福昕高级PDF编辑器

高效·安全·专业

立即下载　　点击购买

- OFFICE格式互转
- OCR文字识别
- 文本图像编辑
- 加密和签署
- 交互式动态表单
- 互联PDF文档

数据库实验 8: 简单的论坛网页

1 简介

我们这次实现了一个简单的论坛网页，用的数据库为 mysql，网页使用 django 搭建，主要完成了注册、发起主题（帖子）、回复帖子等相关功能。

2 论坛设计分析

论坛的话，主要需求就是用户、版块、帖子和回复，然后需要把这些功能通过网页的形式展现出来，后端与数据库相连接进行操作。但本次实验是数据库实验，而不是前端开发大作业，只要能完成前端、后端和数据库的连接，就算完成了实验任务。而前端只是用来展示，因此我们做得相对来说比较简陋。具体应用分析如下：

- 用户发帖需要注册
- 用户只能在版块内发起主题（发帖）
- 每个主题一定属于所发起的版块
- 在一个主题下，用户可以对主题进行回复（跟帖）
- 用户可以关注其他用户、也可以拉黑其他用户

2.1 E-R 图设计

根据上述要求，设计成 E-R 图如下。其中有实体集主题、用户和版块，联系集有发帖人、回复、版块帖

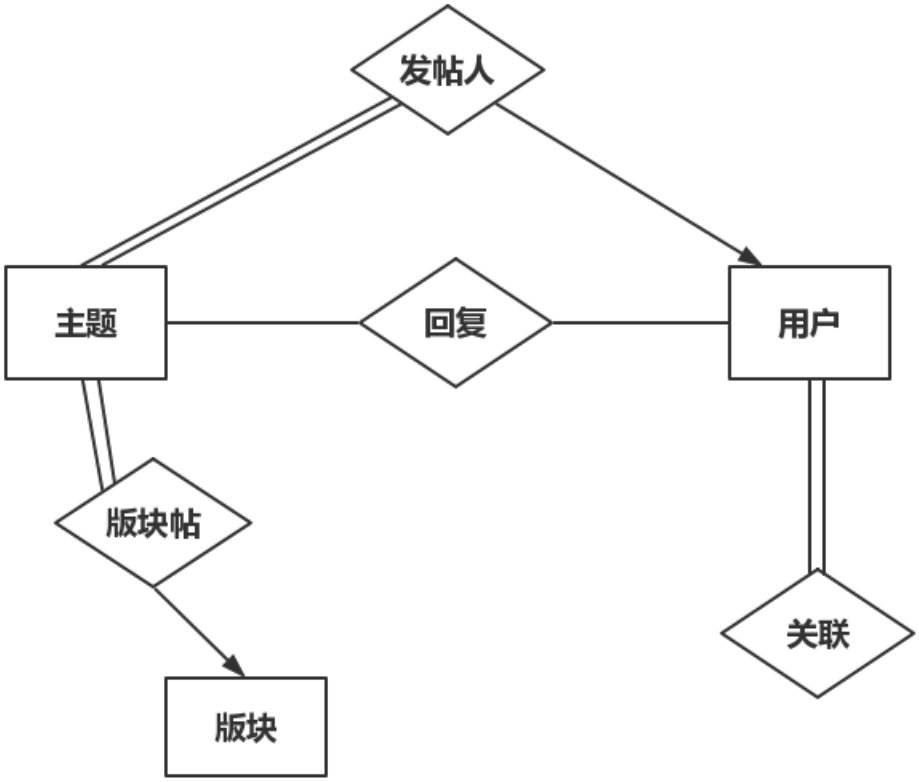

图 1: 简单论坛的 E-R 图

2.1.1 用户和主题

用户和主题之间有 2 个联系集。

对于回复这个联系集，显然这个联系是多对多的，一个用户可以在不同主题下回复，也可以在同一个主题下回复多次。

而对于发帖人这个联系集，注意到主题是全部参与这个联系集的，因为主题一定有且仅有一个发帖人。同时一个用户可以发起多个主题，因此用户是一对多参与的。

2.1.2 版块和主题

主题和版块通过版块帖这个联系集相连接。注意到主题是全部参与到这个版块帖的，因为每个帖子一定要属于一个母版块。同时版块是一对多的，因为显然不能一个主题属于多个版块。

2.1.3 用户和用户

用户和用户之间可以相互关注，也可以拉黑，这个功能通过关联这个联系集实现。关联记录了用户和用户之间的关系，使得这些功能得以实现。显然这个关系也是多对多的。

2.2 MySQL 表设计

有了 E-R 图后，用 MySQL Workbench 自带的 EER Diagram 就可以直接画图，然后导出成对应的 MySQL 脚本语言。实际上，有些联系集直接可以作为一个表的外键引用和一个实体集合并，然后生成一个表。因此在 MySQL 中画出的表会比上面简单。如图 2 所示，theme 即为实体集主题，版块帖被合并到 theme 中，作为一个外键引用到板块 plate 这个实体集对应的表；回复这个联系集因为是多对多的，所以独立生成一个表 reply，同时拥有对 theme 和 user 的外键引用；link 这个表表示着实体集关联，用于实现拉黑、关注的功能。

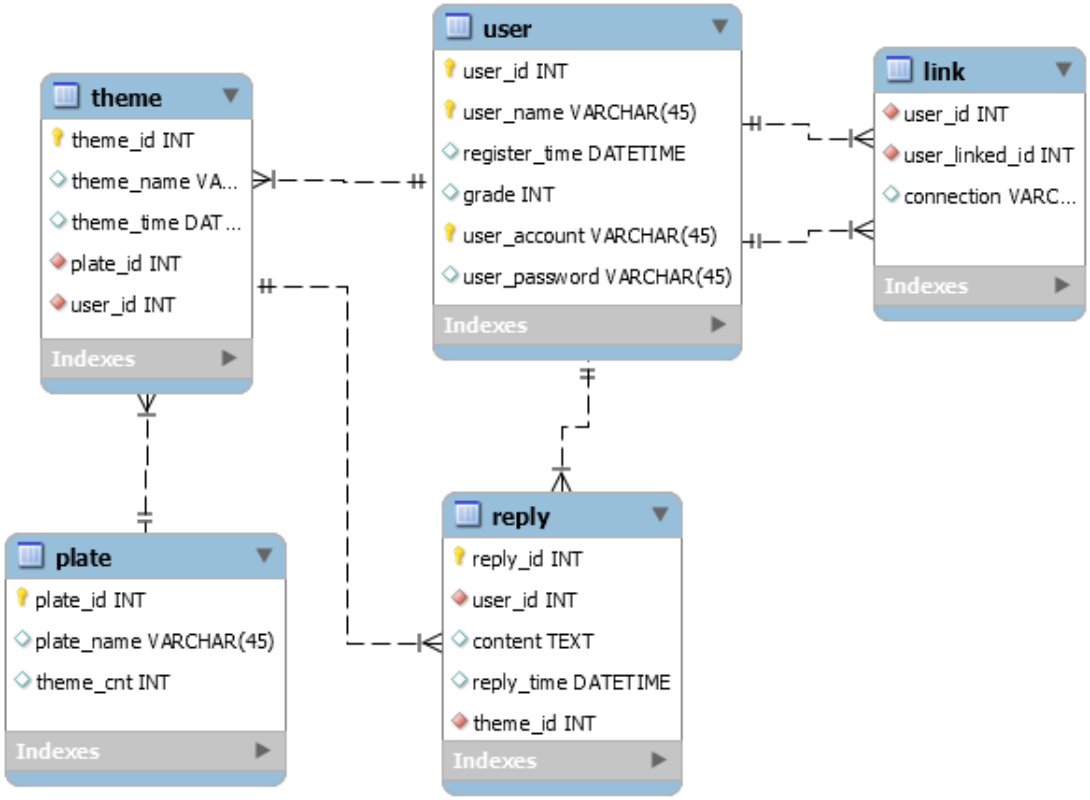

图 2: 简单论坛 MySQL 关系图设计

导出的数据见附带的 sql 文件。

3 数据创建

主要是创建板块、主题和回复，用户在登陆界面就注册生成数据了。

创建代码如下，主要是重复生成一些相同的数据，主要还是用来测试是否成功显示。

```
def generate_data():
    s = sql()

    # 生成版块
    s.insert_into('plate', ('1', '中山大学', 0))
    s.insert_into('plate', ('2', '数据库', 0))
    table_name = 'user'
    for i in range(9):
        s.insert_into(table_name, (i, 'name'+str(i), '2018-12-29
            00:05:03', i, 'acount'+str(i), 'password'+str(i)))

    # 生成主题(发帖)
    table_name = 'theme'
    for i in range(9):
        s.insert_into(table_name, (i, '主题'+str(i), '2018-12-29
            00:05:03', 1, str(i)))

    # 生成reply
    for i in range(9):
        for j in range(9):
            s.do_reply(i, "asdasdsadada", j)
```

4 结果展示

网页设计得比较简陋。具体如下：

首先进入首页，可以看到有 2 个版块，分别为中山大学和数据库。后面的数字显示了在该版块下有多少个主题。

图 3: 登陆界面

下面是注册账户的界面

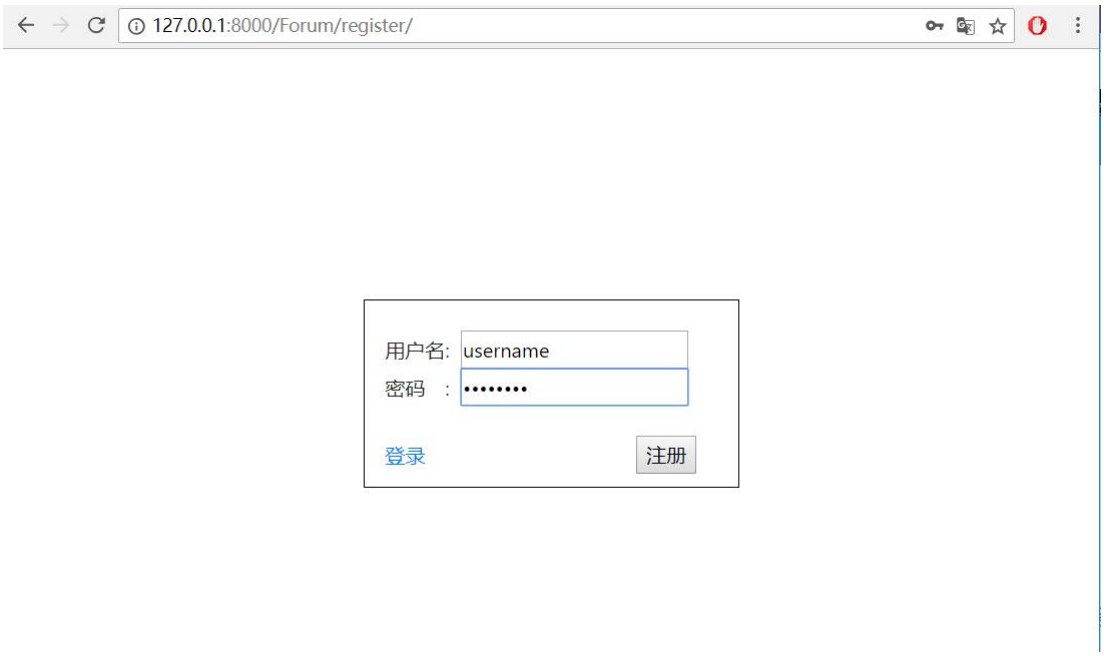

图 4: 注册界面

下面是注册成功的显示。

图 5: 注册成功

然后查看数据库的 user 表，可以看出确实生成了如下的元组，说明数据库成功更新了。

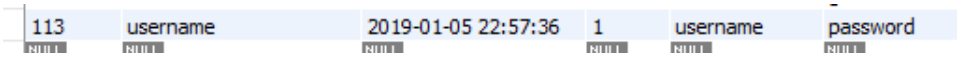

图 6: 数据库 user 更新

之后用刚刚注册的用户进行登录，如下图所示

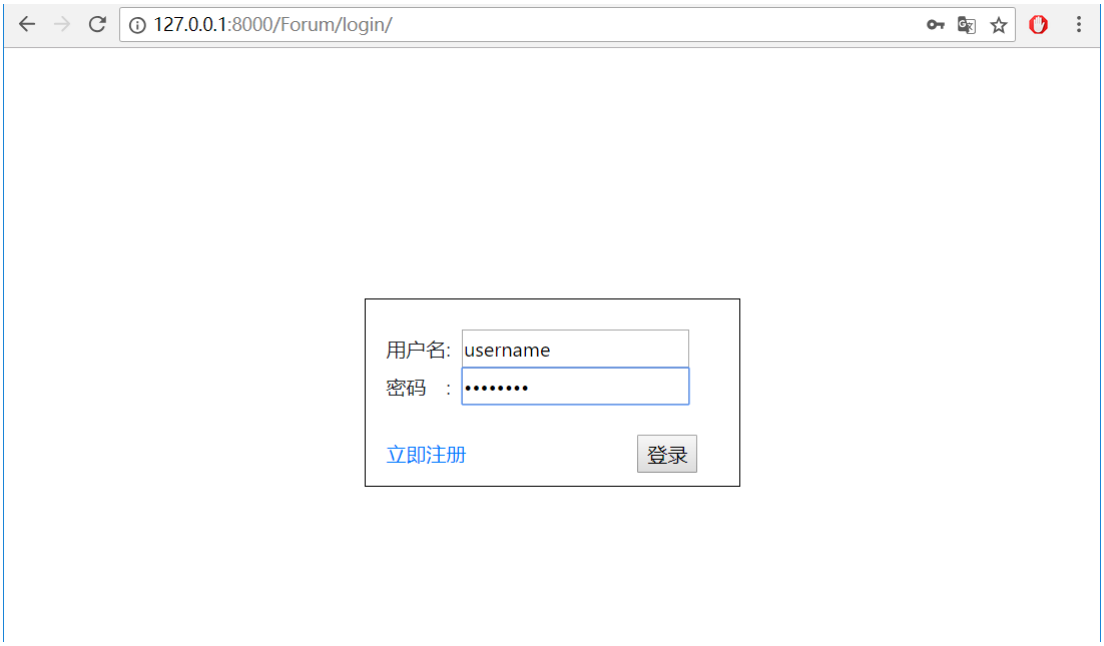

图 7: 用户登录

登录成功后，再次转移到首页，可以看到右上角已经能显示自己的用户名了。

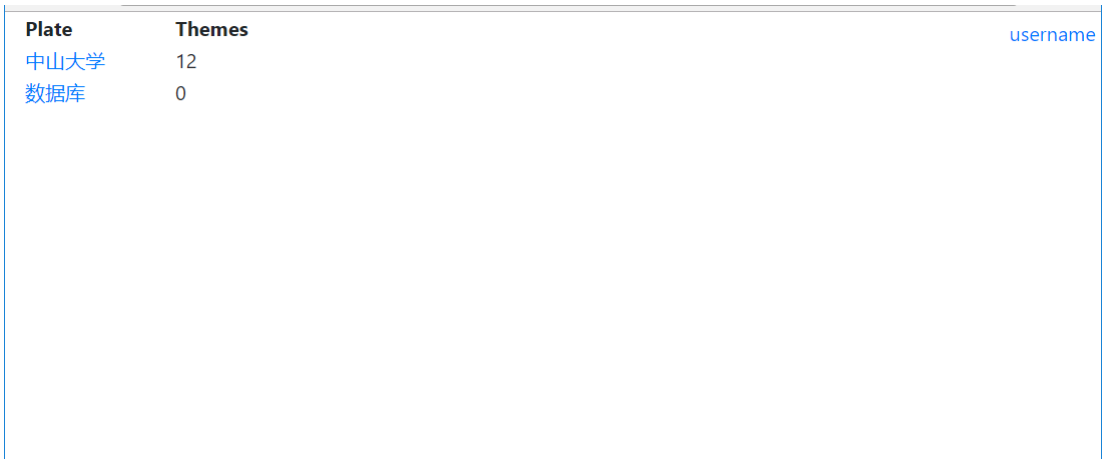

图 8: 登录后的首页

进入版块后，显示板块内的内容，上面为发表主题（帖子）的地方，下面为主题。

图 9: 板块内部内容

然后在中山大学版块进行发帖，主题如下所示。

图 10: 板块内部内容

发主题结果如下，可以看到主题成功被放到了最上面，右边显示了时间。

图 11: 主题内部界面

另外，点击用户名可以查看用户的个人信息，结果如下。这与下面的数据库的内容也是符合的。

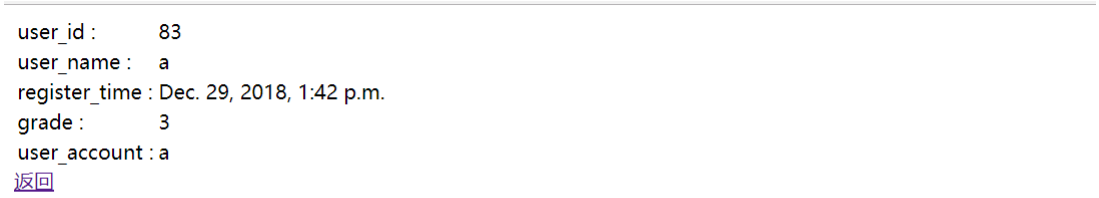

图 12: 用户个人信息界面

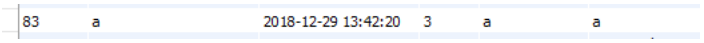

图 13: 该用户的数据库信息

进入别人发的主题后，可以进行回复操作，如下图所示。

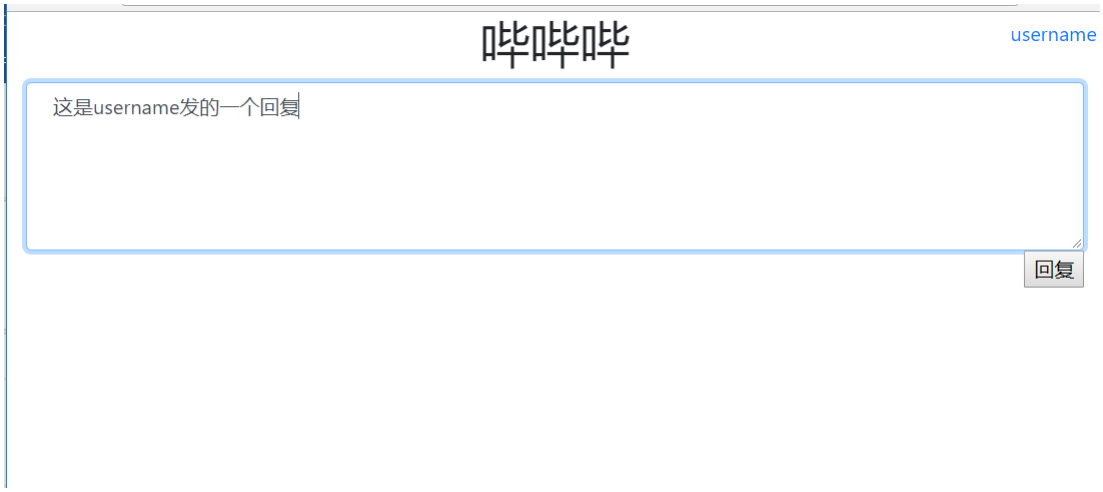

图 14: 该用户的数据库信息

结果如下，可以看到回复成功。

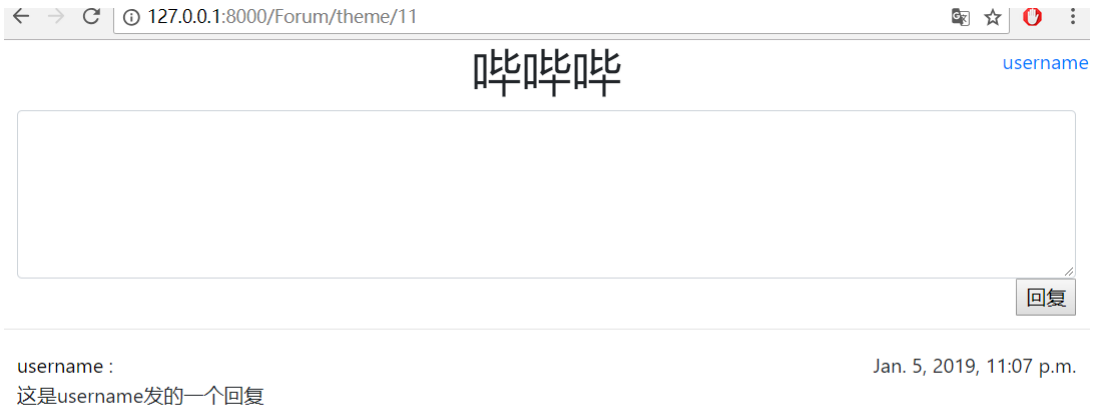

图 15: 该用户的数据库信息

5 代码

django 是 python 下的一个框架，因此想要连接数据库需要用相应的库。虽然 django 自带 model 可以连接数据库，但是为了分工方便，我们只是用了 pymysql 这个库来连接数据库和前端语言。我们设计了一个类，先做好最基本的增删查改功能，然后在这基础上给前端实现一些高级的接口来方便使用。具体如下。

```
class sql():
    def __init__(self):
        # self.conn = pymysql.connect(host="172.18.35.138",
            user="wuzy", password="519519519", db="forum")
        self.conn = pymysql.connect(host="localhost", user="root",
            password="", db="forum")

    def __del__(self):
        self.conn.close()

    def select_from(self, table_name, attribute='*', predicate=''):
        """获得table_name信息,返回元组，失败返回None"""
        with self.conn.cursor() as cursor:
            sql_lang = "SELECT " + str(attribute) + " FROM " +
                str(table_name) + " " + predicate
            try:
```

```
                cursor.execute(sql_lang)
                self.conn.commit()
                data = cursor.fetchall()
            except Exception as e:
                print('SELECT FAILED!\n Error message:', str(e))
                self.conn.rollback()
                return ()
        if data:
            return data
        return ()

    def insert_into(self, table_name, tuple):
        """插入表信息，成功返回True，失败返回False"""
        with self.conn.cursor() as cursor:
            sql_lang = "INSERT INTO " + table_name + " VALUES " +
                str(tuple)
            try:
                cursor.execute(sql_lang)
                self.conn.commit()
            except Exception as e:
                print('INSERT FAILED!\n Error message:', str(e))
                print(sql_lang)
                self.conn.rollback()
                return False
        return True

    def delete_from(self, table_name, predicate=''):
        """获得table_name信息,成功返回True,失败返回False"""
        with self.conn.cursor() as cursor:
            sql_lang = "DELETE FROM " + table_name + ' ' + predicate
            try:
                cursor.execute(sql_lang)
                self.conn.commit()
            except Exception as e:
                print('DELETE FAILED!\n Error message:', str(e))
```

```
                self.conn.rollback()
                return False
        return True

    def update(self, table_name, attribute, predicate=''):
        """更新表的内容，成功返回True，失败返回False"""
        with self.conn.cursor() as cursor:
            sql_lang = "UPDATE " + table_name + ' SET ' + attribute + ' '
                + str(predicate)
            try:
                cursor.execute(sql_lang)
                self.conn.commit()
            except Exception as e:
                print('UPDATE FAILED!\n Error message:', str(e))
                self.conn.rollback()
                return False
        return True

    def check_user(self, user_account):
        """检查用户是否在user表中"""
        data = self.select_from('user', '*', "where
            user_account='"+str(user_account)+"'")
        if len(data) > 0:
            return True
        return False

    def get_password_by_account(self, user_account):
        """检查用户是否在user表中"""
        data = self.select_from('user', '*', "where
            user_account='"+str(user_account)+"'")
        if len(data) > 0:
            return data[0][5]
        return ()

    def get_user_info(self, user_name):
```

```
        """获取用户的信息的字典，依次为id, name, register_time, grade,
            user_account"""
        data = self.select_from('user', '*', "where
            user_name='"+str(user_name)+ "'")
        dic = {}
        if len(data > 0):
            dic['id'] = data[0]
            dic['name'] = data[1]
            dic['register_time'] = data[2]
            dic['grade'] = data[3]
            dic['user_account'] = data[4]
        return dic

    def get_user_info_by_id(self, user_id):
        """获取用户的信息的字典，依次为id, name, register_time, grade,
            user_account"""
        data = self.select_from('user', '*', "where
            user_id='"+str(user_id)+ "'")
        dic = {}
        if len(data) > 0:
            dic['user_id'] = data[0][0]
            dic['user_name'] = data[0][1]
            dic['register_time'] = data[0][2]
            dic['grade'] = data[0][3]
            dic['user_account'] = data[0][4]
        return dic

    def get_linked_num(self, id):
        """获取被关注的数目"""
        data = self.select_from('link', '*', 'where
            user_linked_id='+str(id))
        return len(data)

    def get_all_theme(self, plate_id):
        """返回一个版块所有的主题dict的list，依次顺序为theme_id,
```

```
            theme_name, theme_time, plate_id, user_id"""
        data = self.select_from('theme', '*', 'where
            plate_id='+str(plate_id) + " order by theme_time desc ")
        res = []
        for i in range(len(data)):
            tmp_dict = {'theme_id':data[i][0],
                        'theme_name':data[i][1],
                        'theme_time':data[i][2],
                        'plate_id':data[i][3],
                        'user_id':data[i][4]}
            res.append(tmp_dict)
        return res

    def create_user(self, account, password, user_name):
        """创建用户，输入账号、密码、用户名即可"""
        data = self.select_from('user')
        num = len(data)
        time_str = self.get_time()
        return self.insert_into('user', (num, user_name, time_str, 0,
            account, password))

    def get_reply(self, theme_id):
        """获取指定theme_id下的所有回复，为字典的list，依次属性为reply_id,
            user_id, content, reply_time, theme_id"""
        data = self.select_from('reply', '*', 'where
            theme_id='+str(theme_id) + " order by reply_time desc ")
        res = []
        for i in range(len(data)):
            tmp_dic = {'reply_id':data[i][0],
                       'user_id':data[i][1],
                       'content':data[i][2],
                       'reply_time':data[i][3],
                       'theme_id':data[i][4]}
            res.append(tmp_dic)
        return res
```

```
    def do_reply(self, user_id, content, theme_id):
        """回复一个主题，传入用户id，内容，主题号"""
        data = self.select_from('reply')
        reply_id = len(data)
        datetime_str = self.get_time()
        print(data)
        print(reply_id)
        return self.insert_into('reply', (reply_id+1, user_id, content,
            datetime_str, theme_id))

    def raise_theme(self, theme_name, plate_id, user_id):
        """发起一个主题，传入主题名、版块id和用户id"""
        datetime_str = self.get_time()
        data = self.select_from('theme')
        theme_id = len(data)
        print(data)
        print(theme_id)
        return self.insert_into('theme', (theme_id, theme_name,
            datetime_str, plate_id, user_id))

    def get_user_id_by_account(self, user_account):
        """通过用户id获取用户的account"""
        data = self.select_from('user', '*', "where user_account='" +
            str(user_account)+"'")
        if len(data) > 0:
            return data[0][0]
        return None

    def get_time(self):
        """获取当前时间格式"""
        return datetime.datetime.now().strftime("%Y-%m-%d
            %H:%M:%S")
```

具体前端和数据库导出的 sql 文件见代码文件，因为这次实验的不是重点，就不再详细叙述了。

6 实验总结

在数据库的设计上，因为形式设计的相对简单，因此没有太多的难度。虽然在设计的时候考虑了非常多，但是最后还是决定用最简单的实体-联系集来表示，然后导出。但是如果论坛规模非常大的话，显然用上面简单的表来存所有内容是不行的。查找资料得知其实还有 NOSQL 可以用来实现这种大型的数据库。当然，因为之前没有学过，也没有现成的熟悉的工具，然后又要自学前端，因此就没有去用了。

总的来看的话，最大的障碍还是在网页的显示上，以及 python 在使用 sql 语言时候的引号问题。因为 sql 查询中如果有字符串（也就是需要单引号）的话，在 python 中也需要用单引号表示出来。而我们一直习惯用单引号来表示字符串，而 sql 语句也是通过字符串的形式传到数据库中执行的。这样就会导致 python 的引号和 sql 引号的问题。而且，python 的 tuple 在用 str 函数时，会自动给里面所有的字符串再加一个双引号，然而单单对一个字符串使用 str 这个函数就没有影响，因为这个问题，坑了我们非常久的时间。