COMP 3311 Database Management Systems Spring 2015

Lab 8. Programming with ODBC 2

Objectives of the Lab

- After this lab, you should be able to:
  - know about the different data-types of ODBC (which is important in calling the functions),
  - issue prepared statements using the ODBC interface,
  - check for error messages under ODBC.

Reminder: connecting to ODBC 1

- You have seen how to connect to the Oracle server through the ODBC interface in Lab 7.
- Four steps are involved in connecting to the Oracle through the ODBC interface:
  - Include the proper headers (<sql.h>, <sqlext.h>) to the C program,
  - Initialize ODBC environment,
  - Allocate a connection handle,
  - Connect to the data source corresponds to the Oracle server.

Reminder: connecting to ODBC 2

- To initialize the ODBC environment:

```
HENV   henv;
/* Allocate environment handle */
SQLAllocEnv( &henv);
```

- To allocate a connection handle:

```
HDBC   hdbc;
/* Allocate connection handle */
SQLAllocConnect(henv, &hdbc);
```

- To call the SQLConnect() function:

```
SQLConnectA(hdbc, (SQLCHAR*) "comp3311.cse.ust.hk",
SQL_NTS, (SQLCHAR*) "comp3311stu120", SQL_NTS,
(SQLCHAR*) “123456", SQL_NTS);
```

ODBC Data types 1

- To enable real ODBC programming, one needs to be able to declare ODBC data types.
- ODBC defines two sets of data types:
  - C data types – indicate the data type of the data stored in the local variables of the ODBC programs.
  - SQL data types – indicate the data type of data stored at the data source (i.e. the DataBase Management System)

ODBC Data types 2

- Some common C data types

C type identifier (i.e. parameter passed to SQLBindCol and SQLGetData functions to specify target variable datatype)	ODBC C typedef (define variables in the program	Corresponding C data type
SQL_C_CHAR	SQLCHAR *	unsigned char *
SQL_C_WCHAR	SQLWCHAR *	wchar_t *
SQL_C_SSHORT	SQLSMALLINT	short int
SQL_C_USHORT	SQLUSMALLINT	unsigned short int
SQL_C_SLONG	SQLINTEGER	long int
SQL_C_ULONG	SQLUINTEGER	unsigned long int
SQL_C_FLOAT	SQLREAL	float
SQL_C_DOUBLE	SQLDOUBLE, SQLFLOAT	double

ODBC Data types 3

- Some common SQL data types

SQL type identifier (i.e. the SQL data type of the data being stored in the DBMS)	Actual SQL data type	Type description
SQL_CHAR	CHAR(n)	Character string of length n
SQL_VARCHAR	VARCHAR(n)	Variable length character string upto n characters
SQL_DECIMAL	DECIMAL(p,s)	Signed numeric value with precision of at least p and scale of s (p significant digits and s digits after the decimal point, p<=15)
SQL_NUMERIC	NUMERIC(p,s)	Signed numeric value with precision of exactly p and scale of s (p<=15)
SQL_SMALLINT	SMALLINT	Numeric value with precision 5 and scale 0
SQL_INTEGER	INTEGER	Numeric value with precision 10 and scale 0
SQL_FLOAT	FLOAT(p)	Signed numeric value with a binary precision of at least p
SQL_DOUBLE	DOUBLE PRECISION	Signed numeric value with a binary precision 53.

The prepared statement 1

- To Prepare and execute a SQL statement, one needs to:
  - call SQLPrepare() function to prepare the statement (pre-compiled at the server to improve efficiency),
  - call the SQLBindParameter() function to set the value(s) of the parameter(s),
  - call SQLExecute() function to execute the statement.

The prepared statement 2

- The SQL statement could contain place-holders which indicate values obtained from the program during the execution.
- The SQLBindParameter() function binds local variables to the place-holders and specify the data types of the variables and the columns associated with the parameters.

The prepared statement 3

- To call SQLPrepare(), one needs to pass a statement handle and the SQL query as the parameters, a question mark in the SQL query indicates the location of a place-holder:

```
SQLPrepareA(hstmt, (SQLCHAR*) "SELECT room_number FROM departments WHERE department_id=?", SQL_NTS);
```

- The possible return codes of SQLPrepare() are SQL_SUCCESS, SQL_ERROR, SQL_SUCCESS_WITH_INFO, and SQL_INVALID_HANDLE

The prepared statement 4

- To bind parameters to the placeholders in the SQL query, one needs the SQLBindParameter() function. The following example binds the string deptid to the question mark on the last slide:

```
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 0, 0, deptid, 50, &deptid_n);
```

- The possible return codes of SQLBindParameter() are SQL_SUCCESS, SQL_ERROR, SQL_SUCCESS_WITH_INFO, and SQL_INVALID_HANDLE

The prepared statement 5

- The full syntax of calling the SQLBindParameter() function:

```
SQLBindParameter( SQLHSTMT StatementHandle, SQLUSMALLINT ParameterNumber, SQLSMALLINT InputOutputType, SQLSMALLINT ValueType, SQLSMALLINT ParameterType, SQLULEN ColumnSize, SQLSMALLINT DecimalDigits, SQLPOINTER ParameterValuePtr, SQLLEN BufferLength, SQLLEN * StrLen_or_IndPtr);
```

The prepared statement 6

- ParameterNumber indicates the particular placeholder (question mark) you want the local variable to be bound with.
- InputOutputType indicates the type of the variable to be bound. Possible values for this parameter are :
  - SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, SQL_PARAM_OUTPUT_STREAM, SQL_PARAM_INPUT_OUTPUT and SQL_PARAM_INPUT_OUTPUT_STREAM
  - (see http://msdn.microsoft.com/en-us/library/ms710963(v=VS.85).aspx for the details)
- ValueType indicates the C data type of the parameter (see slide 6 for the list of data types)
- ParameterType indicates the SQL data type of the parameter (see slide 7 for the list)
- ColumnSize indicates the max length of the parameter, it is used by the function if ParameterType equals to SQL_CHAR, SQL_GRAPHIC, SQL_DECIMAL or SQL_NUMERIC. For other parameter types, this parameter is unused/ignored.
- DecimalDigits indicates the number of digits of the parameter, it is used by the function if the ParameterType equals to SQL_DECIMAL,SQL_NUMERIC, or SQL_TIMESTAMP. For other parameter types, this parameter is unused/ignored.
- ParameterValuePtr is the pointer that points to a buffer that contains the actual data for the parameter, i.e. a local variable/array is been bound to the SQL parameter through this pointer.
- BufferLength indicates the length of the buffer pointed to by the ParameterValuePtr (50 for the previous example).
- strLen_or_IndPtr holds the length of the parameter value stored in *ParameterValuePtr , in the previous example we initialize it to SQL_NTS.

The prepared statement 7

- To execute the SQL statement, one needs to call the SQLExecute() function:

  SQLExecute(hstmt);

  Where hstmt is the statement handle.

- The possible return codes of SQLExecute() are SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, and SQL_PARAM_DATA_AVAILABLE.

The prepared statement 8

- Finally retrieve the result by binding the result to a local variable.

```
SQLBindCol(hstmt,1, SQL_C_SLONG,&room,1,&room_n);
```

- The possible return codes of SQLBindCol() are SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, and SQL_PARAM_DATA_AVAILABLE.

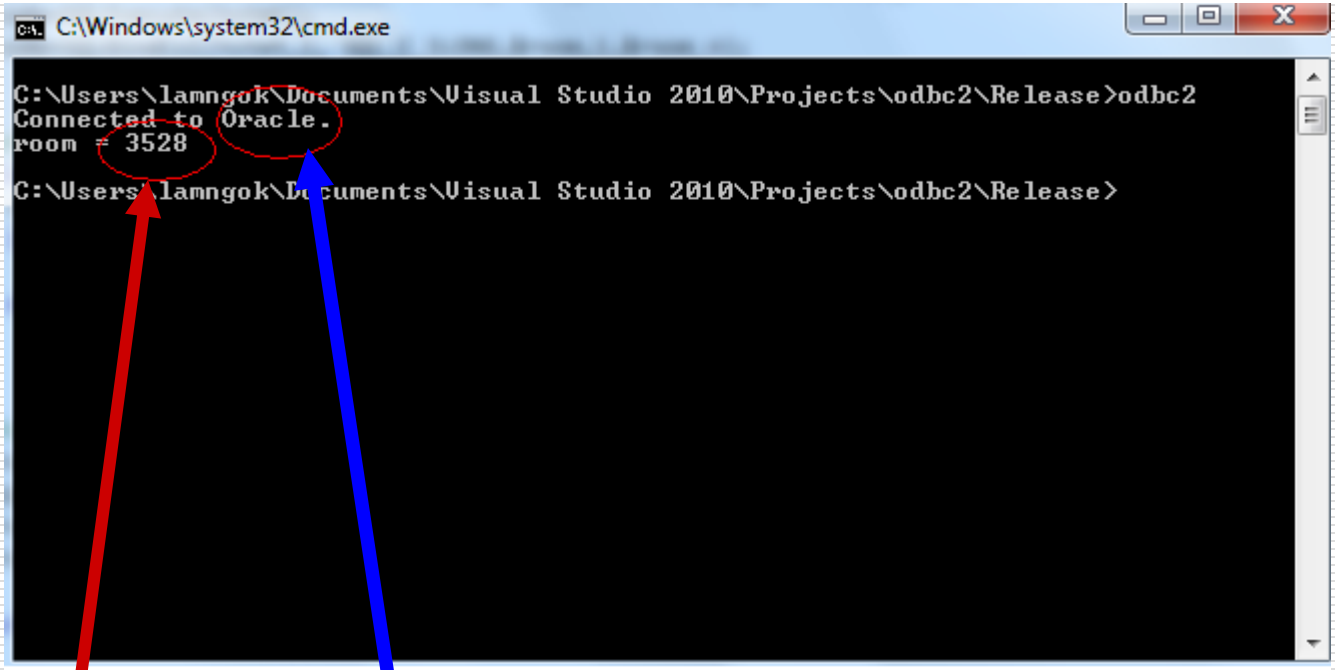
- The script file for building the database is available at[+]: http://course.cse.ust.hk/comp3311/labs/lab8.sql
- The complete piece of code is available at: http://course.cse.ust.hk/comp3311/labs/odbc2.cpp

[+] make sure you type "commit;" at the sqlplus prompt after running the script. Before you "commit" (or "exit"), the data will not be written.

Running the example 1

- First, make sure the data source has been set up properly as "comp3311.cse.ust.hk" (refer to the appendix 1 of lab7 for the detailed steps).
- Second, make sure you have ran the script lab8.sql at the SQL*Plus client (and "commit" it by typing "commit;" at the SQL*Plus prompt).
- Finally, start a new project and compile the code under Visual Studio (refer to the appendix 2 of lab7 for the detailed steps), and run the compiled program.

Running the example 2



Connection to Oracle through ODBC is successful!

Prepared statement ran successfully, retrieved value is bound to a C++ local variable "room" and is displayed to the screen

Getting error information 1

- To obtain ODBC error information, one can use the SQLGetDiagRec() function.
- The function will return the
  - SQLSTATE,
  - the native error code
  - the diagnostic message

  for the error.

Getting error information 2

- The syntax of the SQLGetDiagRec() function:

```
SQLGetDiagRec(    SQLSMALLINT HandleType,
SQLHANDLE Handle,      SQLSMALLINT RecNumber,
SQLCHAR * SQLState,      SQLINTEGER * NativeErrorPtr,
SQLCHAR * MessageText,      SQLSMALLINT BufferLength,
SQLSMALLINT * TextLengthPtr);
```

- The possible return codes are SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, and SQL_INVALID_HANDLE.

Getting error information 3

- HandleType is a handle identifier and the value can be SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT or SQL_HANDLE_DESC.
- Handle is the input handle for getting the specific error. It must be of the same type as declared by HandleType.
- When there are multiple errors, RecNumber allows the programmer to indicate which error to be retrieved. The first error message starts at RecNumber=1.
- SQLState is a pointer that points to the buffer that the five-character SQLSTATE code will be stored.
- NativeErrorPtr is a pointer that points to the buffer where the native error code will be stored. The native error code is specific to the particular data source (DMBS).
- MessageText is a pointer to the buffer where the diagnostic message (a character string) will be stored.
- BufferLength is the length of the MessageText buffer in characters.
- TextLengthPtr is a pointer to the buffer where the size of the MessageText string (in number of characters) is stored.

Getting error information 4

- A piece of code that calls the SQLGetDiagRec() function to retrieve the error message:

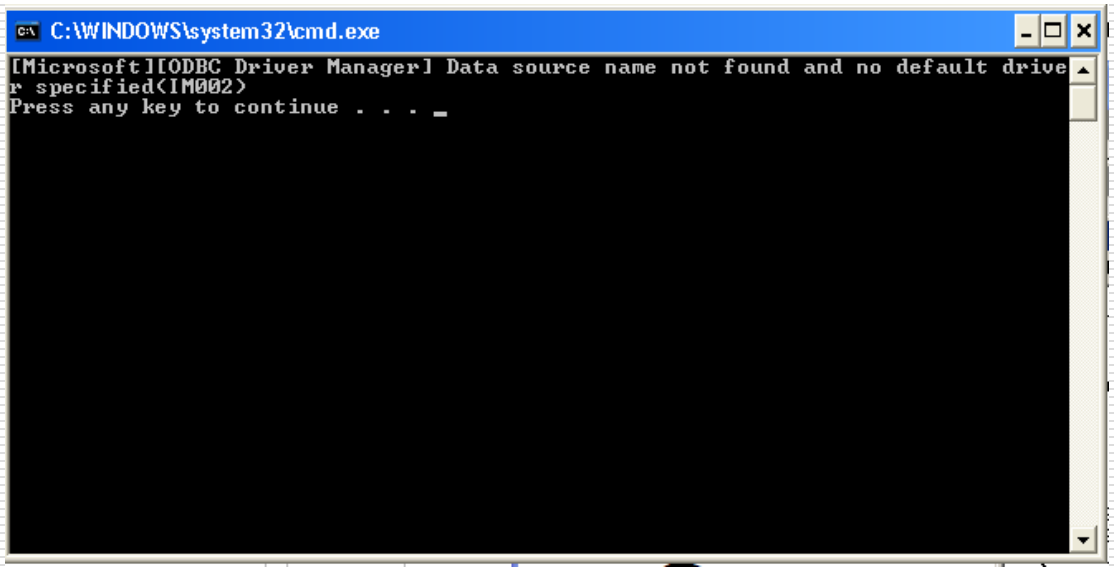
```
retcode = SQLConnectA(hdbc, (SQLCHAR*) "comp3331.cse.ust.hk", SQL_NTS,
(SQLCHAR*) "comp3311stu212", SQL_NTS, (SQLCHAR*) "123456", SQL_NTS);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
    printf ("Connected to Oracle.\n");}
else {
    SQLGetDiagRecA(SQL_HANDLE_DBC,hdbc,1,sqlstate, &sqlcode,
msg,4000,&len);
    printf("%s(%s)\n",msg,sqlstate);
    exit;}
```

- In the above code, the data source name is incorrect. So we should expect the error message to complain about that.

Getting error information 5

- The following is the error message returned:



- The complete code is available at:
  http://course.cse.ust.hk/comp3311/labs/odbc3.cpp

Getting error information 6

- Some common error codes:

SQLSTATE	Error
01000	General warning
08002	Connection name in use
08003	Connection not open
08007	Connection failure during transaction
22012	Division by zero
28000	Invalid authorization specification

- The complete list of error codes is available at: http://msdn.microsoft.com/en-us/library/ms714687(v=VS.85).aspx

Conclusion

- We covered the following topics in this lab:
  - the different data-types of ODBC,
  - the prepared statement,
  - function for getting ODBC error messages.

Appendix 1: List of all ODBC functions and datatypes

- The following page contains detailed information about all the ODBC functions:

http://msdn.microsoft.com/en-us/library/ms712628(v=VS.85).aspx

- More C datatypes are available at:

http://msdn.microsoft.com/en-us/library/ms714556(v=VS.85).aspx

- More SQL datatypes are available at:

http://msdn.microsoft.com/en-us/library/ms710150(v=VS.85).aspx

Appendix 2: Working with Virtual Lab 1

- You can not connect to the Oracle server through the ODBC interface from home this year (CSsystem is not allowing this).
- If you want to work remotely from home, the only choice you have is to utilize the CS virtual lab*.
- You could refer to the link below for the details of connecting to virtual lab using various OSes:

  http://cssystem.cse.ust.hk/home.php?docbase=UServices/vmvie &req_url=UServices/vmview/vmguide.html

- Our notes here will concentrate on connecting through the MS Windows.

**Warning: the virtual lab can only support up to 50 logins simultaneously and could be very slow when heavily loaded. Working on a normal lab machine will be a much better experience.*

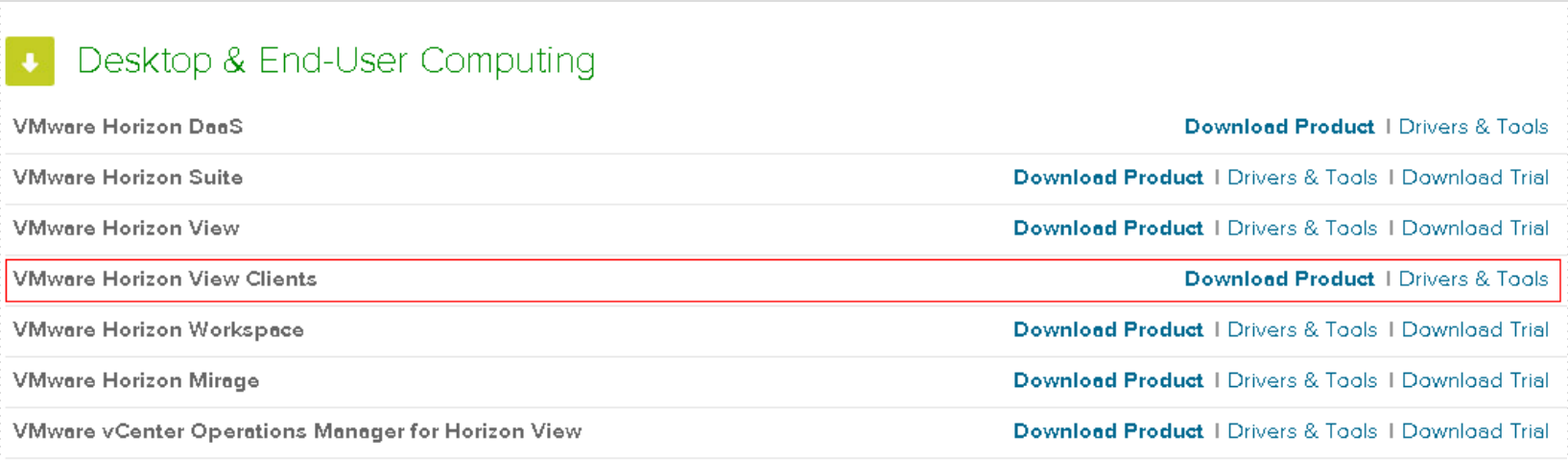
Appendix 2: Working with Virtual Lab 2

- To enable connecting to the virtual lab through Windows, first you will need to install a virtual machine client.
- You could download a free virtual machine client called “VMware horizon View” at the following link:

  https://my.vmware.com/web/vmware/downloads

Appendix 2: Working with Virtual Lab 3

- Click on the previous link and scroll down the appeared webpage you will see the following. Select "VMware Horizon View Clients"



Desktop & End-User Computing

VMware Horizon DaaS	**Download Product** | Drivers & Tools
VMware Horizon Suite	**Download Product** | Drivers & Tools | Download Trial
VMware Horizon View	**Download Product** | Drivers & Tools | Download Trial
VMware Horizon View Clients	**Download Product** | Drivers & Tools
VMware Horizon Workspace	**Download Product** | Drivers & Tools | Download Trial
VMware Horizon Mirage	**Download Product** | Drivers & Tools | Download Trial
VMware vCenter Operations Manager for Horizon View	**Download Product** | Drivers & Tools | Download Trial

Appendix 2: Working with Virtual Lab 4

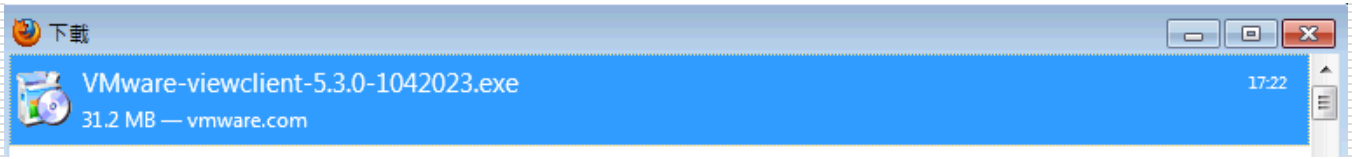
- Then follow the links to select the proper version of client for your OS.
- Two quick direct links are provided here:
  - 32-bit windows

    https://download3.vmware.com/software/view/viewclients/VMware-viewclient-5.3.0-1042023.exe
  - 64-bit windows

    https://download3.vmware.com/software/view/viewclients/VMware-viewclient-x86_64-5.3.0-1042023.exe

Appendix 2: Working with Virtual Lab 5

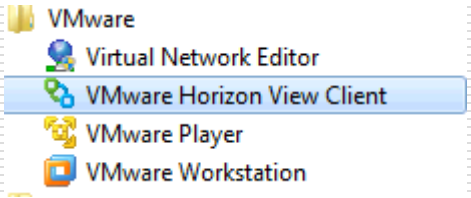
- Once you have downloaded the client, double click on the downloaded file and start the installation process.



- Follow the instructions to finish the whole installation process (this should be quite straight forward)

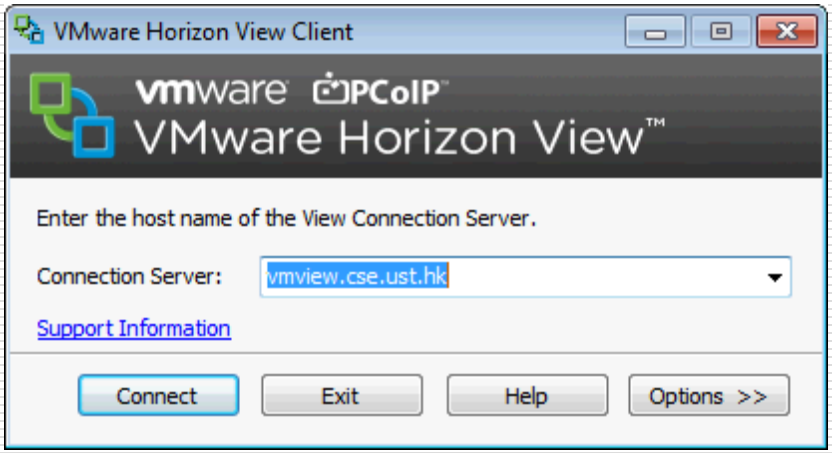
Appendix 2: Working with Virtual Lab 6

- Click the “start” button, and expand the folder “VMware”.
- Click on the “VMware Horizon View Client” icon shown below to start it.



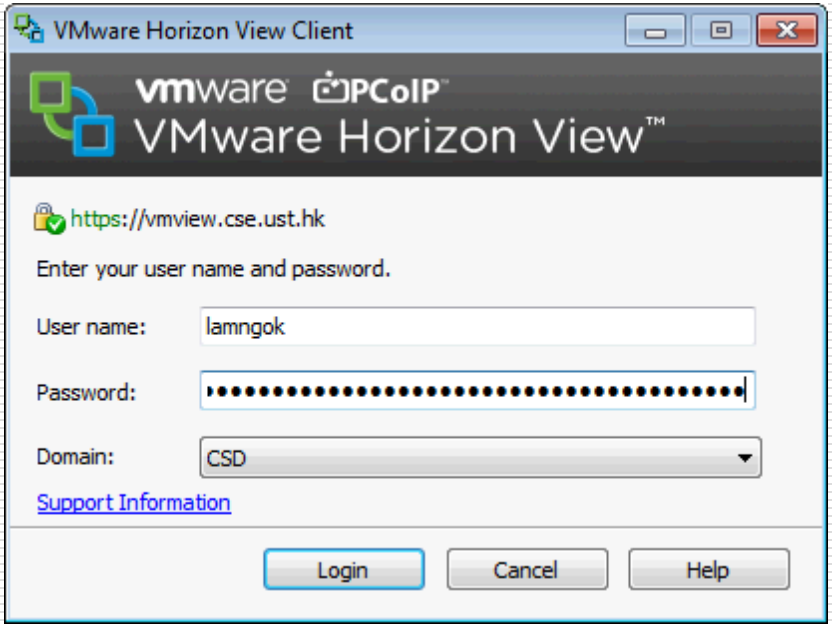
Appendix 2: Working with Virtual Lab 7

- Provide the connection server name "vmview.cse.ust.hk" and click "connect"



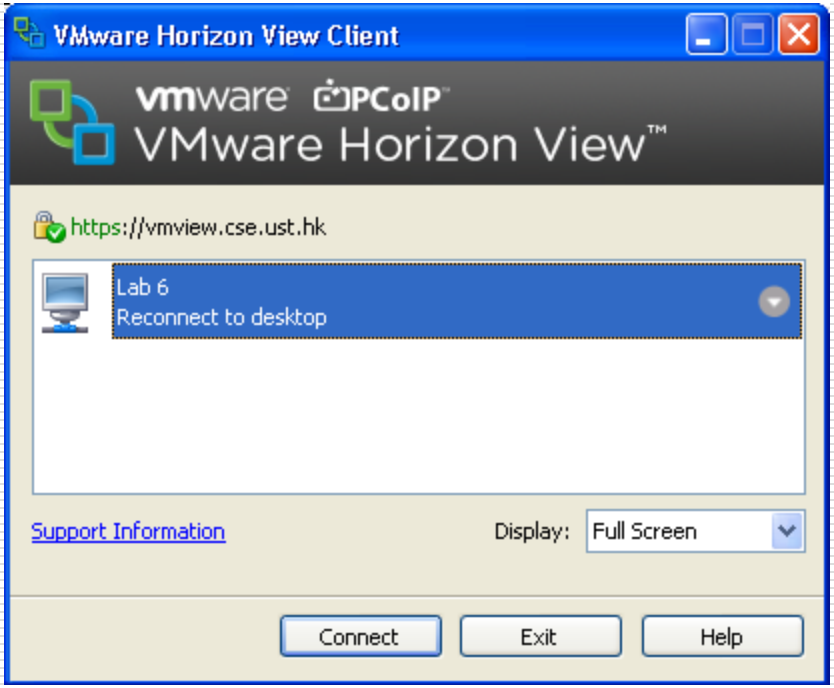
Appendix 2: Working with Virtual Lab 8

- Provide your CSD account user name and password (i.e. the username, password you use to login a CS lab machine)



Appendix 2: Working with Virtual Lab 9

- Choose Lab 6. Then you will log in the lab machine.



Appendix 2: Working with Virtual Lab 10

- Once you are logged in, follow the steps on slides 22-28 of lab7 note set to set up the data source.
- Then follow the steps on slides 29-37 of lab7 note set to work with Visual Studio.

Appendix 2: Working with Virtual Lab 11

- If you can not log in successfully to the Virtual Lab, please report to CSSystem the time of your login attempt and the username you use for the login attempt.
- Always remember to backup the assignment program (send the program through emails to yourself via the Virtual Machine's web browser).
- Warning: the virtual lab could be very slow when heavily loaded.