**Міністерство освіти і науки України**

**Київський Фаховий Коледж електронних приладів**

**Тема**

**"візуальний інтерфейс бази даних інтернет магазину"**

Пояснювальна записка до курсового проекту

БД 121ХХ ПЗ

Розробив\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Пожарський Н. С.)

**підпис**

“\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_ 2020 р.

Керівник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Гончаренко Н.Є.)

**підпис**

“\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_ 2020 р.

2020

**Зміст**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

*БД12116ПЗ*

Разраб.

Пожарський Н.С

Провер.

Реценз

Н. Контр.

Утверд.

*БД12116ПЗ*

Лит.

Листов

39

1 ЗАГАЛЬНА ЧАСТИНА

1.1 Постановка задачі ...............................................................................3

1.2 Дослідження і аналіз об’єкту програмування ....................................3

1.3 Використані програмні засоби ............................................................6

1.4 Вимоги до апаратного та програмного забезпечення .....................7

2 ПРАКТИЧНА ЧАСТИНА

2.1 Створення та налагодження програми..............................................8

2.2 Опис програми та її алгоритмів ........................................................9

2.3 Схема зв’язків в таблицях .................................................................11

2.4 Інструкція програміста.......................................................................12

2.5 Інструкція оператора .........................................................................15

ВИСНОВОК...............................................................................................25

ЛІТЕРАТУРА.............................................................................................27

ДОДАТОК А (код програми)...................................................................28

**1 ЗАГАЛЬНА ЧАСТИНА**

**1.1 Постановка задачі**

Створити графічний інтерфейс до бази даних інтернет магазину. На цю тему мене

надихнуло вивчення нового фреймворку до с++ та саме з темою вивчення нових можливостей цього фреймворку виникла ідея написання цією програми ще на початку навчального року, що згодом стало ідеєю для курсової роботи

**1.2 Дослідження і аналіз об’єкту програмування**

На початку проектування роботи, треба як слід обдумати всі процеси, які можуть нашкодити програмі, наприклад при введені символьного типу даних туди де потрібен числовий може виникнути помилка. Завдяки Фреймворку Qt C++ та окремо класу QSqlRelationTableModel який спроектований у фреймворку. Завдяки ньому на екран виводиться дані у повному обсязі, без «забруднення» рук.

QSqlRelationTableModel – клас для роботи з базою даних, цей клас представляє у вигляді таблиці усі дані бази даних, запити, тощо… При грамотному проектуванні, клас може виводити повну інформацію про базу даних. У моїй програмі ми можемо побачити не тільки повну інформацію про базу даних, а й можливість корегувати її, створювати складні запити за допомогою текстового поля, або шукати інформацію про конкретного клієнта у базі, тобто річ, яку він збирається придбати, кількість яку він хоче придбати, номер його телефону та ціна за 1 штуку товару, який клієнт бажає придбати.

На початку програми ми маємо вибрати базу даних з якою збираємося працювати, у нас з’являється стартове діалогове вікно (так прийнято називати спроектовані класи, які успадковують структуру QDialog Qt фреймворку) яке дозволяє вибрати користувачу базу даних.

Проблеми, які виникли при проектуванні діалогового вікна – потоки підключення до бази даних змішувались та програма не працювала, згодом проблема була вирішена доданням ще одної змінної, яка під’єднувалася до бази в окремому діалоговому вікні та після його використання потік закривався, що не заважало створенню іншого.

Перебравшись до нашої основної програми, розуміємо, що для того щоб редагувати інформацію у табличках нам потрібні відповідні кнопки, тому було прийнято рішення про створення 5-ти основних кнопок

1. Add Row (додання строки)
2. Delete Row(видалення строки)
3. Submit (Підтвердження останніх змін)
4. Select (Вивід на екран вибраної таблиці)
5. Revert (Відмінити останні зміни)

Також у реалізації присутні два меню, перше – з іменами клієнтів, за якою можно знаходити детальну інформацію про не підтверджені покупки клієнта, друге – з назвами таблиць, їх може бути безліч, але має бути 3 основних таблиці, за якими буде проводитися пошук інформації – client, item, relation.

Client – таблиця з існуючими клієнтами.

Item – таблиця з продуктом продажу.

Relation – відношення між клієнтом та продуктом продажу.

Ці три таблиці мають бути для коректної роботи програми, тому що кожен сайт має таблиці з інформацією про клієнтів, про товар та їх взаємо зв’язок.

В програмі існує строка, яка приймає в себе повноцінні запити на додавання інформації до таблиць та додавання таблиць зовсім.

Присутні також два режиму роботи програми:

-Standart

-Inner Join

Перший за умовчуванням виконує всі дії, а коли програма в другому стані, вона виконує лише запит inner Join, більш легкої версії, яку я методом створення алгоритму скоротив.

З стандартного запиту Inner Join на 3 таблиці

select client.name, item.name, relation.kolvo, item.price, client.phone, relation.date from item inner join relation on

item.id = relation.id\_it inner join client on relation.id\_cl = client.id

Мені вдалося створити

«OUTPUT client.name as clientName item.name relation.kolvo item.price client.phone FROM item relation client ON item.id relation.id\_it relation.id\_cl client.id»

Він не занадто коротший, але більш простий.

Як можна побачити, у останьому варіанті існує свій синтаксис, який методом парсингу даних витягує зі строки корисні дані та перетворює її у перший запит.

3 головних слова – OUTPUT, FROM, ON, без яких не буде працювати запит сприймається як ключі, за якими програма сприймає запит як inner Join, чи як будь-який інший запит.

Важливо, що програма у стані \*Inner Join не буде виконувати стандартні функції, такі як додати, видалити, тощо…

Після формування запиту, або будь-якої взаємодії з функціональністю програми, відсилається певний запит до бази даних на локальному сервері «postgresql» після чого дані опрацьовуються та надсилають певний результат до моделі, яка згодом перетворює цей запит у візуальну картинку, яку ми бачимо на екрані.

**1.3 Використані програмні засоби**

-Qt creator version 5.12.0

-Microsoft \_word

-PgAdmin 4 (база даних на локальному сервері)

**1.4 Вимоги до апаратного та програмного забезпечення**

|  |  |
| --- | --- |
| **Процессор:** | Intel Core 2 Quad CPU Q6600 2.40GHz (4 CPUs) / AMD Phenom 9850 Quad-Core Processor (4 CPUs) 2.5GHz |
| **Оп.память:** | 4GB |
| **Видеокарта:** | NVIDIA 9800 GT 1GB / AMD HD 4870 1GB |

**2 ПРАКТИЧНА ЧАСТИНА**

**2.1 Створення та налагодження програми**

Створення програми проходило поетапно.

1 – етап проектування. Етап на якому програміст продумує всі можливі складнощі та проблеми, які можуть виникнути. Продумування концепції та мети програми.

2- Написання коду. Написання коду можна розділити на дві частини, так як програма використовує два віджети. Перше діалогове вікно (також називається віджетом) програмувалося без особих проблем, так як це мала частина проекту, та сама найпростіша. На цьому етапі створилось декілька слотів (

слот – функція, яка заміняє «call back» у С++. Є ідеалом завдяки своєї оптимізованості та простоті в використанні), декілька public функцій, та приватних полів.

Наступний етап кодування – головна програма, в якій проходить більшість дій. Тут в конструкторі за замовчуванням створюється весь головний функціонал, йде під'єднання до слотів різних кнопок та дій (QAction), для зручного та швидкого фідбеку до користувача.

Головна функція main, яка знаходиться у файлі main.cpp

Має не більше 10 строк коду, в який йде виклик класу “Test” для подальшого користування програмою

**2.2 Опис програми та її алгоритмів**

Програма виконує функцію полегшувача в використанні бази даних, саме для цього вона була придумана.

Алгоритми програми не складні та базуються в основному на створені запитів.

Функцій у програмі доволі багато, тому запропоную переглянути декілька доволі цікавих функцій.

QString Test::**queryInnerSearch**(QStringList &lst)

QString – String C++. QStringList – List<String> C++.

Функція, яка робить з нашого не гарного запиту inner Join більш красивіший, та простіший запит. Базується програма на головному циклі, який проходить лише одну ітерацію, тому що в тілі циклу змінна «і» набуває кінечного значення.

Алгоритм побудований на основі зчитування деяких даних з строки та перетворення її на коректний запит методом додавання до тексту невистачаючих спец. Слів. До цього ж, цикл вміє працювати з командою as, яка може вказати будь-яке ім’я таблиці під котрим ми виведемо інформацію.

Конструктор по замовчуванню класу “Test” – голова цієї програми.

Саме в ній створюється основне з’єднання до бази даних та розміщення усіх елементів графічного інтерфейсу на свої місця за допомогою layouts. Заповнення двох меню також відбувається саме тут, як і під’єднання їх до слотів, в яких в свою чергу обробляється кожен з дій (QAction) меню.

Виконується це методом раннього створення запиту, з якого зчитування йде у масив строкових значень, який потім перетворюється на меню.

Метод пошуку інформації за клієнтом відбувається за допомогою запиту inner Join та конкатенації з використанням умови “…Where id = “+str+”’;” де str – id шуканого Клієнта за ім’ям.

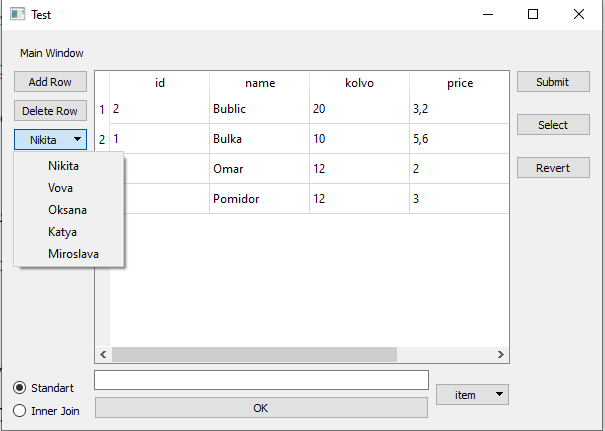


Рис 1. Головне меню програми

**2.3 Схема зв’язків в таблицях**

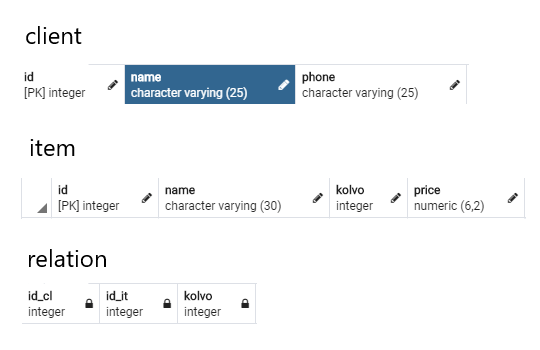
****

Рис 2. Схема таблиць

**2.4 Інструкція програміста**

Програма складається з 6 файлів.

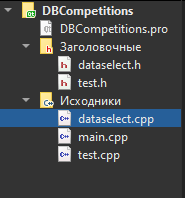


Рис 3. –Скріншот з середи програмування

Два заголовки, ще два – виконавчі, один pro файл, який має бути у кожному qt проекту та main.cpp, файл в якому знаходиться головна функція main(). Клас «Test» містить 8 слотів, 6 функції та 14 змінних, model та db відповідають за саме коректне під’єднання до бази даних.



Рис 4. –Скріншот з файлу test.h

Клас “DialogMain”, який реалізован у файлі “dataselect.cpp” містить у собі 2 слоти, 5 функцій та 4 змінних.

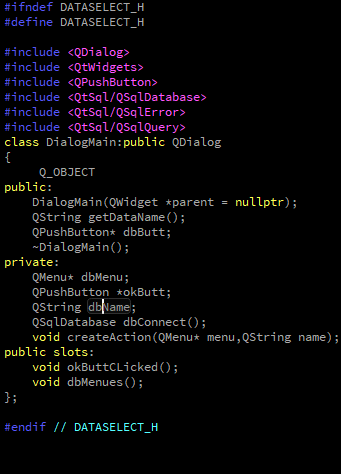


Рис 5. – Скріншот з файлу dialogmain.h

Реалізація усього функціоналу знаходиться у cpp файлах.

З’єднання з базою відбувається на драйвері «QPSQL» з хост ім’ям «127.0.0.1», назва користувача та пароль – «postgres», “2426”

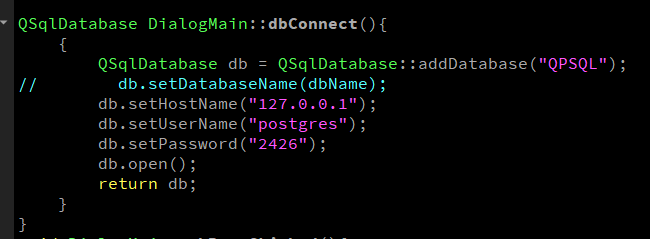


Рис 6. – Скріншот функції, яка під’єднує до бази даних

**2.5 Інструкція оператора**

На початку програми користувача зустрічає діалогове вікно, в якому треба визначитися з базою даних

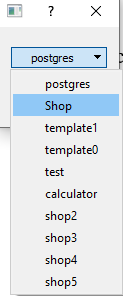
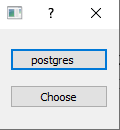


Рис 7. Стартове вікно програми

Після того, як ви вибрали з якою базою даних працювати, перед вами відчиняється головне вікно, яке має такий вигляд

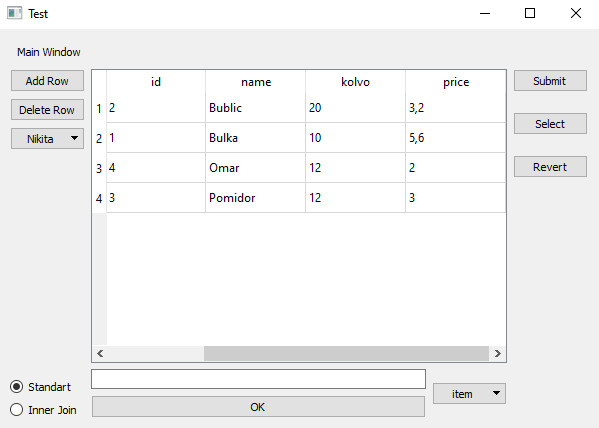


Рис 8. Головне меню програми

Користувач має змогу змінювати активну таблицю в правому кутку, в меню треба обрати таблицю, яка нас цікавить.

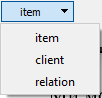


Рис 9. –Список таблиць

Можна додати інформацію, натиснувши на кнопку “add Row”

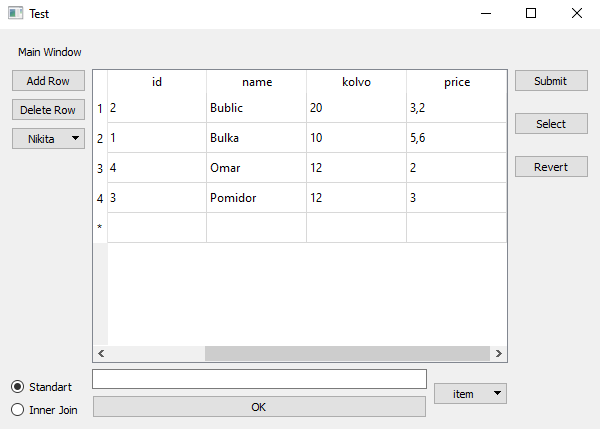


Рис 10. –Добавлення нового поля

Після занесення даних, натискаємо кнопку submit для того, щоб зафіксувати останні дії та зберегти дані у базі даних.

Після натиску, елементу присвоюється id згідно останього доданого елемента.

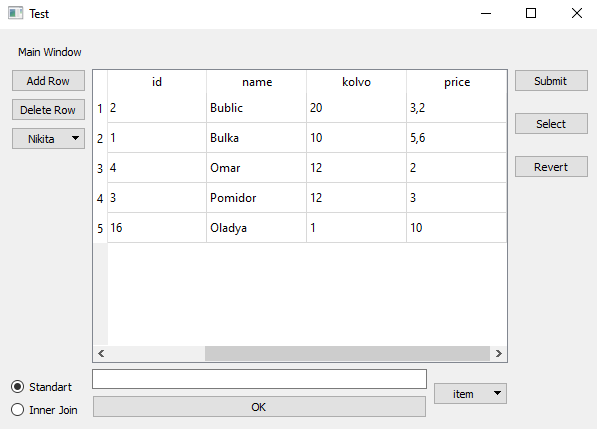


Рис 11. – Добавлення нового поля

Також ми можемо натиснути на кнопку «Select» яка має поновити вікно, тобто «освіжити» таблицю з даними, якщо були внесені зміни.

Кнопка «revert» відповідає за відмову останніх дій та повернення на стадію до внесення змін. ВАЖЛИВО! Цю кнопку треба натискати до кнопки revert, тому що після натиску «revert» запит відправляється на базу даних і процес не повернути назад, тільки якщо видалити строку.

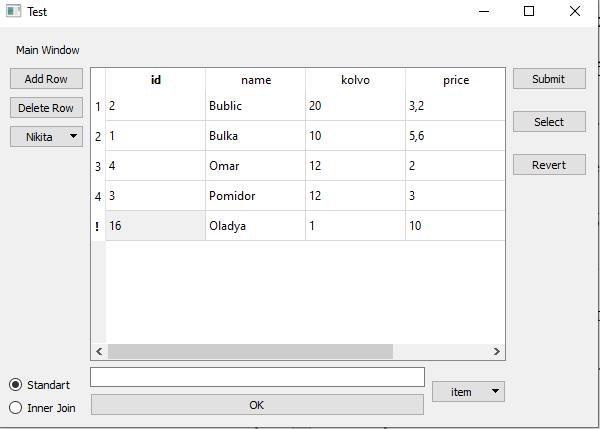
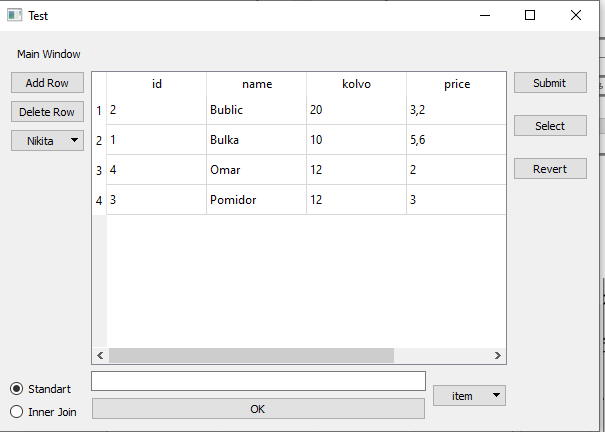
Кнопка «delete Row» видаляє строку. Після вибору рядка треба натиснути на кнопку, а тоді на “Submit”

Рис 12. – Видалення строки

  
Рис 13. – Стартове вікно з видаленною строкою

Нижче можна побачити флаги, зараз програма знаходиться в режимі стандартного флагу, щоб його змінити необхідно натиснути на кружечок біля іншого стану. Якщо вибрати флаг «Inner Join», то в нас буде працювати лише строка та вона буде чекати на запит у строку. Запит має формуватися наступним чином

«OUTPUT client.name as clientName item.name relation.kolvo item.price client.phone FROM item relation client ON item.id relation.id\_it relation.id\_cl client.id;» це дозволить вивести інформацію про всіх клієнтів, які додали до корзини товар. Після чого потрібно нажати «ОК» та побачити результат

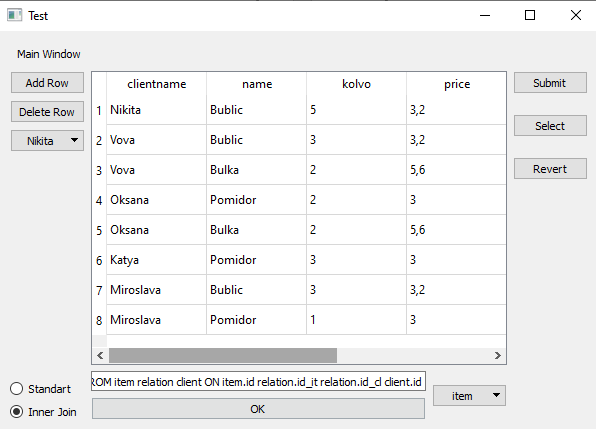


Рис 14. – Зразок роботи команднох строки

В стандартному режимі Рядок також працює, але приймає звичайні sql запити, наприклад «select\* from client»

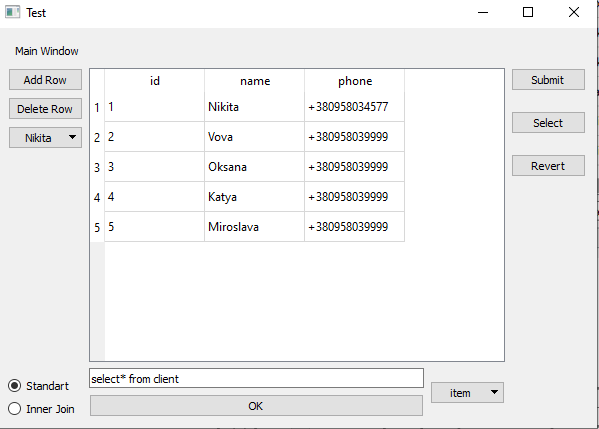


Рис 15. –Вивід таблиці cient на екран

Крапку з комою в кінці не обов'язково вводити, як у мові sql. Програма сама її доставить.

**Висновок**

Отже, програма створена для зручного та легкого використання баз даних наприклад для інтернет магазинів. Вона працює без перебоїв, стабільно, а головне дані захищені від ураження(від бовдура). Інформація виводиться чітко та зрозуміло.

Основна особливість – доволі непоганий функціонал враховуючи легкість освоювання.

Елементарна в зрозумінні, ховає реалізацію від користувача, що робить її повністю доступною для всіх.

Мінуси – для повного функціонування програми, має бути стабільно 3 таблиці з однаковою назвою, або деякі можливості будуть відсутні. Вона одноразова, тобто, якщо вам потрібна буде інша база, потрібно буде вийти з додатку та запустити знову.

Значно зручніше користуватися програмою ніж власноруч використовувати Базу даних без стороннього софту. За наявності графічного інтерфейсу користувач може здійснити багато операцій без знань мови SQL.

Софт призначений для користувачів, які не мають багатого досвіду у знанні SQL, або зовсім без них.

Додаток доволі простий та корисний, рекомендую до використання!

**ЛІТЕРАТУРА**

1. Атре, Ш. Структурный подход к организации баз данных / Ш. Атре. - М.: Финансы и статистика, **2010**. - 317 c.
2. Бланшет, Саммерфилд - Qt4 Программирование GUI на С++. 2ed. – 2008
3. Макс Шпее ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ НА Qt 5.10 C++ Санкт-Петербург « БХВ-Петербург» 2018
4. <https://www.cyberforum.ru/> - cyberforum
5. <https://github.com/> - github

**dataselect.h**

#ifndef DATASELECT\_H

#define DATASELECT\_H

#include <QDialog>

#include <QtWidgets>

#include <QPushButton>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlError>

#include <QtSql/QSqlQuery>

class **DialogMain**:public QDialog

{

Q\_OBJECT

public:

**DialogMain**(QWidget \*parent = nullptr);

QString **getDataName**();

QPushButton\* dbButt;

~***DialogMain***();

private:

QMenu\* dbMenu;

QPushButton \*okButt;

QString dbName;

QSqlDatabase **dbConnect**();

void **createAction**(QMenu\* menu,QString name);

public slots:

void **okButtCLicked**();

void **dbMenues**();

};

#endif // DATASELECT\_H

**Test.h**

#ifndef TEST\_H

#define TEST\_H

#include <QtWidgets>

#include <QPushButton>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlRelationalTableModel>

#include <QtSql/QSqlError>

#include <QtSql/QSqlQuery>

#include <QDialog>

#include "dataselect.h"

class **QPushButton**;

class **Test** : public QWidget

{

Q\_OBJECT

public:

**Test**(QWidget\* widget = nullptr);

void **createAction**(QMenu\* menu,QString name);

// void createActionForData(QMenu\* menu,QString name);

QString **queryInnerSearch**(QStringList& lst);

QSqlDatabase **dbConnect**();

QString dbName; // dbName;

~***Test***();

public slots:

void **addRowButtonClicked**();

void **deleteRowButtonClicked**();

// void innerJoinButtonClicked();

// void leftJoinButtonClicked();

void **submitButtonClicked**();

void **selectButtonClicked**();

void **revertButtonClicked**();

void **okButtonClicked**();

void **clientNamesAction**();

void **MenuTableSelected**();

// void MenuDatabaseSelected();

private:

QTableView \*table;

QLineEdit\* command;

QSqlDatabase db;

QSqlRelationalTableModel\* model;

QGroupBox \* group;

QRadioButton\* standart;

QRadioButton\* innerJoin;

// QRadioButton\* leftJoin;

QPushButton\* cmd;

QPushButton\* cmd2;

QMenu \* menu, \*menu2;

QStringList dbList;

QStringList namesLst;

QStringList idLst;

};

#endif // TEST\_H

**dataSelect.cpp**

#include "dataselect.h"

#include <QDialog>

DialogMain::**DialogMain**(QWidget \*parent)

: QDialog(parent)

{

{ QVBoxLayout\* mainLayout = new QVBoxLayout();

QSqlDatabase db2 = dbConnect();

okButt = new QPushButton("Choose");

QStringList dbList;

dbButt = new QPushButton("postgres");

QMenu \* menu2 = new QMenu(dbButt);

{QSqlQuery checkQuery;

checkQuery.exec("Select\* from pg\_database;");

while(checkQuery.next()){

QString str = checkQuery.value(0).toString();

dbList<<str;

}}

// qDebug()<<dbList;

for(int i=0;i<dbList.size();i++){

QAction\* setTable = new QAction(dbList[i]);

connect(setTable,SIGNAL(triggered()), SLOT(dbMenues()));

menu2->addAction(setTable);

}

dbButt->setMenu(menu2);

mainLayout->addWidget(dbButt);

mainLayout->addWidget(okButt);

setLayout(mainLayout);

connect(okButt, SIGNAL(clicked()), SLOT(okButtCLicked()));

db2.close();}

}

QSqlDatabase DialogMain::**dbConnect**(){

{

QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");

// db.setDatabaseName(dbName);

db.setHostName("127.0.0.1");

db.setUserName("postgres");

db.setPassword("2426");

db.open();

return db;

}

}

void DialogMain::**okButtCLicked**(){

dbName = dbButt->text();

close();

}

QString DialogMain::**getDataName**(){

return dbName;

}

void DialogMain::**dbMenues**(){

QString str = ((QAction\*) sender())->text();

dbButt->setText(str);

}

DialogMain::~***DialogMain***(){

}

**Test.cpp**

#include "test.h"

#include <vector>

Test::**Test**(QWidget\* widget)

:QWidget(widget)

{

DialogMain dialog;

dialog.setModal(true);

dialog.resize(100,100);

dialog.*exec*();

dbName = dialog.getDataName();

//connect

// if(dbName.isEmpty())

db = dbConnect();

db.setDatabaseName(dbName);

if(!db.open()){

qDebug()<<db.lastError().text();

return;

}

else

qDebug()<<"Success\n";

// creating LAYOUTS...

QHBoxLayout\* hLayout = new QHBoxLayout();

QHBoxLayout\* hLayout2 = new QHBoxLayout();

QVBoxLayout\* lay\_Center = new QVBoxLayout();

QVBoxLayout\* vLayout= new QVBoxLayout();

QVBoxLayout\* vLayout2= new QVBoxLayout();

QVBoxLayout\* vLayout3 = new QVBoxLayout();

// MainLayout

QVBoxLayout\* mainLayout = new QVBoxLayout();

// Menu and other BUTTONS and WIDGETS FOR GUI

cmd = new QPushButton();

menu = new QMenu(cmd);

QStringList list = db.tables();

for(int i =0; i<list.size();i++)

createAction(menu,list[i]);

cmd->setText(list[0]);

cmd->setMenu(menu);

{

model = new QSqlRelationalTableModel(this,db);

model->*setEditStrategy*(QSqlTableModel::*OnManualSubmit*);

model->*setTable*(list[0]);

model->*select*();

}

cmd2 = new QPushButton();

menu2 = new QMenu(cmd2);

QSqlQuery checkQuery;

checkQuery.exec("select name from client;");

while(checkQuery.next()){

QString str = checkQuery.value(0).toString();

namesLst<<str;

} if(!namesLst.isEmpty()){

// qDebug()<<namesLst;

for(int i=0;i<namesLst.size();i++){

QAction\* setTable = new QAction(namesLst[i]);

connect(setTable,SIGNAL(triggered()), SLOT(clientNamesAction()));

menu2->addAction(setTable);

}

cmd2->setText(namesLst[0]);

cmd2->setMenu(menu2);

}

else cmd2->setText(" ");

checkQuery.exec("select id from client;");

while(checkQuery.next()){

QString str = checkQuery.value(0).toString();

idLst<<str;

}

QPushButton\* addRow = new QPushButton("Add Row");

QPushButton\* deleteRow = new QPushButton("Delete Row");

QPushButton\* submit = new QPushButton("Submit");

QPushButton\* select = new QPushButton("Select");

QPushButton\* revert = new QPushButton("Revert");

QToolBar \*bar = new QToolBar();

table = new QTableView();

command = new QLineEdit();

QPushButton \* ok = new QPushButton("OK");

//RadioButtons and GroupBox

group = new QGroupBox("Type");

standart = new QRadioButton("Standart");

innerJoin = new QRadioButton("Inner Join");

// leftJoin = new QRadioButton("Left Join");

standart->setChecked(true);

//bar

bar->addAction("Main Window");

//LEFT MENU LAYOUT

vLayout->addWidget(addRow,1);

vLayout->addWidget(deleteRow,2);

vLayout->addWidget(cmd2);

vLayout->addStretch(1);

vLayout->addWidget(standart);

vLayout->addWidget(innerJoin);

// vLayout->addWidget(leftJoin);

//BOTTOM MENU ROW AND BUTTON "OK"

vLayout3->addWidget(command);

vLayout3->addWidget(ok);

//ADD BOTTOM MENU IN HORISONTAL LAYOUT

hLayout2->addLayout(vLayout3);

hLayout2->addWidget(cmd);

//CENTER LAYOUT ADD TABLE AND LAYOUT WITH BOTTOM MENU

lay\_Center->addWidget(table);

lay\_Center->addLayout(hLayout2);

//temporary space

int\* space = new int(14);

//RIGHT MENU LAYOUT

vLayout2->addWidget(submit,1);

vLayout2->addSpacing(\*space);

vLayout2->addWidget(select,2);

vLayout2->addSpacing(\*space);

vLayout2->addWidget(revert,3);

vLayout2->addStretch(1);

delete space;

//CREATING ONE LAYOUT WHICH HAVE ALL MENU

hLayout->addLayout(vLayout);

hLayout->addLayout(lay\_Center);

hLayout->addLayout(vLayout2);

//MAIN LAYOUT WHICH HAVE HORISONTAL MENU AND TOOL BAR

mainLayout->addWidget(bar);

// mainLayout->addWidget(cmd2);

mainLayout->addLayout(hLayout);

//SET MAIN LAYOUT

setLayout(mainLayout);

//SET DATABASE MODEL FOR VIEW

table->*setModel*(model);

//CONNECTS

connect (addRow, SIGNAL(clicked()), SLOT(addRowButtonClicked()));

connect (deleteRow, SIGNAL(clicked()), SLOT(deleteRowButtonClicked()));

connect (submit, SIGNAL(clicked()), SLOT(submitButtonClicked()));

connect (select, SIGNAL(clicked()), SLOT(selectButtonClicked()));

connect (revert, SIGNAL(clicked()), SLOT(revertButtonClicked()));

connect (ok, SIGNAL(clicked()), SLOT(okButtonClicked()));

connect (cmd, SIGNAL(clicked()), SLOT (MenuTableSelected()));

}

void Test::**addRowButtonClicked**(){

qDebug()<<"Inserting Row";

model->insertRow(model->*rowCount*());

}

void Test::**deleteRowButtonClicked**(){

int selectRow = table->currentIndex().row();

if(selectRow >=0) model->removeRow(selectRow);

else qDebug()<<"no row Selected";

}

void Test::**submitButtonClicked**(){

model->submitAll();

}

void Test::**selectButtonClicked**(){

model->*select*();

}

void Test::**revertButtonClicked**(){

model->revertAll();

}

void Test::**okButtonClicked**(){

QString str= command->text();

if(standart->isChecked()){

QStringList lst;

lst<<str.split(" ");

if(lst[0]=="set"){

QString buffer;

if(lst[lst.size()-1][lst[lst.size()-1].size()-1] == ';'){

for(int i = 0;i<lst[lst.size()-1].size()-1;i++){

buffer +=lst[lst.size()-1][i];

}

lst[lst.size()-1] = buffer;

}

model->*setTable*(lst[1]);

model->*select*();

table->*setModel*(model);

}

else{

QSqlQueryModel\* query1 = new QSqlQueryModel();

query1->setQuery(str);

if(lst[0] == "select" || lst[0] =="select\*"){

table->*setModel*(query1);

}

}

}else if(innerJoin->isChecked()){

QStringList lst;

lst<<str.split(" ");

QSqlQueryModel\* query1 = new QSqlQueryModel();

QString query = queryInnerSearch(*lst*);

query1->setQuery(query);

table->*setModel*(query1);

//qDebug()<<lst;

}

// else if(leftJoin->isChecked()){}

}

void Test::**MenuTableSelected**(){

if(standart->isChecked()){

QString str = ((QAction\*) sender()) ->text();

cmd->setText(str);

model->*setTable*(str);

model->*select*();

table->*setModel*(model);

}

}

void Test::**createAction**(QMenu\* menu,QString name){

QAction\* setTable = new QAction(name);

connect (setTable,SIGNAL(triggered()),SLOT(MenuTableSelected()));

menu->addAction(setTable);

}

QString Test::**queryInnerSearch**(QStringList &lst){

QStringList newList;

QString queryRow = "select ";

int counter=0;

for(int i=0;i<lst.size();i++){

newList<<lst[i].toLower();

if(newList[i] == "output" || newList[i]=="from" || newList[i]=="on") counter++;

}

if(counter == 3){

for(int i=1;i<newList.size();i++){

while(newList[i]!= "from"){

queryRow+=newList[i];

if(newList[i+1] == "as") queryRow+=" ";

if(newList[i] == "as"){

queryRow+=" ";

i++;

queryRow+=newList[i];

queryRow+=" ";

}

if(newList[i+1]!= "from" && newList[i+1] != "as")

queryRow+=", ";

i++;

}

queryRow+=" ";

queryRow += newList[i];

i++;

queryRow+=" ";

QStringList tables;

while(newList[i]!= "on"){

tables<<newList[i];

i++;

}

queryRow+= tables[0];

i++;

// ITEM queryRow+=newList[i];

for(int j=1;j<tables.size();j++){

queryRow+=" Inner Join ";

queryRow+=tables[j];

//if(sz%2 != 0) qDebug()<<"Incorrect relation";

queryRow+=" on ";

queryRow+=newList[i];

i++;

queryRow+=" = ";

queryRow+=newList[i];

i++;

}

}

}

else qDebug()<<"Incorrect Query";

return queryRow;

}

QSqlDatabase Test::**dbConnect**(){

{

QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");

// db.setDatabaseName(dbName);

db.setHostName("127.0.0.1");

db.setUserName("postgres");

db.setPassword("2426");

db.open();

return db;

}

}

void Test::**clientNamesAction**(){

if(standart->isChecked()){

QString str = ((QAction\*) sender())->text();

cmd2->setText(str);

QString id;

for(int i =0;i<namesLst.size();i++){

if(str == namesLst[i]) id = idLst[i];

}

QSqlQueryModel\* query1 = new QSqlQueryModel();

QString query = "select client.name, item.name, relation.kolvo, item.price, client.phone from item inner join relation on item.id = relation.id\_it inner join client on relation.id\_cl = client.id where id\_cl ="+id+";";

query1->setQuery(query);

table->*setModel*(query1);

}

}

Test::~***Test***(){

}

**Main.cpp**

#include "test.h"

#include "dataselect.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

// dialog->show();

Test t;

t.setWindowTitle("Test");

t.resize(600,400);

t.show();

return a.exec();

}