北京郵電大學

数据库系统原理

第二次实验

实验七　数据库接口实验

姓　　名　郭毓恒

学　　院　计算机学院

专　　业　计算机科学与技术

班　　级　2020211302

学　　号　2020211262

任课教师　邓芳

2022 年　11　月

实验目的

1. 华为的 GaussDB(for openGauss)支持基于 C、Java 等应用程序的开发。了解它相关的系统结构和相关概念，有助于更好地开发和使用 GaussDB(for openGauss)数据库。
2. 通过实验了解通用数据库应用编程接口 ODBC/JDBC 的基本原理和实现机制，熟悉连接 ODBC/JDBC 接口的语法和使用方法。
3. 熟练 GaussDB(for openGauss)的各种连接方式与常用工具的使用。
4. 利用 C 语言(或其它支持 ODBC/JDBC 接口的高级程序设计语言)编程实现简单的数据库应用程序，掌握基于 ODBC 的数据库访问基本原理和方法。

实验平台和环境

1. 本实验环境为华为云 GaussDB(for openGauss)数据库；
2. 为了满足本实验需要，实验环境采用以下配置：
   1) 设备名称：数据库
   2) 设备型号：GaussDB(for openGauss) 8 核 | 64 GB
   3) 软件版本：GaussDB(for openGauss) 2020 主备版

实验内容

1. 本实验内容通过使用 ODBC/JDBC 等驱动开发应用程序。
2. 连接语句访问数据库接口，实现对数据库中的数据进行操作（包括增、删、改、查等）；
3. 要求能够通过编写程序访问到华为数据库，该实验重点在于 ODBC/JDBC 数据源配置和高级语言(C/C++/JAVA/PYTHON)的使用。

实验步骤

在 Windows 控制面板中通过管理工具下的 ODBC 数据源工具在客户端新建连接到华为分布式数据库服务器的 ODBC 数据源，测试通过后保存，注意名字应与应用程序中引用的数据源一致。

1) 编译程序并调试通过；
2) 实验过程要求：
   (1) 以 PGSQL 语言相关内容为基础，课后查阅、自学 ODBC/JDBC 接口有关内容，包括 ODBC 的体系结构、工作原理、数据访问过程、主要 API 接口的语法和使用方法等。
   (2) 以实验二建立的数据库为基础，编写 C 语言(或其它支持 ODBC/JDBC 接口的高级程序设计语言) 数据库应用程序，按照如下步骤访问数据库：
      (a) Step1. ODBC 初始化，为 ODBC 分配环境句柄；
      (b) Step2. 建立应用程序与 ODBC 数据源的连接；
      (c) Step3. 实现数据库应用程序对数据库中表的数据查询、修改、删除、插入等操作。
      (d) Step4. 结束数据库应用程序。
      (e) 由于不是程序设计练习，因此针对一张表进行操作，即可完成基本要求。
      (f) 若程序结构和功能完整，界面友好，可适当增加分数。
   (3) 实验相关语句要求：
      所编写的数据库访问应用程序应使用到以下主要的 ODBC API 函数：
      (a) SQLALLocEnv：初始化 ODBC 环境，返回环境句柄；
      (b) SQLALLocConnect：为连接句柄分配内存并返回连接句柄；
      (c) SQLConnect：连接一个 SQL 数据资源；
      (d) SQLDriverConnect：连接一个 SQL 数据资源，允许驱动器向用户询问信息；
      (e) SQLALLocStmt：为语句句柄分配内存, 并返回语句句柄；
      (f) SQLExecDirect：把 SQL 语句送到数据库服务器，请求执行由 SQL 语句定义的数据库访问；
      (g) SQLFetchAdvances：将游标移动到查询结果集的下一行(或第一行)；
      (h) SQLGetData：按照游标指向的位置，从查询结果集的特定的一列取回数据；
      (i) SQLFreeStmt：释放与语句句柄相关的资源；
      (j) SQLDisconnect：切断连接；
      (k) SQLFreeConnect：释放与连接句柄相关的资源；
      (l) SQLFreeEnv：释放与环境句柄相关的资源。

实验结果及分析

实验源代码

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <sqlext.h>
#include <string.h>

void sql_insert(void);                  //插入
void sql_delete(void);                  //删除
void sql_update(void);                  //更新
void sql_select_all(void);              //查询所有列

SQLHENV             V_OD_Env;           // Handle ODBC environment
SQLHSTMT            V_OD_hstmt;         // Handle statement
SQLHDBC             V_OD_hdbc;          // Handle connection
SQLRETURN           V_OD_erg;

int main(int argc, char* argv[])
{
    //申请环境句柄
    V_OD_erg                =               SQLAllocHandle(SQL_HANDLE_ENV,
SQL_NULL_HANDLE, &V_OD_Env);
    if    ((V_OD_erg    !=    SQL_SUCCESS)    &&    (V_OD_erg    !=
SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }
    //设置环境属性（版本信息）
    SQLSetEnvAttr(V_OD_Env,                 SQL_ATTR_ODBC_VERSION,
(void*)SQL_OV_ODBC3, 0);
    //申请连接句柄
    V_OD_erg    =    SQLAllocHandle(SQL_HANDLE_DBC,    V_OD_Env,
&V_OD_hdbc);
    if    ((V_OD_erg    !=    SQL_SUCCESS)    &&    (V_OD_erg    !=
SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
```

```
    //设置连接属性
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,
(SQLPOINTER)SQL_AUTOCOMMIT_ON, 0);
    printf("*****", V_OD_hdbc);
    //连接数据源
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*)"PostgreSQL35W",
SQL_NTS, (SQLCHAR*)"bupt2020211262", SQL_NTS,
(SQLCHAR*)"bupt@2022", SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg !=
SQL_SUCCESS_WITH_INFO))
    {
        printf("Error SQLConnect %d\n", V_OD_erg);
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    printf("Connected !\n");
    //设置语句属性
    SQLSetStmtAttr(V_OD_hstmt, SQL_ATTR_QUERY_TIMEOUT,
(SQLPOINTER*)3, 0);

    while (1)
    {
        int input;

        printf("新增记录请输入 1\n 删除记录请输入 2\n 查询记录请输入 3\n
更新记录请输入 4\n 退出请输入其他任意数字\n");
        scanf("%d", &input);

        switch (input)
        {
        case 1:
            sql_insert();
            break;
        case 2:
            sql_delete();
            break;
        case 3:
            sql_select_all();
            break;
        case 4:
            sql_update();
            break;
        default:
            break;
```

```
            }

            if (input > 4 || input < 1)
                break;
        }
        //断开数据源连接和释放句柄资源
        SQLDisconnect(V_OD_hdbc);
        SQLFreeHandle(SQL_HANDLE_DBC, V_OD_hdbc);
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        return(0);
}

void sql_insert(void)
{
        SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
        char insert_stmt[1000] = "INSERT INTO 全国各省累计数据统计
VALUES('";
        char dateIn[11], proIn[100], caseStr[100], cureStr[100], deathStr[100];
        int caseIn, cureIn, deathIn;

        printf("输入要插入的记录：\n 日期：");
        scanf("%s", dateIn);
        printf("省：");
        scanf("%s", proIn);
        printf("累计确诊：");
        scanf("%d", &caseIn);
        printf("累计治愈：");
        scanf("%d", &cureIn);
        printf("累计死亡：");
        scanf("%d", &deathIn);
        itoa(caseIn, caseStr, 10);
        itoa(cureIn, cureStr, 10);
        itoa(deathIn, deathStr, 10);

        strcat(insert_stmt, dateIn);
        strcat(insert_stmt, "', '");
        strcat(insert_stmt, proIn);
        strcat(insert_stmt, "', ");
        strcat(insert_stmt, caseStr);
        strcat(insert_stmt, ", ");
        strcat(insert_stmt, cureStr);
        strcat(insert_stmt, ", ");
        strcat(insert_stmt, deathStr);
        strcat(insert_stmt, ");");
```

```
    V_OD_erg = SQLExecDirect(V_OD_hstmt, (SQLCHAR*)insert_stmt,
SQL_NTS);
    if (V_OD_erg == SQL_SUCCESS || V_OD_erg ==
SQL_SUCCESS_WITH_INFO)
    {
        printf("插入成功\n");
    }
    else
    {
        printf("插入失败\n");
        if (V_OD_erg == SQL_ERROR)
        {
            SQLTCHAR state[128] = { 0 };
            SQLTCHAR msg[128] = { 0 };
            SQLError(V_OD_Env, V_OD_hdbc, V_OD_hstmt, state, NULL,
msg, sizeof(msg), NULL);
            printf("SQL_ERROR:\t%s\t%s\n", state, msg);
        }
    }

    printf("\n\n");
    SQLFreeHandle(SQL_HANDLE_STMT, V_OD_hstmt);
}

void sql_delete(void)
{
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
    char delete_stmt[1000] = "DELETE FROM 全国各省累计数据统计
WHERE ";
    char condition[1000];

    printf("输入删除条件：");
    getchar();
    gets(condition);

    strcat(delete_stmt, condition);
    strcat(delete_stmt, ";");

    V_OD_erg = SQLExecDirect(V_OD_hstmt, (SQLCHAR*)delete_stmt,
SQL_NTS);
    if (V_OD_erg == SQL_SUCCESS || V_OD_erg ==
SQL_SUCCESS_WITH_INFO)
    {
```

```
            printf("删除成功\n");
        }
        else
        {
            printf("删除失败\n");
            if (V_OD_erg == SQL_ERROR)
            {
                SQLTCHAR state[128] = { 0 };
                SQLTCHAR msg[128] = { 0 };
                SQLError(V_OD_Env, V_OD_hdbc, V_OD_hstmt, state, NULL,
msg, sizeof(msg), NULL);
                printf("SQL_ERROR:\t%s\t%s\n", state, msg);
            }
        }

        printf("\n\n");
        SQLFreeHandle(SQL_HANDLE_STMT, V_OD_hstmt);
}

void sql_select_all(void)
{
        SQLCHAR dateOut[11], proOut[100];
        SQLINTEGER caseOut, cureOut, deathOut;
        SQLLEN lenCol[5];

        SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
        char select_stmt[1000] = "SELECT * FROM 全国各省累计数据统计 ";
        char condition[1000];

        printf("请输入 SQL 选择语句的后续部分：");
        getchar();
        gets(condition);

        strcat(select_stmt, condition);

        SQLExecDirect(V_OD_hstmt, (SQLCHAR*)select_stmt, SQL_NTS);
        V_OD_erg = SQLFetch(V_OD_hstmt);
        if (V_OD_erg == SQL_SUCCESS || V_OD_erg ==
SQL_SUCCESS_WITH_INFO)
        {
            while (V_OD_erg == SQL_SUCCESS || V_OD_erg ==
SQL_SUCCESS_WITH_INFO)
            {
                SQLGetData(V_OD_hstmt, 1, SQL_C_CHAR,
```

```
(SQLPOINTER)&dateOut, 11, &lenCol[0]);
                SQLGetData(V_OD_hstmt,          2,          SQL_C_CHAR,
(SQLPOINTER)&proOut, 100, &lenCol[1]);
                SQLGetData(V_OD_hstmt,          3,          SQL_C_ULONG,
(SQLPOINTER)&caseOut, 0, &lenCol[2]);
                SQLGetData(V_OD_hstmt,          4,          SQL_C_ULONG,
(SQLPOINTER)&cureOut, 0, &lenCol[3]);
                SQLGetData(V_OD_hstmt,          5,          SQL_C_ULONG,
(SQLPOINTER)&deathOut, 0, &lenCol[4]);

                printf("日期: %s,\t 省: %s,\t 累计确诊: %u,\t 累计治愈: %u,\t 累计
死亡: %u\n", dateOut, proOut, caseOut, cureOut, deathOut);
                V_OD_erg = SQLFetch(V_OD_hstmt);
            };
        }
        else
        {
            printf("查询失败\n");
            if (V_OD_erg == SQL_ERROR)
            {
                SQLTCHAR state[128] = { 0 };
                SQLTCHAR msg[128] = { 0 };
                SQLError(V_OD_Env,  V_OD_hdbc,  V_OD_hstmt,  state,  NULL,
msg, sizeof(msg), NULL);
                printf("SQL_ERROR:\t%s\t%s\n", state, msg);
            }
        }

        printf("\n\n");
        SQLFreeHandle(SQL_HANDLE_STMT, V_OD_hstmt);
}

void sql_update(void)
{
        SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
        char update_stmt[1000] = "UPDATE  全国各省累计数据统计  SET ";
        char setStr[1000], condition[1000];

        printf("输入更新方法：");
        getchar();
        gets(setStr);
        printf("输入更新条件：");
        gets(condition);
```

```
    strcat(update_stmt, setStr);
    strcat(update_stmt, " WHERE ");
    strcat(update_stmt, condition);
    strcat(update_stmt, ";");

    V_OD_erg = SQLExecDirect(V_OD_hstmt, (SQLCHAR*)update_stmt,
SQL_NTS);
    if (V_OD_erg == SQL_SUCCESS || V_OD_erg ==
SQL_SUCCESS_WITH_INFO)
    {
        printf("更新成功\n");
    }
    else
    {
        printf("更新失败\n");
        if (V_OD_erg == SQL_ERROR)
        {
            SQLTCHAR state[128] = { 0 };
            SQLTCHAR msg[128] = { 0 };
            SQLError(V_OD_Env, V_OD_hdbc, V_OD_hstmt, state, NULL,
msg, sizeof(msg), NULL);
            printf("SQL_ERROR:\t%s\t%s\n", state, msg);
        }
    }

    printf("\n\n");
    SQLFreeHandle(SQL_HANDLE_STMT, V_OD_hstmt);
}
```

代码分别使用了四个函数实现对“全国各省累计确诊”表的插入、删除、更新和查询所有列的操作，分别是 sql_insert()、sql_delete()、sql_update()和 sql_select_all()。

在函数 sql_insert()中，用户需逐列输入所有列的数据。在函数 sql_delete()中，用户需自行输入 WHERE 条件，输入需要符合 SQL 语法。在函数 sql_update()中，用户需自行输入 SET 的内容和 WHERE 条件，输入需要符合 SQL 语法。在函数 sql_select_all()中，用户可以输入内容，拼接在基础的“SELECT * FROM 全国各省累计确诊”之后，输入要符合 SQL 语法。

```
*****Connected !
新增记录请输入1
删除记录请输入2
查询记录请输入3
更新记录请输入4
退出请输入其他任意数字
3
请输入SQL选择语句的后续部分：ORDER BY 日期 LIMIT 10;
日期：2020-11-22,        省：江西省,      累计确诊：935,   累计治愈：934,   累计死亡：1
日期：2020-11-22,        省：内蒙古自治区,        累计确诊：310,   累计治愈：291,   累计死亡：1
日期：2020-11-22,        省：北京市,      累计确诊：950,   累计治愈：936,   累计死亡：9
日期：2020-11-22,        省：台湾,        累计确诊：1,     累计治愈：1,     累计死亡：0
日期：2020-11-22,        省：吉林省,      累计确诊：157,   累计治愈：155,   累计死亡：2
日期：2020-11-22,        省：天津市,      累计确诊：298,   累计治愈：277,   累计死亡：3
日期：2020-11-22,        省：江苏省,      累计确诊：677,   累计治愈：670,   累计死亡：0
日期：2020-11-22,        省：云南省,      累计确诊：217,   累计治愈：210,   累计死亡：2
日期：2020-11-22,        省：山东省,      累计确诊：849,   累计治愈：841,   累计死亡：7
日期：2020-11-22,        省：上海市,      累计确诊：1308,  累计治愈：1236,  累计死亡：7
```

输入使得查询结果按日期升序排序，输出十行。

```
新增记录请输入1
删除记录请输入2
查询记录请输入3
更新记录请输入4
退出请输入其他任意数字
1
输入要插入的记录：
日期：2000-1-1
省：北京市
累计确诊：0
累计治愈：0
累计死亡：0
插入成功


新增记录请输入1
删除记录请输入2
查询记录请输入3
更新记录请输入4
退出请输入其他任意数字
3
请输入SQL选择语句的后续部分：ORDER BY 日期 LIMIT 10;
日期：2000-01-01,        省：北京市,      累计确诊：0,     累计治愈：0,     累计死亡：0
日期：2020-11-22,        省：内蒙古自治区,        累计确诊：310,   累计治愈：291,   累计死亡：1
日期：2020-11-22,        省：江苏省,      累计确诊：677,   累计治愈：670,   累计死亡：0
日期：2020-11-22,        省：台湾,        累计确诊：1,     累计治愈：1,     累计死亡：0
日期：2020-11-22,        省：吉林省,      累计确诊：157,   累计治愈：155,   累计死亡：2
日期：2020-11-22,        省：天津市,      累计确诊：298,   累计治愈：277,   累计死亡：3
日期：2020-11-22,        省：北京市,      累计确诊：950,   累计治愈：936,   累计死亡：9
日期：2020-11-22,        省：云南省,      累计确诊：217,   累计治愈：210,   累计死亡：2
日期：2020-11-22,        省：江西省,      累计确诊：935,   累计治愈：934,   累计死亡：1
日期：2020-11-22,        省：山东省,      累计确诊：849,   累计治愈：841,   累计死亡：7
```

在插入新记录 2000-1-1 北京市的累计确诊后，可见输出内容有此项记录，插入成功。

```
新增记录请输入1
删除记录请输入2
查询记录请输入3
更新记录请输入4
退出请输入其他任意数字
4
输入更新方法：日期 = '2001-1-1'
输入更新条件：日期 = '2000-1-1'
更新成功


新增记录请输入1
删除记录请输入2
查询记录请输入3
更新记录请输入4
退出请输入其他任意数字
3
请输入SQL选择语句的后续部分：ORDER BY 日期 LIMIT 10;
日期：2001-01-01,        省：北京市,      累计确诊：0,     累计治愈：0,     累计死亡：0
日期：2020-11-22,        省：内蒙古自治区,        累计确诊：310,   累计治愈：291,   累计死亡：1
日期：2020-11-22,        省：江苏省,      累计确诊：677,   累计治愈：670,   累计死亡：0
日期：2020-11-22,        省：台湾,        累计确诊：1,     累计治愈：1,     累计死亡：0
日期：2020-11-22,        省：吉林省,      累计确诊：157,   累计治愈：155,   累计死亡：2
日期：2020-11-22,        省：天津市,      累计确诊：298,   累计治愈：277,   累计死亡：3
日期：2020-11-22,        省：北京市,      累计确诊：950,   累计治愈：936,   累计死亡：9
日期：2020-11-22,        省：云南省,      累计确诊：217,   累计治愈：210,   累计死亡：2
日期：2020-11-22,        省：江西省,      累计确诊：935,   累计治愈：934,   累计死亡：1
日期：2020-11-22,        省：山东省,      累计确诊：849,   累计治愈：841,   累计死亡：7
```

选择更新记录，将日期为 2000-1-1 的记录的日期改为 2001-1-1，更新成功。

```
新增记录请输入1
删除记录请输入2
查询记录请输入3
更新记录请输入4
退出请输入其他任意数字
2
输入删除条件：日期 = '2001-1-1'
删除成功


新增记录请输入1
删除记录请输入2
查询记录请输入3
更新记录请输入4
退出请输入其他任意数字
3
请输入SQL选择语句的后续部分：ORDER BY 日期 LIMIT 10;
日期：2020-11-22,        省：江西省,      累计确诊：935,   累计治愈：934,   累计死亡：1
日期：2020-11-22,        省：内蒙古自治区,        累计确诊：310,   累计治愈：291,   累计死亡：1
日期：2020-11-22,        省：北京市,      累计确诊：950,   累计治愈：936,   累计死亡：9
日期：2020-11-22,        省：台湾,        累计确诊：1,     累计治愈：1,     累计死亡：0
日期：2020-11-22,        省：吉林省,      累计确诊：157,   累计治愈：155,   累计死亡：2
日期：2020-11-22,        省：天津市,      累计确诊：298,   累计治愈：277,   累计死亡：3
日期：2020-11-22,        省：江苏省,      累计确诊：677,   累计治愈：670,   累计死亡：0
日期：2020-11-22,        省：云南省,      累计确诊：217,   累计治愈：210,   累计死亡：2
日期：2020-11-22,        省：山东省,      累计确诊：849,   累计治愈：841,   累计死亡：7
日期：2020-11-22,        省：上海市,      累计确诊：1308,  累计治愈：1236,  累计死亡：7
```

将日期等于 2001-1-1 的记录删除，删除成功。

实验小结

通过本次实验，学会了如何进行数据库实例连接，然后编写代码读写数据库内容。借此，可以在程序设计中利用数据库存储和操作数据，提高程序效率。在实验过程中，主要遇到的问题出于对 OBDC 开发的不熟悉。例如，当执行完 SQL 查询语句，利用 SQLFetch 函数输出多行结果后，需要对语句句柄重新分配资源，直接执行之后的 SQL 会导致 SQL_ERROR 错误。

在程序开发语言中，复杂的数据库操作和通信被数据库驱动抽象成为访问接口。数据库驱动是应用程序和数据库存储之间的一种接口，担任类似翻译员功能，将开发语言对数据库的调用语言翻译成数据库自己的语言，实现数据库调用。

ODBC，Open Database Connectivity，开放数据库互联，是由 Microsoft 公司基于 X/OPEN CLI 提出的用于访问数据库的应用程序编程接口。使用 ODBC 进行应用开发时，首先需要申请环境句柄资源，然后设置环境属性。接着，申请连接句柄和设置连接属性，进行数据源连接。连接成功后，申请语句句柄，可以开始执行 SQL 语句。对需要处理并输出结果集的 SQL 语句，在输出完毕后，需要释放语句句柄，然后再次申请，才能执行之后的 SQL 语句。执行完毕后，断开数据源连接，释放所有句柄资源，结束程序。