**武汉大学计算机学院**

**本科生课程实验报告**

**数据库实验**

专 业 名 称 ：XXX

课 程 名 称 ：XXX

指 导 教 师 ：XXX 职称

学 生 学 号 ：XXX

学 生 姓 名 ：XXX

学 年 学 期 ：XXX

完 成 时 间 ：XXX

成 绩 ：

二○XX年X月

**郑 重 声 明**

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名： 日期：二○XX年X月

摘 要

数据库是一种用于存储和管理数据的软件系统，它可以实现数据的增删改查、备份恢复、安全控制、并发控制等功能。数据库的主要组成部分是数据模型、数据结构、数据操作语言和数据管理系统。

GaussDB是华为开发的一款分布式数据库，它支持多种数据模型，如关系型、文档型、图形型等，可以满足不同场景的数据处理需求。GaussDB具有高性能、高可用、高可靠、高安全等特点，可以应用于云计算、大数据、人工智能等领域。GaussDB采用开源的PostgreSQL作为内核，兼容SQL标准，支持JDBC、ODBC等多种接口，方便用户进行开发和使用。

微信是一款基于互联网的即时通讯和社交软件，它提供了多种功能，如聊天、朋友圈、支付、小程序等。微信的数据量非常庞大，需要使用数据库来存储和管理。数据库可以为微信提供数据的持久化、一致性、安全性、可扩展性等保障，同时也可以支持微信的数据分析、挖掘、推荐等业务。

为了学习和掌握数据库的基本概念、操作和应用，本实验利用GaussDB数据库，仿真微信的部分功能，例如用户登录、好友管理、转账记录、预警规则等。本实验采用Java语言开发程序，通过JDBC接口与GaussDB数据库建立连接，执行SQL语句来对数据进行各种操作，包括增加、删除、修改和查询，进而培养学生的数据库设计、编程和调试技能，同时加强学生对数据库和微信的理解和运用能力。

本实验旨在设计并实现GaussDBDemo程序，从而深入探究数据库管理系统的组成和功能。通过本实验，我们对数据库管理系统的内部结构和运行机制有了更加全面的认识。在这个过程中，我们不仅掌握了GaussDB数据库的特性和功能，还亲身实践了JDBC接口的使用和SQL语句的编写，为后续研究大数据和云计算领域的数据存储和处理方案打下坚实的基础。

**关键词：**数据库系统；GaussDB；JDBC；微信功能仿真

目录

[**数据库实验** 1](#_Toc170493479)

[摘 要 3](#_Toc170493480)

[**1 实验环境** 5](#_Toc170493481)

[**2 连接数据库** 10](#_Toc170493482)

[**3 SQL语句实现** 17](#_Toc170493483)

[**3.1 基本的Insert、update、delete编程** 17](#_Toc170493484)

[**3.2 SELECT查询编程** 19](#_Toc170493485)

[**3.3 SELECT思考题** 26](#_Toc170493486)

[**4 小结** 31](#_Toc170493487)

**1 实验环境**

数据库（Database）是指按照数据结构来组织、存储和管理数据的仓库，它可以实现数据的共享、保护和处理。数据库的主要特点是数据的独立性、冗余度低、易扩展、易维护等。

模式（Schema）是指对数据库中数据的逻辑结构和特征的描述，它是数据库的框架，决定了数据库的组织方式。模式分为三个层次：外模式、概念模式和内模式。外模式是指用户视图，即用户看到的数据库的部分数据，它反映了用户的需求和应用。概念模式是指全局视图，即数据库中所有数据的逻辑结构和约束，它反映了数据库的整体设计。内模式是指存储视图，即数据在物理存储设备上的存储方式和地址，它反映了数据库的物理结构和存储效率。

关系（Relation）是指一种二维表的数据结构，它是关系型数据库的基本组成单位。关系由行和列组成，每一行称为一个元组（Tuple），表示一个实体或对象的信息，每一列称为一个属性（Attribute），表示实体或对象的一个特征或特性。关系的特点是每个元组的属性值都是单值的，每个元组都是唯一的，每个属性都有一个名称和一个数据类型，属性的顺序和元组的顺序都是无关紧要的。关系的完整性约束包括实体完整性、参照完整性和用户定义的完整性。

实验所用的数据库环境是华为云GaussDB，其具备如下的配置参数：

表**1.1** 设备明细表

|  |  |  |
| --- | --- | --- |
| 设备名称 | 设备型号 | 软件版本 |
| 数据库 | GaussDB 4 核 | 16 GB | GaussDB服务 |

建立GaussDB数据库模式的步骤如下所示：

1. 第一步：登录华为云官网，选择IAM用户，输入租户名，IAM用户名和IAM用户密码，登录到供学生使用的华为云数据库。
2. 第二步：登录后，点击右上方的控制台，进入到控制台界面，选择左上角的按钮，在服务中选择数据库——>数据管理服务DAS，进入到数据管理页面。
3. 第三步：在DAS界面中点击进入开发工具，在开发工具界面中，选择“新增数据库登录实例”，在弹出的对话框中输入信息，点击测试连接，确认数据库连接正常。
4. 第四步：在DAS开发工具界面中，点击登录按钮，进入到数据库列表界面，选择指定的数据库（例如Wechat)，进入到该数据库模式列表。
5. 第五步：用户具备创建模式权限，可以创建模式，或者用下面的翻页箭头选择指定的模式。例如用户选择创建模式，在弹出的对话框中输入模式名称，并点击确定按钮。
6. 第六步：找到用户创建的模式，并点击，进入到该模式的对象管理界面，点击右上角的SQL窗口按钮，进入到SQL界面，用户输入SQL语句，并点击执行SQL命令，就可以完成相应的SQL操作。

在使用应用程序连接GaussDB数据库之前，需要进行一些配置和设置，以保证连接的成功和安全。连接前的准备工作主要分为以下两个步骤：

* 弹性公网IP的绑定：为了使应用程序能够通过互联网访问数据库，必须为 GaussDB 数据库实例分配一个弹性公网 IP 地址。在控制台的“连接信息”区域，可以找到主节点的“绑定”按钮，从而选择一个已经购买的弹性公网 IP 地址进行分配。分配完成后，需要记下这个 IP 地址，以便后续的连接。
* 开放8000端口：GaussDB数据库实例的安全组需配置 8000 端口，该端口为数据库的默认端口，应用程序与数据库的通信依赖于该端口。内网安全组的链接位于控制台的“节点列表”区域，点击后进入安全组的设置页面，在“入方向规则”中新增一条规则，端口号填写 8000，协议选择 TCP，授权对象设置为 0.0.0.0/0，最后点击“确定”。

在完成前述两个步骤之后，便可建立客户端与GaussDB数据库的连接。连接数据库时，应用程序须提供数据库实例的弹性公网IP地址、端口号、数据库名称、用户名和密码等参数，以创建连接对象和执行SQL语句。

接下来给出微信数据库模式示例中涉及的所有关系的集合，并用下划线标识主码属性：

UserInfo (UID, Name, RegTime, privilege)

Login (UID, LogTime, DID)

Device (DID, Name)

Friend (UID1, UID2, AddTime)

Messages (UID1, UID2, Content, SendTime, DID)

Transfer (UID1, UID2, Amount, SendTime, DID)

Warning (UID, ActTime)

其中各种关系所涉及的具体情境中的概念详细阐述如下：

1. 用户表UserInfo，各个字段的类型以及含义如表1.2。

表**1.2** 用户表UserInfo

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 含义 |
| UID | INT | 用户的微信号 |
| Name | VARCHAR(40) | 用户的微信名 |
| RegTime | TIMESTAMP | 用户注册的日期 |
| Balance | INT | 用户的余额表，值大于0. |
| Status | INT | 用户的状态，1:上线2:退出。 |
| Password | VARCHAR(10) | 存储用户的密码 |

1. 用户登录表Login，当用户申请成功微信后，用户每次登录微信后，都会在该表中增加一条记录，各个字段的类型以及含义如表1.3。

表**1.3** 登录表Login

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 含义 |
| UID | INT | 用户的微信号 |
| LogTime | TIMESTAMP | 用户的登录时间 |
| DID | INT | 设备的编号，具体的含义见Device表。 |

1. 设备编号表Device，各个字段的类型以及含义如表1.4。

表**1.4** 编号表Device

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 含义 |
| DID | INT | 设备编号，大于0. |
| Name | CHAR(20） | 设备的操作系统名称 |

1. 微信好友表Friend，各个字段的类型以及含义如表1.5，假设用户1和2是好友，在关系中存储元组(1,2)和(2,1)。

表**1.5** 微信好友表Friend

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 含义 |
| UID1 | INT | 用户1的微信号 |
| UID2 | INT | 用户2的微信号 |
| AddTime | TIMESTAMP | 用户1和用户2成为好友的时间 |

1. 用户好友之间的消息表Messages，各个字段的类型以及含义如表五。

表**1.6** 消息表Messages

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 含义 |
| UID1 | INT | 用户1的微信号 |
| UID2 | INT | 用户2的微信号 |
| Content | VARCHAR(200) | 用户1向用户2发送消息的内容。 |
| SendTime | TIMESTAMP | 发生消息的时间 |
| DID | INT | 发送消息的设备 |

1. 用户转账表Transfer，记录用户之间的转账信息，各个字段的类型以及含义如表1.7。

表**1.7** 转账表Transfer

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 含义 |
| UID1 | INT | 用户1的微信号 |
| UID2 | INT | 用户2的微信号 |
| Amount | INT | 用户1向用户2转账的余额。 |
| SendTime | TIMESTAMP | 发生消息的时间 |
| DID | INT | 发送消息的设备 |

1. 微信中存在很多诈骗行为，为此设了一个用户预警表Warning，列入此表中的用户账号将收到监控，各个字段的类型以及含义如表1.8。

表**1.7** 预警表Warning

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 含义 |
| UID | INT | 用户的微信号 |
| ActTime | TIMESTAMP | 最近一次转账的时间 |

**2 连接数据库**

本实验旨在利用Java语言开发一个简易的数据库应用程序，对微信数据进行增删改查的操作。实现该目标需借助JDBC（Java Database Connectivity）技术，它是一套规范的Java API，可用于接入和操纵各种数据库。JDBC提供了一组接口和类，用于执行SQL语句，处理查询结果，管理事务等。JDBC的优点是可以实现数据库的独立性，即同一套代码可以适用于不同的数据库，只需更换相应的驱动程序和连接参数即可。

数据库应用程序的编写和运行需要在Java环境下进行，因此，首先要对Java环境进行配置。这一过程涉及到Java开发工具包（JDK）的安装，Java环境变量的设置，以及Java集成开发环境（IDE）的选择。

请按照以下步骤下载并安装JDK：

* 访问Oracle官网，选择适合你的计算机系统的JDK安装包，建议使用java8版本。
* 下载完成后，双击安装包，按照提示进行安装，一般无需更改默认设置。

为了在命令行中使用Java命令，必须配置JDK环境变量。在系统属性中，要设置以下三个环境变量：

* JAVA\_HOME：JDK的安装目录
* path：JDK的bin目录和jre的bin目录
* CLASSPATH：Java类路径，通常为“.”即可

具体的设置步骤可以参考网上的教程。

使用JDBC驱动程序连接GaussDB云数据库应遵循以下原则：

1. 下载并安装GaussDB的JDBC驱动程序，它是一个jar文件，用于实现Java与GaussDB的通信。华为云官网提供了GaussDB的客户端工具包，下载后进行解压操作，根据数据库实例类型和系统架构选择合适的驱动包，例如GaussDB-Kernel-V500R002C10-EULER-64bit-Jdbc.tar.gz，再次解压后，即可获取opengaussjdbc.jar文件。我们需要将这个jar文件放在Java的类路径下，或者在编译和运行时指定它的位置。
2. 调用Class.forName方法加载和注册JDBC驱动程序，该方法会把驱动程序类的字节码载入到内存，并在DriverManager中自动进行注册。其需要传入驱动程序类的全限定名作为参数，如"com.huawei.opengauss.jdbc.Driver"。
3. 数据库连接的建立，依赖于DriverManager.getConnection方法，该方法返回一个Connection对象，代表了与数据库的连接状态。该方法需要接收三个参数，分别是数据库的URL，用户名和密码，如"jdbc:opengauss://110.41.120.191:8000/Wechat?ApplicationName=app1"，"root"和"abc888888@"。
4. 通过Connection.createStatement方法，可以创建并执行SQL语句，该方法返回一个Statement对象，代表了一个SQL语句的实例。Statement对象提供了Statement.executeUpdate和Statement.executeQuery两种方法，分别用于执行更新语句（如INSERT、UPDATE、DELETE等）和查询语句（如SELECT等）。执行结果可以是一个int值，表示受影响的行数，或者一个ResultSet对象，表示查询的结果集。
5. ResultSet对象用于处理查询结果，它是一个游标，可以遍历结果集中的每一行。我们可以调用ResultSet.next方法，将游标移动到下一行，如果返回值为true，说明还有未读取的数据，如果为false，说明已经读完所有数据。我们可以根据列名或列索引，使用ResultSet.getInt、ResultSet.getString等方法，获取当前行的某一列的值。
6. 为了释放与数据库的连接，关闭SQL语句和结果集等，我们需要调用Connection.close、Statement.close、ResultSet.close等方法来关闭资源。这些资源都是有限的，所以我们应该在不需要使用的时候尽快关闭它们，防止资源浪费和内存泄漏的发生。

实验代码的基本架构如下所示，第3节将展示SQL功能的具体实现方式。：

import java.sql.\*;

import java.util.Scanner;

public class GaussDBDemo {

static final String JDBC\_DRIVER = "com.huawei.opengauss.jdbc.Driver";

static final String DB\_URL = "jdbc:opengauss://110.41.120.191:8000/Wechat?ApplicationName=app1";

// 数据库的用户名与密码，需要根据自己的设置

static final String USER = "root";

static final String PASS = "abc888888@";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

Scanner sc = new Scanner(System.in); // 创建一个扫描器对象，用于接收用户输入

try {

// 注册 JDBC 驱动

Class.forName(JDBC\_DRIVER);

// 打开链接

System.out.println("连接数据库...");

conn = DriverManager.getConnection(DB\_URL, USER, PASS);

// 执行查询

System.out.println(" 实例化 Statement 对象...");

stmt = conn.createStatement();

String sql;

ResultSet rs;

int choice; // 用于存储用户的选择

do {

System.out.println("\n欢迎使用GaussDBDemo程序，我们提供以下服务：");

System.out.println("1. 简单增删改");

System.out.println("2. 查询");

System.out.println("3. 进阶查询");

System.out.println("0. 退出程序");

System.out.println("请输入你想要执行的功能编号：");

choice = sc.nextInt(); // 接收用户输入的整数

switch (choice) {

case 1:

// 功能1：执行简单增删改功能

// 省略具体的功能实现

break;

case 2:

// 功能2：执行查询功能

// 省略具体的功能实现

break;

case 3:

// 功能3：执行进阶查询功能

// 省略具体的功能实现

break;

case 0:

// 退出程序

System.out.println("\n感谢使用，再见！");

break;

default:

// 输入无效的编号

System.out.println("请输入有效的编号！");

break;

}

} while (choice != 0); // 循环输入直到用户输入0

// 完成后关闭

stmt.close();

conn.close();

} catch (SQLException se) {

// 处理 JDBC 错误

se.printStackTrace();

} catch (Exception e) {

// 处理 Class.forName 错误

e.printStackTrace();

} finally {

// 关闭资源

try {

if (stmt != null)

stmt.close();

} catch (SQLException se2) {

} // 什么都不做

try {

if (conn != null)

conn.close();

} catch (SQLException se) {

se.printStackTrace();

}

if (sc != null) // 关闭扫描器对象

sc.close();

}

System.out.println("Goodbye!");

}

}

程序中可添加简易语句，以检验连接是否成功。

把Java源码和jar驱动放在同一目录，运行如下命令即可编译并执行：

javac -encoding utf-8 -cp .\opengaussjdbc.jar GaussDBDemo.java;java -cp ".;./opengaussjdbc.jar" GaussDBDemo

这两条命令的意思是：

* 用Java编译器，以utf-8的编码格式，把GaussDBDemo.java这个文件编译成字节码文件，并且在编译的过程中，需要用到d:\download\opengaussjdbc.jar这个jar文件里的类和资源。
* 用Java虚拟机，以.;d:/download/opengaussjdbc.jar这个类路径，运行GaussDBDemo这个Java程序。

这里需要注意的是两个方面：

1. Windows系统支持两种路径分隔符，反斜杠\和斜杠/。反斜杠是Windows的默认分隔符，但是斜杠也可以被识别，因为它是Unix和Linux系统的标准分隔符。所以，你可以用任何一种分隔符来输入路径，Windows都会正确地处理它。
2. -cp .;d:/download/opengaussjdbc.jar是一个选项，它指定了类路径（classpath），也就是Java运行时环境需要查找类定义的位置。这里是.;d:/download/opengaussjdbc.jar，这是两个路径，用分号隔开，表示同时查找两个地方。第一个是.，表示当前目录，也就是运行命令的目录，这里假设是GaussDBDemo.class文件所在的目录。第二个是d:/download/opengaussjdbc.jar，这是一个jar文件，也就是Java的打包文件，它包含了一些类和资源，这里是OpenGauss的JDBC驱动，它可以让Java程序连接和操作OpenGauss数据库。

分号(;)在Windows操作系统的命令提示符（CMD）中是结束命令的标志。但是，在Java命令的环境下，分号是用来区分类路径（classpath）中的不同项目的。运行这个命令时，应该用分号隔开路径。如果在Windows的CMD中出现问题，要把整个类路径字符串用引号括起来，这样可以避免分号被误解。

**3** **SQL语句实现**

本次实验，我们使用了简单的switch语句来实现不同的SQL语句功能。

整体结构如下：

1. **int** choice;//用户输入
2. **switch**(choice){
3. **case** 1:
4. funct1();
5. **break**;
6. **case** 2:
7. funct2();
8. **break**;
9. **case** 3:
10. .......
11. }

**3.1 基本的Insert、update、delete编程**

**3.1.1 用户1退出微信，用SQL语句更新用户的状态。**

1. // 增删改1：更新用户1的状态为2（离线）
2. sql = "UPDATE \"Schema\_ljh\_zzh\".UserInfo SET Status = 2 WHERE UID = 1";
3. stmt.executeUpdate(sql); // 执行更新语句
4. System.out.println("用户1的状态已更新为离线。");
5. **break**;

**3.1.2 鸿蒙操作系统名称是Harmony，在登录设备中增加一项，设设备编号是6。**

1. // 增删改2：在登录设备中增加一项，设备编号为6，名称为Harmony
2. sql = "INSERT INTO \"Schema\_ljh\_zzh\".Device VALUES (6, 'Harmony')";
3. stmt.executeUpdate(sql); // 执行插入语句
4. System.out.println("登录设备中已增加一项，设备编号为6，名称为Harmony。");
5. **break**;

**3.1.3 华为发布了openEulerc操作系统，请在登录设备中增加一项。**

1. // 增删改3：在登录设备中增加一项，设备编号为最大编号加1，名称为openEuler
2. sql = "INSERT INTO \"Schema\_ljh\_zzh\".Device(DID, Name) SELECT MAX(DID) + 1, 'openEuler' FROM \"Schema\_ljh\_zzh\".Device";
3. stmt.executeUpdate(sql); // 执行插入语句
4. System.out.println("登录设备中已增加一项，设备编号为最大编号加1，名称为openEuler。");
5. **break**;

**3.1.4 清除预警表中的所有历史记录。**

1. // 增删改4：删除预警表中的所有记录
2. sql = "DELETE FROM \"Schema\_ljh\_zzh\".Warning";
3. stmt.executeUpdate(sql); // 执行删除语句
4. System.out.println("预警表中的所有记录已删除。");
5. **break**

**3.1.5 预警规则1是单笔转账超过400的用户，将该用户的微信UID以及转账的时间存入Warning表。**

1. // 增删改5：将单笔转账超过400的用户的微信UID以及转账的时间存入Warning表
2. sql = "INSERT INTO \"Schema\_ljh\_zzh\".Warning SELECT UID1, SendTime FROM \"Schema\_ljh\_zzh\".Transfer WHERE Amount >= 400";
3. stmt.executeUpdate(sql); // 执行插入语句
4. System.out.println("单笔转账超过400的用户的微信UID以及转账的时间已存入Warning表。");
5. **break**;

**3.1.6 预警规则2是累计转出超过1000的用户以及最后一次转账时间存入预警表。**

1. // 增删改6：将累计转出超过1000的用户以及最后一次转账时间存入预警表
2. sql = "INSERT INTO \"Schema\_ljh\_zzh\".Warning SELECT UID1, MAX(SendTime) FROM \"Schema\_ljh\_zzh\".Transfer GROUP BY UID1 HAVING SUM(Amount) >= 1000";
3. stmt.executeUpdate(sql); // 执行插入语句
4. System.out.println("累计转出超过1000的用户以及最后一次转账时间已存入预警表。");
5. **break**;

**3.2 SELECT查询编程**

**3.2.1 列出每一个用户在每一个设备类型上登录微信次数,按照微信号从小到大输出，相同的微信号，按照登录次数从高到低输出。**

1. // 查询1：列出每一个用户在每一个设备类型上登录微信次数,按照微信号从小到大输出，相同的微信号，按照登录次数从高到低输出。
2. sql = "SELECT UID, Name, COUNT(\*) AS Total FROM \"Schema\_ljh\_zzh\".Login, \"Schema\_ljh\_zzh\".Device WHERE Login.DID = Device.DID GROUP BY UID, Name ORDER BY UID, Total DESC";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户ID\t设备名称\t登录次数");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt("UID");
8. String name = rs.getString("Name");
9. **int** total = rs.getInt("Total");
10. // 输出数据
11. System.out.print(uid + "\t" + name + "\t" + total + "\n");
12. }
13. rs.close(); // 关闭结果集对象
14. **break**;

**3.2.2 列出用户1的好友UID。**

1. // 查询2：列出用户1的好友UID。
2. sql = "SELECT UID2 FROM \"Schema\_ljh\_zzh\".Friend WHERE UID1 = 1";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户1的好友UID：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid2 = rs.getInt("UID2");
8. // 输出数据
9. System.out.print(uid2 + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.3 列出每一个用户的好友数目。**

1. // 查询3：列出每一个用户的好友数目。
2. sql = "SELECT UID1, COUNT(\*) AS Friends FROM \"Schema\_ljh\_zzh\".Friend GROUP BY UID1 ORDER BY UID1";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户ID\t好友数目");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("UID1");
8. **int** friends = rs.getInt("Friends");
9. // 输出数据
10. System.out.print(uid1 + "\t" + friends + "\n");
11. }
12. rs.close(); // 关闭结果集对象
13. **break**;

**3.2.4 列出用户1和用户3的共同好友。**

1. // 查询4：列出用户1和用户3的共同好友。
2. sql = "SELECT f1.UID2 FROM \"Schema\_ljh\_zzh\".Friend f1 WHERE f1.UID1 = 1 AND f1.UID2 IN (SELECT f2.UID2 FROM \"Schema\_ljh\_zzh\".Friend f2 WHERE f2.UID1 = 3)";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户1和用户3的共同好友：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid2 = rs.getInt("UID2");
8. // 输出数据
9. System.out.print(uid2 + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.5 统计用户1和用户3的共同好友的数量。**

1. // 查询5：统计用户1和用户3的共同好友的数量。
2. sql = "SELECT COUNT(\*) AS Common FROM \"Schema\_ljh\_zzh\".Friend f1, \"Schema\_ljh\_zzh\".Friend f2 WHERE f1.UID1 = 1 AND f2.UID1 = 3 AND f1.UID2 = f2.UID2";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户1和用户3的共同好友的数量：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** common = rs.getInt("Common");
8. // 输出数据
9. System.out.print(common + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.6 统计每一个用户对之间的好友数量，如果他们之间没有好友就不显示，每个用户对只显示一次，比如1和2有5个好友，那么显示（1，2，5）或者（2，1，5）。**

1. // 查询6：统计每一个用户对之间的好友数量，如果他们之间没有好友就不显示，每个用户对只显示一次，比如1和2有5个好友，那么显示（1，2，5）或者（2，1，5）。
2. sql = "SELECT f1.UID1 AS UID1\_1, f2.UID1 AS UID1\_2, COUNT(\*) AS Mutual FROM \"Schema\_ljh\_zzh\".Friend f1, \"Schema\_ljh\_zzh\".Friend f2 WHERE f1.UID1 < f2.UID1 AND f1.UID2 = f2.UID2 GROUP BY f1.UID1, f2.UID1";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("每一个用户对之间的好友数量：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("UID1\_1");
8. **int** uid2 = rs.getInt("UID1\_2");
9. **int** mutual = rs.getInt("Mutual");
10. // 输出数据
11. System.out.print("(" + uid1 + "," + uid2 + "," + mutual + ")\n");
12. }
13. rs.close(); // 关闭结果集对象
14. **break**;

**3.2.7 显示没有共同好友的用户对。**

1. // 查询7：显示没有共同好友的用户对。
2. sql = "SELECT u1.UID AS UID1, u2.UID AS UID2 FROM \"Schema\_ljh\_zzh\".UserInfo u1, \"Schema\_ljh\_zzh\".UserInfo u2 WHERE u1.UID < u2.UID AND NOT EXISTS (SELECT \* FROM \"Schema\_ljh\_zzh\".Friend f1, \"Schema\_ljh\_zzh\".Friend f2 WHERE f1.UID1 = u1.UID AND f2.UID1 = u2.UID AND f1.UID2 = f2.UID2)";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("没有共同好友的用户对：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("UID1");
8. **int** uid2 = rs.getInt("UID2");
9. // 输出数据
10. System.out.print("(" + uid1 + "," + uid2 + ")\n");
11. }
12. rs.close(); // 关闭结果集对象
13. **break**;

**3.2.8 查询聊天记录中包含‘账号’两个字的用户聊天记录。**

1. // 查询8：查询聊天记录中包含‘账号’两个字的用户聊天记录。
2. sql = "SELECT \* FROM \"Schema\_ljh\_zzh\".Messages WHERE Content LIKE '%账号%'";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.printf("%-8s\t%-8s\t%-16s\t%-24s\t%-8s\n", "发送者ID", "接收者ID", "内容", "发送时间","设备ID");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("UID1");
8. **int** uid2 = rs.getInt("UID2");
9. String content = rs.getString("Content");
10. Timestamp sendTime = rs.getTimestamp("SendTime");
11. **int** did = rs.getInt("DID");
12. // 输出数据
13. System.out.printf("%-8d\t%-8d\t%-16s\t%-24s\t%-8d\n", uid1, uid2, content, sendTime,did);
14. }
15. rs.close(); // 关闭结果集对象
16. **break**;

**3.2.9 查询只有聊天记录，但是没有转账记录的用户微信UID。**

1. // 查询9：查询只有聊天记录，但是没有转账记录的用户微信UID。
2. sql = "(SELECT UID1 FROM \"Schema\_ljh\_zzh\".Messages UNION SELECT UID2 FROM \"Schema\_ljh\_zzh\".Messages) EXCEPT (SELECT UID1 FROM \"Schema\_ljh\_zzh\".Transfer UNION SELECT UID2 FROM \"Schema\_ljh\_zzh\".Transfer)";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("只有聊天记录，但是没有转账记录的用户微信UID：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt(1);
8. // 输出数据
9. System.out.print(uid + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.10 统计用户1主动向哪些用户发送至少5条消息，列出聊天对象和消息数。**

1. // 查询10：统计用户1主动向哪些用户发送至少5条消息，列出聊天对象和消息数。
2. sql = "SELECT UID2, COUNT(\*) AS Total FROM \"Schema\_ljh\_zzh\".Messages WHERE UID1 = 1 GROUP BY UID2 HAVING Total >= 5";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户1主动向以下用户发送至少5条消息：");
5. System.out.printf("%-8s\t%-8s\n", "聊天对象", "消息数");
6. **while** (rs.next()) {
7. // 通过字段检索
8. **int** uid2 = rs.getInt("UID2");
9. **int** total = rs.getInt("Total");
10. // 输出数据
11. System.out.printf("%-8d\t%-8d\n", uid2, total);
12. }
13. rs.close(); // 关闭结果集对象
14. **break**;

**3.2.11 按照用户统计该用户分在2023年分别向哪些用户发送了多少条消息，列出用户微信UID1，接受对象的微信UID2，消息总条数。**

1. // 查询11：按照用户统计该用户在2023年分别向哪些用户发送了多少条消息，列出用户微信UID1，接受对象的微信UID2，消息总条数。
2. sql = "SELECT UID1, UID2, COUNT(\*) AS Total FROM \"Schema\_ljh\_zzh\".Messages WHERE date\_part('year', SendTime) = 2023 GROUP BY UID1, UID2";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.printf("%-8s\t%-8s\t%-8s\n", "用户", "接受对象", "消息总条数");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("UID1");
8. **int** uid2 = rs.getInt("UID2");
9. **int** total = rs.getInt("Total");
10. // 输出数据
11. System.out.printf("%-8d\t%-8d\t%-8d\n", uid1, uid2, total);
12. }
13. rs.close(); // 关闭结果集对象
14. **break**;

**3.2.12 查询使用‘KylinOS’登录微信账号的用户名。**

1. // 查询12：查询使用‘KylinOS’登录微信账号的用户名
2. sql = "SELECT U.Name FROM (\"Schema\_ljh\_zzh\".UserInfo U JOIN \"Schema\_ljh\_zzh\".Login L ON U.UID = L.UID) JOIN \"Schema\_ljh\_zzh\".Device D ON L.DID = D.DID WHERE D.Name = 'KylinOS'";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("使用‘KylinOS’登录微信账号的用户名：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. String name = rs.getString("Name");
8. // 输出数据
9. System.out.print(name + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.13 查询只使用KylinOS设备登录微信的用户账号。**

1. // 查询13：查询只使用KylinOS设备登录微信的用户账号
2. sql = "SELECT UID FROM \"Schema\_ljh\_zzh\".Login L JOIN \"Schema\_ljh\_zzh\".Device D ON L.DID = D.DID GROUP BY UID HAVING COUNT(DISTINCT CASE WHEN D.Name = 'KylinOS' THEN L.DID END) = COUNT(DISTINCT L.DID)";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("只使用KylinOS设备登录微信的用户账号：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt("UID");
8. // 输出数据
9. System.out.print(uid + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.14 至少在两种设备上登录过微信账号的用户UID。**

1. // 查询14：至少在两种设备上登录过微信账号的用户UID
2. sql = "SELECT UID FROM \"Schema\_ljh\_zzh\".Login GROUP BY UID HAVING COUNT(DISTINCT DID) >= 2";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("至少在两种设备上登录过微信账号的用户UID：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt("UID");
8. // 输出数据
9. System.out.print(uid + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.15 查询在所有设备上均实现登录的微信用户名。**

1. // 查询15：查询在所有设备上均实现登录的微信用户名。
2. sql = "SELECT Name FROM \"Schema\_ljh\_zzh\".UserInfo U JOIN ( SELECT UID FROM \"Schema\_ljh\_zzh\".Login GROUP BY UID HAVING COUNT(DISTINCT DID) = ( SELECT COUNT(\*) FROM \"Schema\_ljh\_zzh\".Device ) ) AS AllDevices ON U.UID = AllDevices.UID";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("在所有设备上均实现登录的微信用户名：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. String name = rs.getString("Name");
8. // 输出数据
9. System.out.print(name + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.2.16 查询连续七天之内有两笔向外转账的用户微信ID。**

1. // 查询16：查询连续七天之内有两笔向外转账的用户微信ID
2. sql = "SELECT DISTINCT UID1 FROM \"Schema\_ljh\_zzh\".Transfer t1 WHERE EXISTS ( SELECT \* FROM \"Schema\_ljh\_zzh\".Transfer t2 WHERE t1.UID1 = t2.UID1 AND t1.SendTime < t2.SendTime AND TIMESTAMPDIFF(DAY, t1.SendTime, t2.SendTime) <= 7 )";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("连续七天之内有两笔向外转账的用户微信ID：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("UID1");
8. // 输出数据
9. System.out.print(uid1 + "\n");
10. }
11. rs.close(); // 关闭结果集对象
12. **break**;

**3.3 SELECT思考题**

前提：对于微信好友表Friend(UID1,UID2,AddTime)，如果用户1和2是好友，在关系中存储元组(1,2)，或者(2, 1)，不会同时存储(1,2)和(2, 1)。

**3.3.1 重建Friend表。**

1. // 进阶查询1：重建Friend表
2. stmt.executeUpdate("DELETE FROM \"Schema\_ljh\_zzh\".Friend"); // 清空Friend表
3. sql = "INSERT INTO \"Schema\_ljh\_zzh\".Friend (UID1, UID2, AddTime) VALUES"; // 定义插入语句的前半部分
4. System.out.println("请输入你想要重建的Friend表的元组，元组间以逗号分隔，以分号结束：");
5. sc.nextLine(); // 清除缓冲区
6. String rebuild = ""; // 初始化重建语句为空字符串
7. String line = ""; // 初始化每一行的输入为空字符串
8. **do** {
9. line = sc.nextLine(); // 接收用户输入的一行
10. rebuild += line + "\n"; // 将每一行的输入拼接到重建语句中，并换行
11. } **while** (!line.endsWith(";")); // 循环直到用户输入的一行以分号结束
12. **try** {
13. stmt.executeUpdate(sql + rebuild); // 执行重建语句
14. System.out.println("Friend表已重建。");
15. } **catch** (SQLException se) {
16. // 处理SQL错误
17. System.out.println("重建Friend表失败，请检查你输入的元组格式是否正确。");
18. se.printStackTrace();
19. }
20. **break**;

**3.3.2 列出用户1的好友。**

1. // 进阶查询2：列出用户1的好友
2. sql = "SELECT UID2 FROM \"Schema\_ljh\_zzh\".Friend WHERE UID1 = 1 UNION SELECT UID1 FROM \"Schema\_ljh\_zzh\".Friend WHERE UID2 = 1";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户1的好友：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt("UID2");
8. // 输出数据
9. System.out.print(uid + "\n");
10. }
11. rs.close(); // 关闭结果集对象

**break**;

**3.3.3 统计每一个用户的好友数目。**

1. // 进阶查询3：统计每一个用户的好友数目
2. sql = "SELECT f1.UID1, count(\*) + (SELECT COUNT(\*) FROM \"Schema\_ljh\_zzh\".Friend F2 WHERE f2.uid2 = f1.uid1) AS Friends FROM \"Schema\_ljh\_zzh\".Friend f1 GROUP BY f1.UID1 UNION SELECT f3.UID2 AS UID, count(\*) + (SELECT COUNT(\*) FROM \"Schema\_ljh\_zzh\".Friend F4 WHERE f4.uid1 = f3.uid2) AS Friends FROM \"Schema\_ljh\_zzh\".Friend f3 GROUP BY f3.UID2";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("用户ID\t好友数目");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt("UID1");
8. **int** friends = rs.getInt("Friends");
9. // 输出数据
10. System.out.print(uid + "\t" + friends + "\n");
11. }
12. rs.close(); // 关闭结果集对象
13. **break**;

**3.3.4 列出具有4个共同好友以上（包含），但自身不是好友的用户对。**

1. // 进阶查询4：列出具有4个共同好友以上（包含），但自身不是好友的用户对，每一个用户对只列一次
2. sql = "SELECT a AS uid1, b AS uid2 FROM (SELECT f1.uid1 AS a, f2.uid1 AS b, f1.uid2 AS c FROM \"Schema\_ljh\_zzh\".Friend f1, \"Schema\_ljh\_zzh\".Friend f2 WHERE f1.uid2 = f2.uid2 AND f1.uid1 < f2.uid1 UNION SELECT f3.uid1 AS a, f4.uid2 AS b, f3.uid2 AS c FROM \"Schema\_ljh\_zzh\".Friend f3, \"Schema\_ljh\_zzh\".Friend f4 WHERE f3.uid2 = f4.uid1 AND f3.uid1 < f4.uid2 UNION SELECT f5.uid2 AS a, f6.uid2 AS b, f5.uid1 AS c FROM \"Schema\_ljh\_zzh\".Friend f5, \"Schema\_ljh\_zzh\".Friend f6 WHERE f5.uid1 = f6.uid1 AND f5.uid2 < f6.uid2 UNION SELECT f7.uid2 AS a, f8.uid1 AS b, f7.uid1 AS c FROM \"Schema\_ljh\_zzh\".Friend f7, \"Schema\_ljh\_zzh\".Friend f8 WHERE f7.uid1 = f8.uid2 AND f7.uid2 < f8.uid1) tmp WHERE tmp.a NOT IN (SELECT f9.uid1  FROM \"Schema\_ljh\_zzh\".Friend f9  WHERE tmp.b = f9.uid2 UNION SELECT f10.uid2 FROM \"Schema\_ljh\_zzh\".Friend f10  WHERE tmp.b = f10.uid1) GROUP BY a, b HAVING count(\*) >= 4";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("具有4个共同好友以上，但自身不是好友的用户对：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("uid1");
8. **int** uid2 = rs.getInt("uid2");
9. // 输出数据
10. System.out.print("(" + uid1 + "," + uid2 + ")\n");
11. }
12. rs.close(); // 关闭结果集对象
13. **break**;

**3.3.5 查询2024年3月份，既有转出又有转入的用户，以及各自相应的总金额。**

1. // 进阶查询5：查询2024年3月份，既有转出又有转入的用户，以及各自相应的总金额
2. sql = "SELECT UID1, tmp1.AllOut, tmp2.AllIn FROM (SELECT UID1, SUM(amount) AS AllOut FROM \"Schema\_ljh\_zzh\".Transfer WHERE SENDTIME BETWEEN '2024-03-01 00:00:00' AND '2024-03-31 23:59:59' GROUP BY UID1) tmp1 JOIN (SELECT UID2, SUM(amount) AS AllIn FROM \"Schema\_ljh\_zzh\".Transfer WHERE SENDTIME BETWEEN '2024-03-01 00:00:00' AND '2024-03-31 23:59:59' GROUP BY UID2) tmp2 ON tmp1.UID1 = tmp2.UID2";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.printf("%-8s\t%-8s\t%-8s\n", "用户ID", "转出金额", "转入金额");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt("UID1");
8. **int** allOut = rs.getInt("AllOut");
9. **int** allIn = rs.getInt("AllIn");
10. // 输出数据
11. System.out.printf("%-8s\t%-8s\t%-8s\n", uid, allOut, allIn);
12. }
13. rs.close(); // 关闭结果集对象

**break**;

**3.3.6 查询2024年3月份，每一个用户转出总额，转入总额，如果两项都为零，不需要列出。如果一项有，另外一项没有，没有的显示为0。**

1. // 查询6：查询2024年3月份，每一个用户转出总额，转入总额，如果两项都为零，不需要列出。如果一项有，另外一项没有，没有的显示为0。例如用户3只收到1笔1000转入金额，那么显示（3，null，1000）
2. sql = "SELECT UID1, tmp1.AllOut, tmp2.AllIn FROM (SELECT UID1, SUM(amount) AS AllOut FROM \"Schema\_ljh\_zzh\".Transfer WHERE SENDTIME BETWEEN '2024-03-01 00:00:00' AND '2024-03-31 23:59:59' GROUP BY UID1) tmp1 LEFT JOIN (SELECT UID2, SUM(amount) AS AllIn FROM \"Schema\_ljh\_zzh\".Transfer WHERE SENDTIME BETWEEN '2024-03-01 00:00:00' AND '2024-03-31 23:59:59' GROUP BY UID2) tmp2 ON tmp1.UID1 = tmp2.UID2 UNION SELECT UID2, tmp3.AllOut, tmp4.AllIn FROM (SELECT UID1, SUM(amount) AS AllOut FROM \"Schema\_ljh\_zzh\".Transfer WHERE SENDTIME BETWEEN '2024-03-01 00:00:00' AND '2024-03-31 23:59:59' GROUP BY UID1)tmp3 RIGHT JOIN (SELECT UID2, SUM(amount) AS AllIn FROM \"Schema\_ljh\_zzh\".Transfer WHERE SENDTIME BETWEEN '2024-03-01 00:00:00' AND '2024-03-31 23:59:59' GROUP BY UID2) tmp4 ON tmp3.UID1 = tmp4.UID2";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("每一个用户转出总额，转入总额：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid = rs.getInt("UID1");
8. **int** allOut = rs.getInt("AllOut");
9. **int** allIn = rs.getInt("AllIn");
10. // 输出数据
11. System.out.print("(" + uid + "," + allOut + "," + allIn + ")\n");
12. }
13. rs.close(); // 关闭结果集对象
14. **break**;

**3.3.7 统计每一对用户之间的发送信息的数目，例如1向2发送了5条信息，同时2向139871了4条消息，那么他们之间的消息就是9。**

1. // 查询7：统计每一对用户之间的发送信息的数目，例如1向2发送了5条信息，同时2向1发送了4条消息，那么他们之间的消息就是9条
2. sql = "SELECT tmp1.UID1,tmp1.UID2,tmp1.total+tmp2.total AS total FROM (SELECT UID1,UID2,Count(\*) AS total FROM \"Schema\_ljh\_zzh\".Messages GROUP BY UID1,UID2) tmp1 JOIN (SELECT UID1,UID2,Count(\*) AS total FROM \"Schema\_ljh\_zzh\".Messages GROUP BY UID1,UID2) tmp2 ON tmp1.UID1=tmp2.uid2 and tmp1.uid2=tmp2.uid1 UNION SELECT tmp5.UID1,tmp5.UID2,tmp5.total1 FROM ((SELECT UID1,UID2,Count(\*) as total1 FROM \"Schema\_ljh\_zzh\".Messages GROUP BY UID1,UID2) tmp3 LEFT  JOIN (select UID1,UID2,Count(\*) as total2 FROM \"Schema\_ljh\_zzh\".Messages GROUP BY UID1,UID2) tmp4 ON tmp3.UID1=tmp4.uid2 and tmp3.uid2=tmp4.uid1) AS tmp5(uid1,uid2,total1,uid3,uid4,total2) WHERE total2 IS null";
3. rs = stmt.executeQuery(sql); // 执行查询语句
4. System.out.println("每一对用户之间的发送信息的数目：");
5. **while** (rs.next()) {
6. // 通过字段检索
7. **int** uid1 = rs.getInt("UID1");
8. **int** uid2 = rs.getInt("UID2");
9. **int** total = rs.getInt("total");
10. // 输出数据
11. System.out.print("(" + uid1 + "," + uid2 + "," + total + ")\n");
12. }
13. rs.close(); // 关闭结果集对象
14. **break**;

**4 小结**

本实验利用Java语言和JDBC技术，有效地演示了与GaussDB数据库的交互方式。实验过程中，我们先搭建了Java环境和JDBC驱动，再编写了一个Java应用程序，该程序能够完成数据库的基本操作，如增删改查。通过这个实验，我们不仅深入了解了SQL语句的用法，也掌握了如何在Java程序中建立数据库连接、执行SQL语句和处理查询结果。

此外，实验中的程序设计考虑了用户交互，提供了一个简单的菜单驱动界面，使用户能够根据需求选择不同的数据库操作。这种设计方式不仅提高了程序的可用性，也使得程序更加用户友好。

本实验是对数据库编程技能的有益训练，它让我们不仅掌握了数据库管理系统的运行机制，也了解了一个具体的应用程序开发过程。这些技能和知识对我们未来从事数据库管理、数据分析和软件开发等相关工作大有裨益。

教师评语评分

评语：

评分：

评阅人：

年 月 日