JavaScript 和 jQuery 库打造的 Web 页面对比其 Windows 应用有着可移植性强、访问简单、开发便捷的优点。

在客户端的功能设计方面，主要分为三个部分，第一个部分为基本的 MySQL 语句输入功能，对应着数据库的基本功能，包括增删查改、跨数据表操作、索引等功能。只需要在文本框中输入对应的 MySQL 语句，后端便可以直接执行。下面给出该部分的前端核心代码：

代码 4　MySQL 基本功能实现

输入：SQL 语句

输出：若 SQL 语句是 SELECT 语句则返回结果，否则直接执行

```
<script type="text/javascript">
    function basic() {
        $.ajax({
            type: "POST", //方法类型
            url: "http://101.xxx.xxx.xx:5000/basic", //url
            data: $('#form1').serialize(),
            success: function(result) {
                if (result != "ERROR") {
                    alert("SUCCESS");
                    var _template = template(document.getElementById('_tem
plate').innerHTML);
                    var html = _template({
                        data: result
                    });
                    document.getElementById('wrap').innerHTML = html;
```

```
                    template.config({
                        compress: true
                    });
                    var _template = template(document.getElementById('_template').innerHTML);
                    $(document).ready(function() {
                        // And make them fancy
                        $("#sampleTableA").fancyTable({
                            sortColumn: 0,
                            pagination: true,
                            perPage: 25,
                            globalSearch: true
                        });
                        // 设置删除的点击事件
                        $(".DELETE").click(function() {
                            $(this).parent().parent().remove();
                        })
                    });
                } else {
                    alert("FALSE");
                }
            },
            error: function() {
                alert("异常！");
            }
        });
    }
</script>
```

第二个部分为事务功能，对应有三个按钮，分别是执行、提交与回滚。三个按钮点击后对应着不同的 ajax 函数，向服务端发送不同的 Action，从而对应着不同的请求。下面给出该部分的前端代码：

代码 5　MySQL 事务功能实现

输入：事务语句

输出：无

```
 <script type="text/javascript">
    function execute() {  //事务执行
        $.ajax({
            type: "POST", //方法类型
            url: "http://101.xxx.xxx.xx:5000/Transaction", //url
            data: $('#form2').serialize() + "&action=execute",
```

```
            success: function(data, textStatus, xhr) {
                if (xhr.status != 250) {
                    alert("SUCCESS");
                } else {
                    alert("FALSE");
                }
            },
            error: function() {
                alert("异常！");
            }
        });
    }
</script>

<script type="text/javascript">
    function rollback() {  //回滚
        $.ajax({
            type: "POST", //方法类型
            url: "http://101.xxx.xxx.xx:5000/Transaction", //url
            data: $('#form2').serialize() + "&action=rollback",
            success: function(data, textStatus, xhr) {
                if (xhr.status != 250) {
                    alert("SUCCESS");
                } else {
                    alert("FALSE");
                }
            },
            error: function() {
                alert("异常！");
            }
        });
    }
</script>

<script type="text/javascript">
    function commit() {
        $.ajax({
            type: "POST", //方法类型
            url: "http://101.xxx.xxx.xx:5000/Transaction", //url
            data: $('#form2').serialize() + "&action=commit",
            success: function(data, textStatus, xhr) {
                if (xhr.status != 250) {
                    alert("SUCCESS");
                } else {
```

```
                    alert("FALSE");
                }
            },
            error: function() {
                alert("异常！");
            }
        });
    }
</script>
```

最后一部分为多用户性能查询功能，置于页面的底端，点击查询按钮后，便可以返回各个用户的性能情况。下面给出这一部分的前端代码：

代码 6　多用户性能查询

输入：无

输出：数据库各用户的情况

```
<script>
        $('.find').click(function() {
            $.ajax({
                type: "GET", //方法类型
                url: "http://101.xxx.xxx.xx:5000/MultipleUser", //url
                success: function(data, textStatus, xhr) {
                    var tpl = template(document.getElementById('tpl1').innerHTML);
                    var html = tpl({
                        data: data
                    });
                    document.getElementById('wrap1').innerHTML = html;
                    template.config({
                        compress: true
                    });
                    var tpl = template(document.getElementById('tpl1').innerHTML);
                    console.log(data);
                },
                error: function() {
                    alert("异常！");
                }
            });
        })
```

```
23.     </script>
```

以上就是客户端部分的核心代码，下面给出客户端的各功能截图。

图 11　客户端页面概览

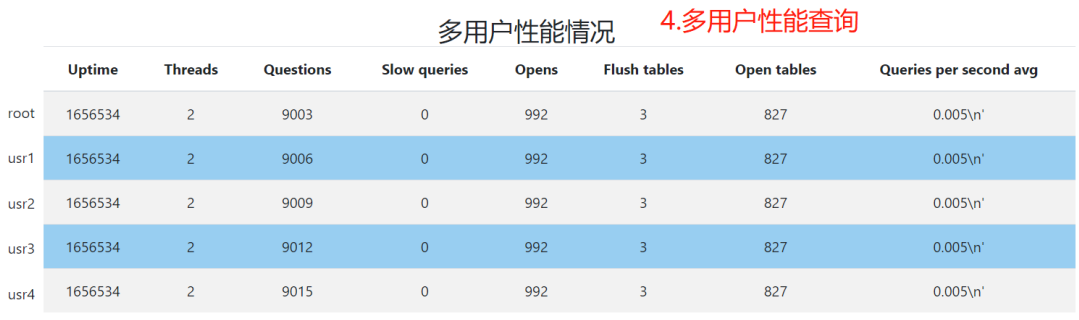

图 12　多用户性能情况查询

8. 功能展示

8.1 基本功能

8.1.1 创建数据表

在 Web 页面中输入 MySQL 如下的建表语句。

```
CREATE TABLE IF NOT EXISTS `bookTable`(
   `id` INT UNSIGNED AUTO_INCREMENT,
   `title` VARCHAR(100) NOT NULL,
   `author` VARCHAR(40) NOT NULL,
   `submission_date` DATE,
   PRIMARY KEY ( `id` )
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

执行后，可在客户端中查看到建表已成功。

图 13　创建表示意图

8.1.2 插入数据

在 Web 页面中输入 MySQL 如下的插入语句。

```
INSERT INTO bookTable ( title, author, submission_date ) VALUES("学习 MySQL","charfole",NOW());
INSERT INTO bookTable ( title, author, submission_date ) VALUES("学习 Flask","charfole",NOW());
INSERT INTO bookTable ( title, author, submission_date ) VALUES("学习 Linux","charfole",NOW());
INSERT INTO bookTable ( title, author, submission_date ) VALUES("学习 HTML","charfole",NOW());
```

执行后，可在客户端中查看到插入已成功。

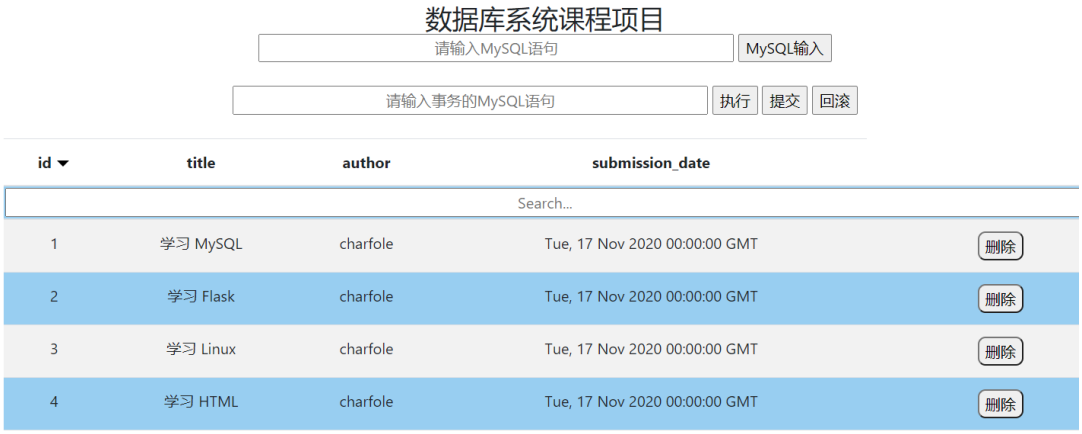

图 14　插入数据示意图

8.1.3 删除数据

在 Web 页面中输入 MySQL 如下的删除语句。

```
1.  DELETE FROM bookTable WHERE id=3;
```

执行后，可在 Navicat 中查看删除操作已成功。

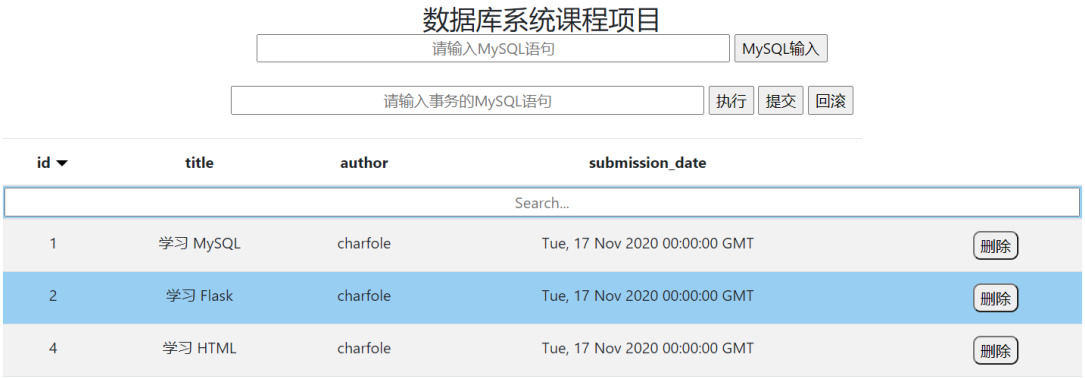

图 15　删除表示意图

8.1.4 修改数据

在 Web 页面中输入 MySQL 如下的修改表语句。

```
1.  UPDATE bookTable SET title='学习 C++' WHERE id=2;
```

执行后，可在客户端中查看到已修改成功。

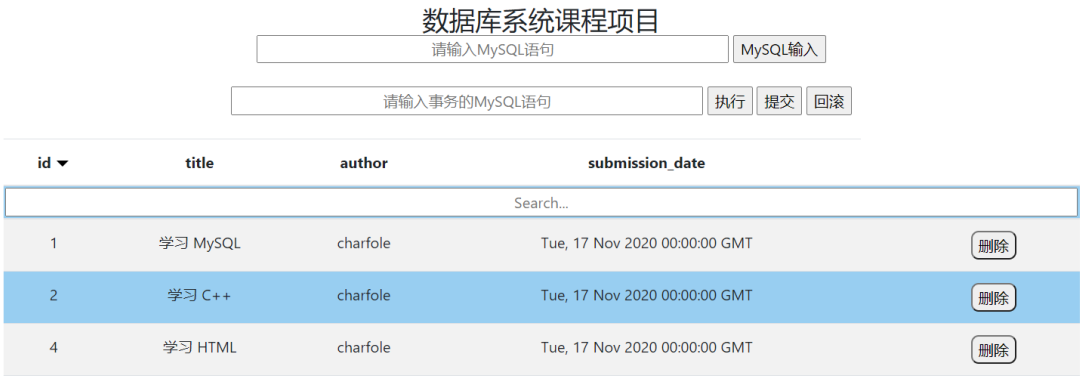

图 16　修改表示意图

8.1.5 跨数据表操作

在 Web 页面中输入 MySQL 如下语句。

```
1.  CREATE TABLE userBookTable AS (SELECT * FROM userTable FULL JOIN bookTable);
```

执行后，可在客户端中查看到跨数据表的操作已成功。

图 17　跨数据表操作示意图

8.1.6 删除数据表操作

在 Web 页面中输入 MySQL 如下语句。

```
1.  DROP TABLE userBookTable;
```

执行后，可在 Navicat 中查看到该表已删除。

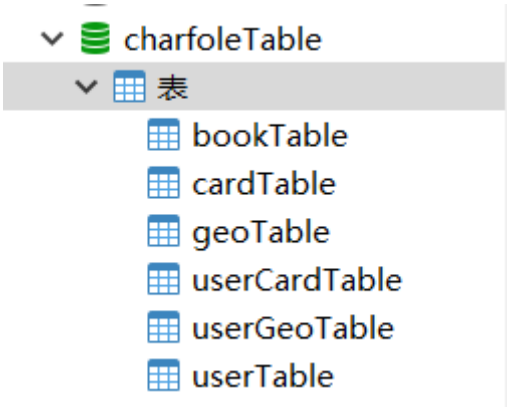

图 18　删除表示意图

8.2 额外功能

8.2.1 索引支持

在 Web 页面中输入 MySQL 如下的建立索引语句。

```
1.  CREATE INDEX book_id ON bookTable (id);
```

执行后，可在 Navicat 中查看到索引已建立成功。

图 19　创建索引示意图

8.2.2 事务支持

在 Web 页面中输入一条插入的 MySQL 语句，随后点击执行，再点击提交，

则完成了一个事务。

执行后，可在客户端中查看到事务已完成。

图 20　提交事务示意图

在 Web 页面中输入一条更新的 MySQL 语句，随后点击执行，再点击回滚，则该事务回滚撤销，其后再次查询数据表，发现没有变化。

```
1.  UPDATE bookTable SET title='学习 Flask' WHERE id=9;
```

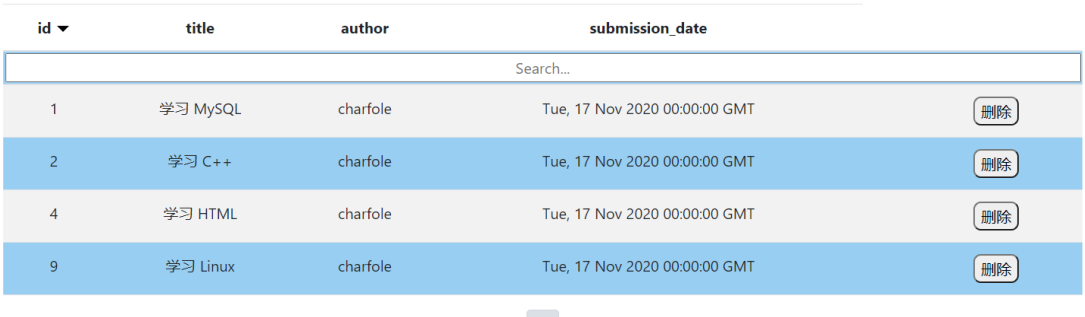

图 21　撤销事务示意图

8.2.3 多用户性能查询

点击多用户性能查询，在 Web 端可以看到当前的用户情况。

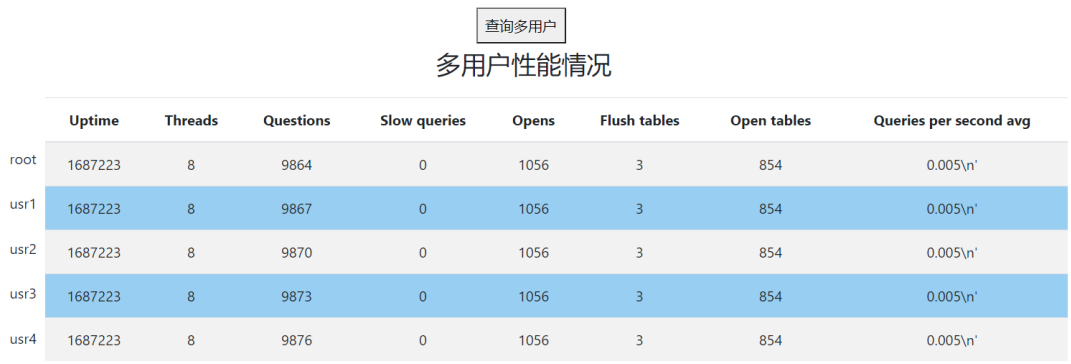

图 22　多用户性能查询示意图

9. 总结

本次实验中，我按照实验的要求，在安装了 CentOS 系统的阿里云服务器中部署好了 Gunicorn 服务器、Flask 微框架以及 MySQL，并在此基础上编写了服务端代码，在后端实现了增删查改、索引支持等功能，同时还实现了事物功能以及多用户性能查询功能。此外，我还借助 JavaScript、HTML、CSS、jQuery 等工具开发了数据库客户端，实现了数据表的展示、基本 SQL 语句执行、事务执行和对多用户性能的展示。对于实验所要求的功能，目前的项目已经基本实现，但除此之外，有不少方面仍待提高，如：（1）目前仅支持输入 SQL 语句的方式来与数据库交互（2）在查询大量数据的时候，由于后端代码缺少特别处理与服务器的性能受限，导致速度较慢。

本次实验使我获益匪浅，让我对 Linux 下的工具部署、后端开发、Web 开发有了更好的理解。在实验的过程中，我遇到了不少的问题，比如事务处理流程后端代码的逻辑问题、Web 客户端的数据接收预处理问题等，但通过询问同学与在互联网搜索，这些问题都被迎刃而解了，这同时也让我解决问题的能力得到了很好的提高。