Relational Databases

MySQL C API

Setting Up

Setting Up Visual C/C++ For MySQL Applications

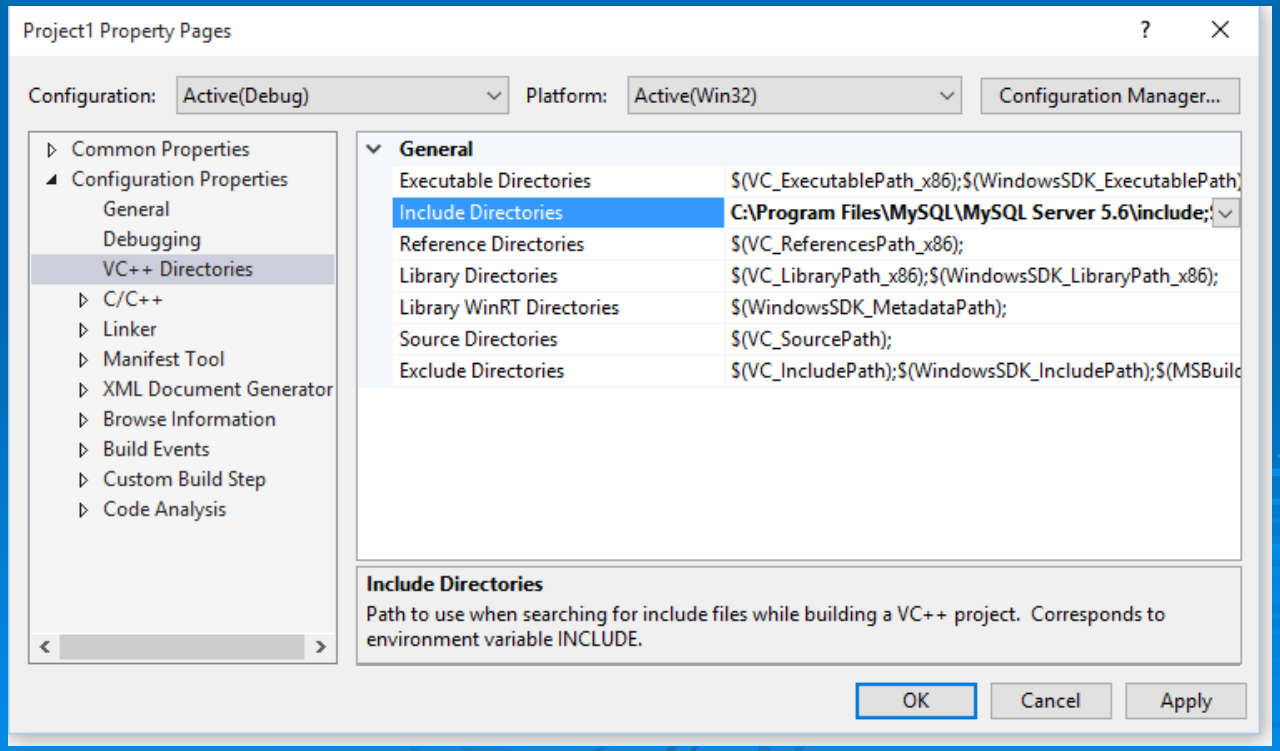
- You have to set up Visual C++'s directories and project settings, as well as including #include files and creating and copying files.

#includes in Your Code

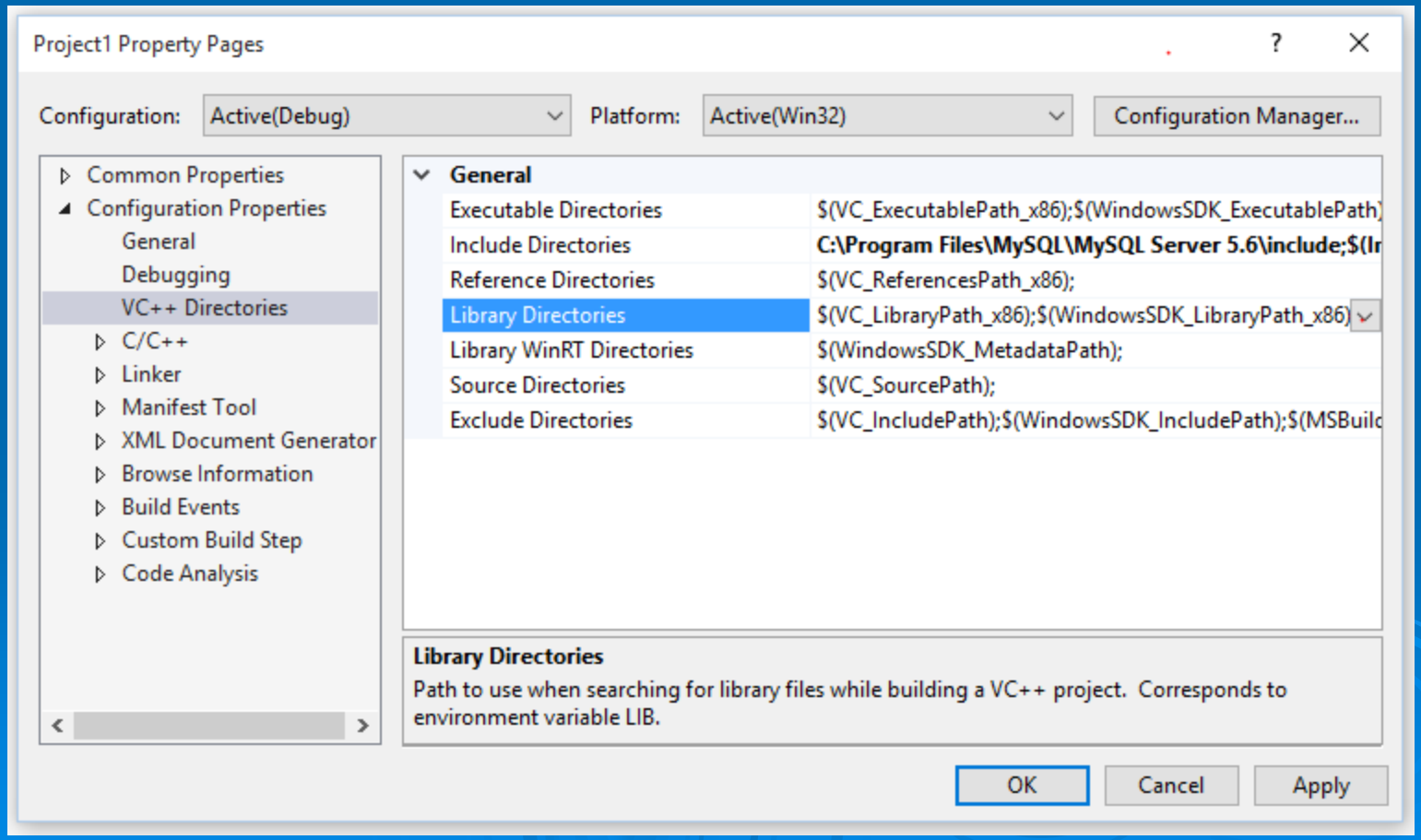
- Include the following files IN THIS ORDER:
  - #include <stdio.h>
  - #include <stdlib.h>
  - #include <my_global.h>
  - #include <mysql.h>

Setting Up Directories

- Click Project->Properties->Configuration Properties->VC++ Directories
  - Click in the data area for Include Directories
    - Click the drop-down arrow and choose Edit
    - Add the directory “C:\Program Files\MySQL Server *X.X*\include” where *X.X* is the MySQL server version you installed

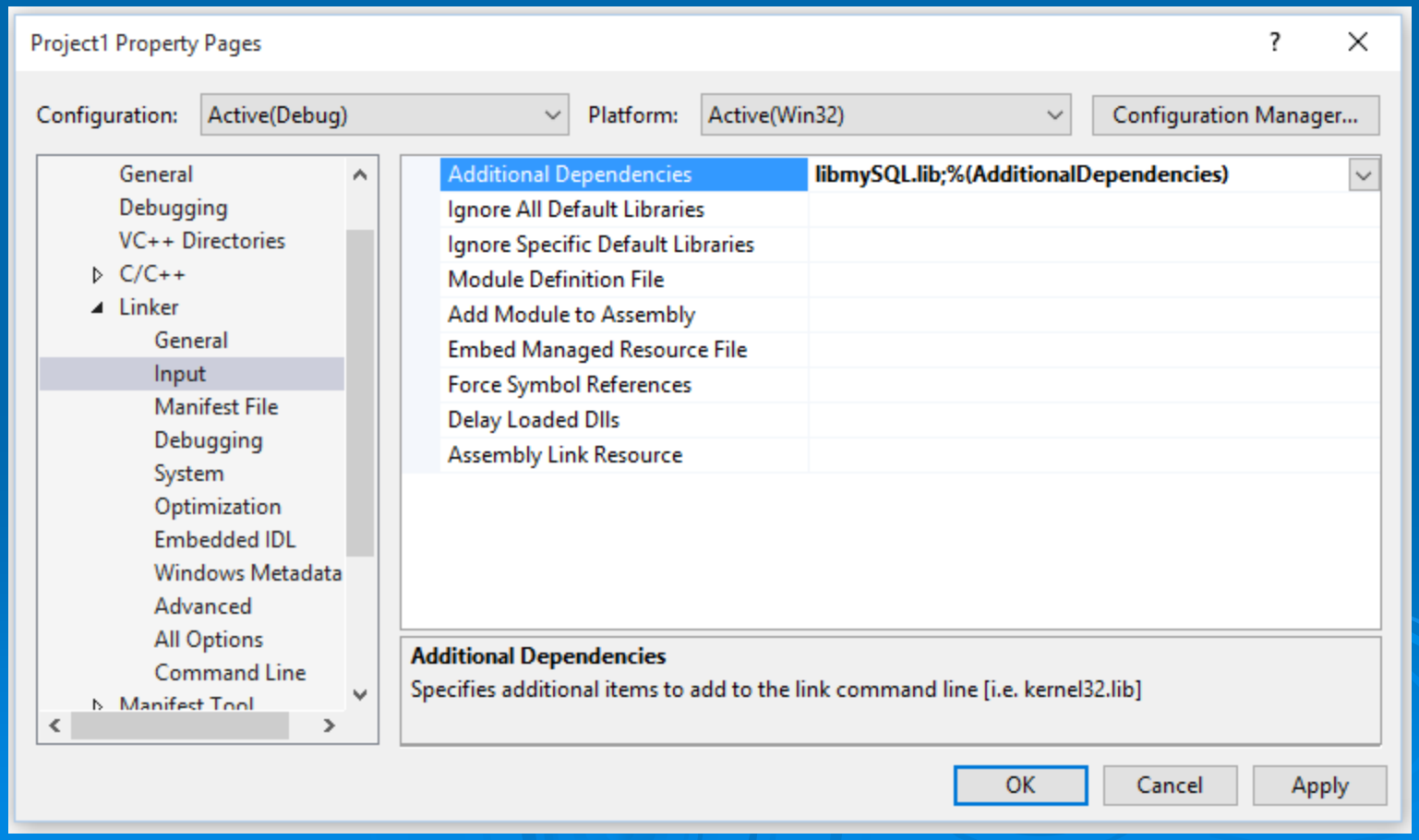


- Click in the data area for Library Directories
  - Click the drop-down arrow and choose Edit
  - Add the directory "C:\Program Files\MySQL\MySQL Server *X.X*\lib" where *X.X* is the version of MySQL installed



Project Properties

- Click: Projects -> Properties -> Configuration Properties-> Linker -> Input
  - In Additional Dependencies add “libmySQL.lib”



- If this sample main() function compiles, you did everything right and are ready to begin programming.

```
int main(void)
{
    MYSQL db;
    mysql_init(&db);

    return 0;
}
```

Unresolved References

- It is really important that you create an executable for the platform that matches your library – if the MySQL library is 32-bit, then you must use the 32-bit platform; if you are using the 64-bit library, then you must use the x64 platform
- Use the Configuration Manager to change the Active Platform if you need to

Cannot Find libmysql.dll

- Even if you build successfully, it is possible that the libmysql.dll cannot be found at runtime
- To solve, add the directory where it exists to the path

Using the MySQL C API

The MYSQL Variable

- You must create a MYSQL variable (actually a struct) that you'll feed to the API calls
  - e.g. MYSQL mysql;

mysql_init()

- Initializes a MYSQL data structure variable.
- Takes a pointer to a MYSQL variable as a parameter.
  - If you want, you can pass NULL and take the return value as a pointer to a MYSQL variable that will be deallocated some time later using mysql_close()

mysql_real_connect()

- Needed to connect to the database on the server.
- Takes a large number of parameters.
- All but the first parameter can be 0 or NULL.

mysql_real_connect() Parameters

- MYSQL *mysql: pointer to variable from init call
- const char *host: IP address of host w/server
- const char *user: NULL assumes the name of the user executing the program
- const char *passwd: NULL indicates empty password
- const char *db: name of default database
- uint port: port number used for TCP/IP connections (0 indicates 3306, the default port)
- const char *unix_socket: path to a socket (NULL if none)

More Parameters ...

- uint client_flag: bit field with option constants
  - CLIENT_COMPRESS: compresses communication between client and server
  - CLIENT_FOUND_ROWS: returns matched rows instead of affected row
  - CLIENT_IGNORE_SPACE: permits spaces between function names and parentheses
  - CLIENT_INTERACTIVE: uses interactive_timeout instead of wait_timeout
  - CLIENT_NO_SCHEMA: doesn't allow periods to specify database, table, and column names
  - CLIENT_ODBC: the client is an ODBC client
    - "Open Database Connectivity" is a dominant way of working w/databases on Windows
  - CLIENT_SSL: encrypts communications with SSL

mysql_real_connect() Return Value

- Returns pointer to MYSQL block or NULL if error
- Errors (retrieve with mysql_error()):
  - CR_COMMANDS_OUT_OF_SYNC,
    CR_SERVER_GONE_ERROR,
    CR_SERVER_LOST,
    CR_UNKNOWN_ERROR,
    ER_ACCESS_DENIED_ERROR,
    ER_BAD_DB_ERROR,
    ER_UNKNOWN_COM_ERROR,
    ER_WRONG_DB_NAME

Doing Queries

- Use mysql_query() primarily
- Can use mysql_real_query() if you need to have null characters in the query string
- Can use mysql_send_query() if you don't want to wait for the response
  - Useful for keeping responsive to the user
  - Can use mysql_read_query_result() to get the result later (but there may be a timeout)

mysql_query()

- int mysql_query(MYSQL *mysql, const char *query);
  - don't use a semicolon at the end of the string
  - returns 0 if OK and non-zero otherwise
- you sometimes get result sets back (also called them recordsets)
  - follow the call with mysql_store_result() or mysql_use_result()
- mysql_query() is also used for non-query SQL statements

Getting Query Results

- can have the results buffered or read one row at a time
- need to specify how you want to do it
- mysql_use_result() sets up for reading one row at a time
  - You must retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.
- mysql_store_result() gets the result set into a buffer (most often used)

mysql_use_result()

- Sets up to send rows back one at a time, through mysql_fetch_row()
- MYSQL_RES *mysql_use_result(MYSQL *mysql);
- Must retrieve all rows, even if you don't want to
  - If not, the rows will be returned as part of the next result set

Why Use mysql_use_result()?

- The client requires less memory for the result set because it maintains only one row at a time.
- Disadvantages:
  - you must process each row quickly to avoid tying up the server
  - you don't have random access to rows within the result set
  - you don't know how many rows are in the result set until you have retrieved them all

mysql_store_result()

- Puts the result of a query into an internal buffer
- MYSQL_RES *mysql_store_result(MYSQL *mysql);
- The pointer returned is sent to mysql_fetch_row() and mysql_data_seek()
- When done, must use mysql_free_result()
  - void mysql_free_result(MYSQL_RES *result);

Why Use mysql_store_result()?

- You can move back and forth in the result set using mysql_data_seek() or mysql_row_seek() to change the current row position within the result set.
- You can also find out how many rows there are by calling mysql_num_rows().
- On the other hand, the memory requirements for mysql_store_result() may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

mysql_fetch_row()

- mysql_fetch_row() gets each row
- MYSQL_ROW mysql_fetch_row(MYSQL_RES *res);
  - Returns NULL if no more rows to get or on error
- Use mysql_errno() to see if there is an error
- The columns come back as an array of strings
- Get column lengths with mysql_fetch_lengths()

- Recommended usage:
  - Use mysql_num_fields to get the number of fields in the result set
    - Store it outside the loop
  - Loop while mysql_fetch_row() doesn't return NULL
  - Get the lengths of the columns for this row by using lengths = mysql_fetch_lengths(result);
  - Loop for all fields, using row[i] to access the field

mysql_num_rows()

- mysql_num_rows() gets the number of rows
- my_ulonglong mysql_num_rows(MYSQL_RES *res);
- my_ulonglong is a 64-bit integer

mysql_num_fields()

- mysql_num_fields() gives the number of columns in the result set
- unsigned int mysql_num_fields(MYSQL_RES *res);

mysql_fetch_lengths()

- mysql_fetch_lengths() gives an array of lengths for the columns in the current row in the result set
- Ulong *mysql_fetch_lengths(MYSQL_RES *res);

Creating a Database

- Creating a database is done using the CREATE DATABASE statement in the mysql_query() function
  - mysql_create_db() is deprecated

mysql_affected_rows()

- my_ulonglong mysql_affected_rows(MYSQL *mysql)
- Returns the number of rows changed by the last UPDATE, deleted by the last DELETE or inserted by the last INSERT statement.
- May be called immediately after mysql_query() for UPDATE, DELETE, or INSERT statements. For SELECT statements, mysql_affected_rows() works like mysql_num_rows().

- Return value: An integer greater than zero indicates the number of rows affected or retrieved.
  - Zero indicates that no records where updated for an UPDATE statement, no rows matched the WHERE clause in the query or that no query has yet been executed.
  - -1 indicates that the query returned an error or that, for a SELECT query, mysql_affected_rows() was called prior to calling mysql_store_result().

Errors

- uint mysql_errno(MYSQL *mysql);
  - returns 0 if no error
- char *mysql_error(MYSQL *mysql);
  - returns "" if no error

Exercise

- Enable your Visual Studio environment to use MySQL API's
- Write a program to read a table in MySQL and display the results on the screen