云南大学2022年秋季学期信息学院2020级

《数据库技术》期末实验大作业

**满分：100分 任课教师：**

**学院：信息学院 专业：智能科学与技术 学号： 姓名：*Steven***

**成绩：＿＿＿＿**

# 一、题目

基于数据库的含标签简易资源管理器

# 二、需求分析

## 1. 数据库需求

(1) 数据库粒度：

考虑到本身设定为“资源管理器”而非“文件管理器”，因此并不需要严格按照扩展名进行区分管理，而应当按照文件所属的资源类型进行管理。因此，数据库也只建立了几种资源的数据库：音乐、视频、图片，以及在实际文件系统中绕不开的文件和文件夹库。

(2) 部分文件的属性：

对于音乐、视频、图片，他们都是文件的子类，因此在构建数据库时，这些类型的属性应当为文件属性的超集。

(3) 音乐文件应构建多表：

此外还注意到，音乐文件中往往包含一些标签信息，如标题、专辑、作者、发行年份等，应当根据这些信息建立不同的实体，以满足多样化、高效的查找相信需求。因此，对于音乐类资源，在数据库中建立了音乐、专辑、作者、MV四个数据表，分别存储对应的信息。而对于其他类型的文件则仅需单个数据表即可存储。

(3) 特色设计：

我在曾用过许多Windows上不错的文件管理器，如Files(Files Community)、XYplorer、Billfish等，我发现为文件添加标记作为一项非常实用的功能，却鲜有文件管理器完善，因此在本程序中为每种文件都留了Tags属性，以便于快速查找。

(4) 数据库客户端选择：考虑到：

**①**软件开发应当具备较好的可移植性，而MySQL本体较大，且需要配置环境变量，不便于移植和安装；

**②**作为本地资源管理程序，数据量通常并不大，对性能要求不高。

因此选择了Sqlite作为默认数据库，不过因为Python中Sqlite和MySQL的操作接口一致，在软件中也保留了关于MySQL的操作。

## 2. 界面需求

(1) 信号回调：

本程序应当能从数据库中读取信息，通过GUI显示出来，同时能够通过GUI编辑部分信息，将信息写回数据库，因此应当在界面中定义输入输出对象，并在其上监听相关事件，以便于进行响应。

(2) 多级窗口：

此外，如果只通过一个窗口展示所有内容，由于不同文件的属性不完全相同，势必会造成表单中不同行存在不同的空缺，而且使得整个界面缺乏重点——应当展示基本信息，而将附加信息放在二级菜单显示。至此，本程序仿照Windows文件资源管理器做了一级菜单，并做了一个简单的二级菜单以显示更多信息。

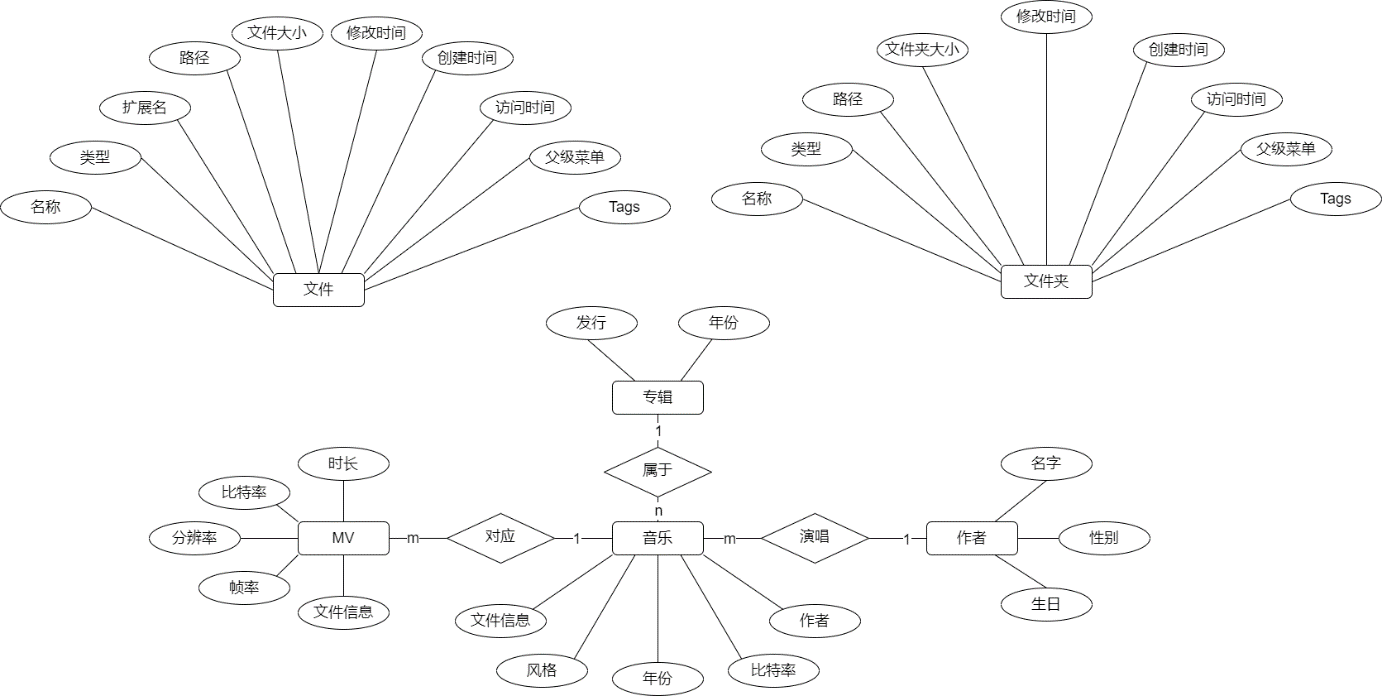
# 三、系统设计（包括功能设计与数据设计，重点阐述数据库的概念结构设计、逻辑结构设计等）

## 1. 功能设计

功能说白了其实就是通过GUI操作数据库，但是完成该功能有几点需要注意：

1. 后端：
   1. 能够根据文件后缀自动识别资源类型。
   2. 能够通过根据资源类型实例化对应的类，并自动存储进不同的数据库。
2. 前端：
   1. 能够通过地址栏、侧栏、双击文件夹，以及前进后退键进行页面的跳转，并保持逻辑。
   2. 能够监听各种窗口事件，获取窗口中的信息，并反馈到后端。

## 2. 数据库概念结构



（图1 E-R图）

E-R图如上图所示，其中，由于图片、视频和文件一样，只用单表存储即可，所以为节省空间没有画出，以文件作为代表。

而对于音乐文件，则应当根据音乐划分不同主体：音乐、作者、专辑、MV，并根据一定的联系连接起来。

## 3. 数据库逻辑结构

数据库的关系模式如下所示：

File(路径, 文件名, 类型, 扩展名, 大小, 大小的单位, 修改时间, 创建时间, 访问时间, 父级菜单, Tags)

Image(路径, 名称, 类型, 扩展名, 大小, 大小的单位, 修改时间, 创建时间, 访问时间, 父级菜单, 宽度, 高度, Tags)

Video(路径, 文件名, 类型, 扩展名, 大小, 大小的单位, 修改时间, 创建时间, 访问时间, 父级菜单, 时长, 宽度, 高度, 比特率, 帧率, Tags)

Dir(路径, 文件夹名, 类型, 大小, 大小的单位, 修改时间, 创建时间, 访问时间, 父级菜单, Tags)

Music(路径, 类型, 扩展名, 大小, 大小的单位, 修改时间, 创建时间, 访问时间, 父级菜单, 歌手, 歌名, 专辑, 年份, 风格, 时长, MV, 比特率, Tags)

Singer(名字, 性别, 生日, 地区)

MV(歌名, 专辑, 时长, 扩展名, 比特率, 宽度, 高度, 帧率, 路径)

Album(专辑名, 发行, 年份)

由于上述关系模式相对来说比较简单，所以设计数据库时也都使其满足了BCNF。

# 四、系统实现（包括数据库管理系统、开发平台与语言、核心代码、系统截图等）

## 1. 数据库管理系统

考虑到以下原因：

1. 由于设定为用户软件，通常处理的数据量不大，因此对DBMS性能没有什么要求
2. 同样由于设定是软件，所以需要有较高的可移植性，MySQL本体较大并不合适，而Sqlite很小，可以方便地嵌入代码中。

所以最后选择的默认DBMS为Sqlite，不过由于Python为Sqlite和MySQL提供的调用接口完全相同，所以也在程序中保留了改用MySQL的方式。

## 2. 开发平台与语言

开发平台：Windows 10 22H2

语言：Python 3(基于Python 3.9的Anaconda)

界面构建：PyQt 5, Qt Designer

## 3. 核心代码

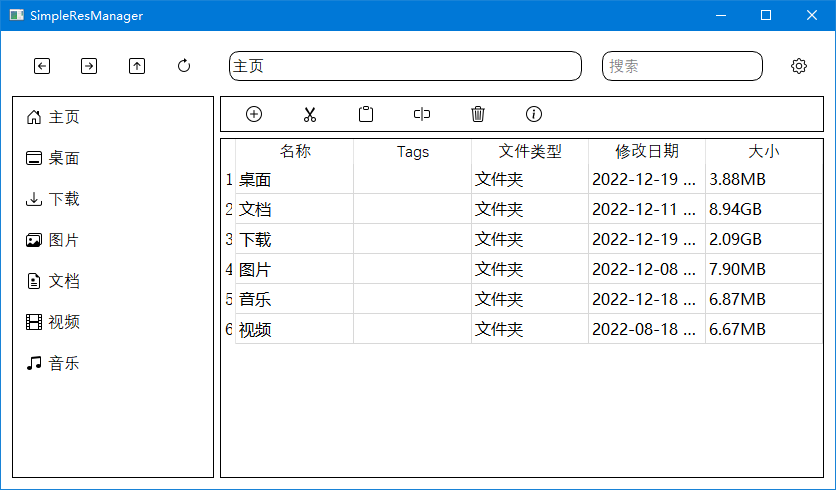
(1) 前端界面：代码使用Qt Designer自动生成，内容较多且结构杂乱，故不在此展示。

(2) 后端代码：后端代码主要为各种资源的子类，以及负责从文件夹中获取信息的FileController类和控制数据库读写的DBController类。篇幅有限，这里只展示DBController的updateDB方法。

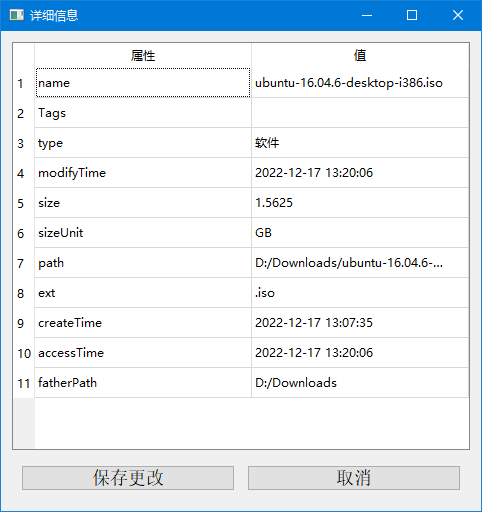
|  |
| --- |
| def updateDB(self, data: list[dict]):  *"""  更新数据库  :param data:dict:数据  """* unique\_key = 'path'  for item in data: *# 对每个对象进行操作* info = {\*\*item['basicInfo'], \*\*item['otherInfo']} *# 合并basicInfo和otherInfo* key, value = zip(\*info.items())  key = list(key)  value = list(value)  if info['type'] in ('文件', '快捷方式', '软件', '文档（不可读）'):  table = 'file'  elif info['type'] == '音乐':  table = 'music'  elif info['type'] == '图片':  table = 'image'  elif info['type'] == '视频':  table = 'video'  elif info['type'] == '文件夹':  table = 'dir'  else: *# 未知类型，跳过* continue  *# 更新数据库* if info['type'] == '音乐': *# 一些前期检查，依次检查歌手和专辑是否存在记录，不存在则插入，然后再判断歌曲并插入/更新  # 检查歌手* singer = info['singer']  self.cursor.execute("SELECT \* FROM music\_singer WHERE name = '%s'" % singer)  fetch = self.cursor.fetchone()  if not fetch: *# 如果不存在，则插入* self.cursor.execute(  "INSERT INTO music\_singer (name) VALUES ('%s')" % singer) *# 因为从歌曲文件的tags中只能知道歌手名* self.conn.commit()  *# 检查专辑，差不多* album = info['album']  self.cursor.execute("SELECT \* FROM music\_album WHERE name = '%s'" % album)  fetch = self.cursor.fetchone()  if not fetch: *# 如果不存在，则插入* self.cursor.execute("INSERT INTO music\_album (name) VALUES ('%s')" % album)  self.conn.commit()  *# 剩下的就是更新主表，因为文件/配置文件、文件夹都只有一个表，所以跳过前面那些复杂的准备工作  # 先将旧记录删掉* self.cursor.execute("DELETE FROM %s WHERE %s = '%s'" % (table, unique\_key, info[unique\_key]))  self.conn.commit()  *# 然后插入新记录* values = ', '.join(["'" + v + "'" if isinstance(v, str) else str(v) for v in value])  self.cursor.execute("INSERT INTO %s (%s) VALUES (%s)" % (table, ','.join(key), values))  self.conn.commit() |

(3) 控制代码：控制代码用于接收前端的信号，根据信号在后端进行响应，并接收后端回传的数据，处理后发送至前端。

(4) 部分截图：



（图2 进程主界面）



（图3 详细信息界面，此处Tags的值可修改，并保存更改）

# 五、使用说明

## 1. 环境配置

安装好Python，在命令行中移动到项目根目录，输入pip install -r requirements.txt，自动安装依赖；或使用conda等其他包管理器安装依赖。

打开根目录下的config.json文件，修改“homePageQuickDirs”和“projectPath”的地址为项目在本机的实际路径。

## 2. 运行

在IDE中运行main.py，或直接在Powershell中移动到根目录，输入“python main.py”运行。

由于本项目的工程量太大，尽管已经写了700余行（若考虑上前端代码则超过1200行），但也仅仅是完成了基本功能，地址栏下方的功能按钮还未完成，此外该程序健壮性也有待提升，偶尔会闪退，但已经来不及完善了。