北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

数据库设计与开发

Assignment3

设计文档

题　　目：数据库 Assignment3 设计文档

学　　院：计算机学院

专业名称：软件工程

学　　号：

姓　　名：邬政钢

Contract

I have listened carefully and understand the "Database Design and Development" course requirements; I received this copy of the course requirements. I have a complete and full comprehension of these requirements and fully accept its contents.

I solemnly declare that I will finish all the courses experiments and assignments independently, if I have any cheating or plagiarism thing or teacher believe that I have the above behavior, I am willing to accept zero score of this course grade, and without any appeal.

- Signature
- Date

目录

任务查找表

一，需求分析

选题针对的问题已在上一部分介绍，为了解决问题，对问题进行简要的需求分析。[1]

1.1 选题介绍

本人 Assignment3 的选题为“跨音乐 App 歌单管理器”。此选题主要针对当前广泛存在的音乐版权问题，在国内，不同的歌曲版权分别归属于不同的音乐 App。一个简单的例子，周杰伦的歌曲大部分版权都归 QQ 音乐，那么在网易云音乐中就不能对无版权歌曲做任何的操作；但是，网易云音乐拥有大量 QQ 音乐所没有的民谣和小众歌手的歌曲。也就是说，二者的版权关系是有相交但不相互包含的集合关系。那么作为音乐爱好者，在不同的音乐 App 中建立歌单管理喜欢的音乐的时候，就会出现版权问题，不能把相同类型的歌曲归类到同一个歌单中，导致对于歌单歌曲的管理出现问题。本选题致力于解决这种问题。

1.2 软件前景

广大音乐爱好者深受版权问题困扰，在进行歌单管理时十分难受，因此开发出一款软件“跨音乐 App 歌单管理器”进行跨音乐 App 的歌单管理。当前软件版本为 1.0.0，为软件的初代版本，亦是之后成熟软件的初代原型系统。如果今后还有机会，此软件将会进行大改，因为需求确实存在，我认为此软件有成为一个优秀开源软件的潜力。

1.3 软件功能

软件功能主要为对歌单进行管理，支持对于歌单的普通操作，如创建，添加歌曲，删除歌曲等操作，详细的软件功能见表 Tabel.1。

需求表		
功能	**功能需求**	
歌单歌曲管理	歌单相关	创建歌单
		删除歌单
		导入其他音乐 App 歌单
	歌曲相关	加入歌曲
		删除歌曲
		播放歌曲
	显示所有已知歌曲	
用户管理	注册新用户	自定义账号，密码，昵称
		选择所属地区
		添加音乐 App 子账号
	用户登录	
	用户信息更新	更改用户密码
		更改用户昵称

[1] 尽管学习了软件需求工程课程，但是这里不会完整的写出整个需求规格文档，原因一是没必要，需求规格文档中的很多要求在 Assignment 中并未涉及，二是没有甲方即需求提出方，所有需求为独立开发者挖掘，因此此需求分析具有相当的主观性

用户交互	UI 界面	界面刷新
		UI 交互
管理员操作	管理员登录	
	所有用户信息查询	
	删除用户	

Table.1 软件整体功能表

1.4 运行环境

软件逻辑上分为客户端，服务端，数据库以及音乐 App 接口服务（后简称 App 接口）四个部分。其中客户端为 delphi 开发，在 window10 系统上运行。服务端使用 Java 开发，数据库使用 openGauss 数据库，App 接口有 3 个，都是基于 JavaScript 的 node.js 服务的开源代码[2]，三者一起运行于服务器上，作为整合服务端，在 CentOS 系统上运行。

1.5 设计和实现限制

在本软件设计和开发中，存在以下限制。

- 必须使用 openGauss 数据库
- 必须使用 delphi 开发 UI
- 数据库表至少有 6 张
- 必须使用 C/S 开发架构

1.6 外部接口

本软件不提供除用户 UI 外的其他接口，本软件用户 UI 承担用户的所有交互操作，需要用户操作的所有功能均通过 UI 实现。

1.7 性能和安全

因为是初版软件以及原型系统，因此对性能和安全要求不高。

本软件对性能要求为及时处理用户操作，要求最长响应时间不超过 3s；对安全要求为禁止对数据的违法操作。

---

[2] 开源代码地址：
网易云音乐 Api：https://github.com/Binaryify/NeteaseCloudMusicApi
QQ 音乐 Api：https://github.com/rain120/qq-music-api 和
https://github.com/jsososo/QQMusicApi

二，软件功能设计

2.1 总体功能分配

软件采用瘦客户端开发模式，在客户端只运行少量的业务逻辑，大部分业务逻辑通过 socket 发送到服务端运行。软件不同部分的功能如表 Table.2 所示。

软件功能设计	
客户端	信息显示
	用户提示信息
	判断具体操作
	输入合法性判断
	组合 SQL 语句
	请求服务端
	接收服务端返回信息
	处理并显示服务端返回信息
服务端	接受客户端请求
	判断业务逻辑并执行
	执行结果返回客户端
数据库	数据存储
	数据库业务逻辑（增删改查）
App 接口	请求音乐 App 官方服务

Table.2 软件分部分功能

2.2 客户端功能

- 信息显示：将信息显示在 UI 界面上，方便用户操作
- 用户提示信息：通过用户提示信息提示用户操作结果和约束用户极端行为
- 判断具体操作：客户端根据用户的具体操作判断用户行为
- 输入合法性判断：客户端判断需要用户的输入是否合法
- 组合 SQL 语句：判断用户具体操作后，如果用户的操作涉及数据库操作，则在客户端将 SQL 语句组装好。
- 请求服务端：客户端将用户操作，SQL 语句等信息整合成 Json 格式字符串后发送请求服务端服务。
- 接收服务端返回信息：请求服务端之后，客户端等待并接收服务端的返回信息
- 处理并显示服务端返回信息：客户端处理接收到的服务端返回信息，如果需要显示到 UI 或者用户提示则进行显示和执行相关提示

2.3 服务端功能

- 接受服务端请求：服务端接收客户端请求信息并解包
- 判断业务逻辑并执行：服务端根据解包后的信息判断业务逻辑并执行
- 执行结果返回客户端：服务端将业务逻辑的执行结果进行自定义组合后返回客户端

2.4 数据库

- 数据存储：数据库存储软件所有的数据
- 数据库业务逻辑：根据服务端的业务逻辑，数据库执行对应的操作，并返回结果

2.5 App 接口

- 请求音乐 App 官方服务：软件的一些业务逻辑需要请求各音乐 App 的官方服务器服务，App 接口实现此功能

三，软件结构设计

3.1 客户端设计

如 Fig.1 所示，客户端的结构较为简单，与用户交互得到信息后进行判断组合然后请求服务端进行操作。

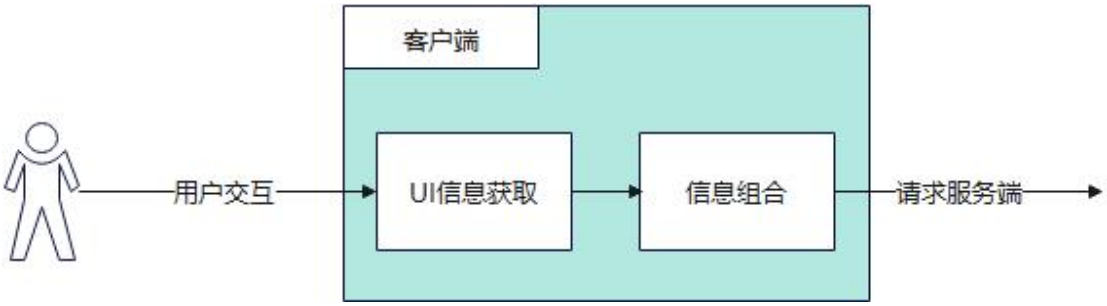

Fig.1 客户端结构

客户端使用 delphi 进行开发，由于客户端的逻辑基本上都是串行逻辑，并不复杂。由于 delphi 特殊的开发机制，单独的一个窗口即为一个文件，因此在开发客户端中，将功能按照窗口划分开发。客户端一共有 5 个窗口，其功能分割如下。

- Login 窗口：登录页面。登录，注册页面进入，管理员管理页面进入
- SignUp 窗口：注册页面。注册信息输入，注册
- Admin 窗口：管理员管理页面。管理员账号登录，查询所有注册用户，删除用户
- MainWindow 窗口：歌单歌曲管理主页面。歌单歌曲管理的全部功能，进入个人信息更新页面
- UpdataAccountInfo 窗口：更改个人信息页面。更改账户密码和昵称

其跳转关系如图 Fig.2 所示。

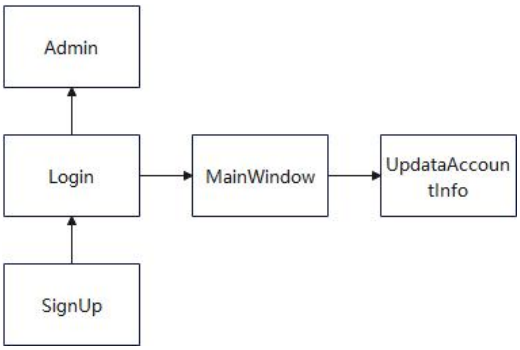

Fig.2 客户端页面跳转关系

3.2 服务端设计

服务端是整个软件的核心部分，连接了客户端，数据库和 App 接口，绝大部分业务逻辑通过服务端进行实现。服务端的结构如 Fig.3 所示。

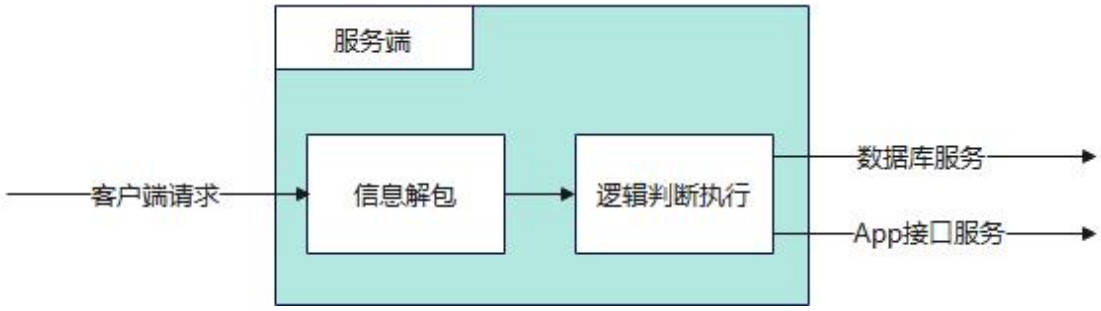

Fig.3 服务端结构

服务端使用 Java 进行开发，使用 JDBC 连接 openGauss 数据库，通过端口访问开源 App 接口服务。服务端的逻辑也并不复杂，监听客户端请求，接收到请求后解包，根据解包信息中的 action 属性执行对应操作，操作完成后进入下一循环继续监听客户端。

服务端除了主运行文件外可以分为几个模块，解包模块 DataTypeAdapters，数据库模块 DbWorkers 和 App 接口模块 MusicAppApis，其余的包都是辅助的信息存储，不具备实际功能。

3.3 数据库设计

软件的数据库设计分为概念结构设计，逻辑结构设计和物理结构设计。

1. 概念结构设计

从需求中可以抽象出一系列实体关系图，如图 Fig.4 所示。

Fig.4 实体分析 ER 图

对于这些实体进行综合分析之后，可以得到如下实体：账号（账号，密码，昵称），歌曲（名字，歌曲 mid[3]，歌曲 mid 所属音乐 App），专辑（名字），歌手（名字，简介），已支持音乐 App 数据字典（音乐 App 名字），音乐 App 账号表（账号，音乐 App 账号，音乐 App 账号密码），地区（账号，省，市，县）[4]。

再分析上述实体之后可以得到数据库中的表单。通过 ER 图形式展示数据库中的表单结

[3] 这个是音乐 App 的歌曲标识 id

[4] 这里通过数据字典实现，在 Assignment2 中已经有体现，这里不赘述

构如 Fig.5 所示。**注意：这张 ER 图并不是实体分析的 ER 图，只是为了方便展示数据库中的表的结构。**

Fig.5 数据库 ER 图

2. 逻辑结构设计

数据库设计需要规范到 3NF 或者 BCNF。找到出现的表，并验证其是否满足 3NF 或者 BCNF。

在验证每张表是否满足范式之前，先说明每张表的设计原因。

- 账号表：注册登录
- 所属地区表：账号信息
- 省市县表：所属地区的数据字典
- 已支持音乐 App 数据字典：已支持官方服务器请求的音乐 App 数据字典
- 音乐 App 账号表：用户拥有的各音乐 App 账号
- 歌单表：歌单信息存储，功能核心表
- 歌曲表：歌曲信息存储，功能核心表
- 专辑表：专辑信息存储，核心功能表
- 歌手表：歌手信息处处，核心功能表
- 歌单歌曲多对多关系表：由歌单和歌曲的多对多关系生成的关系表
- 歌手歌曲多对多关系表：由歌手和歌曲的多对多关系生成的关系表
- 专辑歌手多对多关系表：由专辑和歌手的多对多关系生成的关系表

需要特别说明的是，有些人认为歌曲和歌曲所在的专辑的歌手是一回事，如上的设计会导致歌手信息重复，实则不然，一张真正的专辑信息如 Fig.5 所示，可以观察到其中一首歌曲的歌手与专辑的歌手并不一致。事实上，专辑的歌手是专辑的发行歌手，具体歌曲的歌手是歌曲的演唱歌手，这个需要进行分辨。

Fig.5 真实专辑示例

对于表本身的说明结束之后，验证是否每张表都满足 3NF 或者 BCNF。

- 账号表（账号名，密码，昵称）：函数依赖集为{账号名->密码，账号名->昵称}，所有函数依赖左边都是超键，满足 BCNF。
- 所属地区表（账号名，省 ID，市 ID，县 ID）：函数依赖集为{账号名->省 ID*市 ID*县 ID，县 ID->市 ID，市 ID->省 ID}，所有函数依赖左边都是键的一部分即主属性，满足 3NF。
- 省份表（省 ID，省份名）：函数依赖集为{省 ID->省份名}，所有函数依赖左边都是超键，满足 BCNF。
- 地级市表（市 ID，地级市名，省 ID）：函数依赖集为{市 ID->省 ID，市 ID->地级市名}，所有函数依赖左边都是超键，满足 BCNF。
- 县城表（县 ID，县名，市 ID）：函数依赖集为{县 ID->市 ID，县 ID->县名}，所有函数依赖左边都是超键，满足 BCNF。
- 已支持音乐 App 数据字典（音乐 AppID，音乐 App 名字）：函数依赖集为{音乐 AppID->

音乐 App 名字}，所有函数依赖左边都是超键，满足 BCNF。

- 音乐 App 账号表（账号名，音乐 AppID，音乐 App 账号名，音乐 App 密码）：函数依赖集为{账号名*音乐 AppID*音乐 App 账号名->音乐 App 密码}，所有函数依赖左边都是超键，满足 BCNF。
- 歌单表（歌单 ID，歌单名，账号名，音乐 AppID）：函数依赖集为{歌单 ID->歌单名，歌单 ID->账号名，歌单 ID->音乐 AppID}，所有函数依赖左边都是超键，满足 BCNF。
- 歌曲表（歌曲 ID，歌曲名，歌曲 mid，专辑 ID，音乐 AppID）：函数依赖集为{歌曲 ID->歌曲名，歌曲 ID->歌曲 mid，歌曲 ID->专辑 ID，歌曲 ID->音乐 AppID}，所有函数依赖左边都是超键，满足 BCNF。
- 专辑表（专辑 ID，专辑名字）：函数依赖集为{专辑 ID->专辑名字}，所有函数依赖左边都是超键，满足 BCNF。
- 歌手表（歌手 ID，歌手名字，歌手简介）：函数依赖集为{歌手 ID->歌手名字，歌手 ID->歌手简介}，所有函数依赖左边都是超键，满足 BCNF。
- 歌单歌曲多对多关系表（歌单 ID，歌曲 ID），歌手歌曲多对多关系表（歌手 ID，歌曲 ID），专辑歌手多对多关系表（专辑 ID，歌手 ID）：此三个表都是实体表的多对多关系生成的关系表，属性为多对多关系的两张表的主键。关系表不存在非平凡函数依赖，满足 BCNF。

综上所述，数据库中的所有表至少满足 3NF，因此数据库的设计满足 3NF。

除对于表的逻辑设计之外，为了方便后续对于数据库的使用，建立相关视图，函数和存储过程。

- 视图 get_all_songs：查询所有歌曲
- 函数 insert_bigaccount：注册账号
- 函数 insert_area：新建账号所属地区
- 函数 insert_singers：新建歌手
- 函数 insert_albums：新建专辑
- 函数 insert_songs：新建歌曲
- 函数 insert_songlists：新建歌单
- 存储过程 delete_bigaccount：删除账号

3. 物理结构设计

物理结构可以设计的空间不大。使用的数据库为 openGauss 数据库。另外当前场景中的数据量不大，因此不选择使用分区表实现。各种类型的数据的主要存储方式主要有，ID 通过整型数字 INT 存储，名字，简介等通过 char 数组存储。

四，Q&A：开发人员问答

4.1 为什么使用这样的四部分开发结构？

在软工的学生群中，其他同学谈论地很多的开发方式为使用 ODBC 直连数据库。我对此持怀疑态度。我认为 C/S 架构应该有一个执行业务逻辑的服务端，然后使用服务端连接数据库。ODBC 是微软开发的跑在 windows 上的，难道同学们的服务器跑在 windows 上。这我不知道，但是我的服务器跑在 centos 上，数据库也在 centos 上，服务端是 java 写的，所以使用 JDBC 连接数据库顺理成章。

所以这样就有客户端，服务端，数据库三个部分，另外因为我的服务需要连接到音乐 App 的官方服务器请求信息，所以还有一个 App 接口部分，一共四个部分。

4.2 为什么在表的设计中要留一个没有用到的音乐 App 账号表？

并不能说这个表没有用，因为这个表在后续的迭代版本中肯定是会用到的，只不过是否需要更改还需要检验。

对于 Ass3 我开发的这个软件，我并不视此为一个短期的大作业式软件，现在音乐，书籍什么都在谈版权。音乐是我为数不多的兴趣爱好之一，可以说，在我使用电脑的同时，至少同时开着两个音乐 App，我对于不同的 App 拥有不同歌曲的版权有很大的意见，所以我一直希望有一个软件能解决这个问题。这次也正好借这个契机做了一下尝试，发现商业软件对于自身的保护十分严密。因此在提交作业前的短开发周期内肯定是不能做到在自开发软件层面做到真正的登录到官方 App 上的功能，尽管这是我最想实现的。所以我不是自己造几个假的歌单然后像 Excel 那样简单输入去管理，而是舍近求远去找 Api。并且这些 Api 并不完善，我找到的这几个 Api 虽说有实现简单的登录功能，但是必须通过浏览器进行使用。我目前能力不够，肯定做不到自己写程序实现这些功能。

但是现在做不到，不代表以后，不代表我现在不能有一个构想，我希望之后能够完全实现，所以将这个冗余的表留在这里。

4.3 对于 delphi 的看法？

我上手 delphi 很快，但是我有 Qt 开发的经验，我不知道其他同学如果没有图形界面开发经验会怎么样。

但我对于 delphi 的好评和坏评对半开，好评在于 delphi 简单，上手快，如果开发小型可视化软件绝对是利器中的利器；坏评在于 delphi 它本身基于 pascal 语言，pascal 语言在近十年中使用场景是越来越少，这造成的直接问题是如果开发 delphi 遇到 bug，能在网上检索到的信息很少，翻来覆去就是那些，甚至在 Google，StackOverflow 上找到的都是零几年的问题。最直接的一个体现就是我遇到的 socket 传中文乱码问题，我已经在发送之前将字符串转换为了 UTF8，但是 delphi 的发送函数会在发送的时候改变编码然后发送，必须给发送函数指定采用 UTF8 进行发送才不会乱码。就这么一个小小的问题，我花费了 10 个小时解决，原因是网上根本找不到资料，最后是在一个小角落里面发现他的函数比我的多一个参数，尝试之后发现问题解决了。解决之后才想到应该是发送函数不指定编码的话应该是全部转换成一个默认编码发送，这样导致的问题。但是如果是基于 C++的 Qt，遇到同样的问题找到的资料会比 delphi 多太多，解决难度也会低很多。所以为啥 delphi 的母公司又开发了一个基于 C++的类似 delphi 的 C++ Builder 其实也不难理解。

delphi 另一个好坏参半的点是他的 ide，delphi 的 ide 在很多地方不够智能，比如代码的自动提示，文件的更改等，尽管我安装了 cnpack，但是仍然感到不习惯。另外是 delphi 的调试也差 VS,Jetbrain 系列太多，当然这也可能和我刚接触 delphi，使用地没有之前的 ide 熟练的缘故。但是我觉得 delphi 的 ide 做的好的一点在于他可以同一个窗口开很多个项目，这个我认为还是很不错的，特别是在开发大系统的不同组件的时候。

4.4 对于 openGauss 数据库的看法？

这几天因为要准备 Ass4 的论文，所以了解了很多其他的数据库。当前世界上使用最多的数据库时 Oracle，MySQL 和 SQL Server，这三家前两个都是甲骨文公司的，Oracle 面向高安全性场景，MySQL 面向低端场景，SQL Server 是微软的，基本上绑定 Windows。

观察这三家以及 openGauss 的祖先 postgresql，我觉得 openGauss 的处境不是很好，openGauss 作为后起之秀，其实我觉得并没有太多出彩到可以和其他广泛使用的数据库掰手腕的地步，另外华为也没有 Windows 这样的操作系统来强绑定推广 openGauss 数据库。尽管华为有 openEuler 操作系统，但是作为 Linux 发行版，我很难认为 openEuler 有巨大的潜力。

我认为，当前，openGauss 数据库的对手应该是 postgresql 和 MySQL，其中一个是他祖先，二者功能相似性很高，后者有免费社区版，而且使用广泛。如果 openGauss 有一天能客观地和这二者比较，我觉得 openGauss 可能会有良好的前景。但是目前 openGauss 数据库显然不具备这样的能力。一个最简单的例子，如果现在我需要实际在工程中使用数据库，openGauss 并不能说服我去使用它。

但是可以看到目前 openGauss 在飞速发展中，希望未来也能良好地发展。