Practice 8.

Relational databases in Qt

A relational database is a database based on a relational data model.

1. Class adding for relational connections.

To provide relational connections in Qt, use the class

QSqlRelationalTableModel.

So, we will replace the QSqlTableModel class, used earlier, with QSqlRelationalTableModel:

– in header file:

```
// data model
QSqlRelationalTableModel *model;
```

– in the file with realization:

```
// declare the data model
model = new QSqlRelationalTableModel(this);
```

2. Adding a field for connection.

Task:

Write a SQL query that adds a new field "***group_id***" (group number) to the "***person***" table.3.

Try to display the contents of this table (this is to check the correctness of your SQL query, after you have done this - return everything to its original state).

To do this, replace

```
model->setTable("person");
```

on

```
model->setTable("group_list");
```

4. Adding connections.

```
model->setRelation(5, QSqlRelation ("group_list", "group_id", "group_name"));
```

// 5 - foreign key column, "group_list" - the table with which to link,

"group_id" is the primary key column, "group_name" is the substituted field

Pay particular attention to the foreign key column number (numbering columns starts at zero).

Test your application.

5. Adding a new window to display the table.

Suppose we need to review and edit table "group_list", for this you need to create a new window, place on it a ***Table View*** element and associate it with the new model.

We will implement all this through the menu and the toolbar.

5.1. Creation of the menu.

Double click on the mainwindow.ui form - the visual editor will open (fig. 1). Find the words ***Write here*** and follow this direction.

Create a standard menu Authorization with items: ***Authorization, Logout***.

Add a separator between the ***Login*** and ***Logout*** menu items.

Create another ***Administration menu*** with the names of your tables (for example, Group Table).

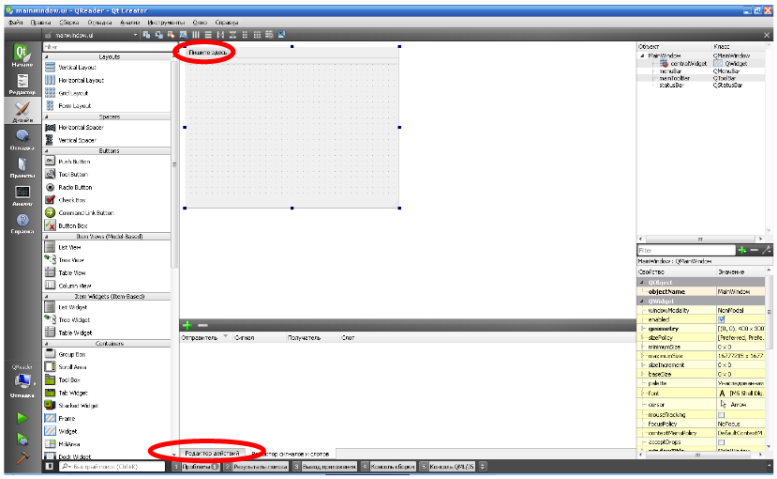

Fig.1. Editing the main window

5.2. Editing actions.

If you look at the bottom of the screen (Fig. 1, highlighted in red at the bottom), you will find that there appeared the so-called ***Action*** - reactions on pressing the corresponding menu items.

Let's modify them by replacing the names with more meaningful ones, for example:

action_Login, action_Quit and action_tbl_St, action_tbl_Gr, as well as add hotkeys to them.

This is done simply: in the action editing window. To get into it, use double-click on the action (Fig. 2), press: ***Ctrl + L, Ctrl + Q*** for the corresponding *authorization* and *logout* actions.

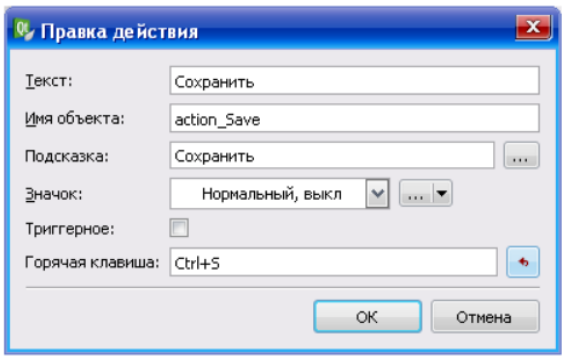

Fig.2. Action editing window

Note!: enter the names of actions carefully, for example, do not accidentally put a space at the end in it (this is not so easy to notice).

5.3. In order to make the menu more attractive, we will add our own icon to each action, and so that the icons have a place to live, we will create a resource file (Fig. 3).

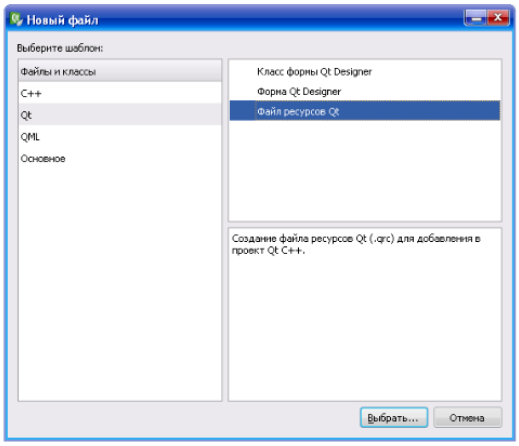

Fig.3. Creating a resource file

You will be asked to enter its name (use the name of the project), as well as the location of the resource file (it can be placed anywhere, but it is better to place it in the project directory, which the program helpfully recommends). Click ***Next*** and then ***Finish***.

Now use double click on ***QBD.qrc*** (if you named the file that way) in the sidebar.

Then click on the ***Add*** button and select ***Add Prefix***. You can change the name of the prefix to something more meaningful.

Now we need to find (or create ourselves) the files that we will use as icons. The size of the icons should be 32x32.

It is recommended to take the files from **"<Qt Installation Path> \ <Qt Version #> \\ mingw492_32 \ examples \ widgets \ mainwindows \ application \ images".**

Add them to the resource file.

It is better to move the image files to the program directory and place them, for example, in the "images" folder you created.

Compile the project.

Let's go back to the form editing window and set icons for all actions.

This can be done through the ***Action edit window*** (Fig. 2), or through the ***properties edit window*** (Fig. 4). Thus, set icons for all menu items (Fig. 5)

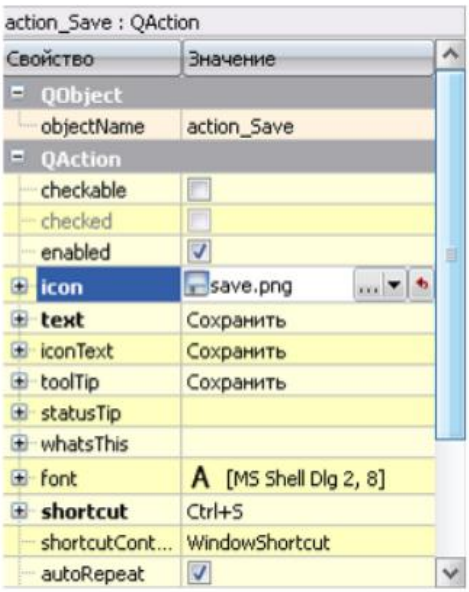

Fig.4. ***Properties edit window***

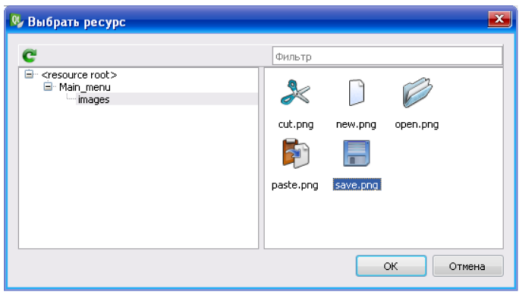

Fig.5. Selection of icons

5.4. Next to the action editor there is an editor for signals and slots (Fig. 6), go to it and sequentially select: action_Quit, triggered (), MainWindow, close ().

Now, by clicking on the ***Exit menu item***, the main application window will close.

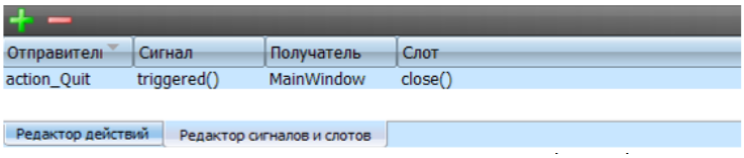

Fig.6. action_Quit

5.5. The application should be beautiful: change the title of the window to "The best client application to the database" (or any other one of your choice) and add an icon to the window (having previously connected it in the resource file).

See the WindowTitle property.

5.6. Toolbar.

When you try to add something to the toolbar, you can run into some difficulties, since the context menu does not allow you to add anything other than an object called the **Separator**.

Everything is quite simple: select the desired action (in the action editor) and drag it to the toolbar - it will appear on the toolbar (Fig. 7).

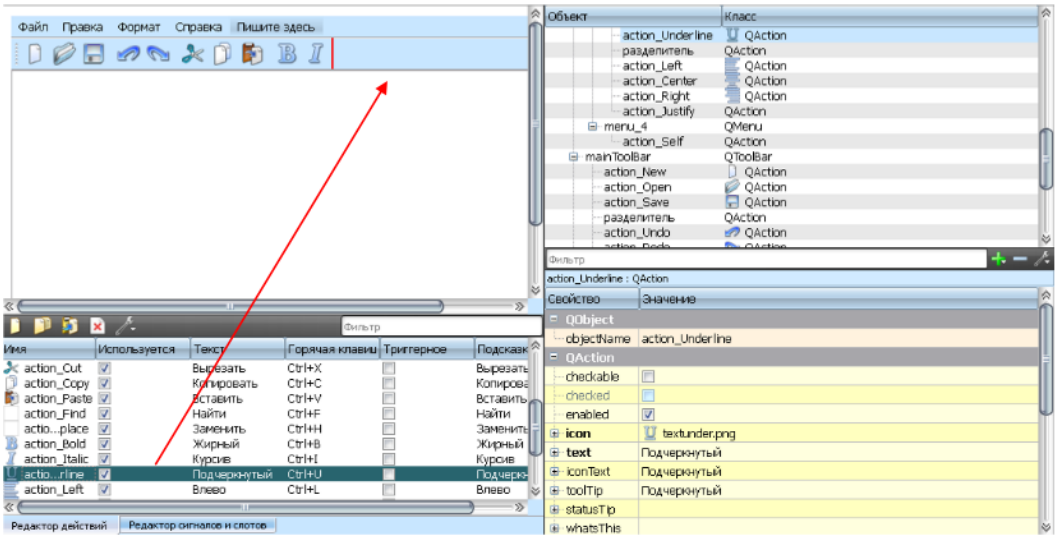

Fig.7. Adding actions to the toolbar

You don't need to write anything else: the procedures you have already created are automatically connected to these buttons. All that remains is to add *separators*, positioning them the same as in the menu.

Select the Toolbar in the ***Object Inspector***, or simply click on it on the form.

In the ***Properties window***, find ***iconSize*** and set the width and height values to 96 and 48, respectively (Figure 8).

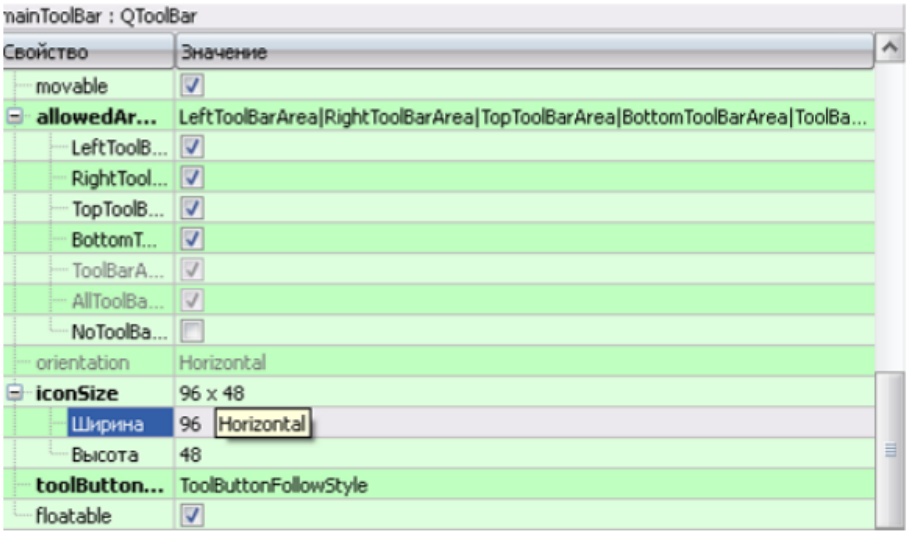

Рисунок 8.17 – Задание размеров иконок

Fig.8. Sizing icons

5.7. Connecting a new application window to display a table with group numbers.

In order to connect a new form to the program, you should: right-click on the root folder in the project tree (see Fig. 2), then select ***Add New ...,*** in the window that appears, select, respectively, **Qt** and ***Qt Designer Form Class***.

Then set the class name: "***Tbl_Gr_Dialog***", respectively, the form file will be named "***tbl_gr_dialog.ui***". The header file and the source code file will have the same names, but with a different extension.

Right-click on ***action_tbl_Gr*** in the action editor and select: ***Go to slot ...,*** then (Fig. 9), click ***OK***.

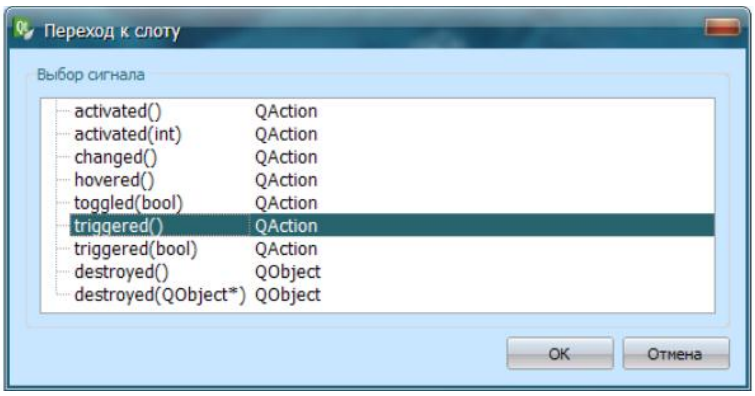

Fig 9.

The connection is implemented like this (in the mainwindow.cpp file):

```
#include "tbl_gr_dialog.h"
```

In `void MainWindow::on_action_tbl_gr_triggered():`

```
Tbl_Gr_Dialog form_group;
 form_group.exec();
```

there you need to add the following code so that the data in the main table is updated:

```
model->select();
```

and add:

```
model->relationModel(5)->select();
```
//5 - foreign key column,

It updates the data displayed in the dropdown list.

5.8. Form filling.

Add a Table View element to the form and connect it to the "group_list" table, compose the form, give it a title, set an icon for the window.

Note !: if you want the changes not to be immediately saved to the database, but to accumulate until the "***Save changes***" button is pressed, then you can create two new actions (***action_Save, action_Undo***) and write this implementation in the constructor:

```
connect(ui->action_Save,SIGNAL(triggered()),model,SLOT(submitAll()));
connect(ui->action_Undo,SIGNAL(triggered()),model,SLOT(revertAll()));
```

In this case, you will need to remove submitAll () in the program text for this code to function normally. This note is not a guide to action, but only the information - it is better to leave the program code as it is.

6. To select group numbers from the available list of groups on our main table we need to add just one line to the constructor:

```
ui-> tableView-> setItemDelegate (new QSqlRelationalDelegate (ui->tableView));
```

Add the table "group_list" to have enough record to select.

7. To implement the join operation of relational algebra, the constants ***InnerJoin*** and ***LeftJoin*** are used:

```
model->setJoinMode(QSqlRelationalTableModel::LeftJoin);
```

By default, the ***InnerJoin*** mode is set, which prevents the display of records with zero foreign keys (i.e., you will not see data for students who do not have a group number). ***LeftJoin*** - will display all fields.

8. Task:

- provide an opportunity to add and remove groups;
- implement authorization (login and password entry) through a separate window launched before the start of the main application window (which will start only when the correct password is entered);

note that ***QLineEdit*** also supports ***echoMode: NoEcho*** and ***Password***.