Readme starts at page 13

PROCEDURE for this NOTE:

update in ICS168Notes and paste into "Database USING for ANY PROJECT in any path.docx"

Or vice-versa if no cpu memory complaint.

Get used to doing this.

\*\*Command prompt, using SQLite, SQL, and pretty much ALL stuff you do

Doesn't require these notes.

They are so basic and simple you just do them as routine.

Create and Edit a Database by using Command Prompt and Notepad:

Everything checked again and works (04/27/14 10:27am)

**Trying sqlite for the first time:**

In Command Prompt:

cd..

cd..

cd sqlite

sqlite3

Since in repository:

cd dataPathToRepository C:\.....\........

sqlite3 <------Must run this by simply saying

sqlite3 like this first

sqlite \_\_\_\_\_\_.db <--To run a database file as a program

Otherwise, skip this step

Then,

**Create a .db file/Open a .db file:**

.open testDB.db

\*\*If it doesn’t exist, the .db is automatically created. If it exists, that is HOW you open the .db in order to insert code in

In NotePad:

Do the SQL syntax there

Back to Command Prompt:

Insert code here in command prompt

Copy and paste

.close

NOTE 2:

**How to do sql syntax in Command Prompt:**

\*\*Do this directly for ICS 168 HW and other projects. Above is just practice.

1. Put the sqlite folder (the one containing the bin, include, and lib)

into the visual studio project and in the folder holding the visual studio project.

2. Do the following (Do the exact same thing as ex) shown for ANY project, except cd to the project file \_\_\_\_ instead:

ex)

//Using Project "Test"

1.

Create .db in the path to the project before running it

cd C:\

cd Users

cd Jordan Hua

cd Desktop

cd ICS 168 HW and Scratch Work

cd TestDatabaseCoder

cd Test

cd Test

---IF for ics repo: cd Users cd Jordan Hua cd My Documents cd GitHub cd swarch cd swarch\_solo cd SwarchSolo cd SwarchSolo

-IF for any server project, cd …… folder that holds server project cd server

sqlite3 -The only reason you can run this is b/c you set up C:\sqlite3 inside path settings

.open \_\_\_.db

…….//Steps below here

.close

.quit

2.

Once .db is created, create a table in database

CREATE TABLE sky(

NAME CHAR(50),

NUMBER INT);

3.

If you want to insert values into a table, use:

INSERT INTO sky

VALUES('String', #);

INSERT INTO sky

VALUES('String2', #);

....

\*\*Notice how you insert the values in according to

how you wrote "Create Table sky...."

4.

If you want to read values from a table, use this:

SELECT \*

FROM sky;

<----"test" is the name of the .db file.

<----"sky" is the name of the table

Summary:

In command prompt

* cd to path
* sqlite3
* .open \_\_\_\_\_.db ---Must ALWAYS open this before your project will work. Also do this to create the \_\_\_\_.db file for the first time.

\*\*If you don’t want to do this, add syntax in your project that creates a .db file in the path you want.

Then,

For Project:

* cmake the sql for your project
* Set up path to that \_\_\_\_.db
* Make syntax to open the \_\_\_\_.db

If you came back to project later:

Work on the visual studio project as is directly. YES. 04/27/14 10:58am

\*\*AS OF NOW TO START PROJECT:

UPDATE: Checked and works. 04/27/14 11:13am

IF the sqlite is already set up,

You don’t need command prompt if using sqlite in Visual Studio.

Just open, insert, condition, close, and run the sql stuff by using syntax INSIDE your visual c++ editor in your project.

1. Setting up SQL with cmake:

//If have time

2. Insert build into project folder in ..\project\project

Data link by:

---------3.--------

\*\*Start HERE as of 04/27/14 12:01pm

SYNTAX if using Database in Visual Studio C++ Project and if ICS 168 anything Start:

\*\*AS of Now (04/47/14 11:18am) , everything above is done. Just open project as is . Do this now:

0.

Assuming that C:\sqlite3 is installed in computer settings, and cmake installed lib, include, bin, into the project folder and again in the project\project folder:

IF NEED TO OPEN .db file to check that your .db has the correct stuff:

Command prompt

a) \*\*IF b) doesn’t work, Must run sqlite separately first in order to open .db file first as shown:

-cd C:\

cd sqlite

sqlite3

b) Skip here if a) not needed:

cd …..project\project

sqlite3

3. C++ SQL Syntax:

Assuming that C:\sqlite3 is installed in computer settings, and cmake installed lib, include, bin, into the project folder and again in the project\project folder:

\*\*If this doesn’t work for some reason when you open the project later,

in Command Prompt: open the path to the project\project and type sqlite3 first. Doubt you need to do this though. But just in case, do this.

sub\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.

Database Creation:

\*\*Syntax should be able to create a \_\_\_.db file on its own without the need to create the .db file in command prompt. BUT IN CASE you need to create the .db file in command prompt first,

Do this:

Assuming that C:\sqlite3 is installed in computer settings, and cmake installed lib, include, bin, into the project folder and again in the project\project folder: paste again as reminder :

cd ….project\project

sqlite3 ---WORKS!!!! Can run ANYWHERE on ANY path thanks to making C:\sqlite3 in computer path (syntax to run sqlite first)

.open anyName.db -Creates it

.quit when you are done with the program

IF want to do something else with .db code:

cd ……project\project

sqlite3 *anyName*.db -This doesn’t create a database. This runs a created database using only sql.

**STEP 1: OPEN and create .db like this: See “Note 2” above if this doesn’t work:**

By default, do this first:

.db creation syntax:

sqlite3 \*db;

int error = sqlite3\_open("test.db", &db);

sub\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//After .db file creation is assumed to work:

2.

\*\*Start here if db already made.

Using Database BASE CODE:

#include <iostream>

#include <string>

#include "sqlite3.h"

int main()

{

sqlite3 \*db;

sqlite3\_stmt \*res;

const char \*errMSG;

const char \*tail;

int error = sqlite3\_open("test.db", &db);

if(error)

{

std::cout << "Couldn't open the database" << std::endl;

sqlite3\_close(db);

system("pause");

return 0;

}

std::string query = "SELECT \* FROM test";//Fixed bug. Changed test2 to test

error = sqlite3\_prepare\_v2(db, query.c\_str(), query.length(), &res, &tail);

if(error != SQLITE\_OK)

{

std::cout << "Couldn't prepare sql" << std::endl;

sqlite3\_close(db);

system("pause");

return 0;

}

while(sqlite3\_step(res) == SQLITE\_ROW)

{

std::cout << "Name: " << sqlite3\_column\_text(res, 0) << std::endl;

std::cout << "Number: " << sqlite3\_column\_text(res, 1) << std::endl << std::endl;

}

sqlite3\_finalize(res);

sqlite3\_close(db);

system("pause");

return 0;

}

….

Milestone 3:

NOTICE how the scene object gets two variants

The loginScene

And

The gameScene

So you can assign the scene object in main.cpp two different types

Just fine!!!!

\*\*NOTICE the functions in GameScene and LoginScene that changes flags so that main.cpp can refer to them and change Scene\* object

to one of its variants when needed.

//TCP SERVER Solution:

//How NetworkManager Project -the Server Project Works:

Critical Base:

3 Objects utilized:

TCPListener object; checks if client connected

Sockets object will be used to send and receive the data

List\* clients -This object used as reference for writing the conditions

In how sockets manages the way it sends and receives our data. YAY!!!!

TO DO NOW:

Client Project send the data

Using

Listener class not needed in Client project

Its own Socket class

Socket objects send packet objects

Packet is sent online by doing

Packet << … << --each << is in order

Data will be received from client on server like this

ServerPacket >> …. >> ……..

Client: Uses Sockets.h, and Packets.h

Sockets send packets

Server:

Uses Sockets.h, and Packets.h

Sockets send packets

We need to set up a

Sfml-dev.org/documentation

Format of Readme:

-Entries broken up by underscores

-Date and time recorded

-Milestone that the date relates to also labeled in entry

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

05/01/14

Milestone 3

//!!!!Layout DOES NOT need to start out this fancy for entries. Just type whatever comes to mind. !!!!!

Client

* Used online library to encrypt the password sent to server
* Player username and password sent from loginScene in client.sln as packets to main.cpp in server.sln
* Mb5.h and md5.cpp does the password encrypting

Server

* Implements if/else conditions to check if username already exists in server’s database
* If not, send y and add
* If so, send n and client has to login in again

What we used: ifs, elses, sqlite call, << for sending the encrypted packets

05/02/14

The important descriptions on how exactly the code was implemented and ran is all shown at the beginning of DatabaseManager.cpp and NetworkManager.cpp

Made sure this packet sent to database doesn't destroy the database:

"; Drop Table swarchTable; <-------This is a sql injection

How we handle this is that when constructing our sql query, we bind the input (like shown above) separately.

EX: "SELECT \* FROM swarchTable WHERE userName = ?"

The '?' can be swapped out by another input by using the command

sqlite3\_bind\_text();

Disconnect Works now

When client that connects has registered,

if from same place and username matches, let him play

Now we just need to get the password validation for the player that reconnects.

Password validation for player connection established.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_