/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* ODBCSQL: a sample program that implements an ODBC command line interpreter.

/\*

/\* USAGE: ODBCSQL DSN=<dsn name> or

/\* ODBCSQL FILEDSN=<file dsn> or

/\* ODBCSQL DRIVER={driver name}

/\*

/\*

/\* Copyright(c) Microsoft Corporation. This is a WDAC sample program and

/\* is not suitable for use in production environments.

/\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Modules:

/\* Main Main driver loop, executes queries.

/\* DisplayResults Display the results of the query if any

/\* AllocateBindings Bind column data

/\* DisplayTitles Print column titles

/\* SetConsole Set console display mode

/\* HandleError Show ODBC error messages

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <windows.h>

#include <sql.h>

#include <sqlext.h>

#include <stdio.h>

#include <conio.h>

#include <tchar.h>

#include <stdlib.h>

#include <sal.h>

#include <iostream>

#include <string>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Macro to call ODBC functions and \*/

/\* report an error on failure. \*/

/\* Takes handle, handle type, and stmt \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define TRYODBC(h, ht, x) { RETCODE rc = x;\

if (rc != SQL\_SUCCESS) \

{ \

HandleDiagnosticRecord (h, ht, rc); \

} \

if (rc == SQL\_ERROR) \

{ \

fwprintf(stderr, L"Error in " L#x L"\n"); \

goto Exit; \

} \

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Structure to store information about \*/

/\* a column.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct STR\_BINDING {

SQLSMALLINT cDisplaySize; /\* size to display \*/

WCHAR\* wszBuffer; /\* display buffer \*/

SQLLEN indPtr; /\* size or null \*/

BOOL fChar; /\* character col? \*/

struct STR\_BINDING\* sNext; /\* linked list \*/

} BINDING;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Forward references \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void HandleDiagnosticRecord(SQLHANDLE hHandle,

SQLSMALLINT hType,

RETCODE RetCode);

void DisplayResults(HSTMT hStmt,

SQLSMALLINT cCols);

void AllocateBindings(HSTMT hStmt,

SQLSMALLINT cCols,

BINDING\*\* ppBinding,

SQLSMALLINT\* pDisplay);

void DisplayTitles(HSTMT hStmt,

DWORD cDisplaySize,

BINDING\* pBinding);

void SetConsole(DWORD cDisplaySize,

BOOL fInvert);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Some constants \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define DISPLAY\_MAX 50 // Arbitrary limit on column width to display

#define DISPLAY\_FORMAT\_EXTRA 3 // Per column extra display bytes (| <data> )

#define DISPLAY\_FORMAT L"%c %\*.\*s "

#define DISPLAY\_FORMAT\_C L"%c %-\*.\*s "

#define NULL\_SIZE 6 // <NULL>

#define SQL\_QUERY\_SIZE 1000 // Max. Num characters for SQL Query passed in.

#define PIPE L'|'

SHORT gHeight = 80; // Users screen height

void convert(string szSource, WCHAR\* wszDest)

{

const size\_t WCHARBUF = 1000;

MultiByteToWideChar(CP\_ACP, MB\_PRECOMPOSED, szSource.c\_str(), -1, wszDest, WCHARBUF);

}

void ExecuteSQLStmt(WCHAR\* wszInput, SQLHSTMT& hStmt)

{

RETCODE RetCode;

SQLSMALLINT sNumResults;

RetCode = SQLExecDirect(hStmt, wszInput, SQL\_NTS);

switch (RetCode)

{

case SQL\_SUCCESS\_WITH\_INFO:

{

HandleDiagnosticRecord(hStmt, SQL\_HANDLE\_STMT, RetCode);

// fall through

}

case SQL\_SUCCESS:

{

// If this is a row-returning query, display

// results

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLNumResultCols(hStmt, &sNumResults));

if (sNumResults > 0)

{

DisplayResults(hStmt, sNumResults);

}

else

{

SQLLEN cRowCount;

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLRowCount(hStmt, &cRowCount));

if (cRowCount >= 0)

{

wprintf(L"%Id %s affected\n",

cRowCount,

cRowCount == 1 ? L"row" : L"rows");

}

}

break;

}

case SQL\_ERROR:

{

HandleDiagnosticRecord(hStmt, SQL\_HANDLE\_STMT, RetCode);

break;

}

default:

fwprintf(stderr, L"Unexpected return code %hd!\n", RetCode);

}

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLFreeStmt(hStmt, SQL\_CLOSE));

Exit:

;

}

int \_\_cdecl wmain(int argc, \_In\_reads\_(argc) WCHAR\*\* argv)

{

SQLHENV hEnv = NULL;

SQLHDBC hDbc = NULL;

SQLHSTMT hStmt = NULL;

WCHAR\* pwszConnStr = NULL;

WCHAR wszInput[SQL\_QUERY\_SIZE];

//Students should modifiy the following variables

string D\_ID = "";

string D\_Name = "";

string Phone\_Number = "";

string Address = "";

string Salary = "";

string Field = "";

string P\_Name = "";

string PPhone\_Number = "";

string P\_ID = "";

string Email = "";

string Sex = "";

string Appointment\_id = "";

string Booked\_Date = "";

string Patient\_Description = "";

string N\_ID = "";

string N\_Name = "";

string Phone\_Number\_ = "";

string nSalary = "";

string Address\_ = "";

//Medical info

string Blood\_type = "";

string Animia = "";

string Blood\_Disorder = "";

string Cancer = "";

string Depression = "";

string Kidney\_Disease = "";

string Lung\_Disease = "";

string Heart\_Problems = "";

string Thyroid\_Disease = "";

string HIV = "";

string Covid\_19 = "";

string Pregnant = "";

string Diabetes = "";

string Blood\_Pressure = "";

//Bill Info

string discount = "";

string bill\_charges = "";

string bill\_no = "";

// Allocate an environment

if (SQLAllocHandle(SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &hEnv) == SQL\_ERROR)

{

fwprintf(stderr, L"Unable to allocate an environment handle\n");

exit(-1);

}

// Register this as an application that expects 3.x behavior,

// you must register something if you use AllocHandle

TRYODBC(hEnv,

SQL\_HANDLE\_ENV,

SQLSetEnvAttr(hEnv,

SQL\_ATTR\_ODBC\_VERSION,

(SQLPOINTER)SQL\_OV\_ODBC3,

0));

// Allocate a connection

TRYODBC(hEnv,

SQL\_HANDLE\_ENV,

SQLAllocHandle(SQL\_HANDLE\_DBC, hEnv, &hDbc));

if (argc > 1)

{

pwszConnStr = \*++argv;

}

else

{

pwszConnStr = L"";

}

//Student must insert his/her connection string

pwszConnStr = L"Driver={SQL Server};Server=DB221341.mssql.somee.com;Database=DB221341;Uid=A221341\_SQLLogin\_1;Pwd=ex4yhk8dda;";

//pwszConnStr = L"Driver={SQL Server};Server=localhost\\SQLEXPRESS;Database=Company;Trusted\_Connection=Yes;";

// Connect to the driver. Use the connection string if supplied

// on the input, otherwise let the driver manager prompt for input.

TRYODBC(hDbc,

SQL\_HANDLE\_DBC,

SQLDriverConnect(hDbc,

GetDesktopWindow(),

pwszConnStr,

SQL\_NTS,

NULL,

0,

NULL,

SQL\_DRIVER\_COMPLETE));

fwprintf(stderr, L"Connected!\n");

TRYODBC(hDbc,

SQL\_HANDLE\_DBC,

SQLAllocHandle(SQL\_HANDLE\_STMT, hDbc, &hStmt));

//User Code

int choice = 10;

while (choice != -1)

{

int doctorChoice = 10;

int patientChoice = 10;

int appointmentChoice = 10;

int nurseChoice = 10;

int medChoice = 10;

int pat\_appChoice = 10;

int billChoice = 10;

int assistChoice = 10;

cout << "==============================" << endl;

cout << "= Welcome to Health Care =" << endl;

cout << "==============================" << endl;

cout << "==============================" << endl;

cout << "= Main Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Doctors Menu" << endl;

cout << "2: Patient Menu" << endl;

cout << "3: Appointments Menu" << endl;

cout << "4: Nurses Menu" << endl;

cout << "5: Medical Info Menu" << endl;

cout << "6: Patient Appointment Info Menu" << endl;

cout << "7: Bill Info Menu" << endl;

cout << "8: Assists Menu" << endl;

cout << "-1: Exit" << endl;

cout << "Choice: ";

cin >> choice;

switch (choice)

{

case 1:

while (doctorChoice != -1)

{

cout << "==============================" << endl;

cout << "= Doctors Menu = " << endl;

cout << "==============================" << endl;

cout << "1: Show All Doctors" << endl;

cout << "2: Insert new Doctor" << endl;

cout << "3: Delete current Doctor" << endl;

cout << "4: Update current Doctor" << endl;

cout << "5: Display Doctors' salary in descending order" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> doctorChoice;

switch (doctorChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from Doctors", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Doctors ID: ";

cin >> D\_ID;

cout << "Enter Doctors name: ";

cin >> D\_Name;

cout << "Enter Doctors phone number: ";

cin >> Phone\_Number;

cout << "Enter Doctors address: ";

cin >> Address;

cout << "Enter Doctors salary: ";

cin >> Salary;

cout << "Enter Doctors field: ";

cin >> Field;

convert("insert into Doctors values(" + D\_ID + ",'" + D\_Name + "','" + Phone\_Number + "','" + Address + "','" + Salary + "','" + Field + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Doctor ID to delete: ";

cin >> D\_ID;

convert("delete from Doctors where D\_ID = " + D\_ID, wszInput);

//cout << "delete from Students where id = " + D\_ID << endl;

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "= Doctors Update Menu = " << endl;

cout << "==============================" << endl;

cout << "1: Update Doctor Name" << endl;

cout << "2: Update Doctor address" << endl;

cout << "3: Update Doctor salary" << endl;

cout << "4: Update Doctor's Field" << endl;

cout << "-1: Back to the main menu" << endl;

cin >> doctorChoice;

switch (doctorChoice)

{

case 1:

cout << "Enter Doctor ID: ";

cin >> D\_ID;

cout << "Enter Doctor name: ";

cin >> D\_Name;

convert("update Doctors set D\_Name = '" + D\_Name + "' where D\_ID = " + D\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 2:

cout << "Enter Doctor ID: ";

cin >> D\_ID;

cout << "Enter Doctor Phone number: ";

cin >> Phone\_Number;

convert("update Doctors set Phone\_Number = '" + Phone\_Number + "' where D\_ID = " + D\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "Enter Doctor ID: ";

cin >> D\_ID;

cout << "Enter Doctor Address: ";

cin >> Address;

convert("update Doctors set Address\_ = '" + Address + "' where D\_ID = " + D\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "Enter Doctor ID: ";

cin >> D\_ID;

cout << "Enter Doctor field: ";

cin >> Field;

convert("update Doctors set Field = '" + Field + "' where D\_ID = " + D\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

case 5:

cout << "==============================" << endl;

convert("Select \* from Doctors ORDER BY Salary DESC", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

}

}

break;

case 2:

while (patientChoice != -1)

{

cout << "==============================" << endl;

cout << "= Patients Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Show All Patients" << endl;

cout << "2: Insert new Patient" << endl;

cout << "3: Delete current Patient" << endl;

cout << "4: Update current Patient" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> patientChoice;

switch (patientChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from Patient", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Patient id: ";

cin >> P\_ID;

cout << "Enter Patient name: ";

cin >> P\_Name;

cout << "Enter Patient phone number: ";

cin >> PPhone\_Number;

cout << "Enter Patient email: ";

cin >> Email;

cout << "Enter Patient sex: ";

cin >> Sex;

convert("insert into Patient values('" + P\_Name + "','" + PPhone\_Number + "','" + P\_ID + "','" + Email + "','" + Sex + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Patient ID to delete: ";

cin >> P\_ID;

convert("delete from Patient where P\_ID = '" + P\_ID + "'", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "= Patients Update Menu = " << endl;

cout << "==============================" << endl;

cout << "1: Update Patient's Name" << endl;

cout << "2: Update Patient's Phone Number" << endl;

cout << "3: Update Patient's Email" << endl;

cout << "4: Update Patient's Sex" << endl;

cout << "-1: Back to the main menu" << endl;

cin >> patientChoice;

switch (patientChoice)

{

case 1:

cout << "Enter Patient ID to update: ";

cin >> P\_ID;

cout << "Enter Patient name to update: ";

cin >> P\_Name;

convert("update Patient set P\_Name = '" + P\_Name + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 2:

cout << "Enter Patient ID to update: ";

cin >> P\_ID;

cout << "Enter the Patient's new phone number to update: ";

cin >> PPhone\_Number;

convert("update Patient set Phone\_Number = '" + PPhone\_Number + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "Enter Patient ID to update: ";

cin >> P\_ID;

cout << "Enter Patient's new email to update: ";

cin >> Email;

convert("update Patient set Email = '" + Email + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "Enter Patient ID to update: ";

cin >> P\_ID;

cout << "Enter Patient sex to update: ";

cin >> Sex;

convert("update Patient set Sex = '" + Sex + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

break;

}

}

break;

case 3:

while (appointmentChoice != -1)

{

cout << "==============================" << endl;

cout << "= Patients Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Show All Appointments" << endl;

cout << "2: Insert new Appointment" << endl;

cout << "3: Delete current Appointment" << endl;

cout << "4: Update current Appointment" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> appointmentChoice;

switch (appointmentChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from Appointments", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Appointment\_id: ";

cin >> Appointment\_id;

cout << "Enter Booked\_Date: ";

cin >> Booked\_Date;

cout << "Enter Patient\_Description: ";

cin >> Patient\_Description;

convert("insert into Appointments values('" + Appointment\_id + "','" + Booked\_Date + "','" + Patient\_Description + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Appointment\_id to delete: ";

cin >> Appointment\_id;

convert("delete from Appointments where Appointment\_id = '" + Appointment\_id + "'", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "Enter Appointment\_id TO UPDATE: ";

cin >> Appointment\_id;

cout << "Enter Booked\_Date TO UPDATE: ";

cin >> Booked\_Date;

cout << "Enter Patient\_Description TO UPDATE: ";

cin >> Patient\_Description;

convert("update Appointments set Booked\_Date = '" + Booked\_Date + "' where Appointment\_id = " + Appointment\_id, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

convert("update Appointments set Patient\_Description = '" + Patient\_Description + "' where Appointment\_id = " + Appointment\_id, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

}

break;

case 4:

while (nurseChoice != -1)

{

cout << "==============================" << endl;

cout << "= Nurses Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Show All Nurses" << endl;

cout << "2: Insert new Nurses" << endl;

cout << "3: Delete current Nurses" << endl;

cout << "4: Update current Nurses" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> nurseChoice;

switch (nurseChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from Nurses", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Nurse\_id: ";

cin >> N\_ID;

cout << "Enter Nurse name: ";

cin >> N\_Name;

cout << "Enter Nurse phone number: ";

cin >> Phone\_Number\_;

cout << "Enter Nurse Salary: ";

cin >> nSalary;

cout << "Enter Nurse Address: ";

cin >> Address\_;

convert("insert into Nurses values('" + N\_ID + "','" + N\_Name + "','" + Phone\_Number\_ + "','" + nSalary + "','" + Address\_ + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Nurse ID to delete: ";

cin >> N\_ID;

convert("delete from Nurses where N\_ID = '" + N\_ID + "'", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "= Nurses Update Menu = " << endl;

cout << "==============================" << endl;

cout << "1: Update Nurse's Name" << endl;

cout << "2: Update Nurse's Phone Number" << endl;

cout << "3: Update Nurse's Salary" << endl;

cout << "4: Update Nurse's Address" << endl;

cout << "-1: Back to the main menu" << endl;

cin >> nurseChoice;

switch (nurseChoice)

{

case 1:

cout << "Enter Nurse\_id to update: ";

cin >> N\_ID;

cout << "Enter Nurse name to update: ";

cin >> N\_Name;

convert("update Nurses set N\_Name = '" + N\_Name + "' where N\_ID = " + N\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 2:

cout << "Enter Nurse\_id to update: ";

cin >> N\_ID;

cout << "Enter Nurse phone number to update: ";

cin >> Phone\_Number\_;

convert("update Nurses set Phone\_Number\_ = '" + Phone\_Number\_ + "' where N\_ID = " + N\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "Enter Nurse\_id to update: ";

cin >> N\_ID;

cout << "Enter Nurse Salary to update: ";

cin >> nSalary;

convert("update Nurses set Salary = '" + nSalary + "' where N\_ID = " + N\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "Enter Nurse\_id to update: ";

cin >> N\_ID;

cout << "Enter Nurse Address to update: ";

cin >> Address\_;

convert("update Nurses set Address\_ = '" + Address\_ + "' where N\_ID = " + N\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

}

}

break;

case 5:

while (medChoice != -1)

{

cout << "==============================" << endl;

cout << "= Medical Info =" << endl;

cout << "==============================" << endl;

cout << "1: Show All Medical Info" << endl;

cout << "2: Insert new Medical Info" << endl;

cout << "3: Delete current Medical Info" << endl;

cout << "4: Update current Medical Info" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> medChoice;

switch (medChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from Medical\_info", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Animia: ";

cin >> Animia;

cout << "Kidney\_Disease: ";

cin >> Kidney\_Disease;

cout << "Lung\_Disease: ";

cin >> Lung\_Disease;

cout << "Heart\_Problems: ";

cin >> Heart\_Problems;

cout << "Thyroid\_Disease: ";

cin >> Thyroid\_Disease;

cout << "HIV: ";

cin >> HIV;

cout << "Covid\_19: ";

cin >> Covid\_19;

cout << "Pregnant: ";

cin >> Pregnant;

cout << "Diabetes: ";

cin >> Diabetes;

cout << "Blood\_Pressure: ";

cin >> Blood\_Pressure;

convert("insert into Medical\_info values('" + Animia + "','" + Kidney\_Disease + "','" + Lung\_Disease + "','" + Heart\_Problems + "','" + Thyroid\_Disease + "', '" + HIV + "' ,'" + Covid\_19 + "' , '" + Pregnant + "','" + Diabetes + "','" + Blood\_Pressure + "', '" + P\_ID + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Patient ID to delete: ";

cin >> P\_ID;

convert("delete from Medical\_info where P\_ID = '" + P\_ID + "'", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "= Medical info Update Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Update Animia Status" << endl;

cout << "2: Update Kidney Disease Status" << endl;

cout << "3: Update Lung Disease Status" << endl;

cout << "4: Update Heart Problems Status" << endl;

cout << "5: Update Thyroid Disease Status" << endl;

cout << "6: Update HIV Status" << endl;

cout << "7: Update Covid 19 Status" << endl;

cout << "8: Update Pregnant Status" << endl;

cout << "9: Update Diabetes Status" << endl;

cout << "10: Update Blood Pressure Status" << endl;

cout << "-1: Back to the main menu" << endl;

cin >> medChoice;

switch (medChoice)

{

case 1:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Animia: ";

cin >> Animia;

convert("update Medical\_info set Animia = '" + Animia + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 2:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Kidney\_Disease: ";

cin >> Kidney\_Disease;

convert("update Medical\_info set Kidney\_Disease = '" + Kidney\_Disease + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Lung\_Disease: ";

cin >> Lung\_Disease;

convert("update Medical\_info set Lung\_Disease = '" + Lung\_Disease + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Heart\_Problems: ";

cin >> Heart\_Problems;

convert("update Medical\_info set Heart\_Problems = '" + Heart\_Problems + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 5:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Thyroid\_Disease: ";

cin >> Thyroid\_Disease;

convert("update Medical\_info set Thyroid\_Disease = '" + Thyroid\_Disease + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 6:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "HIV: ";

cin >> HIV;

convert("update Medical\_info set HIV = '" + HIV + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 7:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Covid\_19: ";

cin >> Covid\_19;

convert("update Medical\_info set Covid\_19 = '" + Covid\_19 + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 8:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Pregnant: ";

cin >> Pregnant;

convert("update Medical\_info set Pregnant = '" + Pregnant + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 9:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Diabetes: ";

cin >> Diabetes;

convert("update Medical\_info set Diabetes = '" + Diabetes + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 10:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Blood\_Pressure: ";

cin >> Blood\_Pressure;

convert("update Medical\_info set Blood\_Pressure = '" + Blood\_Pressure + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

break;

}

}

break;

case 6:

while (pat\_appChoice != -1)

{

cout << "==============================" << endl;

cout << "= PAT\_APP Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Show All Patient Appointments" << endl;

cout << "2: Insert new Patient Appointment" << endl;

cout << "3: Delete current Patient Appointment" << endl;

cout << "4: Update current Patient Appointment" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> pat\_appChoice;

switch (pat\_appChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from PAT\_APP", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Enter Appointment ID: ";

cin >> Appointment\_id;

convert("insert into PAT\_APP values('" + P\_ID + "','" + Appointment\_id + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Patient ID to delete Appointment: ";

cin >> P\_ID;

convert("delete from PAT\_APP where P\_ID = '" + P\_ID + "'", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Enter Appointment ID: ";

cin >> Appointment\_id;

convert("update PAT\_APP set Appointment\_id = '" + Appointment\_id + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

}

break;

case 7:

while (billChoice != -1)

{

cout << "==============================" << endl;

cout << "= Bill Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Show All Bills" << endl;

cout << "2: Insert new Bill" << endl;

cout << "3: Delete current Bill" << endl;

cout << "4: Update current Bill" << endl;

cout << "5: Display Bills in descending order" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> billChoice;

switch (billChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from bill", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Enter Discount: ";

cin >> discount;

cout << "Enter Bill Charges: ";

cin >> bill\_charges;

cout << "Enter Bill Number: ";

cin >> bill\_no;

convert("insert into bill values('" + discount + "','" + bill\_charges + "', '" + bill\_no + "', '" + P\_ID + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Patient ID to delete Appointment: ";

cin >> P\_ID;

convert("delete from bill where P\_ID = '" + P\_ID + "'", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "= Bill Update Menu = " << endl;

cout << "==============================" << endl;

cout << "1: Update Bill Number" << endl;

cout << "2: Update Bill Charges" << endl;

cout << "3: Update Discounts" << endl;

cout << "-1: Back to the main menu" << endl;

cin >> billChoice;

switch (billChoice)

{

case 1:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Enter Bill Number: ";

cin >> bill\_no;

convert("update bill set bill\_no = '" + bill\_no + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 2:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Enter Bill Charges: ";

cin >> bill\_charges;

convert("update bill set bill\_charges = '" + bill\_charges + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "Enter Patient ID: ";

cin >> P\_ID;

cout << "Enter Discount: ";

cin >> discount;

convert("update bill set discount = '" + discount + "' where P\_ID = " + P\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

case 5:

cout << "==============================" << endl;

convert("Select \* from bill ORDER BY bill\_charges DESC", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

}

}

break;

case 8:

while (assistChoice != -1)

{

cout << "==============================" << endl;

cout << "= Assist Menu =" << endl;

cout << "==============================" << endl;

cout << "1: Show All Assists" << endl;

cout << "2: Insert new Assist" << endl;

cout << "3: Delete current Assist" << endl;

cout << "4: Update current Assist" << endl;

cout << "-1: Back to the main menu" << endl;

cout << "Choice: ";

cin >> assistChoice;

switch (assistChoice)

{

case 1:

cout << "==============================" << endl;

convert("Select \* from Assists", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

break;

case 2:

cout << "==============================" << endl;

cout << "Enter Doctor ID: ";

cin >> D\_ID;

cout << "Enter Nurse ID: ";

cin >> N\_ID;

convert("insert into Assists values('" + D\_ID + "','" + N\_ID + "')", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 3:

cout << "==============================" << endl;

cout << "Enter Patient ID to delete Appointment: ";

cin >> P\_ID;

convert("delete from Assists where D\_ID = '" + D\_ID + "'", wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

case 4:

cout << "==============================" << endl;

cout << "Enter Doctor ID: ";

cin >> D\_ID;

cout << "Enter Nurse ID: ";

cin >> N\_ID;

convert("update Assists set N\_ID = '" + N\_ID + "' where D\_ID = " + D\_ID, wszInput);

ExecuteSQLStmt(wszInput, hStmt);

cout << "Changes has been saved successfully!" << endl;

break;

}

}

break;

}

}

Exit:

// Free ODBC handles and exit

if (hStmt)

{

SQLFreeHandle(SQL\_HANDLE\_STMT, hStmt);

}

if (hDbc)

{

SQLDisconnect(hDbc);

SQLFreeHandle(SQL\_HANDLE\_DBC, hDbc);

}

if (hEnv)

{

SQLFreeHandle(SQL\_HANDLE\_ENV, hEnv);

}

wprintf(L"\nDisconnected.");

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* DisplayResults: display results of a select query

/\*

/\* Parameters:

/\* hStmt ODBC statement handle

/\* cCols Count of columns

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DisplayResults(HSTMT hStmt,

SQLSMALLINT cCols)

{

BINDING\* pFirstBinding, \* pThisBinding;

SQLSMALLINT cDisplaySize;

RETCODE RetCode = SQL\_SUCCESS;

int iCount = 0;

// Allocate memory for each column

AllocateBindings(hStmt, cCols, &pFirstBinding, &cDisplaySize);

// Set the display mode and write the titles

DisplayTitles(hStmt, cDisplaySize + 1, pFirstBinding);

// Fetch and display the data

bool fNoData = false;

do {

// Fetch a row

if (iCount++ >= gHeight - 2)

{

int nInputChar;

bool fEnterReceived = false;

while (!fEnterReceived)

{

wprintf(L" ");

SetConsole(cDisplaySize + 2, TRUE);

wprintf(L" Press ENTER to continue, Q to quit (height:%hd)", gHeight);

SetConsole(cDisplaySize + 2, FALSE);

nInputChar = \_getch();

wprintf(L"\n");

if ((nInputChar == 'Q') || (nInputChar == 'q'))

{

goto Exit;

}

else if ('\r' == nInputChar)

{

fEnterReceived = true;

}

// else loop back to display prompt again

}

iCount = 1;

DisplayTitles(hStmt, cDisplaySize + 1, pFirstBinding);

}

TRYODBC(hStmt, SQL\_HANDLE\_STMT, RetCode = SQLFetch(hStmt));

if (RetCode == SQL\_NO\_DATA\_FOUND)

{

fNoData = true;

}

else

{

// Display the data. Ignore truncations

for (pThisBinding = pFirstBinding;

pThisBinding;

pThisBinding = pThisBinding->sNext)

{

if (pThisBinding->indPtr != SQL\_NULL\_DATA)

{

wprintf(pThisBinding->fChar ? DISPLAY\_FORMAT\_C : DISPLAY\_FORMAT,

PIPE,

pThisBinding->cDisplaySize,

pThisBinding->cDisplaySize,

pThisBinding->wszBuffer);

}

else

{

wprintf(DISPLAY\_FORMAT\_C,

PIPE,

pThisBinding->cDisplaySize,

pThisBinding->cDisplaySize,

L"<NULL>");

}

}

wprintf(L" %c\n", PIPE);

}

} while (!fNoData);

SetConsole(cDisplaySize + 2, TRUE);

wprintf(L"%\*.\*s", cDisplaySize + 2, cDisplaySize + 2, L" ");

SetConsole(cDisplaySize + 2, FALSE);

wprintf(L"\n");

Exit:

// Clean up the allocated buffers

while (pFirstBinding)

{

pThisBinding = pFirstBinding->sNext;

free(pFirstBinding->wszBuffer);

free(pFirstBinding);

pFirstBinding = pThisBinding;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* AllocateBindings: Get column information and allocate bindings

/\* for each column.

/\*

/\* Parameters:

/\* hStmt Statement handle

/\* cCols Number of columns in the result set

/\* \*lppBinding Binding pointer (returned)

/\* lpDisplay Display size of one line

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void AllocateBindings(HSTMT hStmt,

SQLSMALLINT cCols,

BINDING\*\* ppBinding,

SQLSMALLINT\* pDisplay)

{

SQLSMALLINT iCol;

BINDING\* pThisBinding, \* pLastBinding = NULL;

SQLLEN cchDisplay, ssType;

SQLSMALLINT cchColumnNameLength;

\*pDisplay = 0;

for (iCol = 1; iCol <= cCols; iCol++)

{

pThisBinding = (BINDING\*)(malloc(sizeof(BINDING)));

if (!(pThisBinding))

{

fwprintf(stderr, L"Out of memory!\n");

exit(-100);

}

if (iCol == 1)

{

\*ppBinding = pThisBinding;

}

else

{

pLastBinding->sNext = pThisBinding;

}

pLastBinding = pThisBinding;

// Figure out the display length of the column (we will

// bind to char since we are only displaying data, in general

// you should bind to the appropriate C type if you are going

// to manipulate data since it is much faster...)

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLColAttribute(hStmt,

iCol,

SQL\_DESC\_DISPLAY\_SIZE,

NULL,

0,

NULL,

&cchDisplay));

// Figure out if this is a character or numeric column; this is

// used to determine if we want to display the data left- or right-

// aligned.

// SQL\_DESC\_CONCISE\_TYPE maps to the 1.x SQL\_COLUMN\_TYPE.

// This is what you must use if you want to work

// against a 2.x driver.

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLColAttribute(hStmt,

iCol,

SQL\_DESC\_CONCISE\_TYPE,

NULL,

0,

NULL,

&ssType));

pThisBinding->fChar = (ssType == SQL\_CHAR ||

ssType == SQL\_VARCHAR ||

ssType == SQL\_LONGVARCHAR);

pThisBinding->sNext = NULL;

// Arbitrary limit on display size

if (cchDisplay > DISPLAY\_MAX)

cchDisplay = DISPLAY\_MAX;

// Allocate a buffer big enough to hold the text representation

// of the data. Add one character for the null terminator

pThisBinding->wszBuffer = (WCHAR\*)malloc((cchDisplay + 1) \* sizeof(WCHAR));

if (!(pThisBinding->wszBuffer))

{

fwprintf(stderr, L"Out of memory!\n");

exit(-100);

}

// Map this buffer to the driver's buffer. At Fetch time,

// the driver will fill in this data. Note that the size is

// count of bytes (for Unicode). All ODBC functions that take

// SQLPOINTER use count of bytes; all functions that take only

// strings use count of characters.

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLBindCol(hStmt,

iCol,

SQL\_C\_TCHAR,

(SQLPOINTER)pThisBinding->wszBuffer,

(cchDisplay + 1) \* sizeof(WCHAR),

&pThisBinding->indPtr));

// Now set the display size that we will use to display

// the data. Figure out the length of the column name

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLColAttribute(hStmt,

iCol,

SQL\_DESC\_NAME,

NULL,

0,

&cchColumnNameLength,

NULL));

pThisBinding->cDisplaySize = max((SQLSMALLINT)cchDisplay, cchColumnNameLength);

if (pThisBinding->cDisplaySize < NULL\_SIZE)

pThisBinding->cDisplaySize = NULL\_SIZE;

\*pDisplay += pThisBinding->cDisplaySize + DISPLAY\_FORMAT\_EXTRA;

}

return;

Exit:

exit(-1);

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* DisplayTitles: print the titles of all the columns and set the

/\* shell window's width

/\*

/\* Parameters:

/\* hStmt Statement handle

/\* cDisplaySize Total display size

/\* pBinding list of binding information

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DisplayTitles(HSTMT hStmt,

DWORD cDisplaySize,

BINDING\* pBinding)

{

WCHAR wszTitle[DISPLAY\_MAX];

SQLSMALLINT iCol = 1;

SetConsole(cDisplaySize + 2, TRUE);

for (; pBinding; pBinding = pBinding->sNext)

{

TRYODBC(hStmt,

SQL\_HANDLE\_STMT,

SQLColAttribute(hStmt,

iCol++,

SQL\_DESC\_NAME,

wszTitle,

sizeof(wszTitle), // Note count of bytes!

NULL,

NULL));

wprintf(DISPLAY\_FORMAT\_C,

PIPE,

pBinding->cDisplaySize,

pBinding->cDisplaySize,

wszTitle);

}

Exit:

wprintf(L" %c", PIPE);

SetConsole(cDisplaySize + 2, FALSE);

wprintf(L"\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* SetConsole: sets console display size and video mode

/\*

/\* Parameters

/\* siDisplaySize Console display size

/\* fInvert Invert video?

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SetConsole(DWORD dwDisplaySize,

BOOL fInvert)

{

HANDLE hConsole;

CONSOLE\_SCREEN\_BUFFER\_INFO csbInfo;

// Reset the console screen buffer size if necessary

hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (hConsole != INVALID\_HANDLE\_VALUE)

{

if (GetConsoleScreenBufferInfo(hConsole, &csbInfo))

{

if (csbInfo.dwSize.X < (SHORT)dwDisplaySize)

{

csbInfo.dwSize.X = (SHORT)dwDisplaySize;

SetConsoleScreenBufferSize(hConsole, csbInfo.dwSize);

}

gHeight = csbInfo.dwSize.Y;

}

if (fInvert)

{

SetConsoleTextAttribute(hConsole, (WORD)(csbInfo.wAttributes | BACKGROUND\_BLUE));

}

else

{

SetConsoleTextAttribute(hConsole, (WORD)(csbInfo.wAttributes & ~(BACKGROUND\_BLUE)));

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* HandleDiagnosticRecord : display error/warning information

/\*

/\* Parameters:

/\* hHandle ODBC handle

/\* hType Type of handle (HANDLE\_STMT, HANDLE\_ENV, HANDLE\_DBC)

/\* RetCode Return code of failing command

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void HandleDiagnosticRecord(SQLHANDLE hHandle,

SQLSMALLINT hType,

RETCODE RetCode)

{

SQLSMALLINT iRec = 0;

SQLINTEGER iError;

WCHAR wszMessage[1000];

WCHAR wszState[SQL\_SQLSTATE\_SIZE + 1];

if (RetCode == SQL\_INVALID\_HANDLE)

{

fwprintf(stderr, L"Invalid handle!\n");

return;

}

while (SQLGetDiagRec(hType,

hHandle,

++iRec,

wszState,

&iError,

wszMessage,

(SQLSMALLINT)(sizeof(wszMessage) / sizeof(WCHAR)),

(SQLSMALLINT\*)NULL) == SQL\_SUCCESS)

{

// Hide data truncated..

if (wcsncmp(wszState, L"01004", 5))

{

fwprintf(stderr, L"[%5.5s] %s (%d)\n", wszState, wszMessage, iError);

}

}

}