数据库实验期末大作业

演示视频链接链接: https://pan.baidu.com/s/1lwfTt7uZsOIZxD5XLExphQ?pwd=8t6f

提取码: 8t6f

1、引言

（1）设计内容&设计目的

设计内容概述：

本次实验的选题是 健身房会员管理系统的设计与实现。需要实现会员信息管理、课程管理、预约管理等功能。其中：会员信息管理负责会员信息的添加、修改和查询；课程管理负责课程信息的添加、修改和查询；预约管理负责预约信息的录入、修改和查询。

我们参考了一些健身房会员管理的服务端和已有项目所实现的功能，在实际的设计过程中为上述三个功能分别构建了统一菜单栏下的三个页面，每个页面中都实现了添加、修改和查询的功能。这三个功能都有专门的数据库支撑，并通过前后端调用将网页操作和数据库操作联系起来。

实验目的：

本次实验旨在通过课程设计的选题，将数据库的设计和操作引入现实应用的场景，从而在关系数据库和相关应用程序的概念设计、逻辑设计、实现、操作和维护方面获得实践的经验。与此同时，实验涉及到前后端的设计和操作，这在本课程中并未涉及，需要适当地查询相关资料，并参考其他相关项目。在这个过程中能够提升自身调查研究、查阅技术文献、资料、手册以及编写技术文献的能力。

（2）设计环境：

前端：使用React框架完成

后端：使用Node.js中的Express框架完成

数据库：使用PostgreSQL，数据库管理工具使用pgAdmin4.

（3）同组人员&分工：

学号	姓名	分工
20319045	刘冠麟	代码编写
21307364	林梓博	实验报告编写

2、概要设计

（1）系统需求分析

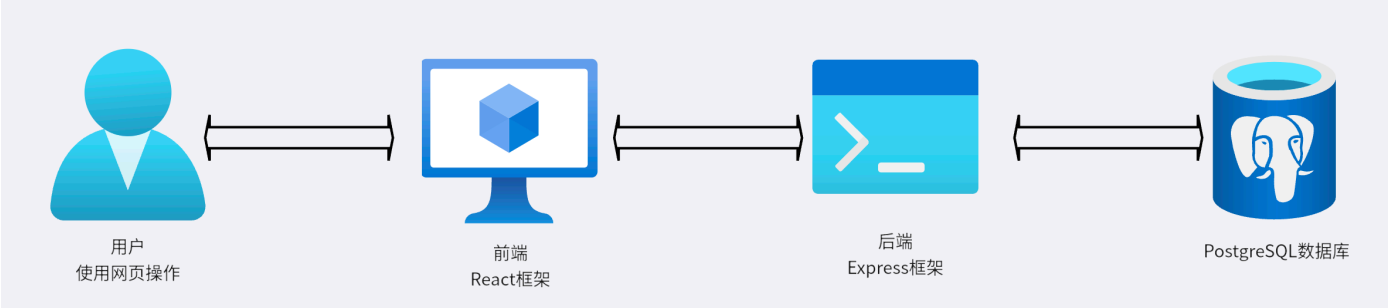
本次实验的目标是设计一个基于数据库的健身房会员管理系统。类比数据库实验课上一直涉及到的school数据库，可以参考school数据库和它对应的管理系统来设计健身房的数据库和相应的管理系统。

类比school中对**学生信息（Students）、教师信息（Teachers）、课程信息（Courses）和选课信息（Choices）**的管理，实验中设计的管理系统的功能分为四个部分，分别是**会员管理（Members）、教练管理（Trainers）、课程管理（Courses）和预约管理（Bookings）**，每一部分都包含对应记录的修改和查看功能，例如会员信息列表的查看和会员的添加、删除或修改。同时，四个部分之间存在一定的逻辑关联，其中课程管理和预约管理不是完全独立的。课程管理中存在一个选课人数的表项，需要记录预约管理中预约记录的个数，因此就需要在用户修改预约管理内容后对课程管理的相应表项做修改。

此外还需要考虑登录健身房会员管理系统的用户本身的管理，本次实验中将健身房管理员作为用户，并且管理员有多个，所以需要数据库有记录**管理员的相关信息**，并通过这些信息完成管理系统和数据库的登入。

（2）系统结构设计

系统结构设计如下：



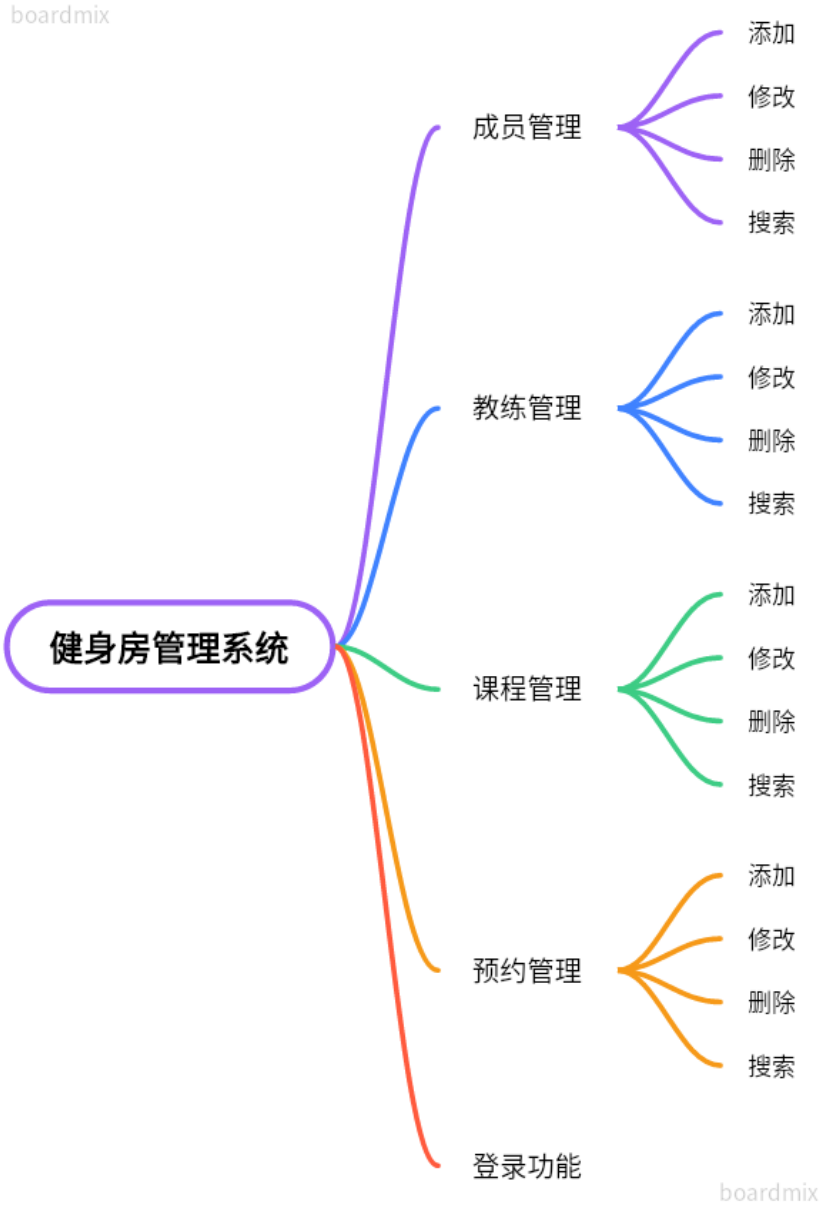
如上图所示，系统结构分为三个部分：

- 前端部分：该部分为管理系统最终展示给用户的部分，也是用户能直接操作的部分，该部分关注于用户界面和用户体验，将数据库的内容以便于用户使用与查看的方式呈现给用户，用户可通过与前端界面的交互来向后端发起请求，从而间接对数据库进行更改，而更改的内容又经由后端处理后返回给前端，前端再做出相应更改呈现给用户。
- 后端部分：这部分是整个系统的中流砥柱，也是直接更改操作数据库的部分，负责处理逻辑和数据处理。后端接收来自前端的请求，根据请求内容对数据库进行一定的更改与操作，完成后再发送更改后内容给前端。
- 数据库部分：为系统中存储数据的部分，负责数据的存储和检索。是最底层、核心的部分，前端中所有数据都来源于数据库。

本管理系统采用前端、后端、数据库分离结构，每一层都可以独立开发和维护，降低了代码复杂性，提高了开发效率。而由于每一层都是独立的，可以根据需要对单独一层进行扩展或优化，而不影响其他层，可以更灵活地应对业务增长与技术变化，同时每个部分的更新和维护可以独立进行，增强整体的安全性。

（3）功能模块设计

按照实验要求，本系统完成的功能模块如下图所示：



本系统完成了会员、教练、课程以及预约系统的添加、修改、删除和搜索功能的实现，另外还完成了登录功能。

3、详细设计

（1）系统数据库设计

针对会员管理、教练管理、课程管理和预约管理四个部分建立相应的数据库关系表。此外，数据库管理系统还需要添加管理员账户表，用于记录多个管理员的相关信息。以下是各个关系表的具体内容：

1、会员信息管理

会员表 (Members)

- 会员ID（主键）
- 姓名
- 联系方式（确保唯一性）

- 邮箱地址（确保唯一性）
- 地址
- 加入日期
- 会员等级
- 密码

2、课程管理

课程表 (Courses)

- 课程ID（主键）
- 课程名称
- 教练ID（外键，关联教练表）
- 开始时间
- 结束时间
- 选课人数
    - 选课人数与预约表之间有一个触发器，在对`Bookings`表进行插入或更新操作时会自动更新`Courses`表中的`Enrollments`字段

3、教练管理

教练表 (Trainers)

- 教练ID（主键）
- 教练姓名
- 专长
- 联系方式
- 密码

4、预约管理

预约表 (Bookings)

- 预约ID（主键）
- 会员ID（外键，关联会员表）
- 课程ID（外键，关联课程表）
- 状态

预约状态表 (BookingStatus)

- 状态ID（主键）
- 状态描述（例如：已确认、待确认、已取消）

5、管理员信息

管理员账户 (admin_table)

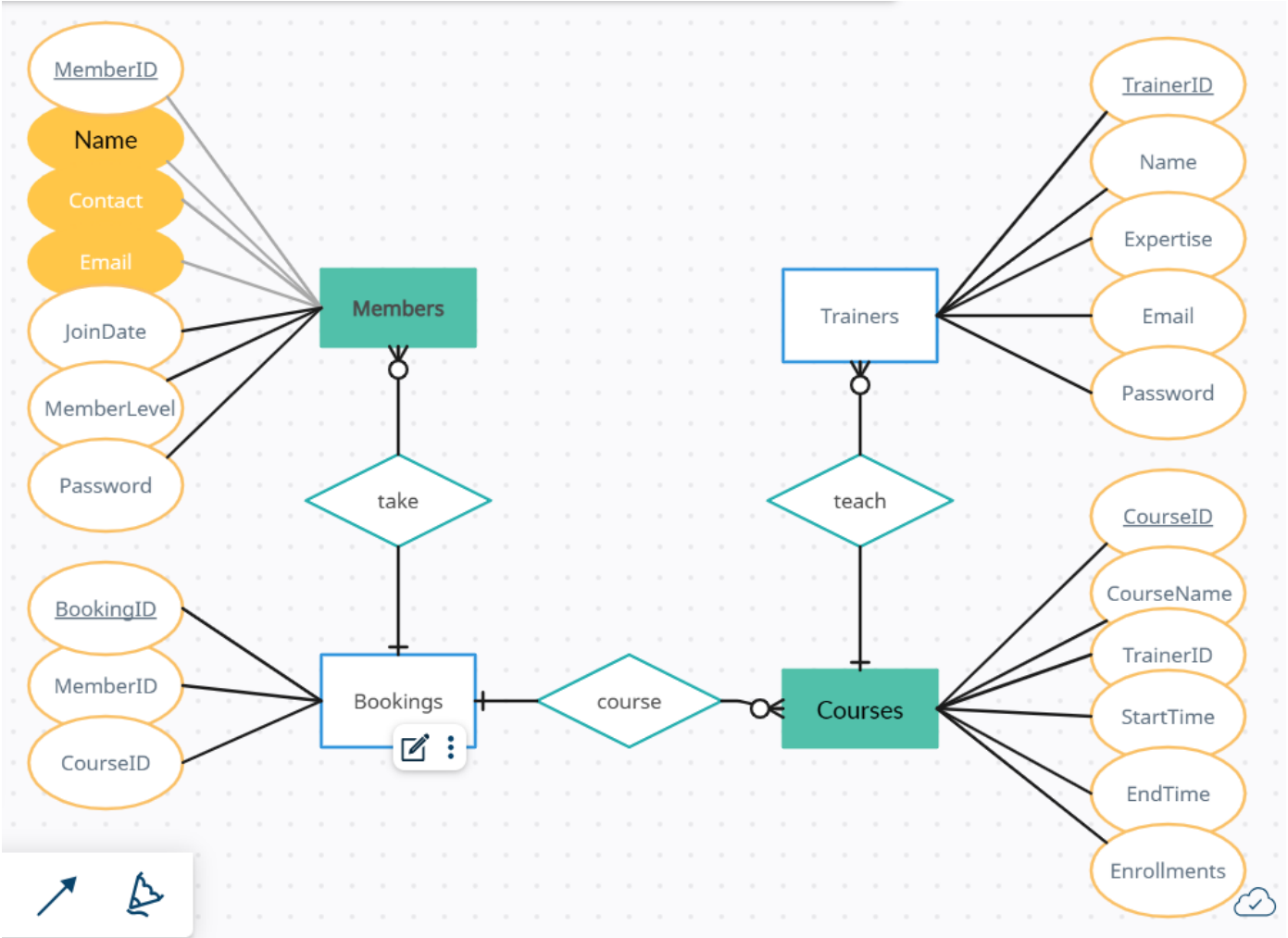
- 管理员ID（主键）
- 管理员姓名（确保唯一性）
- 密码（非空）

6、数据库范式分析

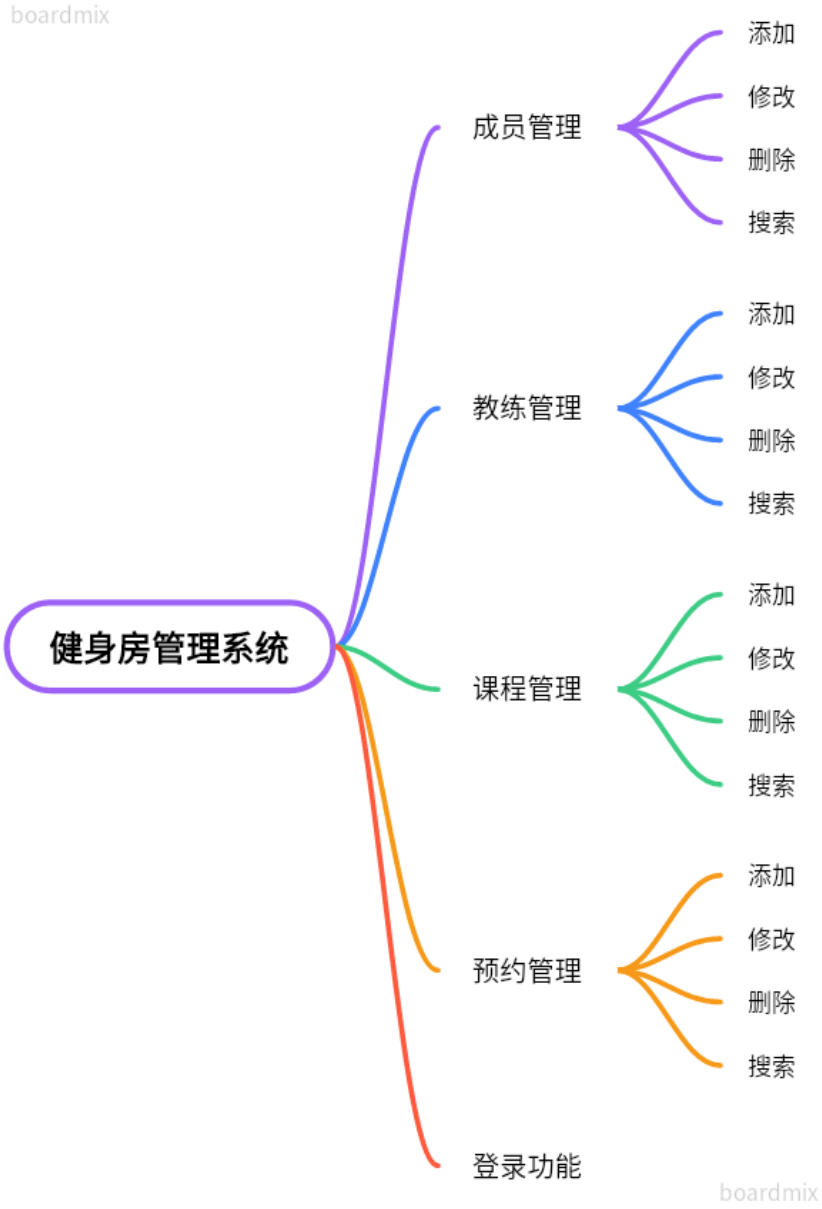
本实验中设计的数据库满足第三范式：

- 第一范式：所有关系表中的属性满足原子性
- 第二范式：所有关系表中仅有一个主键，也就是说候选键的集合中只有一个成员，所有实体也就只有完全函数依赖，二不存在部分函数依赖
- 第三范式：所有关系表中，每一个实体都被相应的主键所决定，可以看到非主键的信息都是由主键推断出来的（由一个实体的ID推断出一个实体的其他具体信息，如姓名、状态和密码等等），任意一个非主键值并不会依赖于另一个非主键值。所以各个关系表中没有传递函数依赖

7、数据库E-R图分析



（2）系统主要功能模块设计（可用流程图表示）



（3）各模块的主要算法对应的原代码

1、数据库创建代码：

```
-- 创建会员表
CREATE TABLE Members (
    MemberID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Contact VARCHAR(255) UNIQUE NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
    Address TEXT,
    JoinDate DATE NOT NULL,
    MemberLevel VARCHAR(100),
    Password VARCHAR(255) NOT NULL
);
```

```sql
-- 创建教练表
CREATE TABLE Trainers (
    TrainerID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Expertise TEXT,
    Email VARCHAR(255) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL
);
-- 创建课程表
CREATE TABLE Courses (
    CourseID SERIAL PRIMARY KEY,
    CourseName VARCHAR(255) NOT NULL,
    TrainerID INT REFERENCES Trainers(TrainerID),
    StartTime TIMESTAMP NOT NULL,
    EndTime TIMESTAMP NOT NULL,
    Enrollments INT NOT NULL,
);
-- 创建预约表
CREATE TABLE Bookings (
    BookingID SERIAL PRIMARY KEY,
    MemberID INT REFERENCES Members(MemberID),
    CourseID INT REFERENCES Courses(CourseID),
    Status VARCHAR(255) NOT NULL
);

-- 创建管理员账户
CREATE TABLE admin_table (
    admin_id SERIAL PRIMARY KEY,
    adminname VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL
);
```

2、后端

后端代码较多，由于各个模块中功能相似，这里只展示“会员管理”模块的后端代码，完整代码可见附件。

```javascript
// server.js
const express = require('express');  //导入express框架
const cors = require('cors');  //导入cors中间件
const bodyParser = require('body-parser');
const app = express();
app.use(cors());
app.use(bodyParser.json()); // 解析 JSON 请求体

const port = 3001;
const db = require('pg-promise')();

//连接数据库
const connection_gym = {
```

```
    host: 'localhost',
    port: 5432,
    database: 'gym',
    user: 'postgres',
    password: '915915'
};

const gym = db(connection_gym);

app.get('/gym/getMembers', async (req, res) => {
    try {
        // 获取查询参数
        const search = req.query.search;
        let query;
        let params;

        // 根据是否有搜索词来决定使用的 SQL 查询
        if (search) {
            // 使用 LIKE 操作符进行模糊匹配
            query = `
                SELECT * FROM Members
                WHERE
                    name ILIKE $1 OR
                    email ILIKE $1 OR
                    contact ILIKE $1
                ORDER BY memberid
            `;
            params = [`%${search}%`]; // 搜索词两边的 % 表示模糊匹配
        } else {
            query = 'SELECT * FROM Members ORDER BY memberid';
            params = [];
        }

        // 执行查询
        const members = await gym.any(query, params);
        res.json(members);
    } catch (error) {
        console.log(error);
        res.status(500).send('Internal Server Error');
    }
});

// 响应会员更新的API
app.post('/gym/updateMember', async (req, res) => {
    try {
        // 获取请求体中的数据
        const { memberid, name, contact, email, address ,password} = req.body;
        // 要在数据库中执行的查询语句
        const values = [name, contact, email, address, password, memberid];
```

```
        const query = 'UPDATE Members SET Name = $1, Contact = $2, Email = $3, Address = $4,Password = $5 WHERE memberid = $6 RETURNING *';

        // 执行查询
        const rows = await gym.any(query, values);
        // 如果有结果则返回第一个匹配的结果，否则返回404
        if (rows.length > 0) {
            res.status(200).json(rows[0]);
        } else {
            res.status(404).send('Member not found');
        }
    } catch (error) {
        console.error('Error while updating member:', error);
        res.status(500).send('Internal Server Error');
    }
});

// 响应会员删除的API
app.delete('/gym/deleteMember/:memberid', async (req, res) => {
    try {
        const { memberid } = req.params; // 获取会员ID
        // SQL查询语句
        const query = 'DELETE FROM Members WHERE memberid = $1 RETURNING *';
        // 执行查询
        const deletedMember = await gym.oneOrNone(query, [memberid]);

        if (deletedMember) {
            // 成功删除会员后的响应
            res.status(200).json(deletedMember);
        } else {
            // 如果没有找到会员或未能删除
            res.status(404).send('Member not found or could not be deleted');
        }
    } catch (error) {
        console.error('Error while deleting member:', error);
        res.status(500).send('Internal Server Error');
    }
});

// 添加新会员的API
app.post('/gym/addMember', async (req, res) => {
    try {
        const { name, contact, email, address, joinDate, memberLevel, password } = req.body;
        // SQL查询语句
        const query = 'INSERT INTO Members (Name, Contact, Email, Address, JoinDate, MemberLevel, Password) VALUES ($1, $2, $3, $4, $5, $6 ,$7) RETURNING *';
```

```
        const values = [name, contact, email, address, joinDate, memberLevel, password];

        // 执行查询
        const newMember = await gym.one(query, values);

        // 返回添加的新会员信息
        res.status(201).json(newMember);
    } catch (error) {
        console.error('Error while adding new member:', error);
        res.status(500).send('Internal Server Error');
    }
});
```

前端

前端代码较为冗长，分为若干个文件夹和十数个代码文件，且不是本课程主要关注的内容，所以这里仅展示页面入口app.js的代码，完整代码可见附件。

前端主要应用了React框架中的Router模块，以及useState、useEffect的使用，还有异步对后端fetch数据。

```
import React ,{useState} from 'react';
import { BrowserRouter as Router, Routes, Route, Link, Navigate } from 'react-router-dom';
import { HomeOutlined, LaptopOutlined, NotificationOutlined, UserOutlined, FormOutlined , SolutionOutlined, LinkOutlined, UserSwitchOutlined} from '@ant-design/icons';
import { Breadcrumb, Layout, Menu, theme } from 'antd';
import Home from './pages/Home/Home.js';
import './App.css';
import Members from './pages/Member/Members.js';
import MembersAdd from './pages/Member/MembersAdd.js';
import Trainers from './pages/Trainer/Trainers.js';
import TrainersAdd from './pages/Trainer/TrainersAdd.js';
import Courses from './pages/Course/Courses.js';
import CoursesAdd from './pages/Course/CoursesAdd.js';
import Bookings from './pages/Booking/Bookings.js';
import BookingsAdd from "./pages/Booking/BookingsAdd.js";
import Login from "./pages/Login/Login.js";
import { useAuth } from './pages/Login/AuthContext';


const { Header: AntHeader, Content, Sider, Footer } = Layout;
const header_item = [
    {
        key: 'home',
        icon: React.createElement(HomeOutlined),
        label: <Link to="/">主页</Link>,
    },
```

```
    {
        key: 'management',
        icon: React.createElement(FormOutlined),
        label: <Link to="/members">管理</Link>,
    },
]
const sider_item = [
    {
        key: 'members',
        icon: React.createElement(SolutionOutlined),
        label: '会员管理',
        children: [
            {
                key: 'membersList',
                label: <Link to="/members">会员信息列表</Link>,
            },
            {
                key: 'membersAdd',
                label: <Link to="/membersadd">会员添加</Link>,
            },
        ],
    },
    {
        key: 'trainers',
        icon: React.createElement(UserSwitchOutlined),
        label: '教练管理',
        children: [
            {
                key: 'trainersList',
                label: <Link to="/trainers">教练信息列表</Link>,
            },
            {
                key: 'trainersAdd',
                label: <Link to="/trainersadd">教练添加</Link>,
            },
        ],
    },
    {
        key: 'Courses',
        icon: React.createElement(LaptopOutlined),
        label: '课程管理',
        children: [
            {
                key: 'coursesList',
                label: <Link to='/courses'>课程信息列表</Link>,
            },
            {
                key: 'coursesAdd',
                label: <Link to='/coursesadd'>课程添加</Link>,
```

```
            }
        ],
    },
    {
        key: 'appointment',
        icon: React.createElement(NotificationOutlined),
        label: '预约管理',
        children: [
            {
                key: 'appointmentAdd',
                label: <Link to='/bookingsadd'>预约录入</Link>,
            },
            {
                key: 'appointmentList',
                label: <Link to='/bookings'>预约信息列表</Link>,
            }
        ],
    },
];


const App = () => {
    const { isAuthenticated } = useAuth();

    const { colorBgContainer, borderRadiusLG } = theme.useToken();

    return (
            <Router>
                {!isAuthenticated ? (
                    // 未认证用户只能看到登录页
                    <Routes>
                        <Route path="/login" element={<Login />} />
                        <Route path="*" element={<Navigate to="/login" />} />
                    </Routes>
                ) : (
                    <Layout>
                        <AntHeader
                            style={{
                                display: 'flex',
                                alignItems: 'center',
                            }}
                        >
                            <div className="demo-logo" />
                            <Menu
                                theme="dark"
                                mode="horizontal"
                                defaultSelectedKeys={['1']}
                                items={header_item}
                                style={{
```

```
                                    flex: 1,
                                    minWidth: 0,
                                }}
                            />
                            <img style="margin-right: 20px;" src="/sysu_logo.jpg" alt="sysu_logo" style={{width: '130px', height: '50px'}}/>
                            <h3 style={{color: 'whitesmoke'}}><pre>  SYSU健身房预约管理系统</pre></h3>
                        </AntHeader>
                        <Layout>
                        <Sider
                                breakpoint="lg"
                                width={200}
                                style={{
                                    background: colorBgContainer,
                                }}
                            >
                                <Menu
                                    mode="inline"
                                    defaultSelectedKeys={['1']}
                                    defaultOpenKeys={['sub1']}
                                    style={{
                                        height: '800px',
                                        borderRight: 0,
                                    }}
                                    items={sider_item}
                                />
                            </Sider>
                            <Layout
                                style={{
                                    padding: '20px 24px 24px',
                                }}
                            >
                                <Content
                                    style={{
                                        padding: 24,
                                        margin: 0,
                                        minHeight: "600px",
                                        background: colorBgContainer,
                                        borderRadius: borderRadiusLG,
                                    }}
                                >
                                    <Routes>
                                        <Route path="/" element={<Home />} />
                                        <Route path="/members" element={<Members />} />
                                        <Route path="/membersadd" element={<MembersAdd />} />
```

```
                                        <Route path="/trainers" element=
{<Trainers />} />
                                        <Route path="/trainersadd" element=
{<TrainersAdd />} />
                                        <Route path="/courses" element={<Courses
/>} />
                                        <Route path="/coursesadd" element=
{<CoursesAdd />} />
                                        <Route path="/bookings" element=
{<Bookings />} />
                                        <Route path="/bookingsadd" element=
{<BookingsAdd />} />
                                    </Routes>
                                </Content>

                                <Footer
                                    style={{
                                        textAlign: 'center',
                                    }}
                                >
                                    LGL Design ©2023 Created by LiuGuanLin
                                </Footer>
                            </Layout>
                        </Layout>
                    </Layout>
                )}
            </Router>
    );
};

export default App;
```

4、调试与运行结果及存在的主要问题

（1）主要问题

在编写数据库的时候我们曾经遇到过的一个问题：课程表中的选课人数和预约表中的条目是有直接关联的，也就是说每个课程的选课人数并不是独立的，而是由预约表中相应的预约该课程的条目数量所决定的。但是预约表中增加和删除对应预约条目的时候并不能伴随地更改对应课程表中课程选课人数。针对这一问题我们两个小组成员讨论了很久，在前端、后端或者数据库上更改设置都设想过，最后决定从底层数据库入手。

（2）解决方案和解决效果

我们的解决方案是在数据库中增加一个触发器，在对`Bookings`表进行插入或更新操作后自动更新`Courses`表中的`Enrollments`字段。也就是说，当`bookings`进行插入或者更新操作后，触发器会触发并执行函数，更新`Courses`中`Enrollments`的字段，其值为为`Bookings`表中具有相同`CourseID`且`Status`为`Confirmed`的行的数量。

代码如下：

```
DROP TRIGGER IF EXISTS update_enrollment_trigger ON Bookings;

CREATE OR REPLACE FUNCTION update_enrollment()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Courses
    SET Enrollments = (SELECT COUNT(*)
                       FROM Bookings
                       WHERE CourseID = NEW.CourseID AND Status = 'Confirmed')
    WHERE CourseID = NEW.CourseID;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_enrollment_trigger
AFTER INSERT OR UPDATE ON Bookings
FOR EACH ROW
EXECUTE FUNCTION update_enrollment();

```

解决效果：当预约表进行更改时，选课表中的选课人数也会相应变化。（解决效果可见视频演示）

5、课程设计小结

（1）工作内容总结

不同于之前的数据库实验的内容，本次实验的内容并不局限于SQL语句的使用和数据库的建立、修改以及管理，还涉及到了数据库的外部调用，包含了前后端开发的内容。其中前端需要实现网页的基础框架和渲染，将来自数据库的相关信息做适当处理后在服务器网页上直观清晰，用户友好地展示出来，同时要接收用户在网页上的操作，将响应向后端传输；后端部分需要实现对来自前端的响应的处理，针对用户不同操作带来的响应正确地调用连接的数据库，控制数据库完成查询、插入、修改和删除等操作，并将数据库更改的信息反馈到前端，从而实现网页信息和数据库信息的一致性，让数据库的架构决定网页功能，也就通过数据库实现了相关的管理系统。

实际操作过程中，本次实验有一部分时间要用于前后端的开发，这和以往数据库部分的内容有较大差别，联系也相对较少。首先是javascrip语言，由于之前没有接触过，就需要一定的时间先了解语法和代码基本框架，再尝试前后端部分的编写；其次是网页部分的渲染，需要css和html部分的语言，涉及到大量的代码细节，完全从零开始实现相对困难。因此，前后端部分适当的参考了已有的基于数据库的管理系统项目，在此过程中一方面学习了其他数据库项目的涉及经验，也从它们的代码中更快速地掌握了语法和代码基本框架。

本次实验选取了健身房会员管理系统的应用场景，能够结合当下的市场环境。同时，会员管理系统中的基本框架和许多其他的应用场景有较高的相似度，例如学校的学生教师和授课管理系统，医院的病人医生和挂号管理系统等等。通过本次实验对健身房会员管理的实现，应当能够在此基础上更快，更完善地实现其他类似场景下的管理系统。

（2）系统可改进的地方

- 由于时间紧迫且不是本门课重点，数据库的前端页面设计地较为简陋；
- 网站的账户功能还待完善，仅完成了登录功能，没有完成管理员账户的修改等。

6、参考文献

无

附录：网站前、后端本地启用配置指南

如果老师您打算在电脑本地使用代码启用网站的前、后端：

1. 首先需要去Node.js官网下载Node.js的JavaScript 运行环境
2. 安装后确保命令行能使用`npm`指令
3. 解压前、后端代码文件
4. 进入前端代码文件，在根目录下执行`npm install`，`npm`会根据文件下的package.json文件安装对应的包和依赖
5. 后端同理
6. 安装完后先在后端文件夹下命令行执行`npm start`启动后端服务器
7. 再在前端文件夹下命令行执行`npm start`启动前端网页