

- 数据库大作业
 - 组员分工
 - 应用需求介绍
 - 应用系统设计
 - 开发工具
 - 功能设计
 - 数据库设计
 - 数据来源
 - 前后端交互
 - 1.安装Django和创建项目
 - 2.Django后端与数据匹配
 - 3.运行服务器
 - 前端设计
 - 主要运行界面
 - 实验总结

数据库大作业

组员分工

林宇浩：Django项目搭建、网页home界面搭建、评论展示功能 王哲：网页的增添、删除、搜索、编辑歌曲功能的实现、绘制E-R图 熊蔚然：获取数据、处理原始数据、初步确定关系表结构、后端搭建

应用需求介绍

随着互联网的快速发展，音乐行业进入了数字化时代，音乐数据的获取、存储和分析变得越来越重要。为了更好地理解和利用音乐数据，我们决定开发一个可以展示歌曲信息的应用系统。该系统基于openGauss实现，能够展示歌曲的详细信息，包括歌曲名、歌手、专辑、歌词、评论。系统包含了五个数据库表，并搭建了Web页面，实现了一个可运行的展示歌曲信息的本地网站。

应用系统设计

开发工具

- openGauss
- data studio
- python
- HTML,CSS,JavaScript
- Django

功能设计

我们的数据库应用系统有以下几项功能：

- 歌曲展示： 提供一个主页，显示随机选择的歌曲列表，最多展示**50**首歌曲。用户可以浏览歌曲、查看艺术家信息和专辑封面。
- 歌曲搜索： 提供搜索功能，用户可以根据歌曲名称、艺术家或专辑搜索歌曲。搜索结果将以列表形式展示。
- 评论查看： 用户可以点击歌曲，查看对应的评论。评论页面显示每首歌曲的前三条评论。
- 歌曲信息编辑： 管理员用户可以通过特定的界面添加、删除和编辑歌曲信息，包括歌曲名称、艺术家、专辑、发布时间等。

数据库设计

数据库系统中用到的所有关系表如下：

1. Song（歌曲表）：

- **id**：作为主键，用于唯一标识每一首歌曲。
- **name**：存储歌曲的名称，以字符串形式保存，最大长度为**150**，允许为空，以应对数据缺失的情况。
- **ar**：使用外键关联到艺术家表（**Artist**），表示歌曲对应的艺术家。通过级联删除保证数据的一致性。
- **al**：使用外键关联到专辑表（**Album**），表示歌曲所属的专辑。通过级联删除保证数据的一致性。
- **no**：存储歌曲的编号，以整数形式保存，允许为空。用于表示歌曲在专辑中的顺序。
- **version**：存储歌曲的版本号，以整数形式保存，允许为空。用于区分不同版本的歌曲。
- **publish_time**：存储歌曲的发布时间，以字符串形式保存，最大长度为**20**，允许为空。以字符串形式存储时间可以灵活处理不同的时间格式。

```
CREATE TABLE IF NOT EXISTS SONG
  (ID INT PRIMARY KEY NOT NULL,
   NAME VARCHAR(150),
   AR_ID INT NOT NULL,
   AL_ID INT NOT NULL,
   NO INT,
   VERSION INT,
   publishTime VARCHAR(20));
```

2. Album（专辑表）：

- **id**：作为主键，用于唯一标识每个专辑。
- **name**：存储专辑的名称，以字符串形式保存，最大长度为100，允许为空。
- **picurl**：存储专辑封面的URL，以字符串形式保存，最大长度为100，允许为空。通过URL保存封面图的路径，方便前端展示。

```
CREATE TABLE IF NOT EXISTS ALBUM
  (ID INT PRIMARY KEY NOT NULL,
   NAME VARCHAR(100),
   PICURL VARCHAR(100));
```

3. Artist（艺术家表）：

- **id**：作为主键，用于唯一标识每个艺术家。
- **name**：存储艺术家的名称，以字符串形式保存，最大长度为50。

```
CREATE TABLE IF NOT EXISTS ARTIST
  (ID INT PRIMARY KEY NOT NULL,
   NAME VARCHAR(50));
```

4. Comment（评论表）：

- **commentid**：作为主键，用于唯一标识每条评论，使用长整数类型。
- **content**：存储评论内容，以字符串形式保存，最大长度为400，允许为空。可以存储评论的具体内容。
- **commenttime**：存储评论时间，以字符串形式保存，最大长度为30，允许为空。以字符串形式存储时间可以灵活处理不同的时间格式。
- **user**：使用外键关联到用户表（**Users**），表示评论的用户。通过级联删除保证数据的一致性。
- **song**：使用外键关联到歌曲表（**Song**），表示评论的歌曲。通过级联删除保证数据的一致性。

```
CREATE TABLE IF NOT EXISTS COMMENT
(COMMENTID BIGINT PRIMARY KEY NOT NULL,
CONTENT VARCHAR(400),
COMMENTTIME VARCHAR(30),
USERID BIGINT NOT NULL,
SONGID INT NOT NULL)
```

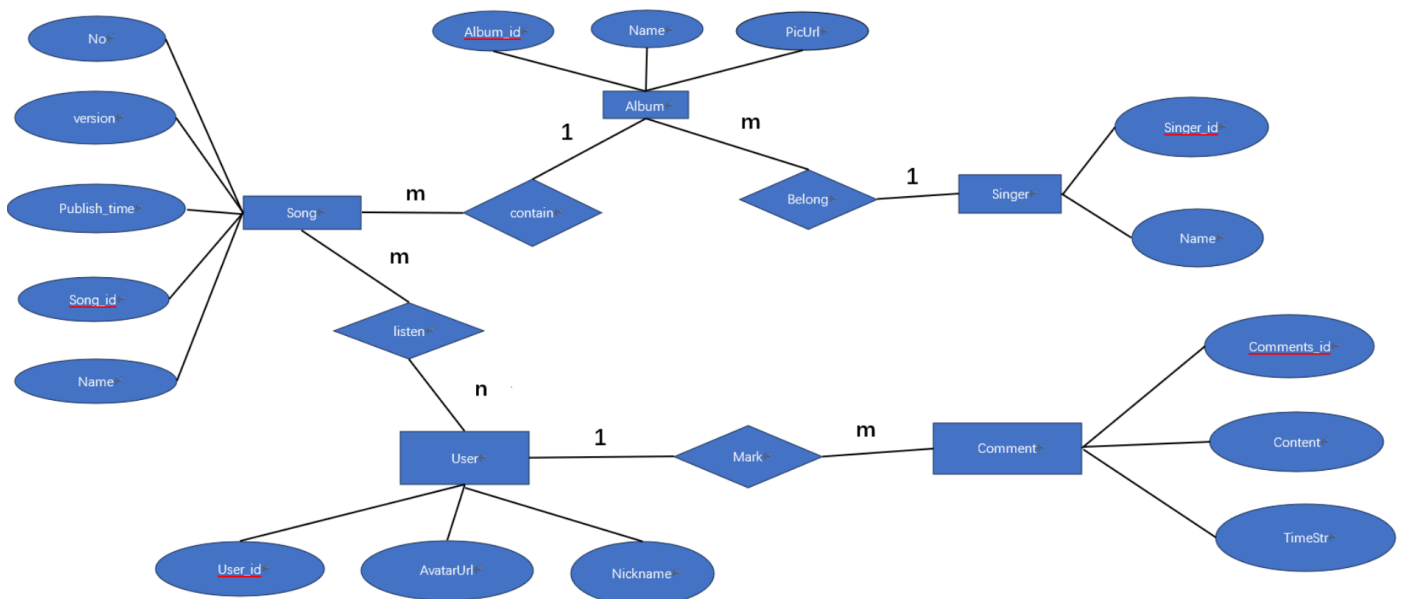
5. Users（用户表）：

- **id**：作为主键，用于唯一标识每个用户，使用长整数类型。
- **username**：存储用户的用户名，以字符串形式保存，最大长度为50。
- **avatarurl**：存储用户头像的URL，以字符串形式保存，最大长度为100，允许为空。通过URL保存头像图的路径，方便前端展示。

```
CREATE TABLE IF NOT EXISTS USERS
(ID BIGINT PRIMARY KEY NOT NULL,
USERNAME VARCHAR(50),
AVATARURL VARCHAR(100))
```

设计中使用了外键关联，通过外键建立了歌曲、艺术家、专辑、评论和用户之间的关系，保证了数据之间的一致性和完整性。每个表都有一个主键用于唯一标识表中的每个记录。字段的类型和长度根据存储的数据特性进行了选择，同时考虑到数据缺失的情况，允许部分字段为空。

数据库的E-R图如下：



数据来源

实验中用到的数据来源于在网上发现的一个可用的网易云音乐api，可通过这个api获取到网易云音乐中的歌曲的各项数据。批量获取实验数据的代码如下：

```
import http.client # 导入用于创建HTTP连接的模块
import json # 导入处理JSON数据的模块

# 歌曲ID列表
song_ids = [str(i).zfill(8) for i in range(20000000, 20050000)] # 生成一系列歌曲ID, 你可以添加更多

# 创建HTTPS连接
conn = http.client.HTTPSConnection("music.163.com")

payload = '' # 定义空的负载 (通常用于POST请求, 这里是GET请求)
# 设置通用的请求头
headers = {
    'User-Agent': 'Apifox/1.0.0 (https://apifox.com)' # 设置用户代理信息
}

# 遍历每首歌曲
with open("song_data.json", "a", encoding="utf-8") as output_file: # 以追加模式打开文件, 用于将歌曲信息写入
    for song_id in song_ids:
        # 构建API请求路径, 包括歌曲ID作为查询参数
        api_path = f"/api/v3/song/detail?id={song_id}&c=%5B%7B%22id%22:%22{song_id}%22%7D%5D"

        # 发起GET请求
        conn.request("GET", api_path, payload, headers=headers)

        # 获取响应
        res = conn.getresponse()

        # 读取响应数据
        data = res.read()
        parsed_data = json.loads(data.decode("utf-8"))

        # 检查是否存在歌曲信息且第一首歌曲有"name"字段
        if "songs" in parsed_data and parsed_data["songs"] and "name" in parsed_data["songs"][0] and parsed_data["songs"][0]["name"]:
            # 格式化输出JSON数据, indent参数用于设置缩进空格数
            formatted_data = json.dumps(parsed_data, indent=4, ensure_ascii=False)

            # 将信息写入文件
            output_file.write(formatted_data + "\n")

# 关闭连接
conn.close()
```

程序的思路就是使用HTTP连接模块建立到网易云音乐API的安全连接，然后设置通用的HTTP请求头，包括用户代理信息。对需要获取的歌曲ID，构建对应的API请求路径。再

使用GET请求向网易云音乐API请求歌曲详细信息，读取API的响应数据并将响应数据解析为JSON格式。如果歌曲信息有效，将格式化的JSON数据写入对应的json文件中。

以上是获取api中歌曲信息的方法，我们还通过同样的方法获取了与每首歌曲对应的评论信息，由于一首歌的评论数量可能会很多，这里只保留每首歌的三条评论。

```
song_ids = [str(i).zfill(8) for i in range(20000000, 20050000)] # 你可以添加更多歌曲ID
# 创建HTTPS连接
conn = http.client.HTTPSConnection("music.163.com")
payload = ''
headers = {
    'User-Agent': 'Apifox/1.0.0 (https://apifox.com)'
}
# 遍历每首歌曲
with open("comments_data.json", "a", encoding="utf-8") as output_file:
    for song_id in song_ids:
        # 构建API请求路径，包括歌曲ID作为查询参数
        api_path = f"/api/comment/resource/comments/get?
rid=R_SO_4_{song_id}&threadId=R_SO_4_{song_id}"

        # 发起GET请求
        conn.request("GET", api_path, payload, headers=headers)

        # 获取响应
        res = conn.getresponse()

        # 读取响应数据
        data = res.read()
        parsed_data = json.loads(data.decode("utf-8"))

        # 检查 "data" 中的 "comments" 是否存在且不为空
        if "data" in parsed_data and "comments" in parsed_data["data"] and
parsed_data["data"]["comments"]:
            # 只保留前三条评论
            parsed_data["data"]["comments"] = parsed_data["data"]["comments"][:3]

            # 将song_id添加到parsed_data中
            parsed_data["id"] = song_id

            # 使用json.dumps格式化输出，indent参数用于设置缩进空格数
            formatted_data = json.dumps(parsed_data, indent=4, ensure_ascii=False)

            # 将信息写入文件
            output_file.write(formatted_data + "\n")

# 关闭连接
conn.close()
```

直接通过api得到的json数据有很多繁琐的信息，所以要从这些数据中提取出我们的数据库表需要的部分，便于将数据插入数据库。将'song_data_2.json' 文件中的歌曲数据进行

处理，提取song表中的特定属性，并为每首歌曲生成一个新的JSON文件：

```
import json

# 打开JSON文件
with open('song_data_2.json', 'r', encoding='utf-8') as file:
    # 从文件中加载JSON数据
    data = json.load(file)

# 现在，变量"data"包含了JSON文件中的数据
# print(len(data["list"]))
# print(data["list"][2014])

def extract_properties(song_data):
    # 从歌曲数据中提取所需的属性
    extracted_data = {
        'name': song_data['name'],
        'id': song_data['id'],
        'ar_id': song_data['ar'][0]['id'] if song_data['ar'] else None,
        'al_id': song_data['al']['id'] if song_data['al'] else None,
        'no': song_data['no'],
        'version': song_data['version'],
        'publishTime': song_data['publishTime']
    }
    return extracted_data

def process_data_list(data_list):
    # 为每个数据提取属性并保存为新的JSON文件
    for i, song_data in enumerate(data_list):
        extracted_data = extract_properties(song_data['songs'][0])

        # 生成新的JSON文件名
        output_filename = f"output_{i + 1}.json"

        # 保存提取后的数据到新的JSON文件
        with open(output_filename, 'w', encoding='utf-8') as output_file:
            json.dump(extracted_data, output_file, indent=4, ensure_ascii=False)

# 处理数据列表
process_data_list(data["list"])
```

提取其他表中的数据的步骤也与以上相同。将每个表项单独放到一个json文件后，向数据库的对应表中插入数据就可以非常容易地用python中的psycopg2模块实现。定义插入数据到数据库表的SQL语句，使用%s作为参数的占位符。连接到数据库后，创建一个游标对象用于执行SQL语句。首先用SQL语句创建所需的表，之后使用循环逐个处理JSON文件，使用执行插入SQL语句的游标，将提取的数据插入到数据库表中。在循环完成后，提交数据库事务，确保所有插入操作都生效。如果在处理过程中发生任何错误，程序将捕获异常，打印错误消息，并回滚数据库事务，以确保数据的一致性。

```

import os
import json
import psycopg2
import datetime

# 数据库连接信息
db_params = {
    'dbname': 'Music',
    'user': 'xiong',
    'password': 'xionG1003.',
    'host': '192.168.89.66',
    'port': '26000'
}

# 获取当前路径下所有以 "output" 开头的 .json 文件
json_files = [file for file in 'song_json' if file.startswith('output') and
file.endswith('.json')]

# SQL 插入语句
insert_sql = """
INSERT INTO SONG (id, name, ar_id, al_id, no, version, publishTime)
VALUES (%s, %s, %s, %s, %s, %s, %s);
"""

# 连接到数据库
connection = psycopg2.connect(**db_params)
cursor = connection.cursor()

# 创建表
cursor.execute('''CREATE TABLE SONG
    (ID INT PRIMARY KEY      NOT NULL,
    NAME          VARCHAR(150) ,
    AR_ID          INT        NOT NULL,
    AL_ID          INT        NOT NULL,
    NO             INT,
    VERSION        INT,
    publishTime    VARCHAR(20));''')

try:
    # 逐个处理 JSON 文件
    for json_file in json_files:
        with open(json_file, 'r', encoding='utf-8') as file:
            data = json.load(file)
            publish_time_seconds = data['publishTime']/ 1000

            if(publish_time_seconds<0):
                formatted_time='null'
            else:
                # 使用 datetime 模块转换为正常时间格式
                publish_time = datetime.datetime.fromtimestamp(publish_time_seconds)

                # 格式化为字符串 (以年-月-日 时:分:秒 的格式)
                formatted_time = publish_time.strftime("%Y-%m-%d")

        # 使用 datetime 模块转换为正常时间格式

```



```
# 提取 JSON 数据并插入数据库
extracted_data = (
    data['id'],
    data['name'],
    data['ar_id'],
    data['al_id'],
    data['no'],
    data['version'],
    formatted_time
)
print(json_file)
cursor.execute(insert_sql, extracted_data)

# 提交事务
connection.commit()

except Exception as e:
    print(f"Error: {e}")
    # 如果出现错误，回滚事务
    connection.rollback()

finally:
    # 关闭连接
    cursor.close()
    connection.close()
```

前后端交互

本次项目中我们使用了Django框架来完成前后端之间的交互。Django是一种后端框架，主要用于处理服务器端逻辑、数据库操作和生成动态内容。在前端方面，Django通常会与其他前端框架等配合使用，以实现全栈开发。Django包含自己的模板引擎，允许在服务器端生成动态的HTML内容。通过在视图中使用模板语言，可以将数据传递到前端并以动态方式呈现。Django还提供了处理静态文件的机制，我们可以在Django项目中定义静态文件目录，然后使用模板中的静态文件标签引用这些文件。

下面介绍本次Django项目的搭建流程：

1.安装Django和创建项目

安装命令：

```
pip3 install django
```

创建Django项目命令：

```
django-admin startproject mysite
```

Django项目是一个包含配置、URL映射、静态文件等的整体Web应用。项目是用来组织和配置整个Web应用的，它可以包含多个应用（APP）。Django项目通常包含一个名为settings.py的配置文件，其中包括数据库设置、静态文件路径、应用列表等。还有一个urls.py文件用于定义项目级别的URL映射。

创建APP命令：

```
python manage.py startapp myapp
```

Django应用（APP）是一个可重用的模块，它包含了实现特定功能的代码、模板、静态文件等。应用是用来组织和封装特定功能的代码块，使得它们可以在不同的项目中重复使用。

由于本次项目的功能都是关于数据库的，为了简便我们将各个模块都放在了myapp这一个应用下面，以便搭建和测试。

2.Django后端与数据匹配

根据数据库结构自动生成数据库模型文件，命令如下

```
python manage.py inspectdb > models.py
```

上面的命令是Django自带的反向映射程序，但是经过测试，发现在我们使用的这个版本的极简版opengauss上并没有得到支持，所以我们只能通过自己编写models.py文件来自行搭建。

根据数据库结构，models.py最终设计如下

```
from django.db import models

# 专辑
class Album(models.Model):
    id = models.IntegerField(primary_key=True) # 专辑ID, 整数类型, 主键
    name = models.CharField(max_length=100, null=True, blank=True) # 专辑名称, 字符串类型, 最大长度100, 可以为空
    picurl = models.CharField(max_length=100, null=True, blank=True) # 专辑封面URL, 字符串类型, 最大长度100, 可以为空
```

```

def __str__(self):
    return f"{self.id} - {self.name}"

class Meta:
    db_table = 'album' # 数据库表名为 'album'

# 艺术家
class Artist(models.Model):
    id = models.IntegerField(primary_key=True) # 艺术家ID, 整数类型, 主键
    name = models.CharField(max_length=50) # 艺术家名称, 字符串类型, 最大长度50

    def __str__(self):
        return f"{self.id} - {self.name}"

    class Meta:
        db_table = 'artist' # 数据库表名为 'artist'

# 评论
class Comment(models.Model):
    commentid = models.BigAutoField(primary_key=True) # 评论ID, 长整数类型, 自增主键
    content = models.CharField(max_length=400, null=True, blank=True) # 评论内容,
    # 字符串类型, 最大长度400, 可以为空
    commenttime = models.CharField(max_length=30, null=True, blank=True) # 评论时
    # 间, 字符串类型, 最大长度30, 可以为空
    user = models.ForeignKey('Users', on_delete=models.CASCADE) # 用户外键, 关联到
    # Users 模型, 级联删除
    song = models.ForeignKey('Song', on_delete=models.CASCADE) # 歌曲外键, 关联到
    # Song 模型, 级联删除

    def __str__(self):
        return f"Comment ID: {self.commentid}"

    class Meta:
        db_table = 'comment' # 数据库表名为 'comment'

# 歌曲
class Song(models.Model):
    id = models.IntegerField(primary_key=True) # 歌曲ID, 整数类型, 主键
    name = models.CharField(max_length=150, null=True, blank=True) # 歌曲名称, 字
    # 符串类型, 最大长度150, 可以为空
    ar = models.ForeignKey('Artist', on_delete=models.CASCADE) # 艺术家外键, 关联到
    # Artist 模型, 级联删除
    al = models.ForeignKey('Album', on_delete=models.CASCADE) # 专辑外键, 关联到
    # Album 模型, 级联删除
    no = models.IntegerField(null=True, blank=True) # 歌曲编号, 整数类型, 可以为空
    version = models.IntegerField(null=True, blank=True) # 歌曲版本, 整数类型, 可
    # 以为空
    publish_time = models.CharField(max_length=20, null=True, blank=True) # 歌曲发
    # 布时间, 字符串类型, 最大长度20, 可以为空

    def __str__(self):
        return self.name

    class Meta:
        db_table = 'song' # 数据库表名为 'song'

# 用户

```

```

class Users(models.Model):
    id = models.BigAutoField(primary_key=True) # 用户ID, 长整数类型, 自增主键
    username = models.CharField(max_length=50) # 用户名, 字符串类型, 最大长度50
    avatarurl = models.CharField(max_length=100, null=True, blank=True) # 用户头像
    URL, 字符串类型, 最大长度100, 可以为空

    def __str__(self):
        return self.username

    class Meta:
        db_table = 'users' # 数据库表名为 'users'

```

之后需要执行下面这两条命令：

```

python manage.py makemigrations
python manage.py migrate

```

第一个命令用于生成数据库迁移文件。迁移文件是一种描述数据库模型变更的文件，包含了对数据库表结构的添加、删除、修改等操作。第二个命令用于应用数据库迁移，即将数据库模型的变更应用到实际的数据库中。运行这个命令会根据迁移文件的指令，执行相应的数据库表结构变更操作，保持数据库与当前代码中的模型类一致。之后如果又修改了models.py文件，也需要运行上面的命令。

运行完成后Django会在opengauss中自动创建我们设计好的表结构，然后我们可以往数据库中插入数据

```

import os
import json
import psycopg2
import datetime

# 数据库连接信息
db_params = {
    'dbname': 'music',
    'user': 'fei',
    'password': 'opengauss@123',
    'host': '192.168.56.101',
    'port': '26000'
}

folder_path = 'comment_json'

# 使用os.listdir()列出文件夹中的所有文件和子文件夹
files_and_folders = os.listdir(folder_path)

# 筛选以 "artist_" 开头且以 ".json" 结尾的文件
json_files = [file for file in files_and_folders if file.startswith('comment_') and file.endswith('.json')]

```

```

# SQL 插入语句
insert_sql = """
INSERT INTO COMMENT (commentid, content, commenttime, user_id, song_id)
VALUES (%s, %s, %s, %s, %s)
"""

# 连接到数据库
connection = psycopg2.connect(**db_params)
cursor = connection.cursor()

# 用一个集合来跟踪已读取的ID
read_ids = set()

try:
    # 逐个处理 JSON 文件
    for json_file in json_files:
        file_path = os.path.join(folder_path, json_file)
        with open(file_path, 'r', encoding='utf-8') as file:
            data = json.load(file)
            publish_time_seconds = data['comment_time'] / 1000

            if (publish_time_seconds < 0):
                formatted_time = 'null'
            else:
                # 使用 datetime 模块转换为正常时间格式
                publish_time = datetime.datetime.fromtimestamp(publish_time_seconds)

                # 格式化为字符串 (以年-月-日 时:分:秒 的格式)
                formatted_time = publish_time.strftime("%Y-%m-%d %H:%M:%S")

            # 获取JSON数据的ID
            comment_id = data['comment_id']

            # 如果ID已存在于已读取的集合中, 则跳过插入操作
            if comment_id in read_ids:
                continue

            # 提取 JSON 数据并插入数据库
            extracted_data = (
                comment_id,
                data['content'],
                formatted_time,
                data['user_id'],
                int(data['song_id'])
            )

            query = "SELECT EXISTS(SELECT 1 FROM song WHERE id = %s);"
            cursor.execute(query, (int(data['song_id']),))

            # 获取查询结果
            result = cursor.fetchone()[0]
            if result:
                cursor.execute(insert_sql, extracted_data)
                print(json_file)
            else:

```

```

        print(f"songid {int(data['song_id'])} 不存在于song表中-----")
    ")

    # 将已读取的ID添加到集合中
    read_ids.add(comment_id)

    # 提交事务
    connection.commit()

except Exception as e:
    print(f"Error: {e}")
    # 如果出现错误，回滚事务
    connection.rollback()

finally:
    # 关闭连接
    cursor.close()
    connection.close()

```

新的插入文件与前面展示的插入文件类似，主要的更改为加入了数据检测，为了保证外键的正确性，如果外键对应的主键并不存在，则将不正确的数据过滤，从而使得我们的数据能够完全符合Django创建的表结构。

3.运行服务器

首先需要更改mysite\mysite\setting.py中的DATABASES

```

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'music', #数据库名
        'USER': 'fei', #用户名
        'PASSWORD': 'opengauss@0', #密码
        'HOST': '192.168.56.101', #虚拟机ip
        'PORT': 26000 #openGauss数据口的端口
    }
}

```

django.db.backends.postgresql_psycopg2是Django中用于与PostgreSQL数据库进行交互的数据库引擎，而OpenGauss兼容PostgreSQL的协议，因此可以使用相同的引擎来连接OpenGauss

运行服务器命令如下

```
python manage.py runserver
```

之后我们可以通过网址[http://127.0.0.1:8000/myapp/home/]访问前端网页

前端设计

我们的前端网站主界面如下：



下面对主界面的布局进行介绍：



左侧是菜单栏可以进行操作选择。中间卡片上方是搜索栏，下方是歌曲滚动列表。右侧是播放列表，由于考虑到版权因素，我们后面取消了实际的播放演示，仅保留界面作为展示。

本次前端设计参考了很多资料，并通过Chrome浏览器的开发者工具对目前主流音乐网站的前端代码进行了学习，最终根据我们的数据结构和要实现的功能采用了上面展示的三分栏设计。

由于前端代码较为庞大，这里展示主要部分

```

<body>
  <div class="scrollable-table">
    <table>
      <thead>
        <tr>
          <th class="title">歌曲</th>
          <th class="player">歌手</th>
          <th class="song-time">发布时间</th>
          <th class="song-album">专辑</th>
          <th class="album-image"></th> <!-- 新添加的列 -->
        </tr>
      </thead>
      <tbody>
        {% for song in songs %}
          <tr>
            <td><a href="{% url 'get_comment' song.id %}">{{ song.name }}
            </a></td>
            <td>{{ song.ar.name }}</td>
            <td>{{ song.publish_time }}</td>
            <td>{{ song.al.name }}</td>
            <td>
              
          </tr>
        {% endfor %}
      </tbody>
    </table>
  </div>
</body>

```

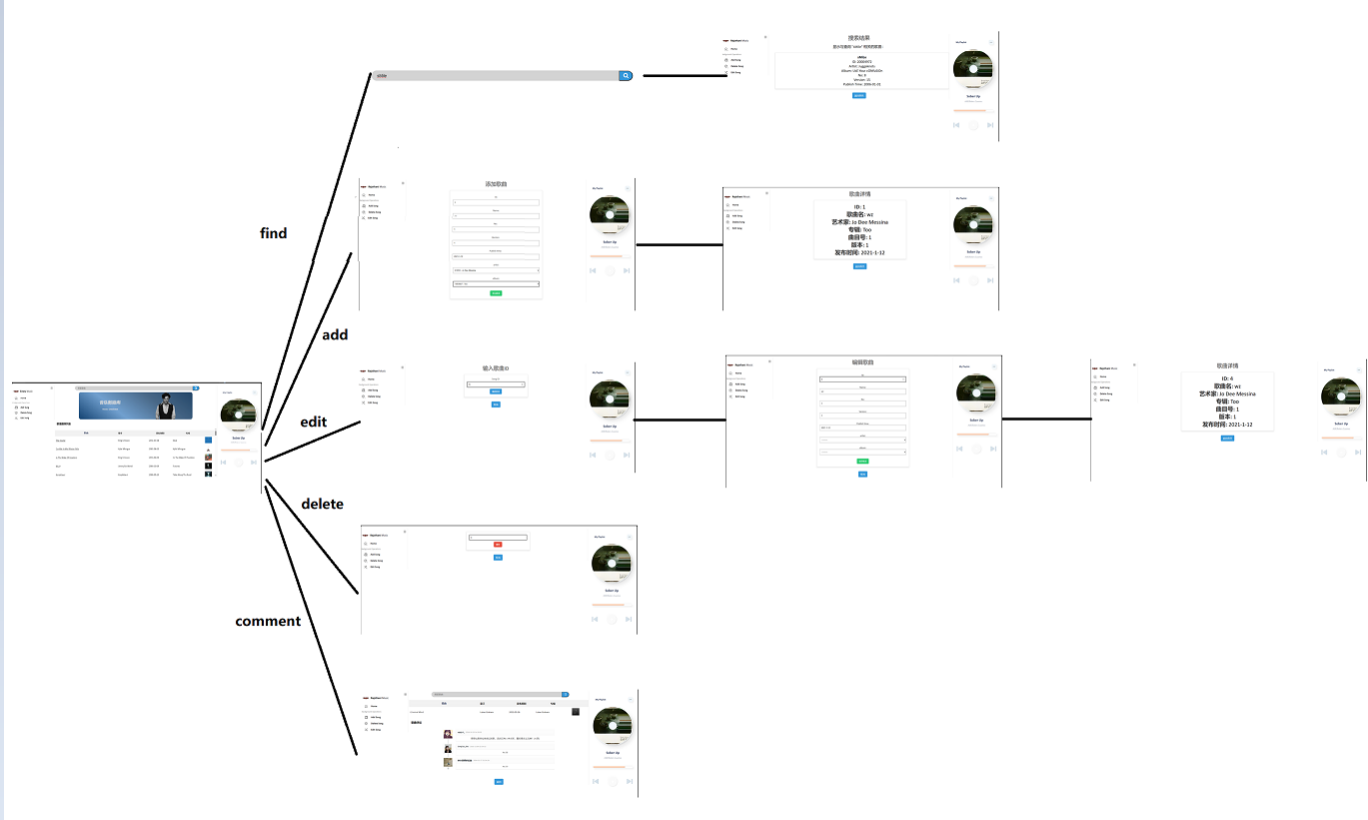
这里使用了Django的模板语言，用于在HTML中嵌入动态数据。例如，`{% for song in songs %}` 这样的标签用于循环遍历歌曲列表，`{{ song.name }}`、`{{ song.ar.name }}`等标签用于插入具体的歌曲数据。`{% url 'get_comment' song.id %}`用于生成歌曲评论页面的URL。这利用了Django的URL模型，允许通过命名URL来生成链接，而不必硬编码URL。``用于在页面上显示专辑封面图，其中`{{ song.al.picurl }}`是从数据库中获取的静态文件的URL。

接下来介绍网站中实现的具体功能：

- 主页 (home.html): 显示随机选择的歌曲列表，最多显示50首歌曲。
- 添加歌曲 (add_song.html): 包含用于添加新歌曲的表单。
- 删除歌曲 (delete_song.html): 包含用于删除歌曲的表单。
- 搜索歌曲 (search_song.html): 包含用于搜索歌曲的表单。
- 歌曲显示 (search_results.html)： 包含用于显示搜索歌曲结果的表单。
- 编辑歌曲 (edit_song.html): 包含用于编辑现有歌曲的表单。
- 输入歌曲ID (input_song_id.html): 包含用于输入歌曲ID的表单。

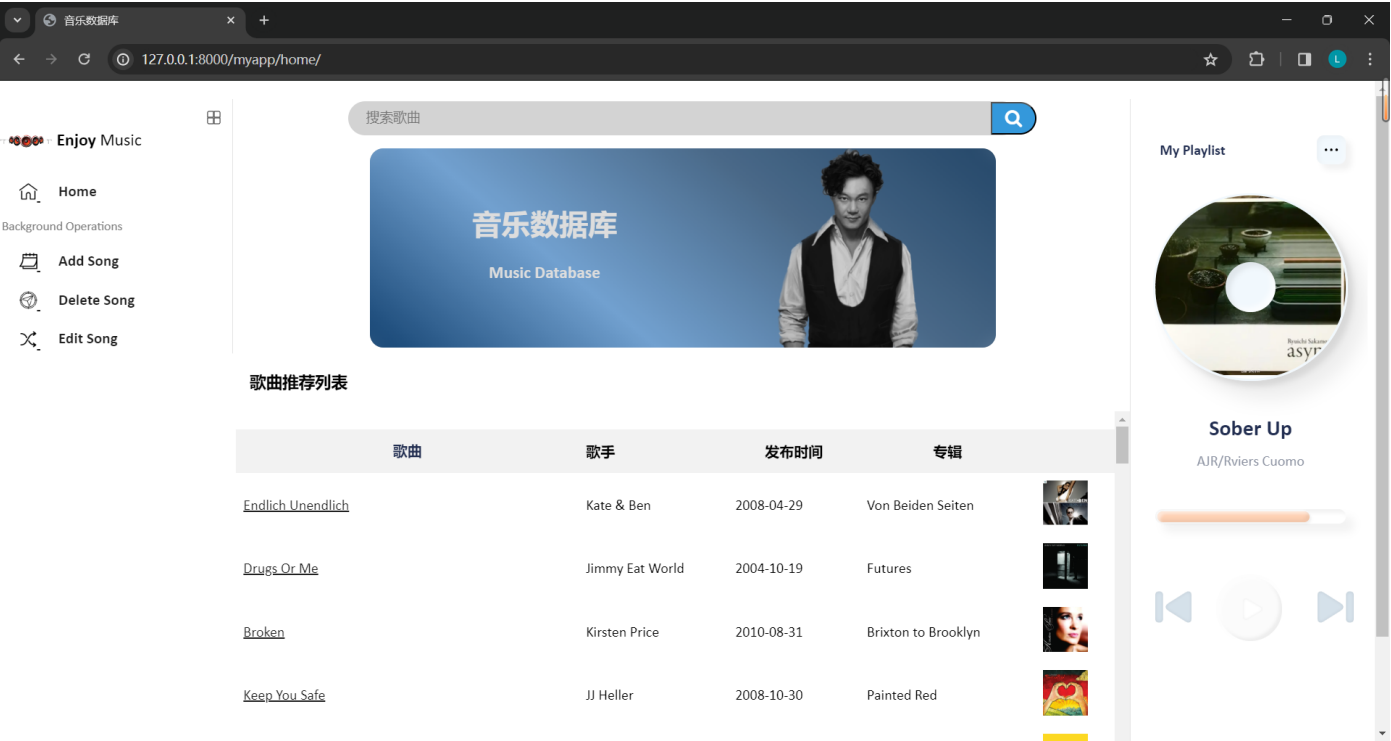
- 歌曲详情 (song_detail.html): 显示特定歌曲的详细信息。
- 评论展示 (get_comment.html)： 点击对应的歌曲即可查看对应的评论。

网站之间的逻辑关系如下图：



主要运行界面

主页：



评论：

🎵 Enjoy Music

Home

Background Operations

Add Song

Delete Song


Edit Song

搜索歌曲


歌曲歌手发布时间专辑

Endlich UnendlichKate & Ben2008-04-29Von Beiden Seiten


歌曲评论

六道骸夫人 2018-12-10 07:55:03

kate和ben是因为一个朋友安利喜欢上的[憨笑][憨笑][憨笑]朋友现在虽然不在了。听着他们的歌就忍不住想起他。也是一种怀念吧

gardenman 2019-03-12 16:51:34


这两人的歌从最前面那首到现在，评论真的越来越少

GillianCCCBKPP 2019-05-25 09:09:28

啦啦啦


返回

My Playlist



Sober Up

AJR/Rviers Cuomo



查找：

🎵 Enjoy Music

Home

Background Operations

Add Song

Delete Song

Edit Song

搜索结果

显示与查询 "tik tok" 相关的歌曲：

Tik Tok

ID: 20036082

Artist: Kesha

Album: Blah Blah Blah

No: 2

Version: 14

Publish Time: 2010-03-02

Tik Tok

ID: 20036109

Artist: Kesha

Album: Tik Tok


No: 1

Version: 47

Publish Time: 2009-08-07


Tik Tok [Fred Falke Club Mix]

My Playlist



Sober Up

AJR/Rviers Cuomo



添加:

🎵 Enjoy Music

Home

Background Operations

Add Song

Delete Song

Edit Song

搜索结果

显示与查询 "tik tok" 相关的歌曲:

Tik Tok

ID: 20036082

Artist: Kesha

Album: Blah Blah Blah

No: 2

Version: 14

Publish Time: 2010-03-02

Tik Tok

ID: 20036109

Artist: Kesha

Album: Tik Tok


No: 1

Version: 47

Publish Time: 2009-08-07

Tik Tok [Fred Falke Club Mix]

My Playlist



Sober Up

AJR/Rviers Cuomo

⏮

▶

⏭

🎵 Enjoy Music

Home

Background Operations

Add Song

Delete Song

Edit Song

歌曲详情

ID: 20051234

歌曲名: hello

艺术家: Kram

专辑: In Your Face


曲目号: 123

版本: 1

发布时间: 2024-01-01

返回首页

My Playlist



Sober Up

AJR/Rviers Cuomo

⏮

▶

⏭

删除：

🎵 Enjoy Music

Home

Background Operations

Add Song

Delete Song


Edit Song

20051234

删除

取消

My Playlist



Sober Up

AJR/Rviers Cuomo

⏮

▶

⏭

编辑：

🎵 Enjoy Music

Home

Background Operations

Add Song

Delete Song

Edit Song


Song ID:

20020716

编辑歌曲

取消

My Playlist



Sober Up

AJR/Rviers Cuomo

⏮

▶

⏭

音乐数据库

127.0.0.1:8000/myapp/edit_song/20020716/

编辑歌曲

Enjoy Music

Home

Background Operations

Add Song

Delete Song

Edit Song

Id:

20020716

Name:

Zingl Went the Strings of My Heart

No:

25

Version:

14

Publish time:


1992-04-22

artist:

album:

保存修改

My Playlist



Sober Up

AJR/Rviers Cuomo

音乐数据库

127.0.0.1:8000/myapp/edit_song/20020716/

编辑歌曲

Enjoy Music

Home

Background Operations

Add Song

Delete Song

Edit Song

Id:

12345679

Name:

New Name

No:

1

Version:

1

Publish time:


1992-04-22

artist:

album:

保存修改

My Playlist



Sober Up

AJR/Rviers Cuomo



实验总结

在本次实验中，我们初步了解到了数据库应用系统的开发流程，包括需求分析、数据库设计、数据处理、后端搭建、前后端交互、前端搭建等方面。实验过程中我们也遇到了一些挑战，例如在Centos上安装openGauss时出现的各种问题、寻找数据来源、前后端的正确交互等。通过团队合作和充分的讨论，我们最终成功地完成了项目。这次实验让我们对数据库有了更深入的理解和认识，锻炼了我们的资料查找能力、代码编写能力、团队合作能力，这次实验经历也为我们未来完成更复杂的项目打下了一些基础。