Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Тихоокеанский государственный университет»

Тимошко А.М., Тусикова А.А.

ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ БАЗЫ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ QT

Методические указания к лабораторной работе №3 по дисциплине «Проектирование приложений баз данных» для студентов специальности 231000.62 «Программная инженерия»

Хабаровск
2016

Проектирование приложения базы данных с использованием Qt: методические указания к лабораторной работе №3 по дисциплине «Проектирование приложений баз данных» для студентов специальности 231000.62 «Программная инженерия» / сост. Тимошко А.А., Тусикова А.А. – Хабаровск: 2016. – 21 с.

Методические указания к лабораторной работе №3 по дисциплине «Проектирование приложений баз данных» составлены студентами для помощи другим студентам при выполнении данной работы. В них изложен доступным языком материал для практического применения: написание скриптов PostgreSQL, работа с моделями таблиц базы данных в Qt, фильтры, поиск и прочее.

Хабаровск, 2016

В качестве примера создадим базу данных rabota со следующей структурой таблиц:

kafera	
Код кафедры	Наименование кафедры
id_kafedra *	n_kafedra
INTEGER	VARCHAR

prepod	
Код преподавателя	Ф.И.О. преподавателя
id_prepod *	fio
INTEGER	VARCHAR

ed_izm		
Код единицы измерения	Наименование единицы измерения	Эквивалент в часах
id_izm *	n_izm	hours
INTEGER	VARCHAR	REAL

sfera		
Код сферы деятельности	Код единицы измерения	Наименование сферы деятельности
id_sfera *	id_izm	n_sfera
INTEGER	INTEGER	VARCHAR

pokazatel		
Код показателя	Код сферы деятельности	Наименование показателя
id_pokazatel *	id_sfera	n_pokazatel
INTEGER	INTEGER	VARCHAR

rabota						
Код работы	Наим. работы	Код показат.	Код кафедры	Код препод.	Учебный год	Объем работы
id_rabota *	n_rabota	id_pokazatel	id_kafedra	id_prepod	year	volume
INTEGER	VARCHAR	INTEGER	INTEGER	INTEGER	INTEGER	REAL

Этап I. Написание и выполнение скрипта SQL

Для нашего примера напишем скрипт, который выполняет SQL-запросы для создания таблиц. В PostgreSQL по умолчанию имена таблиц и столбцов приводятся к нижнему регистру, поэтому для исключения путаницы будем создавать их в нижнем регистре.

Содержимое скрипта Script.sql:

```
CREATE SEQUENCE ids_kaf;
CREATE TABLE kafedra (id_kafedra INTEGER PRIMARY KEY
DEFAULT NEXTVAL('ids_kaf'), n_kafedra VARCHAR UNIQUE NOT
NULL);

CREATE SEQUENCE ids_prep;
CREATE TABLE prepod (id_prepod INTEGER PRIMARY KEY DEFAULT
NEXTVAL('ids_prep'), fio VARCHAR UNIQUE NOT NULL);

CREATE SEQUENCE ids_izm;
CREATE TABLE ed_izm (id_izm INTEGER PRIMARY KEY DEFAULT
NEXTVAL('ids_izm'), n_izm VARCHAR UNIQUE NOT NULL, hours
REAL NOT NULL);

CREATE SEQUENCE ids_sf;
CREATE TABLE sfera (id_sfera INTEGER PRIMARY KEY DEFAULT
NEXTVAL('ids_sf'), id_izm INTEGER NOT NULL, n_sfera
VARCHAR UNIQUE NOT NULL, FOREIGN KEY(id_izm) REFERENCES
ed_izm(id_izm) ON DELETE CASCADE ON UPDATE CASCADE);

CREATE SEQUENCE ids_pokaz;
CREATE TABLE pokazatel (id_pokazatel INTEGER PRIMARY KEY
DEFAULT NEXTVAL('ids_pokaz'), id_sfera INTEGER NOT NULL,
n_pokazatel VARCHAR UNIQUE NOT NULL, FOREIGN KEY(id_sfera)
REFERENCES sfera(id_sfera) ON DELETE CASCADE ON UPDATE
CASCADE);

CREATE SEQUENCE ids_rab;
CREATE TABLE rabota (id_rabota INTEGER PRIMARY KEY DEFAULT
NEXTVAL('ids_rab'), n_rabota VARCHAR, id_pokazatel INTEGER
NOT NULL, id_kafedra INTEGER NOT NULL, id_prepod INTEGER
NOT NULL, year INTEGER NOT NULL, volume REAL NOT NULL,
FOREIGN KEY(id_pokazatel) REFERENCES
pokazatel(id_pokazatel) ON DELETE CASCADE ON UPDATE
CASCADE, FOREIGN KEY(id_kafedra) REFERENCES
kafedra(id_kafedra) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(id_prepod) REFERENCES prepod(id_prepod) ON
DELETE CASCADE ON UPDATE CASCADE,
UNIQUE(n_rabota,id_pokazatel,id_kafedra,id_prepod,year));
```

Свойство auto_increment для id задается в PostgreSQL при помощи последовательностей (SEQUENCE) и функции NEXTVAL. Для возможных ключей ставится свойство UNIQUE. Внешние ключи создаются при помощи следующей конструкции:

FOREIGN KEY (<имя поля текущей таблицы>) REFERENCES <имя таблицы, на которую ссылается>(<имя поля, на которое ссылается>). Политика ON DELETE CASCADE означает, что каждая строка дочерней таблицы, которая ассоциирована с удаляемой родительской строкой, также будет удалена. Для действия ON UPDATE CASCADE значения, сохранённые в зависящем дочернем ключе, будут заменены на новые значения родительского ключа.

Для выполнения скрипта запустите сервер PostgreSQL и выполните команду:

```
psql –U admin –d rabota –f “Script.sql” --single-transaction
```

Описание ключей:

-U – имя пользователя, от которого нужно выполнить команду.

-d – имя базы данных.

-f – использовать файл в качестве источника команд.

--single-transaction – выполнить в пределах одной транзакции. Гарантирует, что, либо все команды будут успешно выполнены, либо никакие изменения не применяются.

Аналогичный скрипт можно создать для заполнения таблиц данными, используя запросы INSERT.

Этап II. Создание приложения БД в Qt

На данном этапе мы рассмотрим создание приложения для ведения нашей базы данных, используя библиотеку Qt. Реализуем форму авторизации пользователя, создадим меню для навигации по таблицам, настроим вывод таблиц и отображение функциональных кнопок, согласно правам пользователей, фильтры, контекстный поиск и добавим сообщения об ошибках и каскадном удалении записей базы данных.

Шаг 1. Создание форм приложения

Для нашего приложения создадим всего две формы. В основной форме будет отображаться текущая таблица, фильтры, строка поиска и

меню. Другая форма будет вызываться для авторизации и соединения пользователя с сервером PostgreSQL.

Поместим на форму mainwindow.ui:

- виджет **Table View** с именем **tableView**. В этом виджете будет отображаться выбранная пользователем таблица.
- виджет **Line Edit** с именем **editSearch**. Этот элемент будет использоваться в качестве строки контекстного поиска.
- кнопку **Push Button** с именем **buttonClear**. Данная кнопка предназначена для очистки результата поиска и возвращения исходной таблицы.
- виджеты **Combo Box** с именами **boxIzm, boxSfera, boxKafedra, boxPrepod, boxYear, boxPokazatel**. Эти виджеты мы будем использовать для организации фильтрации записей в таблице, согласно отбору. Для каждой таблицы устанавливаются свои фильтры, поэтому мы будем делать проверку на то, какая таблица открыта пользователем, и скрывать ненужные для нее Combo Box'сы, а нужные показывать:

```
// скрыть Combo Box, установив макс. размер 0x0
ui->boxIzm->setMaximumWidth(0);
ui->boxIzm->setMaximumHeight(0);
// показать Combo Box
ui->boxSfera->setMaximumWidth(16777215);
ui->boxSfera->setMaximumHeight(16777215);
```

- кнопку **Push Button** с именем **buttonReset**. Эта кнопка предназначена для сбора фильтров.
- кнопки **Push Button** с именами **buttonAdd** и **buttonDel**. Данные кнопки пользователь будет использовать для добавления или удаления записи в таблице базы данных.

Меню приложения создадим программно. Наше меню будет выглядеть следующим образом:

Меню	Основные таблицы	Справочники
Авторизация	Работа	Кафедры
		Преподаватели
		Единицы измерения
		Сферы деятельности
		Показатели

Рис. 1. Схематичное изображение меню приложения

В файле maimwindow.cpp (в конструкторе) пропишем код:

```
menuMain = new QMenu(tr("&Меню"), this);
menuMainTables = new QMenu(tr("&Основные таблицы"), this);
menuOtherTables = new QMenu(tr("&Справочники"), this);

actAutorization = new QAction(tr("&Авторизация"), this);
actRabota = new QAction(tr("&Работа"), this);
actKafedra = new QAction(tr("&Кафедра"), this);
actPrepod = new QAction(tr("&Преподаватели"), this);
actEdIzm = new QAction(tr("&Единицы измерения"), this);
actSfera = new QAction(tr("&Сферы деятельности"), this);
actPokazatel = new QAction(tr("&Показатели"), this);

menuMain->addAction(actAutorization);
menuMainTables->addAction(actRabota);
menuOtherTables->addAction(actKafedra);
menuOtherTables->addAction(actPrepod);
menuOtherTables->addAction(actEdIzm);
menuOtherTables->addAction(actSfera);
menuOtherTables->addAction(actPokazatel);

ui->menuBar->addMenu(menuMain);
ui->menuBar->addMenu(menuMainTables);
ui->menuBar->addMenu(menuOtherTables);
```

Создадим новую форму autorization.ui. Для этого ПКМ по группе «Формы» в окне проектов. Далее выбираем «Класс формы Qt Designer», затем шаблон «Widget» и задаем имя autorization.

На созданную форму поместим две кнопки с именами **ButtonConnect** и **ButtonDisconnect**, три объекта **Line Edit** с именами **editHost, editUser, editPassword** и один объект **Spin Box** с именем **editPort**. Для **editPort** установим свойство ***maximum*** в 10000, а ***value*** – в 5432 (значение порта PostgreSQL по умолчанию).

В итоге получим две формы:

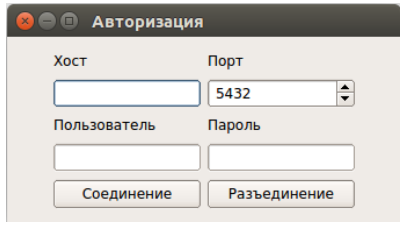

Рис. 2. Форма authorization.ui

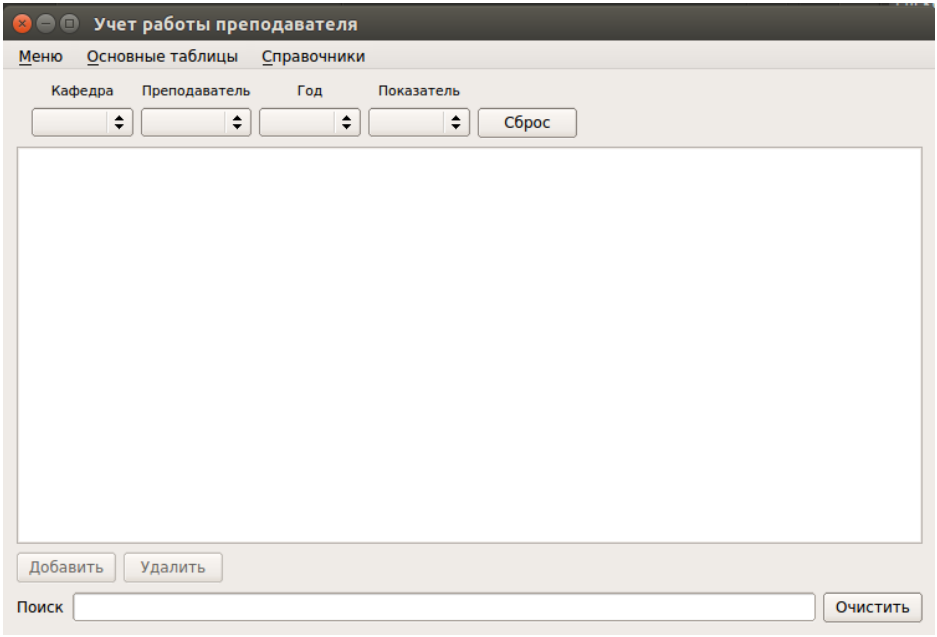

Рис. 3. Форма mainwindow.ui

Для того, чтобы данные с формы authorization.ui передавались в основной класс MainWindow, проделаем следующие действия:

1. Объявим в файле authorization.h сигналы и доступные другим классам переменные:

```
signals:
    void sendConnect();
    void sendDisconnect();
public:
    QString user;
    QString password;
    QString host;
    int port;
```

2. В файле authorization.cpp пропишем код:

```
void Autorization::on_ButtonConnect_clicked()
{
    host = ui->editHost->text();
    port = ui->editPort->value();
    user = ui->editUser->text();
    password = ui->editPassword->text();
    emit sendConnect();
}
```

```
void Autorization::on_ButtonDisconnect_clicked()
{
    ui->editUser->setText("");
    ui->editPassword->setText("");
    emit sendDisconnect();
}
```

При помощи **emit** мы испускаем нужный нам сигнал вручную. Эти сигналы может отлавливать наш основной класс MainWindow.

3. Пропишем в файле mainwindow.cpp (в конструкторе) код:

```
formAutorization = new Autorization();
connect(formAutorization, SIGNAL(sendConnect()), this,
SLOT(Connect()));
connect(formAutorization, SIGNAL(sendDisconnect()),
this, SLOT(Disconnect()));
```

При испускании сигнала sendConnect() вызывается слот Connect() класса MainWindow, аналогично при испускании sendDisconnect() управление передается Disconnect() класса MainWindow.

4. Содержимое слота Connect():

```
{
    db = QSqlDatabase::addDatabase("QPSQL");
    db.setHostName(formAutorization->host);
    db.setPort(formAutorization->port);
    db.setDatabaseName("rabota");
    db.setUserName(formAutorization->user);
    db.setPassword(formAutorization->password);
    formAutorization->hide();

    if(!db.open())
        QMessageBox::critical(this, tr("ERROR!"),
db.lastError().databaseText());
    else
        QMessageBox::information(this, tr("SUCCESS!"),
tr("Соединение установлено! Выберите таблицу для
работы"));

    model = new QSqlRelationalTableModel(0, db);
    ui->tableView->setModel(model);
}
```

5. Содержимое слота Disconnect():

```
{
    model = new QSqlRelationalTableModel(0, db);
    ui->tableView->setModel(model);
    db.close();
    formAutorization->hide();
}
```

Шаг 2. Отображение таблиц

Для отображения всех таблиц будем использовать одну модель QSqlRelationTableModel, а на виджете Table View в данный момент времени будет отображаться только одна таблица, которая выбрана пользователем через меню. Реализуем функцию selectTable(QString nameTable). В качестве параметра она получает имя таблицы, которую выбрал пользователь, внутри функции производятся настройки модели в зависимости от таблицы: имя столбцов, отношение (relation), фильтры Combo Box.

```
void MainWindow::selectTable(QString nameTable)
{
    ui->tableView->show();
    table = new QSqlRelationalTableModel(0, db);
    table->setTable(nameTable);

    if(nameTable == "kafedra")
    {
        table->setHeaderData(1, Qt::Horizontal,
QObject::tr("Наименование кафедры"), Qt::DisplayRole);
    }

    else if(nameTable == "prepod")
    {
        table->setHeaderData(1, Qt::Horizontal,
QObject::tr("Ф.И.О. преподавателя"), Qt::DisplayRole);
    }

    else if(nameTable == "ed_izm")
    {
        table->setHeaderData(1, Qt::Horizontal,
QObject::tr("Название ед. изм."), Qt::DisplayRole);
        table->setHeaderData(2, Qt::Horizontal,
QObject::tr("Эквивалент в часах"), Qt::DisplayRole);
    }

    else if(nameTable == "sfera")
    {
        table->setHeaderData(1, Qt::Horizontal,
QObject::tr("Название ед. изм."), Qt::DisplayRole);
        table->setHeaderData(2, Qt::Horizontal,
QObject::tr("Наименование сферы деятельности"),
Qt::DisplayRole);
```

```
        table->setRelation(1, QSqlRelation("ed_izm",
"id_izm", "n_izm"));
        ui->tableView->setItemDelegate(new
QSqlRelationalDelegate(ui->tableView));
        // установить содержимое фильтра Combo Box
        QSqlQueryModel *filter = new QSqlQueryModel;
        filter->setQuery("SELECT n_izm FROM ed_izm");
        ui->boxIzm->setModel(filter);
        ui->boxIzm->setCurrentIndex(-1);
    }

    else if(nameTable == "pokazatel")
    {
        table->setHeaderData(1, Qt::Horizontal,
QObject::tr("Наименование сферы деятельности"),
Qt::DisplayRole);
        table->setHeaderData(2, Qt::Horizontal,
QObject::tr("Наименование показателя"), Qt::DisplayRole);
        table->setRelation(1, QSqlRelation("sfera",
"id_sfera", "n_sfera"));
        ui->tableView->setItemDelegate(new
QSqlRelationalDelegate(ui->tableView));

        QSqlQueryModel *filter = new QSqlQueryModel;
        filter->setQuery("SELECT n_sfera FROM sfera");
        ui->boxSfera->setModel(filter);
        ui->boxSfera->setCurrentIndex(-1);
    }

    else if(nameTable == "rabota")
    {
        table->setHeaderData(1, Qt::Horizontal,
QObject::tr("Наименование работы"), Qt::DisplayRole);
        table->setHeaderData(2, Qt::Horizontal,
QObject::tr("Наименование показателя"), Qt::DisplayRole);
        table->setHeaderData(3, Qt::Horizontal,
QObject::tr("Наименование кафедры"), Qt::DisplayRole);
        table->setHeaderData(4, Qt::Horizontal,
QObject::tr("Ф.И.О. преподавателя"), Qt::DisplayRole);
        table->setHeaderData(5, Qt::Horizontal,
QObject::tr("Учебный год"), Qt::DisplayRole);
        table->setHeaderData(6, Qt::Horizontal,
QObject::tr("Объем в ед. изм."), Qt::DisplayRole);
        table->setRelation(2, QSqlRelation("pokazatel",
"id_pokazatel", "n_pokazatel"));
```

```
        table->setRelation(3, QSqlRelation("kafedra",
"id_kafedra", "n_kafedra"));
        table->setRelation(4, QSqlRelation("prepod",
"id_prepod", "fio"));
        ui->tableView->setItemDelegate(new
QSqlRelationalDelegate(ui->tableView));

        QSqlQueryModel *filter1 = new QSqlQueryModel;
        filter1->setQuery("SELECT n_pokazatel FROM
pokazatel");
        ui->boxPokazatel->setModel(filter1);
        ui->boxPokazatel->setCurrentIndex(-1);

        QSqlQueryModel *filter2 = new QSqlQueryModel;
        filter2->setQuery("SELECT n_kafedra FROM
kafedra");
        ui->boxKafedra->setModel(filter2);
        ui->boxKafedra->setCurrentIndex(-1);

        QSqlQueryModel *filter3 = new QSqlQueryModel;
        filter3->setQuery("SELECT fio FROM prepod");
        ui->boxPrepod->setModel(filter3);
        ui->boxPrepod->setCurrentIndex(-1);

        QSqlQueryModel *filter4 = new QSqlQueryModel;
        filter4->setQuery("SELECT DISTINCT year FROM
rabota");
        ui->boxYear->setModel(filter4);
        ui->boxYear->setCurrentIndex(-1);
    }

    table->select();
    ui->tableView->setModel(table);
    // скрыть поле ID
    ui->tableView->setColumnHidden(0, true);
    // установить ширину столбцов
    int count = table->columnCount();
    for(int i = 1; i < count; i++)
    {
        ui->tableView->setColumnWidth(i, 250);
    }
    // стратегия редактирования - ручная
    table-
>setEditStrategy(QSqlTableModel::OnManualSubmit);
}
```

Чтобы при выборе пользователем пункта меню открывалась нужная таблица, необходимо в конструкторе описать сигнал:

```
    connect(kafedraAct, SIGNAL(triggered()), this,
SLOT(selectKafedra()));
```

Функция selectKafedra():

```
void MainWindow::selectKafedra()
{
    QString nameTable = "kafedra";
    selectTable(nameTable);
}
```

Шаг 3. Активность функциональных кнопок в зависимости от прав пользователя

В зависимости от прав пользователь может добавлять и удалять записи из определенной таблицы. Если у него нет каких-либо прав, то соответствующие функциональные кнопки должны быть неактивны. Для этого добавим в функцию selectTable(QString nameTable) следующий код:

```
QString userName;
QSqlQueryModel *role = new QSqlQueryModel;
QComboBox *tempRole = new QComboBox;
// получить имя пользователя из формы авторизации
userName = autoForm->user;
// выполнить запрос на проверку прав
role->setQuery(QString("SELECT
has_table_privilege('%1','%2','INSERT')").arg(userName,
nameTable));
tempRole->setModel(role);
// по результату запроса установить кнопку в активное или
неактивное состояние
if(tempRole->currentText() == "true")
    ui->ButtonAdd->setEnabled(true);

role->setQuery(QString("SELECT
has_table_privilege('%1','%2','DELETE')").arg(userName,nam
eTable));
tempRole->setModel(role);

if(tempRole->currentText() == "true")
    ui->ButtonDel->setEnabled(true);
```

Шаг 4. Фильтры Combo Box

При выборе определенной позиции в фильтре Combo Box в таблице должны показываться только те записи, которые удовлетворяют данному фильтру, причем колонка, по которой была выполнена фильтрация, должна стать невидимой (скрытой). Если пользователь добавит новую запись, то значение скрытого поля должно заполняться автоматически, согласно выбранному значению в фильтре Combo Box. Для реализации вышеописанных требований необходимо обработать сигнал Combo Box:

```
connect(ui->boxKafedra, SIGNAL(activated(QString)), this,
SLOT(setFilterKafedra(QString)));
```

Функция setFilterKafedra(QString currentText):

```
void MainWindow::setFilterKafedra(QString currentText)
{
    tempQueryKafedra = new QSqlQueryModel;
    // получить id кафедры по наименованию
    tempQueryKafedra->setQuery(QString("SELECT id_kafedra
FROM kafedra WHERE n_kafedra = '%1'").arg(currentText));
    tempBoxKafedra = new QComboBox;
    tempBoxKafedra->setModel(tempQueryKafedra);
    // начало текста для фильтрации
    QString text = QString("rabota.id_kafedra =
%1").arg(tempBoxKafedra->currentText());
    // если выбран фильтр по показателю, добавить в текст
    if(ui->boxPokazatel->currentIndex() != -1)
    {
        text += " AND rabota.id_pokazatel =
"+tempBoxPokazatel->currentText();
    }
    // если выбран фильтр по преподавателю, добавить в
текст
    if(ui->boxPrepod->currentIndex() != -1)
    {
        text += " AND rabota.id_prepod = "+tempBoxPrepod-
>currentText();
    }
    // если выбран фильтр по году, добавить в текст
    if(ui->boxYear->currentIndex() != -1)
    {
        text += " AND rabota.year = "+ui->boxYear-
>currentText();
    }
    // установить фильтр
    table->setFilter(text);
```

```
    table->select();
    // скрыть колонку, по которой фильтруем
    ui->tableView->setColumnHidden(3, true);
    }
```

Чтобы сбросить все фильтры, напишем обработчик нажатия кнопки «Сброс»:

```
void MainWindow::on_ButtonReset_clicked()
{
    // убрать фильтр
    table->setFilter("");
    table->select();

    int count = table->columnCount();
    for(int i = 1; i < count; i++)
    {
        // колонки снова сделать видимыми
        ui->tableView->setColumnHidden(i, false);
    }
    // текущее значение Combo Box
    ui->boxIzm->setCurrentIndex(-1);
    ui->boxKafedra->setCurrentIndex(-1);
    ui->boxPokazatel->setCurrentIndex(-1);
    ui->boxPrepod->setCurrentIndex(-1);
    ui->boxSfera->setCurrentIndex(-1);
    ui->boxYear->setCurrentIndex(-1);
}
```

Шаг 5. Динамический («живой») поиск

По мере ввода текста пользователем в строку поиска должен отображаться результат поиска. Для этого будем обрабатывать сигнал Line Edit, модель QSqlQueryModel и тот же Table View, в котором у нас отражается текущая таблица. Функция search(QString string):

```
void MainWindow::search(QString string)
{
    QSqlQueryModel *find = new QSqlQueryModel;
    QString str_query, tableName;
    tableName = table->tableName();

    if(tableName == "rabota")
    {
        str_query = "SELECT id_rabota, n_rabota,
n_pokazatel, n_kafedra, fio, year, volume FROM rabota,
pokazatel, kafedra, prepod WHERE (rabota.id_pokazatel =
```

```
pokazatel.id_pokazatel) AND (rabota.id_kafedra =
kafedra.id_kafedra) AND (rabota.id_prepod =
prepod.id_prepod) AND (n_pokazatel || n_kafedra || fio ||
year || volume || n_rabota LIKE '%"+string+"%')";
        find->setQuery(str_query);
        find->setHeaderData(1, Qt::Horizontal,
QObject::tr("Наименование работы"), Qt::DisplayRole);
        find->setHeaderData(2, Qt::Horizontal,
QObject::tr("Наименование показателя"), Qt::DisplayRole);
        find->setHeaderData(3, Qt::Horizontal,
QObject::tr("Наименование кафедры"), Qt::DisplayRole);
        find->setHeaderData(4, Qt::Horizontal,
QObject::tr("Ф.И.О. преподавателя"), Qt::DisplayRole);
        find->setHeaderData(5, Qt::Horizontal,
QObject::tr("Учебный год"), Qt::DisplayRole);
        find->setHeaderData(6, Qt::Horizontal,
QObject::tr("Объем в ед. изм."), Qt::DisplayRole);
    }
    /* аналогично для других таблиц … */
    // показать результат поиска в Table View
    ui->tableView->setModel(find);
}
```

Сигнал Line Edit:

```
    connect(ui->editSearch,   SIGNAL(textChanged(QString)),
this, SLOT(search(QString)));
```

Чтобы при завершении поиска вернуться к исходной модели таблицы, вставим следующий код в обработчик нажатия кнопки «Очистить»:

```
void MainWindow::on_ButtonClear_clicked()
{
    ui->tableView->setModel(table);
}
```

Шаг 6. Добавление новой записи

При добавлении новой записи следует помнить, что, если применен фильтр и соответствующая колонка скрыта, то значение этой колонки должно присваиваться автоматически. Обработчик нажатия кнопки «Добавить»:

```
void MainWindow::on_ButtonAdd_clicked()
{
    QString tableName;
    tableName = table->tableName();
```

```
    // получить число строк
    int count = table->rowCount();
    // вставить новую строку в конце
    table->insertRow(count);

    if(tableName == "sfera" && ui->boxIzm->currentIndex()
!= -1)
    {
        // если применен фильтр, установить его значение в
новой строке в соответствующей колонке
        table->setData(table->index(count, 1), tempBoxIzm-
>currentText());
    }

    else if(tableName == "pokazatel" && ui->boxSfera-
>currentIndex() != -1)
    {
        table->setData(table->index(count, 1),
tempBoxSfera->currentText());
    }

    else if(tableName == "rabota")
    {
        if(ui->boxPokazatel->currentIndex() != -1)
            table->setData(table->index(count, 2),
tempBoxPokazatel->currentText());
        if(ui->boxKafedra->currentIndex() != -1)
            table->setData(table->index(count, 3),
tempBoxKafedra->currentText());
        if(ui->boxPrepod->currentIndex() != -1)
            table->setData(table->index(count, 4),
tempBoxPrepod->currentText());
        if(ui->boxYear->currentIndex() != -1)
            table->setData(table->index(count, 5), ui-
>boxYear->currentText());
    }
}
```

Шаг 7. Удаление записи

При каскадном удалении производится удаление данной записи и всех записей, ссылающихся на данную. Необходимо проверять наличие таких ссылающихся записей и предупреждать пользователя о том, что они также будут удалены. Обработчик нажатия кнопки «Удалить»:

```
void MainWindow::on_ButtonDel_clicked()
{
    QModelIndex ind;
    QString tableName, str_query;
    QSqlQueryModel *find = new QSqlQueryModel;
    QComboBox *temp = new QComboBox;
    tableName = table->tableName();
    // номер выбранной строки
    int row = ui->tableView->selectionModel()-
>selectedRows(0).first().row();
    // получить ID записи
    QString data = (ui->tableView->model()->data(ui-
>tableView->model()->index(row, 0))).toString();
    // текст запроса для проверки наличия ссылающихся
записей на данную
    if(tableName == "kafedra")
    {
        str_query = "SELECT id_rabota FROM rabota WHERE
id_kafedra = "+data;
    }

    else if(tableName == "prepod")
    {
        str_query = "SELECT id_rabota FROM rabota WHERE
id_prepod = "+data;
    }

    else if(tableName == "pokazatel")
    {
        str_query = "SELECT id_rabota FROM rabota WHERE
id_pokazatel = "+data;
    }

    else if(tableName == "sfera")
    {
        str_query = "SELECT id_pokazatel FROM pokazatel
WHERE id_sfera = "+data;
    }

    else if(tableName == "ed_izm")
    {
        str_query = "SELECT id_sfera FROM sfera WHERE
id_izm = "+data;
    }
    // выполнить запрос для проверки
    find->setQuery(str_query);
```

```
    temp->setModel(find);
    // если имеются зависимые записи, вывести диалоговое
сообщение
    if(temp->currentText() != "")
    {
        int result = QMessageBox::question(this,
tr("Каскадное удаление!"), tr("Внимание! Найдены связанные
данные в зависимых таблицах. Удаление приведет к потере
зависимых данных. Продолжить?"), QMessageBox::Yes,
QMessageBox::No|QMessageBox::Default);
        if(result == QMessageBox::Yes)
        {
            // удалить запись и применить изменения в
случае согласия пользователя
            table->removeRow(row, ind);
            if(!table->submitAll())
            {
                QMessageBox::critical(this, tr("ERROR!"),
db.lastError().databaseText());
            }
        }
    }
    // если нет ссылок, выполнить и применить удаление
    else
    {
        table->removeRow(row, ind);
        if(!table->submitAll())
        {
            QMessageBox::critical(this, tr("ERROR!"),
db.lastError().databaseText());
        }
    }
}
```

Шаг 8. Применение изменений при добавлении и изменении

Так как выбрана стратегия редактирования OnManualSubmit, то все изменения применяются только при вызове слота submitAll(). Перед применением изменений необходимо выполнить проверку значений, которые ввел пользователь, на корректность. Для реализации вышеописанных действий будем обрабатывать нажатие клавиши Enter, когда фокус находится на Table View.

```
void MainWindow::keyPressEvent(QKeyEvent *event)
{
    // если фокус находится на Table View и нажата клавиша
Enter (основная или на Numpad)
    if(ui->tableView->hasFocus() && (event->key() ==
Qt::Key_Return || event->key() == Qt::Key_Enter))
        {
            QString tableName = table->tableName();
            // переменная, которая показывает, все ли
записи корректны
            bool flag = true;
            if(tableName == "rabota")
            {
                int curYear = QDate::currentDate().year();
                // проверяем все строки, которые были
активированы пользователем для редактирования
                for(int i = 0; i < listRows.size(); i++)
                {
                    int row = listRows.value(i);
                    float dataVolume = (ui->tableView-
>model()->data(ui->tableView->model()->index(row,
6))).toFloat();
                    int dataYear = (ui->tableView-
>model()->data(ui->tableView->model()->index(row,
5))).toInt();

                    if(dataYear < 1980 || dataYear >
curYear)
                    {
                        flag = false;
                        QMessageBox::critical(this,
tr("ERROR!"), QString("Строка %1: Учебный год должен быть
в пределах от 1980 по %2").arg(row+1).arg(curYear));
                    }

                    if(dataVolume <= 0 || dataVolume >
3000)
                    {
                        flag = false;
                        QMessageBox::critical(this,
tr("ERROR!"), QString("Строка %1: Объем работы должен быть
положительным и не более 3000").arg(row + 1));
                    }
                }
            }
```

```
            // если записи корректны, применяем изменения
            if(flag)
            {
                if(!table->submitAll())
                {
                    QMessageBox::critical(this,
tr("ERROR!"), table->lastError().databaseText());
                }
                // очистить список измененных строк
                else listRows.clear();
            }
        }
}
```

Для формирования списка строк, которые были активированы пользователем для изменения, включая добавленную строку, обработаем сигнал:

```
connect(ui->tableView, SIGNAL(clicked(QModelIndex)), this,
SLOT(activeIndexTable(QModelIndex)));
```

Функция для добавления измененных строк в список:

```
void MainWindow::activeIndexTable(QModelIndex index)
{
    // если в списке нет еще такой строки
    if(!listRows.contains(index.row()))
    {
        listRows.append(index.row());
    }
}
```