Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(ФГБОУ ВО «СибГУТИ»)

*Кафедра прикладной математики и кибернетики*

Расчетно-графическая работа

по дисциплине

«Основы визуального программирования»

Выполнили:

студенты гр. ИП-513

Майоров С.А.

Санин И.В.

Проверил:

доцент кафедры ПМиК

Ситняковская Е.И.

Новосибирск

2017

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. Реляционные БД

2.2. SQL

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Описание собственных классов

5. ТЕКСТ ПРОГРАММЫ

6. РЕЗУЛЬТАТЫ

ПОСТАНОВКА ЗАДАЧИ

Написать специализированную реляционную СУБД для учета студентов. Побочная задача – изучить мультиплатформенную графическую библиотеку - Qt.

ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

Реляционные БД.

**База данных** — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

**Реляционная база данных** — база данных, основанная на реляционной модели данных.

**Реляционная модель данных** (РМД) — логическая модель данных, прикладная теория построения баз данных, которая является приложением к задачам обработки данных таких разделов математики, как теория множеств и логика первого порядка.

Реляционная модель данных включает следующие компоненты:

* Структурный аспект (составляющая) — данные в базе данных представляют собой набор отношений.
* Аспект (составляющая) целостности — отношения (таблицы) отвечают определенным условиям целостности. РМД поддерживает декларативные ограничения целостности уровня домена (типа данных), уровня отношения и уровня базы данных.
* Аспект (составляющая) обработки (манипулирования) — РМД поддерживает операторы манипулирования отношениями (реляционная алгебра, реляционное исчисление).

Термин «реляционный» означает, что теория основана на математическом понятии отношение (relation). В качестве неформального синонима термину «отношение» часто встречается слово таблица. Необходимо помнить, что «таблица» есть понятие нестрогое и неформальное и часто означает не «отношение» как абстрактное понятие, а визуальное представление отношения на бумаге или экране.

Для лучшего понимания РМД следует отметить три важных обстоятельства:

* модель является логической, то есть отношения являются логическими (абстрактными), а не физическими (хранимыми) структурами;
* для реляционных баз данных верен информационный принцип: всё информационное наполнение базы данных представлено одним и только одним способом, а именно — явным заданием значений атрибутов в кортежах отношений; в частности, нет никаких указателей (адресов), связывающих одно значение с другим;
* наличие реляционной алгебры позволяет реализовать декларативное программирование и декларативное описание ограничений целостности, в дополнение к навигационному (процедурному) программированию и процедурной проверке условий.

Таким образом, для создания реляционной БД необходим набор данных в предметной области и связи между этими данными. Управление реляционной БД осуществляется с помощью реляционной СУБД.

SQL.

SQL (англ. structured query language — «язык структурированных запросов») — язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Язык SQL представляет собой совокупность

* операторов
* инструкций
* вычисляемых функций

Операторы SQL делятся на:

* операторы определения данных:
  + CREATE создает объект БД (саму базу, таблицу, представление, пользователя и т. д.),
  + ALTER изменяет объект,
  + DROP удаляет объект;
* операторы манипуляции данными:
  + SELECT выбирает данные, удовлетворяющие заданным условиям,
  + INSERT добавляет новые данные,
  + UPDATE изменяет существующие данные,
  + DELETE удаляет данные;
* операторы определения доступа к данным;
* операторы управления транзакциями.

ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

В работе используются специализированные БД со следующей структурой:

* Факультет – таблица
  + Код – ключевое поле
  + Факультет – поле
  + Курс – поле
  + Количество групп - поле
* Группа – таблица
  + Код – ключевое поле
  + Название группы – поле
  + Фамилия старосты – поле
  + Количество – поле
  + Код факультета – поле, связанное с внешним полем Факультет-Код
* Студент – таблица
  + Код – ключевое поле
  + ФИО – поле
  + Адрес – поле
  + Телефон – поле
  + Код группы – поле, связанное с внешним полем Группа-Код

«Движком» для БД является компактная встраиваемая реляционная база данных SQLite. Это избавляет от необходимости в клиент-серверных отношениях и позволяет производить все операции над БД локально.

ОПИСАНИЕ ПРОГРАММЫ

Программа представляет собой СУБД, способную создавать, открывать и редактировать реляционные БД определенного (см. выше) формата. Вывод БД осуществляется графически в виде набора таблиц с помощью библиотеки Qt.

Описание собственных классов

DataBase – класс, представляющий БД. Осуществляет функционал связанный с открытием, созданием и прямым изменением (с помощью «SQL инъекций», являющихся, по сути, взломом БД) БД.

AddRec – класс, являющийся окном добавления новой записи в БД.

MainWindow – основное окно программы, связывающее всё воедино и реализующее весь функционал.

ТЕКСТ ПРОГРАММЫ

|  |
| --- |
| *main.cpp* |
| #include "mainwindow.h"  #include <QApplication>  int main(int argc, char \*argv[])  {  QApplication a(argc, argv);  a.setApplicationName("Database Laboratory Work #1\n(DataBase Management System)");  a.setApplicationVersion(QString("%1.%2")  .arg(MAJOR\_VERSION)  .arg(MINOR\_VERSION));  MainWindow w;  w.show();  return a.exec();  } |
| *mainwindow.cpp* |
| #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include <QMessageBox>  #include <QInputDialog>  MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  db = new DataBase(this);  lastOrder = Qt::DescendingOrder;  connect(ui->actionAbout\_Qt, SIGNAL(triggered()), qApp, SLOT(aboutQt()),  Qt::UniqueConnection);  }  MainWindow::~MainWindow()  {  delete ui;  }  void MainWindow::on\_pushButton\_Sort\_clicked()  {  if (lastOrder == Qt::AscendingOrder){  view[ui->tabWidget->currentIndex()]->sortByColumn(1, Qt::DescendingOrder);  lastOrder = Qt::DescendingOrder;  } else {  view[ui->tabWidget->currentIndex()]->sortByColumn(1, Qt::AscendingOrder);  lastOrder = Qt::AscendingOrder;  }  }  void MainWindow::on\_lineEdit\_CharSearch\_textChanged(const QString &arg1)  {  QTableView\* curView = view[ui->tabWidget->currentIndex()];  curView->setSelectionMode(QAbstractItemView::MultiSelection);  curView->clearSelection();  for (int i = 0; i < curView->model()->rowCount(); ++i){  QString str = curView->model()->index(i,1).data().toString();  if (str.contains(arg1) == true) curView->selectRow(i);  }  if (arg1 == "") curView->clearSelection();  }  void MainWindow::on\_lineEdit\_WordSearch\_textChanged(const QString &arg1)  {  QTableView\* curView = view[ui->tabWidget->currentIndex()];  curView->setSelectionMode(QAbstractItemView::MultiSelection);  curView->clearSelection();  for (int i = 0; i < curView->model()->rowCount(); ++i){  QString str = curView->model()->index(i,1).data().toString();  if (str == arg1) curView->selectRow(i);  }  }  void MainWindow::on\_pushButton\_AddRecord\_clicked()  {  int curIndex = ui->tabWidget->currentIndex();  QSqlRelationalTableModel\* curMod = mod[curIndex];  QSqlQueryModel\* qmod = new QSqlQueryModel(this);  if (curIndex > 0){  qmod->setQuery(QString("SELECT id, %1 FROM %2").arg(mod[curIndex-1]->headerData(1,Qt::Horizontal).toString(),  mod[curIndex-1]->tableName()));  }  if (ui->actionManualInput->isChecked()){  curMod->insertRow(curMod->rowCount());  } else {  QString str1 = curMod->headerData(1,Qt::Horizontal).toString(),  str2 = curMod->headerData(2,Qt::Horizontal).toString(),  str3 = curMod->headerData(3,Qt::Horizontal).toString(),  str4 = curMod->headerData(4,Qt::Horizontal).toString();  dialogWin = new AddRec(str1,  str2,  str3,  str4,  qmod,  this);  if (dialogWin->exec() == QDialog::Accepted)  {  QString rec = "insert into " + curMod->tableName() + " values(" + dialogWin->data() + ")";  QSqlQuery query;  query.exec(rec);  this->setupTables();  ui->tabWidget->setCurrentIndex(curIndex);  }  }  }  void MainWindow::on\_actionAbout\_LW\_triggered()  {  QMessageBox::about(this, "About programm",  QString("%1 v. %2")  .arg(qApp->applicationName())  .arg(qApp->applicationVersion()));  }  void MainWindow::setupTables()  {  view.clear();  mod.clear();  QTableView\* newView = new QTableView();  QSqlRelationalTableModel\* newMod = new QSqlRelationalTableModel();  newMod->setTable(TABLE1);  newMod->setHeaderData(0, Qt::Horizontal, QObject::tr("id"));  newMod->setHeaderData(1, Qt::Horizontal, QObject::tr(TABLE1\_FAC));  newMod->setHeaderData(2, Qt::Horizontal, QObject::tr(TABLE1\_YEAR));  newMod->setHeaderData(3, Qt::Horizontal, QObject::tr(TABLE1\_GRNUM));  newMod->select();  newMod->setEditStrategy(QSqlTableModel::OnFieldChange);  newView->setModel(newMod);  newView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);  newView->hideColumn(4);  newView->setItemDelegate(new QSqlRelationalDelegate(newView));  mod.push\_back(newMod);  view.push\_back(newView);  newView = new QTableView();  newMod = new QSqlRelationalTableModel();  newMod->setTable(TABLE2);  newMod->setHeaderData(0, Qt::Horizontal, QObject::tr("id"));  newMod->setHeaderData(1, Qt::Horizontal, QObject::tr(TABLE2\_NAME));  newMod->setHeaderData(2, Qt::Horizontal, QObject::tr(TABLE2\_LNAME));  newMod->setHeaderData(3, Qt::Horizontal, QObject::tr(TABLE2\_NUM));  newMod->setHeaderData(4, Qt::Horizontal, QObject::tr(TABLE2\_FAC));  newMod->setRelation(4, QSqlRelation(TABLE1, "id", TABLE1\_FAC));  newMod->select();  newMod->setEditStrategy(QSqlTableModel::OnFieldChange);  newView->setModel(newMod);  newView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);  newView->setItemDelegate(new QSqlRelationalDelegate(newView));  mod.push\_back(newMod);  view.push\_back(newView);  newView = new QTableView();  newMod = new QSqlRelationalTableModel();  newMod->setTable(TABLE3);  newMod->setHeaderData(0, Qt::Horizontal, QObject::tr("id"));  newMod->setHeaderData(1, Qt::Horizontal, QObject::tr(TABLE3\_FN));  newMod->setHeaderData(2, Qt::Horizontal, QObject::tr(TABLE3\_ADDS));  newMod->setHeaderData(3, Qt::Horizontal, QObject::tr(TABLE3\_PNUM));  newMod->setHeaderData(4, Qt::Horizontal, QObject::tr(TABLE3\_IDGR));  newMod->setRelation(4, QSqlRelation(TABLE2, "id", TABLE2\_NAME));  newMod->select();  newMod->setEditStrategy(QSqlTableModel::OnFieldChange);  newView->setModel(newMod);  newView->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);  newView->setItemDelegate(new QSqlRelationalDelegate(newView));  mod.push\_back(newMod);  view.push\_back(newView);  ui->tabWidget->clear();  ui->tabWidget->addTab(view[0], mod[0]->tableName());  ui->tabWidget->addTab(view[1], mod[1]->tableName());  ui->tabWidget->addTab(view[2], mod[2]->tableName());  connect(view[0], SIGNAL(clicked(QModelIndex)), this, SLOT(select2(QModelIndex)),  Qt::UniqueConnection);  connect(view[1], SIGNAL(clicked(QModelIndex)), this, SLOT(select3(QModelIndex)),  Qt::UniqueConnection);  }  void MainWindow::on\_actionInsert\_SQL\_code\_triggered()  {  bool ok;  QString text = QInputDialog::getText(this, "SQL",  "Code:", QLineEdit::Normal,  QString(""), &ok);  if (ok && !text.isEmpty()) {  db->injectSQL(text);  this->setupTables();  }  }  void MainWindow::on\_actionNew\_triggered()  {  QFileDialog\* dial = new QFileDialog(this, "Create file...",  QDir::homePath(), "Database files (\*.db);; All files (\*.\*)");  dial->exec();  QString lFileName = dial->selectedFiles().first();  if(lFileName.isEmpty())  {  return;  }  system("rm -f "+lFileName.toUtf8());  db->connectToDataBase(lFileName);  this->setupTables();  }  void MainWindow::on\_actionOpen\_triggered()  {  QString lFileName = QFileDialog::getOpenFileName(this, "Open file...",  QDir::homePath(),"Database files (\*.db);; All files (\*.\*)");  if(lFileName.isEmpty())  {  return;  }  db->connectToDataBase(lFileName);  this->setupTables();  }  void MainWindow::on\_pushButton\_DeleteRecord\_clicked()  {  int curIndex = ui->tabWidget->currentIndex();  QTableView\* curView = view[curIndex];  QSqlRelationalTableModel\* curMod = mod[curIndex];  curMod->removeRow(curView->currentIndex().row());  this->setupTables();  ui->tabWidget->setCurrentIndex(curIndex);  }  void MainWindow::on\_actionExport\_triggered()  {  QSqlRelationalTableModel\* curMod = mod[ui->tabWidget->currentIndex()];  QString textData;  int rows = curMod->rowCount();  int columns = curMod->columnCount();  if (curMod->tableName() == TABLE1) columns--;  for (int i = 0; i < columns; i++) {  textData += curMod->headerData(i,Qt::Horizontal).toString();  textData += "\t";  }  textData += "\n";  for (int i = 0; i < rows; i++) {  for (int j = 0; j < columns; j++) {  textData += curMod->data(curMod->index(i,j)).toString();  textData += "\t";  }  textData += "\n";  }  QFileDialog\* dial = new QFileDialog(this, "Create file...",  QDir::homePath(), "TSV files (\*.tsv)");  dial->exec();  QString lFileName = dial->selectedFiles().first();  QFile csvFile(lFileName);  if(csvFile.open(QIODevice::WriteOnly | QIODevice::Truncate)) {  QTextStream out(&csvFile);  out << textData;  csvFile.close();  }  }  void MainWindow::select2(QModelIndex index)  {  QString filter = TABLE2\_FAC + tr(" = %1").arg(mod[0]->index(index.row(),0).data().toString());  mod[1]->setFilter(filter);  mod[1]->selectRow(0);  if (mod[1]->index(0,0).data().toString().isEmpty()){  filter = TABLE3\_IDGR + tr(" = %1").arg("0");  } else {  filter = TABLE3\_IDGR + tr(" = %1").arg(mod[1]->index(0,0).data().toString());  }  mod[2]->setFilter(filter);  }  void MainWindow::select3(QModelIndex index)  {  QString filter = TABLE3\_IDGR + tr(" = %1").arg(mod[1]->index(index.row(),0).data().toString());  mod[2]->setFilter(filter);  }  void MainWindow::on\_actionUpdate\_DB\_triggered()  {  int curIndex = ui->tabWidget->currentIndex();  this->setupTables();  ui->tabWidget->setCurrentIndex(curIndex);  } |
| *addrec.cpp* |
| #include "addrec.h"  #include "ui\_addrec.h"  AddRec::AddRec(QString &name2, QString &name3, QString &name4, QString &name5, QSqlQueryModel \*mod, QWidget \*parent) :  QDialog(parent),  lmod(mod),  fifthColumn(name5),  ui(new Ui::AddRec)  {  ui->setupUi(this);  ui->label\_2->setText(name2);  ui->label\_3->setText(name3);  ui->label\_4->setText(name4);  ui->label\_5->setText(name5);  ui->comboBox->setModel(lmod);  ui->comboBox->setModelColumn(1);  if (fifthColumn == TABLE1\_FAKE){  ui->verticalLayout\_5->removeWidget(ui->label\_5);  delete ui->label\_5;  ui->verticalLayout\_5->removeWidget(ui->comboBox);  delete ui->comboBox;  }  connect(ui->pushButton\_add, SIGNAL(clicked()), this, SLOT(accept()));  connect(ui->pushButton\_cancel, SIGNAL(clicked()), this, SLOT(reject()));  }  AddRec::~AddRec()  {  delete ui;  }  QString AddRec::data()  {  QString data = "null, '"+  ui->lineEdit\_2->text()+  "', '"+  ui->lineEdit\_3->text()+  "', '"+  ui->lineEdit\_4->text()+  "', ";  if (fifthColumn == TABLE1\_FAKE || ui->comboBox->currentText().isEmpty())  data += "null";  else  data += lmod->index(ui->comboBox->currentIndex(),0).data().toString();  return data;  } |
| *database.cpp* |
| #include "database.h"  DataBase::DataBase(QObject \*parent) : QObject(parent){}  DataBase::~DataBase()  {  this->closeDataBase();  }  void DataBase::connectToDataBase(const QString &name)  {  if(!QFile(name).exists()){  this->restoreDataBase(name);  } else {  this->openDataBase(name);  }  }  bool DataBase::restoreDataBase(const QString &name)  {  if(this->openDataBase(name)){  return (this->createTables()) ? true : false;  } else {  return false;  }  return false;  }  bool DataBase::openDataBase(const QString &name)  {  db = QSqlDatabase::addDatabase("QSQLITE");  db.setHostName(DATABASE\_HOSTNAME);  db.setDatabaseName(name);  if(db.open()){  return true;  } else {  return false;  }  }  void DataBase::closeDataBase()  {  db.close();  }  bool DataBase::createTables()  {  QSqlQuery query;  if(!query.exec( "CREATE TABLE " TABLE1 " ("  "id INTEGER PRIMARY KEY AUTOINCREMENT, "  TABLE1\_FAC " VARCHAR(255) NOT NULL,"  TABLE1\_YEAR " VARCHAR(255) NOT NULL,"  TABLE1\_GRNUM " VARCHAR(255) NOT NULL,"  TABLE1\_FAKE " INTEGER"  " )"  ) ||  !query.exec( "CREATE TABLE " TABLE2 " ("  "id INTEGER PRIMARY KEY AUTOINCREMENT, "  TABLE2\_NAME " VARCHAR(255) NOT NULL,"  TABLE2\_LNAME " VARCHAR(255) NOT NULL,"  TABLE2\_NUM " VARCHAR(255) NOT NULL,"  TABLE2\_FAC " INTEGER NOT NULL"  " )"  ) ||  !query.exec( "CREATE TABLE " TABLE3 " ("  "id INTEGER PRIMARY KEY AUTOINCREMENT, "  TABLE3\_FN " VARCHAR(255) NOT NULL,"  TABLE3\_ADDS " VARCHAR(255) NOT NULL,"  TABLE3\_PNUM " VARCHAR(255) NOT NULL,"  TABLE3\_IDGR " INTEGER NOT NULL"  " )"  )  ){  return false;  } else {  return true;  }  return false;  }  bool DataBase::injectSQL(const QString &code)  {  QSqlQuery query;  if(!query.exec(code)){  return false;  } else {  return true;  }  return false;  } |

Результаты

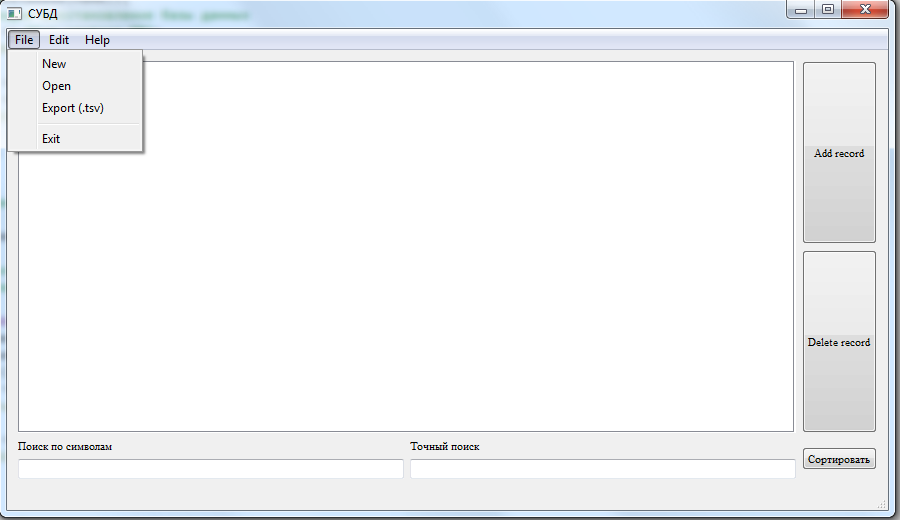


Рис. 1. Общий вид приложения (без БД)

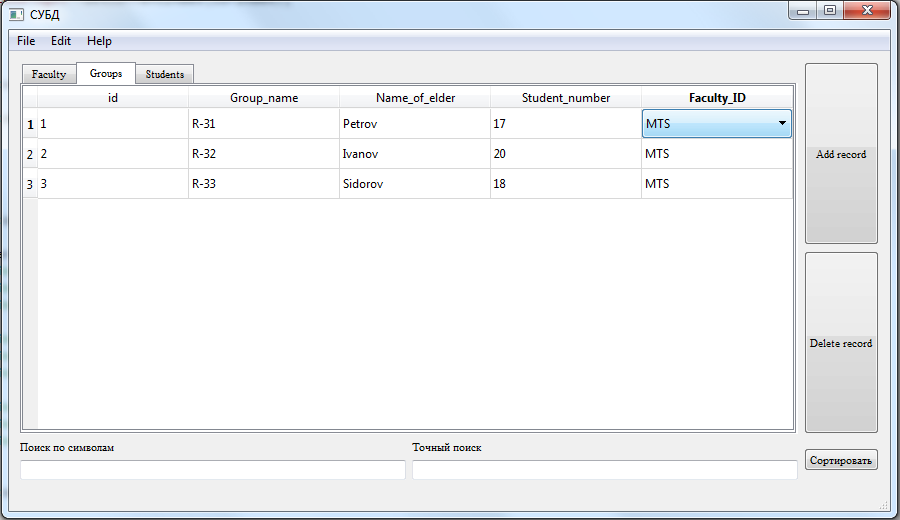


Рис. 2. Общий вид приложения (с БД)