wechat聊天系统的数据库设计与实现

1. 引言

数据库课程设计是指对于一个给定的应用环境，构造设计优化的数据库逻辑模式和物理结构，并据此建立数据库及其应用系统，使之能够有效地存储和管理数据，满足用户的应用需求。信息管理要求是指在数据库中应该存储和管理哪些数据对象；数据操作要求是指对数据对象要进行哪些操作，如查询、加入、删除、修改、统计等操作。数据库课程设计的目标是为了用户和各种应用系统提供一个信息基础设施和高效率的运行环境。高效率的运行环境包括：数据库的存取效率、数据库存储空间的利用率、数据库系统运行管理的效率都是高的。本次课程设计的目的是把所学的数据库的知识应用到实践中去。现今知识已越来越受到人们的重视，仿WECHAT聊天系统是参考腾讯公司发布的腾讯WECHAT，主要运用数据库知识和java将其初步实现[1]。

**1.1课题背景**

现在的信息越来越发达，很多人都在使用WECHAT作为聊天工具，现根据腾讯公司的WECHAT聊天系统制造属于自己的聊天工具，表面上看上去是仿WECHAT聊天系统，其实等到时机成熟可以换成别的名字，相信在不远的将来在老师和同学们的帮助下，再借助互联网，我们一定可以打造出属于自己的聊天系统。当然系统存在着很多不足，需要老师的指点和帮助。

**1.2系统需求分析**

WECHAT聊天系统是大家广泛喜欢经常应用到CS应用程序软件[2]，而现在仿WECHAT聊天系统开发工具，是本人对其程序的深感兴趣，不但可以了解从中的业务，而且还可以加强学习，以及对知识的回顾。同时也是尽快测试此聊天程序，以便和同学进行用自己的工具来交流聊天，由于好奇心促使和同学们一起来探讨此程序的开发，这样增强同学们之间的相互交流，相互学习。

该系统属于应用程序,需要对数据的处理,比如聊天信息要既是反馈给对方,故对数据的及时更新要求较高,保证数据的真实性.该系统在任何操作系统下都可以运行,但必须装有SQL2000数据库。

**2 关键技术简介**

**2.1** **SQL Server2000**

SQL Server 2000 是Microsoft 公司2000年推出的SQL Server 数据库管理系统的版本。它是一个杰出的数据库平台，可用于大型联机事务处理、数据仓库、以及电子商务等。特点: 真正的客户机/服务器体系结构, 图形化用户界面, 丰富的编程接口工具。 SQL Server与Windows NT完全集成, 具有很好的伸缩性,支持Web技术。 SQL Server提供数据仓库功能[3]。

**2.2** **JAVA简介**

**Java**是一种可以撰写[跨平台](http://baike.baidu.com/view/469855.htm)应用程序的[面向对象](http://baike.baidu.com/view/125370.htm)的程序设计语言。[Java](http://baike.baidu.com/subview/29/12654100.htm) 技术具有卓越的通用性、高效性、平台移植性和安全性，广泛应用于[PC](http://baike.baidu.com/subview/13621/4882082.htm)、[数据中心](http://baike.baidu.com/view/754520.htm)、游戏控制台、科学[超级计算机](http://baike.baidu.com/view/296324.htm)、移动电话和互联网，同时拥有全球最大的开发者专业社群。

1：首先根据对其业务逻辑的分析极其全面整合，加之我们所熟悉的WECHAT聊天技术。第一应该做的是界面，界面的美化程度直接反映着用户对此软件喜欢程度，所以借助了互联网搜索了一些好看的图片；

2：对界面的设计，这里所用到的就是JAVA里布局控件，精心的布局，这一点众所周之JAVA里丰富的控件为我们提供了方便，这样做就可以尽可能的让用户用起来比较舒服；

3：各个窗体画好后，接下来自然就是业务逻辑层的代码书写了，由于学习不够精，所以在书写代码方便并没有用到三层技术，但是本人对该系统考虑的比较全面，一些错误和异常都可以将其打印出来

书写代码比较规范，不管从类的书写，还是一些命名都非常规范；

JAVA里提供丰富的控件库，本来就可以自动生成一些代码，对业务逻辑比较明朗，对方法和类之间的命名比较规范，书写代码自然比较快，出错率比较少；方法就是增删改查，

最后结合数据库技术完成业务逻辑。

**3 数据库概念结构设计**

**3.1 构思E-R图原则**

原则1：能独立存在的事物，例如人、物、事、活动、事项等等，在其有多个基本项描述的特性需要关注时，就应把它作为实体。

根据原则1，分析本系统中的实体主要有用户、星座、信息类型、聊天信息、好友策略、好友和血型。

原则2：两个或多个实体间的关联与结合，如主管、从属、组成、占有、作用、配合、协同等等，当需要予以关注时，应作为联系。联系通常是某类行为动作，E-R图关注的是其状态与结果而非其过程。

根据原则2,对本系统间的实体之间的关联进行分析,用户与星座、血型、好友策略之间有组成与被组成的关系;在这里解释一下所谓的好友策略,我们在聊天时,别人可能会加我们为好友,我们可以不通过系统的提示,自动允许,这是一种策略。另外一种策略就是别人要加我们为好友时,先通过系统信息请求我们是否同意。此外,WECHAT消息和消息类型之间也有组成与被组成的关系。

原则3：实体的属性是实体的本质特征。实体应有标识属性，并指定其中一个作为主标识。联系的属性是联系的结果或状态。属性具有如下几个特点：非多值性、非复合性、非导出性。实体的属性还应有非关联性。

根据原则3,分析本系统中的实体属性。WECHAT用户的属性包括WECHAT号(标识属性)、WECHAT密码、用户昵称、用户真实姓名、性别和年龄等;星座的属性包括星座的编号(标识属性)和名称;血型的属性包括血型编号(标识属性)和名称;好友策略的属性包括策略编号(标识属性)和策略名称;消息类型包括消息类型编号(标识属性)和消息类型名称;WECHAT消息的属性包括消息记录编号(标识属性)、消息记录名称、消息接受状态和接发消息的时间

原则4：所有基本项在同一E-R图中作为属性要在仅在一个地方出现[5]。

**3.2设计E-R图**

根据构思E-R图的原则联系本系统的需求分析，将数据库中所涉及的各个实体与它们各自的属性以及各个实体之间的关联用E-R图的形式表现出来，实体用方形表示，属性用椭圆表示，关联关系用没有箭头的直线连接，具体如下图3.1所示：



图3.1 WECHAT系统E-R图

**3.3 E-R图向关系模型转换原则**

(1) 一个1：1联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，每个实体的码均是关系的候选码。如果与某一端实体对应的关系模式合并，则需要在改关系模式的属性中加入另一个关系模式的码和联系本身的属性。

(2)一个1:n联系可以转换为一个独立的关系模式，也可以与n端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为n端实体的码。

(3)一个m:n联系转换为一个关系模式。与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，各实体的码组成关系的码或关系码的一部分。

(4)3个或3个以上实体间的一个多元联系可以转换为一个关系模式。与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，各实体的码组成关系的码或关系码的一部分。

(5)具有相同码的关系模式可合并[6]。

根据这5项实体间的联系将上述E-R图转换为关系模型

WECHAT消息（消息记录编号、消息、消息接受状态、发消息时间）

此为消息实体对应的关系模式，该关系模式已包含了联系“组成”所对应的关系模式。

消息类型（消息类型编号、消息类型名称）

此为消息类型实体对应的关系模式，该关系模式已包含了联系“组成”所对应的关系模式。

用户（WECHAT号、WECHAT密码、昵称、真实姓名、年龄、性别）

此为用户实体对应的关系模式，该关系模式已包含了联系“组成”所对应的关系模式。

星座（星座编号、星座名称）

此为星座实体对应的关系模式，该关系模式已包含了联系“组成”所对应的关系模式。

血型（血型编号、血型名称）

此为血型实体对应的关系模式，该关系模式已包含了联系“组成”所对应的关系模式。

好友策略（好友策略编号、好友策略名称）

此为好友策略实体对应的关系模式，该关系模式已包含了联系“组成”所对应的关系模式。

1. **数据库逻辑结构设计**

**4.1 数据库需求分析**

针对自己所要设计的仿WECHAT聊天系统的需求，设计如下所示的数据项和数据结构：

用户表：WECHAT号码、WECHAT密码、加好友的方式编号、昵称、WECHAT头像编号、性别、年龄、

实姓名、星座编号、血型编号

星座表：星座编号、星座名称

信息类型表：信息类型编号、信息类型

聊天信息表：聊天信息表记录编号、发送信息者WECHAT号、收到信息者WECHAT号、发送信息、信息类型编号、信息状态、发送时间

好友策略表：加好友的方式编号、加好友的方式设置

好友表：表添加记录、发送者的WECHAT、好友的WECHAT

血型表：血型编号、血型 以上内容如图4.1所示：



图4.1 数据流图

根据本系统的业务逻辑关系，得出了此数据流图，每一个箭头都带表着数据的流向，

而数据的流向直接反映给数据库，并将此信息保存在数据库,从而比较正确的设计出数据库。

**4.2 数据库初步关系框架**

User(用户表)( WECHAT号码、WECHAT密码、加好友的方式编号、昵称、WECHAT头像编号、性别、年龄、真实姓名、星座编号、血型编号)

Star(星座表)（星座编号、星座名称）

MessageType(信息类型表)( 信息类型编号、信息类型)

Messages(聊天信息表)( 聊天信息表记录编号、发送信息者WECHAT号、收到信息者WECHAT号、发送信息、信息类型编号、信息状态、发送时间)

FriendshipPolicy(好友策略表)( 加好友的方式编号、加好友的方式设置)

Friends(好友表)( 表添加记录、发送者的WECHAT、好友的WECHAT)

BloodType(血型表)( 血型编号、血型)

**4.3数据库关系模式优化**

该系统的数据库设计是满足数据库设计的第三范式。在设计数据库的时候例如对本数据库中任意一个表，例如用户表；

在关系模式R中的每一个具体关系r中，如果每个属性值 都是不可再分的最小数据单位，则称R是第一范式的关系。例如WECHAT号码，用户昵称，用户真实姓名，年龄（一个人可能只有一个用户昵称，一个WECHAT号码，用户的编号，一个姓名，电话号码……）

如果一是重复存储用户的WECHAT号码和姓名。这样，关键字只能是用户的编号。

很显然这种方法是不可取的；

如果用户的WECHAT号码是关键字，电话号码分为单位电话和住宅电话两个属性。

所以说：用户表中的列是不可再分的（即列的原子性），则可以说明该表满足第一范式！

如果关系模式R（U，F）中的所有非主属性都完全依赖于任意一个候选关键字，则称关系R 是属于第二范式的。

还拿出上面所说的用户表，例如每个用户都自己的详细信息（即，例如WeChat的头像编号，血型，星座……）

那么我们可以简单的举几个例子就可以很简单的看出；

1. 数据冗余，假设同一种血型由50个学生都一样，那么血型就重复了50次。
2. 更新异常，若调整了某种血型，相应的元组血型值都要更新，有可能会出现血型的不同。
3. .插入异常，如果科学家研究出来一种新的血型，由于没人选修，没有WeChat号作为关键字，只能等有人选修才能申请WeChat和录入血型成功。
4. 删除异常，如果WeChat管理员要删除过期的WeChat或者是很久没有人用的WeChat，从当前数据库删除选修记录。那么刚要申请WeChat号的人，则申请的WeChat号和个人的血型都无法保存。

原因是：血型都要依赖于血型的编号。

解决方法：分成两个关系模式分别在用户表血型编号，和血型表中的血型编号建立关系，即需要时再进行自然联接，恢复了原来的关系[7]。

所以说综合上面所叙述，此数据库满足第二范式。

如果关系模式R（U，F）中的所有非主属性对任何候选关键字都不存在传递信赖，则称关系R是属于第三范式的。

仍然按照上面的例子来说明，即例如每个用户都自己的详细信息（即，例如WeChat的头像编号，血型，星座……）

如果只按照WeChat作为关键字，没有部分依赖的问题，肯定是2NF。但这关系肯定有大量的冗余，有关用户属性将重复存储，插入，删除和修改时也将产生类似以上例的情况。

原因：关系中存在传递依赖造成的。假如要插入星座中的白马座，显然它不属于12星座，这样插入很显然是不会成功的，关键是用户星座的编号和星座表中的编号相互联系起来的，也就是函数决定是通过传递依赖。

解决目地：每个关系模式中不能留有传递依赖。

这样就满足了第三范式！

这样做的目的就是：规范化目的是使结构更合理，消除存储异常，使数据冗余尽量小，便于插入、删除和更新

原则：遵从概念单一化 "一事一地"原则，即一个关系模式描述一个实体或实体间的一种联系。规范的实质就是概念的单一化[8]。就是增加系统的稳定性以及系统的成本和后期的维护。

方法：将关系模式投影分解成两个或两个以上的关系模式。

要求：分解后的关系模式集合应当与原关系模式"等价"，即经过自然联接可以恢复原关系而不丢失信息，并保持属性间合理的联系。

从而在建立数据逻辑时能够更加清晰明白避免数据的冗杂，降低了出错的可能性使得该系统更加可靠。

通过上面的数据库关系优化之后我们就可以很简单的写出逻辑关系。

**4.4** **具体逻辑模式设计**

表4.1 用户表Users

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 字段描述 |
| Id | int | WECHAT号码（主键） |
| LoginPwd | varchar | WECHAT密码 |
| FriendshipPolicyId | int | 加为好友的方式编号 |
| NickName | varchar | 昵称 |
| FaceId | int | WECHAT头像的编号 |
| Sex | varchar | 性别 |
| Age | int | 年龄 |
| Name | varchar | 真实姓名 |
| StarId | int | 星座编号 |
| BloodTypeId | int | 血型编号 |

表4.2 星座表Star

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 字段描述 |
| Id | int | 星座编号（主键） |
| Star | varchar | 星座名称 |

表4.3信息类型表MessageType

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 字段描述 |
| Id | int | 信息类型编号（主键） |
| MessageType | varchar | 信息的类型 |

表4.4聊天信息表Messages

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 字段描述 |
| Id | int | 聊天信息表记录编号（主键） |
| FromUserId | int | 发送信息者WECHAT号（外键） |

续表4.4

|  |  |  |
| --- | --- | --- |
| ToUserId | int | 收到信息者的WECHAT号（外键） |
| Message | varchar | 发送的信息 |
| MessageTypeId | int | 信息的类型编号 |
| MessageState | int | 信息的状态（被接收到为1） |
| MessageTime | datetime | 发送的时间 |

表4.5好友策略表FriendshipPolicy

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 字段描述 |
| Id | int | 加为好友的方式编号（主键） |
| FriendshipPolicy | varchar | 加为好友的方式的设置 |

表4.6好友表Friends

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 字段描述 |
| Id | int | 好友表表添加的记录（主键） |
| HostId | int | 发送者的WECHAT（外键） |
| FriendId | int | 好友的WECHAT（外键） |

表4.7血型表BloodType

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 字段描述 |
| Id | int | 血型编号（主键） |
| BloodType | varchar | 血型 |

**4.5** **数据库设计表**

在上述每张表中，第一个数据都作为自己所在表的主键，而在表与表之间也有主外键关系存在，对用户表和星座表来说，星座表是主键表，用户表是外键表；对用户表和血型表来说，血型表是主键表，用户表是外键表；对用户表和好友策略表来说，好友策略表是主键表，用户表是外键表；对聊天信息表和信息类型表来说，信息类型表是主键表，聊天信息表是外键表。

表4.8 设计表Users

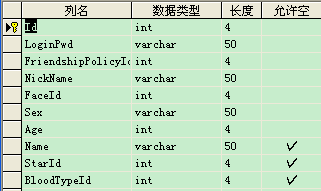


表4.9 设计表BloodType



表4.10 设计表Friends

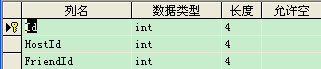


表4.11 设计表FriendshipPolicy



表4.12 设计表Messages

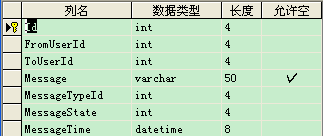


表4.13 设计表MessageType



表4.4 设计表Star



**4.6 关系图**

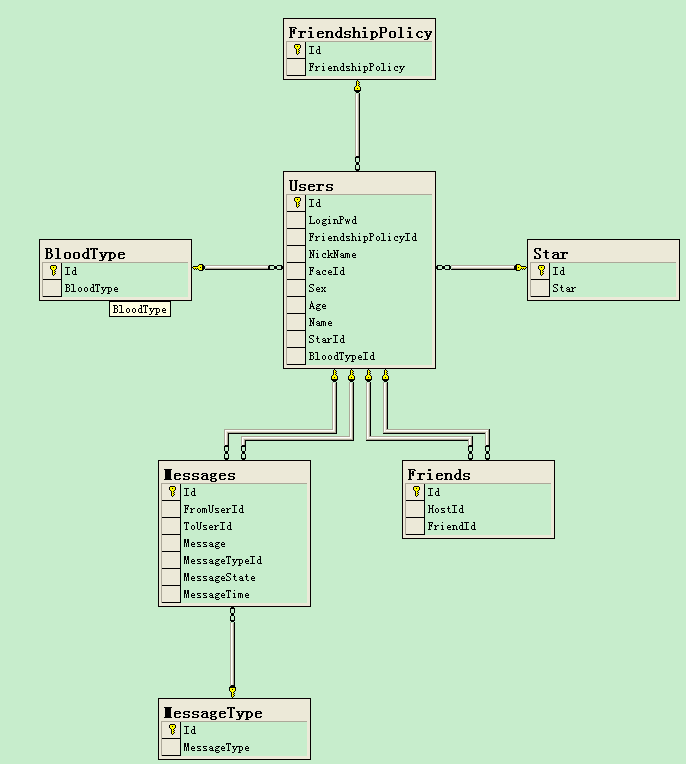
****

图4.2 关系图