Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки Программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Базы данных

Студент: гр. 951007

Воривода М.А.

Руководитель:

Попивненко Е.А.

Минск 2020

**СОДЕРЖАНИЕ**

[Ввеедение 5](#_Toc482833182)

[1 Анализ аналогов программного средства 7](#_Toc482833183)

[1.1 «PGPTools» **–**  надежный шифратор текстовых сообщений](#_Toc482833184) 7

[1.2 Постановка задачи](#_Toc482833186) 8

[2 Цели и задачи 9](#_Toc482833185)

[2.1 Цель курсовой работы 9](#_Toc482833186)

[2.2 Задачи для курсовой работы 9](#_Toc482833187)

[2.3 Необходимые ресурсы для курсовой работы 9](#_Toc482833188)

[3 Проектирование. Разработка курсовой работы 10](#_Toc482833189)

[3.1 Ассиметричное шифрование………………………………………………………………………… 10](#_Toc482833190) [3.2 Структура программного средства……………………………………………………………….. 10](#_Toc482833191)

[3.3 Разработка графического пользовательского интерфейса…...…………………………10](#_Toc482833191)

[4 Тестирование программного средства 2](#_Toc482833205)6

[5 Руководство пользователя 24](#_Toc482833206)

[Заключение 27](#_Toc482833210)

[Список использованных источников 29](#_Toc482833211)

[Приложение А. Исходный код программы 30](#_Toc482833212)

[Приложение Б. Блок-схема метода](#_Toc482833213) 61

**ВВЕДЕНИЕ**

Компьютеры были созданы для решения вычислительных задач, однако со временем они все чаще стали использоваться для построения систем обработки документов, а точнее, содержащейся в них информации. Такие системы обычно и называют информационными. В качестве примера можно привести систему учета отработанного времени работниками предприятия и расчета заработной платы, систему учета продукции на складе, систему учета книг в библиотеке и т.д. Все вышеперечисленные системы имеют следующие особенности:

* для обеспечения их работы нужны сравнительно низкие вычислительные мощности;
* данные, которые они используют, имеют сложную структуру;
* необходимы средства сохранения данных между последовательными запусками системы.

Другими словами, информационная система требует создания в памяти ЭВМ динамически обновляемой модели внешнего мира с использованием единого хранилища - базы данных. Для дальнейшего обсуждения нам необходимо ввести понятие предметной области:

Предметная область - часть реального мира, подлежащая изучению с целью организации управления и, в конечном счете, автоматизации. Предметная область представляется множеством фрагментов, например, предприятие - цехами, дирекцией, бухгалтерией и т.д. Каждый фрагмент предметной области харакетризуется множеством объектов и процессов, использующих объекты, а также множеством пользователей, характеризуемых различными взглядами на предметную область.

Словосочетание "динамически обновляемая" означает, что соответствиебазы данных текущему состоянию предметной области обеспечивается не периодически, а в режиме реального времени. При этом одни и те же данные могут быть по-разному представлены в соответствии с потребностями различных групп пользователей.

Отличительной чертой баз данных следует считать то, что данные хранятся совместно с их описанием, а в прикладных программах описание данных не содержится. Независимые от программ пользователя данные обычно называются метаданными. В ряде современных систем метаданные, содержащие также информацию о пользователях, форматы отображения, статистику обращения к данным и др. сведения, хранятся в словаре базы данных.

Таким образом, система управления базой данных (СУБД) – важнейшийкомпонент информационной системы. Для создания и управления информационной системой СУБД необходима в той же степени, как для разработки программы на алгоритмическом языке необходим транслятор. Основные функции СУБД:

* управление данными во внешней памяти (на дисках);
* управление данными в оперативной памяти;
* журнализация изменениий и восстановление базы данных после сбоев;
* поддержание языков БД (язык определения данных, язык манипулирования данными).

Обычно современная СУБД содержит следующие компоненты (см. рис.):

* ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию,
* процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода,
* подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД
* а также сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

# 1 АНАЛИЗ АНАЛОГОВ ПРОГРАММНОГО СРЕДСТВА

### Приложение «MySQL»

MySQL (МФА: [[maɪ ˌɛskjuːˈɛl]](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82)) — [свободная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E) [реляционная система управления базами данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94). Разработку и поддержку MySQL осуществляет корпорация [Oracle](https://ru.wikipedia.org/wiki/Oracle" \o "Oracle), получившая права на торговую марку вместе с поглощённой [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems" \o "Sun Microsystems), которая ранее приобрела шведскую компанию [MySQL AB](https://ru.wikipedia.org/wiki/MySQL_AB" \o "MySQL AB). Продукт распространяется как под [GNU General Public License](https://ru.wikipedia.org/wiki/GNU_General_Public_License), так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм [репликации](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)).

MySQL является решением для малых и средних приложений. Входит в состав серверов [WAMP](https://ru.wikipedia.org/wiki/WAMP), [AppServ](https://ru.wikipedia.org/w/index.php?title=AppServ&action=edit&redlink=1" \o "AppServ (страница отсутствует)), [LAMP](https://ru.wikipedia.org/wiki/LAMP) и в портативные сборки серверов [Денвер](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BD%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)), [XAMPP](https://ru.wikipedia.org/wiki/XAMPP), [VertrigoServ](https://ru.wikipedia.org/w/index.php?title=VertrigoServ&action=edit&redlink=1" \o "VertrigoServ (страница отсутствует)). Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа [MyISAM](https://ru.wikipedia.org/wiki/MyISAM" \o "MyISAM), поддерживающие полнотекстовый поиск, так и таблицы [InnoDB](https://ru.wikipedia.org/wiki/InnoDB" \o "InnoDB), поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

26 февраля 2008 года [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems" \o "Sun Microsystems) приобрела [MySQL AB](https://ru.wikipedia.org/wiki/MySQL_AB" \o "MySQL AB) за 1 млрд долларов, 27 января 2010 года [Oracle](https://ru.wikipedia.org/wiki/Oracle" \o "Oracle) приобрела [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems" \o "Sun Microsystems) за 7,4 млрд долларов и включила MySQL в свою линейку [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94).

Сообществом разработчиков MySQL созданы различные ответвления кода, такие как [Drizzle](https://ru.wikipedia.org/w/index.php?title=Drizzle&action=edit&redlink=1" \o "Drizzle (страница отсутствует)) ([англ.](https://en.wikipedia.org/wiki/Drizzle_(database_server))), [OurDelta](https://ru.wikipedia.org/w/index.php?title=OurDelta&action=edit&redlink=1" \o "OurDelta (страница отсутствует)), [Percona Server](https://ru.wikipedia.org/w/index.php?title=Percona_Server&action=edit&redlink=1" \o "Percona Server (страница отсутствует)) и [MariaDB](https://ru.wikipedia.org/wiki/MariaDB" \o "MariaDB). Все эти ответвления уже существовали на момент [поглощения](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D1%8F_%D0%B8_%D0%BF%D0%BE%D0%B3%D0%BB%D0%BE%D1%89%D0%B5%D0%BD%D0%B8%D1%8F) компании Sun корпорацией Oracle.

### https://www.mysql.com/common/images/products/mysql_wb_performance_dashboard_osx.png

### Рисунок 1.1 – MySQL

### 1.2 Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства для операций над базами данных.

# 2 ЦЕЛИ И ЗАДАЧИ

### 2.1 Цель курсовой работы

Целью данного курсового проекта является разработка программного средства для работы с базами данных.

### 2.2 Задачи для курсовой работы

Проанализировав аналоги, можно выделить следующие задачи:

* разработать интерфейс для программного средства;
* создание базы данных;
* открытие и сохранение в файл;
* добавление поля(столбца) с различными типами данных;
* удаление поля;
* перемещение поля;
* добавление записи;
* копирование записи;
* удалние записи;
* перемещение записи;
* поиск по базе данных;
* сортировка по заданному условию и столбцу;
* поиск по фильтру.

### 2.3 Необходимые ресурсы для курсовой работы

Для разработки программного средства использован язык программирования C++. Графический редактор – Paint3D. Среда разработки – CLion и Visual Studio. Это одни из самых мощных и популярных интегрированных сред разработки.

# 3 ПРОЕКТИРОВАНИЕ. РАЗРАБОТКА КУРСОВОЙ РАБОТЫ

### 3.1 Структура программного средства

Базовой единицей информации в базе данных является ячейка. В ячейке могут быть представлены данные пяти типов: дата, день недели, время, число, текст. В проекте это классы соответсвенно: DBDate, DBDay, DBTime, DBNumber, DBText. Каждый из них является наследником абстрактного класса DBType:

class DBType {

public:

virtual bool isCorrect() = 0;

virtual int getBuffSize() = 0;

virtual DBString toString() = 0;

virtual bool equals(DBType\* comp) = 0;

virtual int getType() = 0;

virtual DBType\* copy() = 0;

virtual bool biggerThan(DBType\* comp) = 0;

virtual bool lesserThan(DBType\* comp) = 0;

enum types {

tList, tRecord, tDate, tDay, tNumber, tText, tTime

};

static DBString typeToString(int type) {

switch (type) {

case tList:

return "List";

case tRecord:

return "Record";

case tDate:

return "Date";

case tDay:

return "Day";

case tNumber:

return "Number";

case tText:

return "Text";

case tTime:

return "Time";

default:

return "Unknown";

}

}

DBString text{};

int number{};

int day{};

int year{};

int month{};

int date{};

int hours{};

int minutes{};

int seconds{};

int milliseconds{};

};

Каждый из этих классов в свою очередь по-своему реализует функции isCorrect, getBuffSize, toString, getType, copy, equals, biggetThan, lesserThan.

Далее будут представлены реализации этих функций на примере DBDate.

Проверка корректности ячейки:

bool isCorrect() override {

return year != -1 && month != -1 && date != -1;

}

Применяется в множестве точек программы как внутри самого класса, так и вне него.

Получение размера буффера:

int getBuffSize() override {

return toString().length();

}

Применяется для определения размера массива символов, необходимого для записи даты.

Преобразование в встроенный тип DBString:

DBString toString() {

if (!isCorrect()) return {"%ERROR%"};

DBString ans;

ans = ans + (char) (date / 10 + '0') + (char) (date % 10 + '0');

ans = ans + " " + monthName(month) + ", ";

ans = ans + (char) (year / 1000 + '0') +

(char) ((year / 100) % 10 + '0') +

(char) ((year / 10) % 10 + '0') +

(char) (year % 10 + '0');

return ans;

}

Применяется для преобразования в DBString, чтобы в последствии составить в ещё больший DBString, например, для представления базы данных в тексте или преобразования в System::String^ позже при выводе в интерфейс.

Получение типа:

int getType() override {

return type;

}

Применяется для проверки на соответствие типов.

Копирование экзэмпляра класса:

DBType \*copy() override {

return new DBDate(date, month, year);

}

Применяется для резервного копирования как всей базы данных, так и для отдельной записи или ячейки.

Операции сравнения:

bool equals(DBType \*comp) override {

return isCorrect() && comp->isCorrect() &&

year == ((DBDate\*)comp)->year &&

month == ((DBDate\*)comp)->month &&

date == ((DBDate\*)comp)->date;

}

Больше чем:

bool biggerThan(DBType\* comp) override {

if (isCorrect() && ((DBDate\*)comp)->isCorrect()) {

int cYear = ((DBDate\*)comp)->year;

int cMonth = ((DBDate\*)comp)->month;

int cDate = ((DBDate\*)comp)->date;

if (year > cYear) return true;

if(year < cYear) return false;

if (month > cMonth) return true;

if (month < cMonth) return false;

if (date > cDate) return true;

if (date < cDate) return false;

}

return false;

}

Меньше чем:

bool lesserThan(DBType\* comp) override {

if (isCorrect() && ((DBDate\*)comp)->isCorrect()) {

int cYear = ((DBDate\*)comp)->year;

int cMonth = ((DBDate\*)comp)->month;

int cDate = ((DBDate\*)comp)->date;

if (year > cYear) return false;

if (year < cYear) return true;

if (month > cMonth) return false;

if (month < cMonth) return true;

if (date > cDate) return false;

if (date < cDate) return true;

}

return false;

}

Применяются в сортировке, поиске и фильтрации.

Также очень важным является преобразование строки типа DBString в DBDate или иной вид, часть этой функции для DBDate:

switch (intCount) {

case 0:

setYear(-1);

break;

case 1:

setYear(-1);

break;

case 2:

setYear(ints[1]);

for(unsigned int i = 1; i <= 12; i++) {

char \* name = monthName(i);

char \* lower = new char[strlen(name) + 1];

lower[0] = name[0] + 'a' - 'A';

for(unsigned int i = 1; i < strlen(name); i++) {

lower[i] = name[i];

}

lower[strlen(name)] = 0;

if(inDate.indexOf(monthName(i)) > -1 || inDate.indexOf(lower) > -1)

{

setMonth(i);

break;

}

}

setDate(ints[0]);

break;

case 3:

setYear(ints[2]);

setMonth(ints[1]);

setDate(ints[0]);

break;

}

После реализации всех необходимых типов данных необходимо создать представление записи как линейного списка ячеек, за это отвечает класс DBRecord.

В целом это односвязный список с несколькими дополнительными функциям такими как: swap, moveBefore, getType, copy.

Swap – поменять местами две ячейки:

bool swap(int firstIndex, int secondIndex)

{

if (firstIndex == secondIndex) return true;

if (firstIndex > secondIndex)

{

int temp = firstIndex;

firstIndex = secondIndex;

secondIndex = temp;

}

Link first = getLink(firstIndex);

Link second = getLink(secondIndex);

Link firstPrev = getLink(firstIndex - 1);

Link secondPrev = getLink(secondIndex - 1);

if (first == nullptr || second == nullptr) return false;

if (firstPrev == nullptr)

{

begin = second;

}

else {

firstPrev->next = second;

}

if (first->next == second)

{

first->next = second->next;

second->next = first;

}

else {

Link temp = first->next;

first->next = second->next;

second->next = temp;

}

if (secondPrev == first)

{

second->next = first;

}

else {

secondPrev->next = first;

}

return true;

}

MoveBefore – перемещение записи на нужную позицию:

bool moveBefore(int movableIndex, int beforeIndex)

{

if (beforeIndex - 1 == movableIndex || beforeIndex == movableIndex) return

true;

if (beforeIndex > size) return false;

Link movable = getLink(movableIndex);

Link prevMovable = getLink(movableIndex - 1);

Link beforePrev = getLink(beforeIndex - 1);

if (movable == nullptr) return false;

if (prevMovable == nullptr)

{

begin = movable->next;

}

else {

prevMovable->next = movable->next;

}

if(beforeIndex == 0) {

if (begin->next == movable) {

begin->next = movable->next;

}

movable->next = begin;

begin = movable;

}

else {

if (beforePrev == nullptr) return false;

movable->next = beforePrev->next;

beforePrev->next = movable;

}

return true;

}

GetType и Copy – реализует тот же функционал как и ячейки.

Так же DBRecord содержит в себе несколько дополнительных функций, такие как: преобразование в строку, преобразование в строку с заданным разделителем, преобразование в строку для файла.

Следующий необходимый класс это DBFilter. Он используется для создания фильтра из заданной записи условий, записи типов полей и указание на то, в каком режиме должна проходить фильтрация. Основная его функция – проверка на соответствие получившегося фильтра и переданной записи. Функция matches:

bool matches(DBRecord<DBType\*>\* rec) {

if (rec->getSize() != recSize) return false;

for (int j = 0; j < recSize; j++) {

if (filter[3][j]) continue;

DBType\* c1 = comparable->get(j);

DBType\* c2 = rec->get(j);

bool result = isStrict;

if (isStrict) {

if (filter[0][j]) result = result && c2->lesserThan(c1);

if (filter[1][j]) result = result && c2->biggerThan(c1);

if (filter[2][j]) result = result && c2->equals(c1);

}

else {

if (filter[0][j]) result = result || c2->lesserThan(c1);

if (filter[1][j]) result = result || c2->biggerThan(c1);

if (filter[2][j]) result = result || c2->equals(c1);

}

if (!result) return false;

}

return true;

}

Режимы работы фильтра: строгий и нестрогий. В строгом режиме должны совпать все заданные условия. В нестрогом – хотя бы одно.

Основной класс – DataBase. В нём совмещены все предыдущие классы.

DataBase – это двусвязный список записей DBRecord. В нём реализованы все функции что представлены для пользователя:

* добавление/удаление/перемещение поля;
* добавление/удаление/перемещение записи;
* фильтрация;
* сортировка;
* поиск.

Фильтрация:

bool filter(bool mode, DBRecord<DBString\*>\* rules) {

DBFilter filter(mode, rules, &types);

DBEnum\* rec = this->getEnumerator();

int i = 0;

while (rec->hasNext()) {

if (!filter.matches(rec->next())) {

removeRecord(i);

i--;

}

i++;

}

return true;

}

Поиск:

CellLocation findCell(DBString findable, int startCol, int startRow) {

Link current = getLink(startRow);

if (current == nullptr) return CellLocation(-1, -1);

while (current != nullptr) {

if ((startCol = current->data->find(newElementFromString

(DBType::types(getType(startCol)), findable), startCol)) > -1) return CellLocation(startRow, startCol);

startCol = 0;

current = current->next;

startRow++;

}

return CellLocation(-1, -1);

}

### 3.2 Разработка графического пользовательского интерфейса

При разработке графического пользовательского интерфейса был

использован язык C++ и библиотеки .NET. Интерфейс создавался с помощью конструктора форм VisualStudio. Интерфейс содержит своё меню, панель заголовка, панель команд, таблица, панель для осуществления операций, панель подсказок, панель сообщений, кнопка справа для добавления поля и кнопка снизу для добавления записи. Так же в приложении есть описание того, как пользоваться программой и горячие клавиши дублирующие весь функционал, их список пользователь может найти в главном меню.

Интерфейс снабжён множеством подсказок и функций, помогающих пользователю пользоваться программным средством.

Основной задачей в создании пользовательского интерфейса было связать внутренее представление базы данных и элемент DataGridView. Для этого был создан адаптер (класс DBAdapter). В нём хранится ссылка на базу данных и на DataGridView, с которым эта база данных должна быть связана. При написании интерфейса операции совершаются не напрямую с базой данных, а через адаптер, поэтому в нём должны быть реализованы все основные функции базы данных. Должно производиться резервное копирование перед каждой операцией и, в случае ошибки, восстановление. Так же адаптер проверяет, была ли сохранена база данных или нет.

Функция перезагрузки:

bool reload()

{

GW->Rows->Clear();

GW->Columns->Clear();

for (int i = 0; i < DB->getFieldCount(); i++) {

GW->Columns->Add(DB->getTitle(i).toSysString(), DB->

getTitle(i).toSysString());

GW->Columns[GW->ColumnCount - 1]->SortMode =

DataGridViewColumnSortMode::NotSortable;

}

DataBase::DBEnum\* records = DB->getEnumerator();

int j = 0;

while (records->hasNext())

{

GW->Rows->Add();

if (j == 2)

j = 2;

auto record = records->next();

for (int i = 0; i < DB->getFieldCount(); i++)

{

GW[i, j]->Value = record->get(i)->toString().toSysString();

}

j++;

}

Чтение базы данных из файла с расширений «dbvm»:

bool load(System::String^ fileName) {

try

{

System::IO::StreamReader^ stream = gcnew

System::IO::StreamReader(fileName);

System::String^ line;

while ((line = stream->ReadLine())->Length > 0) {

addField(DBType::types(System::Convert::ToInt32(stream->

ReadLine())), new DBString(line));

}

int size = System::Convert::ToInt32(stream->ReadLine());

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

auto rec = new DBRecord<DBType\*>();

int typeIndex = 0;

while ((line = stream->ReadLine())->Length > 0) {

rec->add(DB->newElementFromString(DBType::types(DB->

getType(typeIndex)), DBString(line)));

typeIndex++;

}

addRecord(rec);

}

stream->Close();

}

catch (System::Exception^ e)

{

return false;

}

wasSavedInForm = true;

return true;

}

# ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Заключительным этапом разработки приложения для работы над базами данных является тестирование.

1. Безопасный выход из программы: в случае, если база данных была изменена и не сохранена, то при попытке закрытия программы пользователю предлагается сохранить в файл.

Результат:

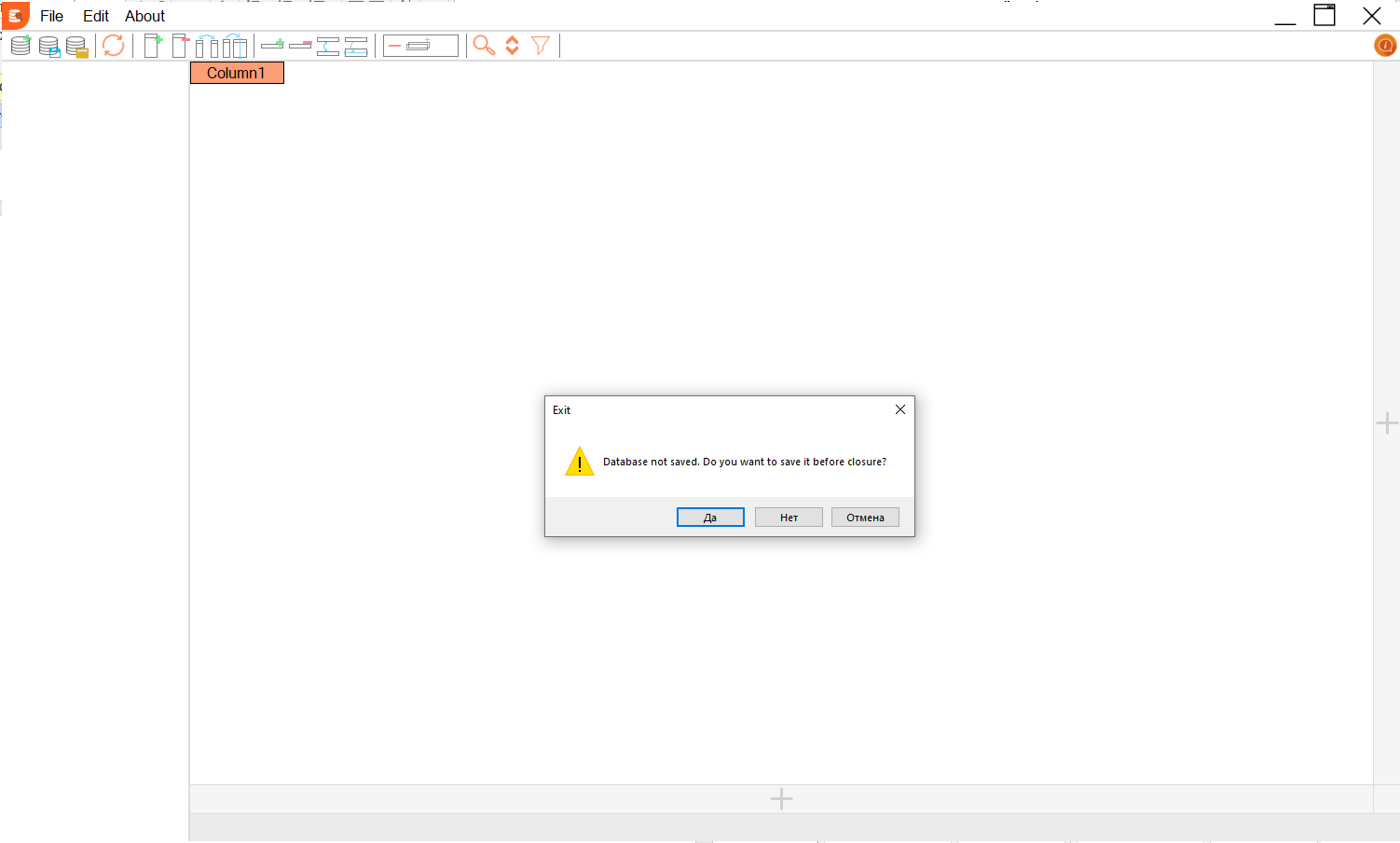


Рисунок 4.1 – закрытие без сохранения файла

2. Проверка правильности введённых значений при добавлении записи.

Результат:

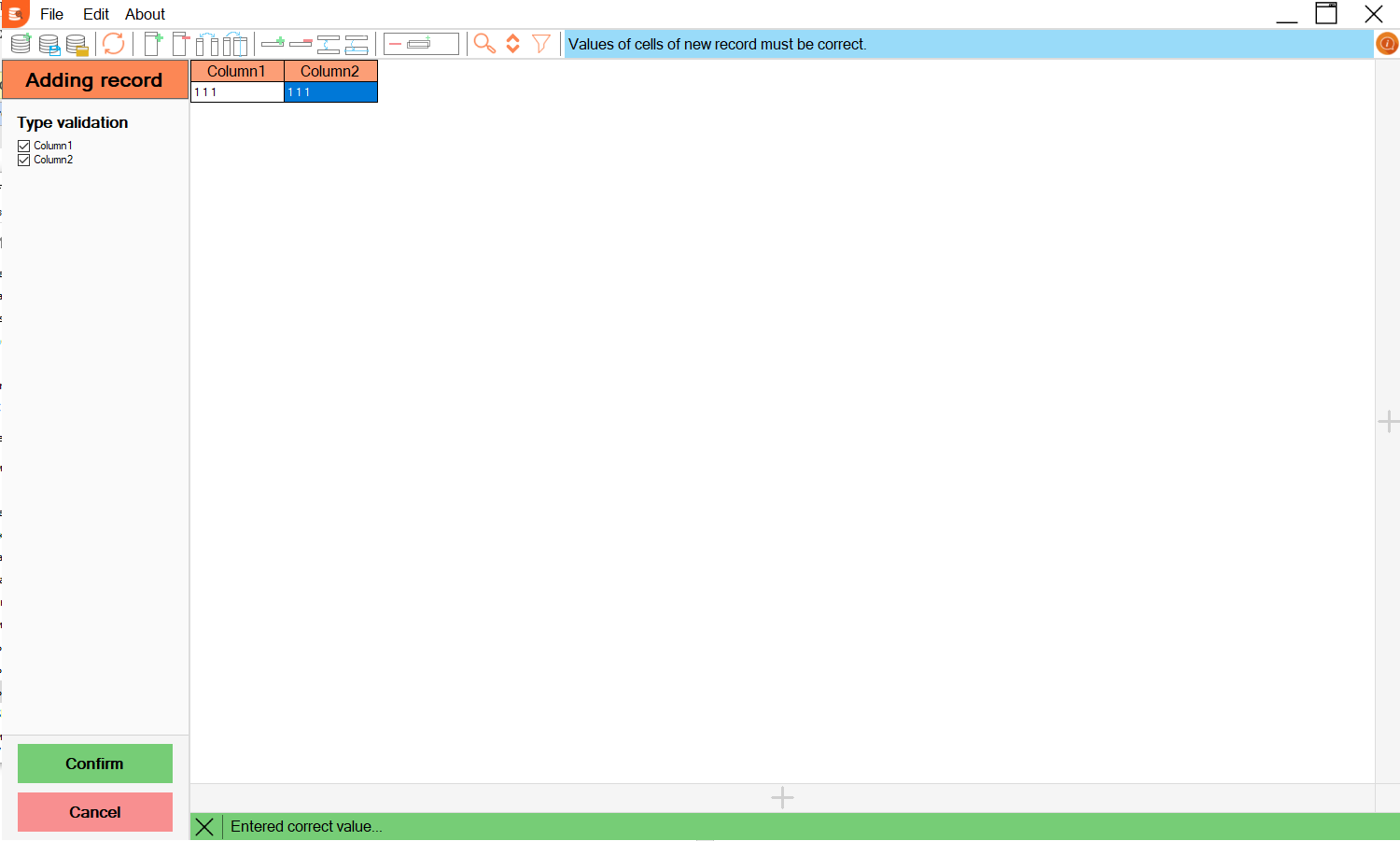
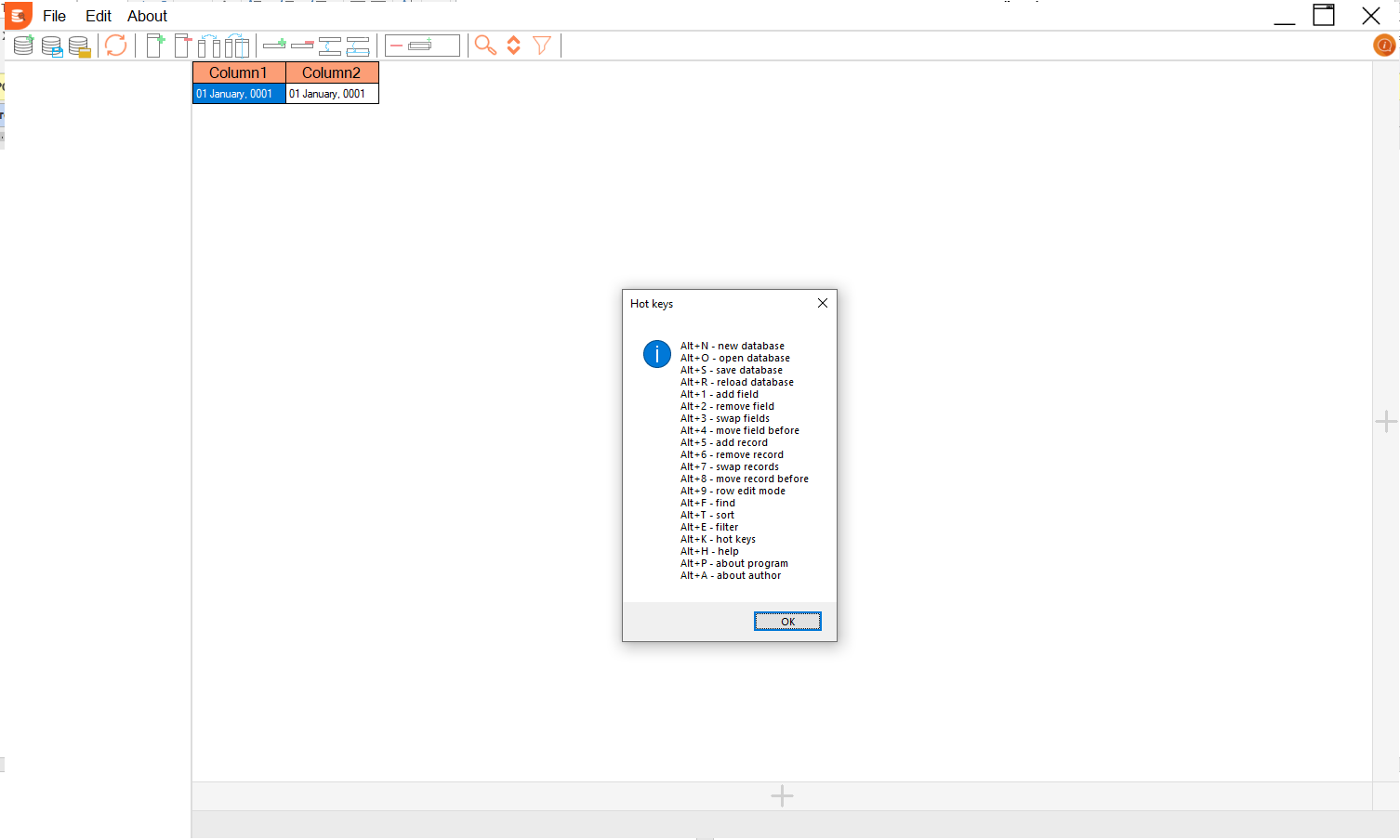


Рисунок 4.2 – введённые значения дат являются правльными

3. Горячая клавиша открытия списка горячих клавиш.

Результат:



# Рисунок 4.3 – результат нажатия сочетания клавиш Alt+K

4. Перемещение окна с несистемным интерфейсом с помощью перетаскивания.

Результат:

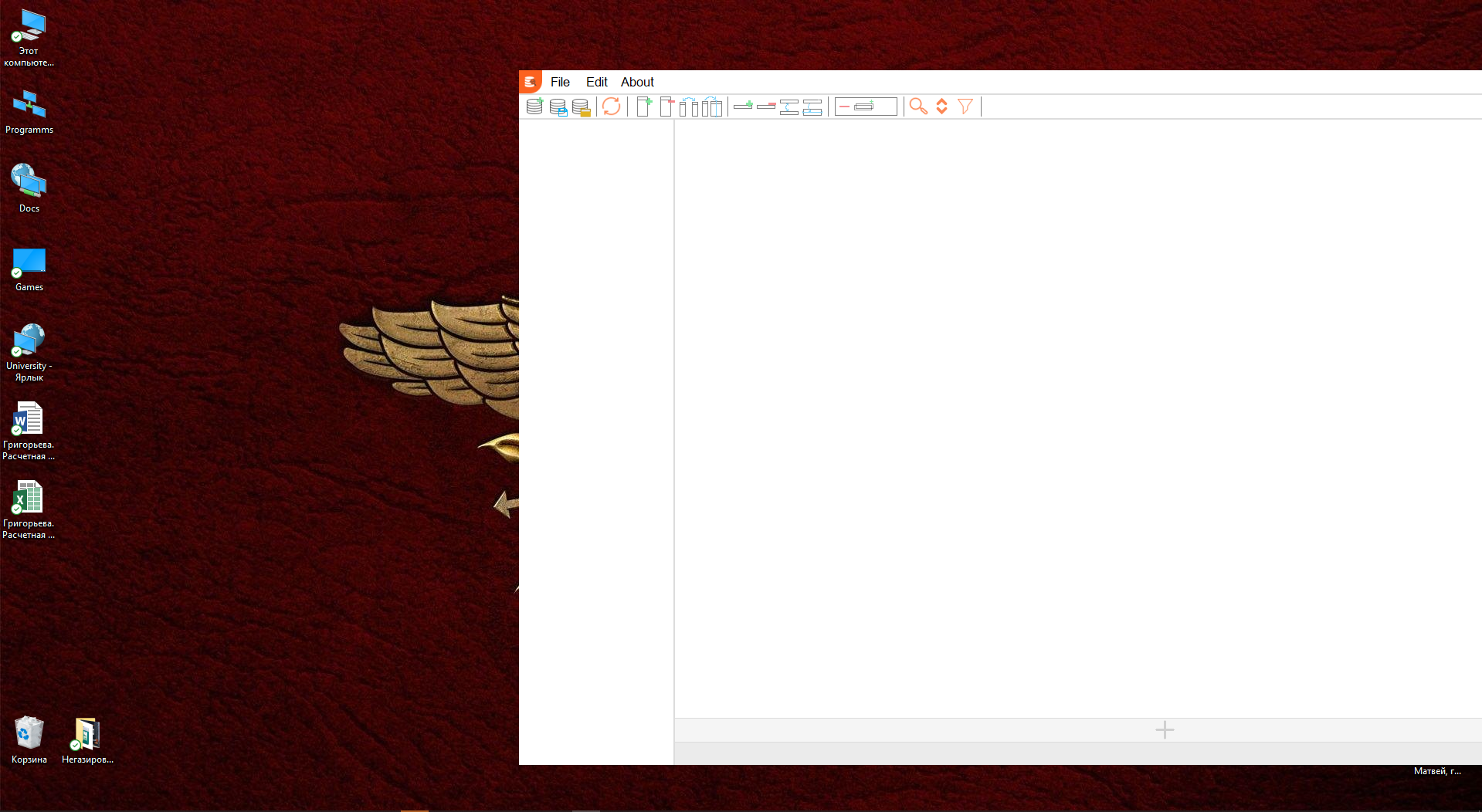
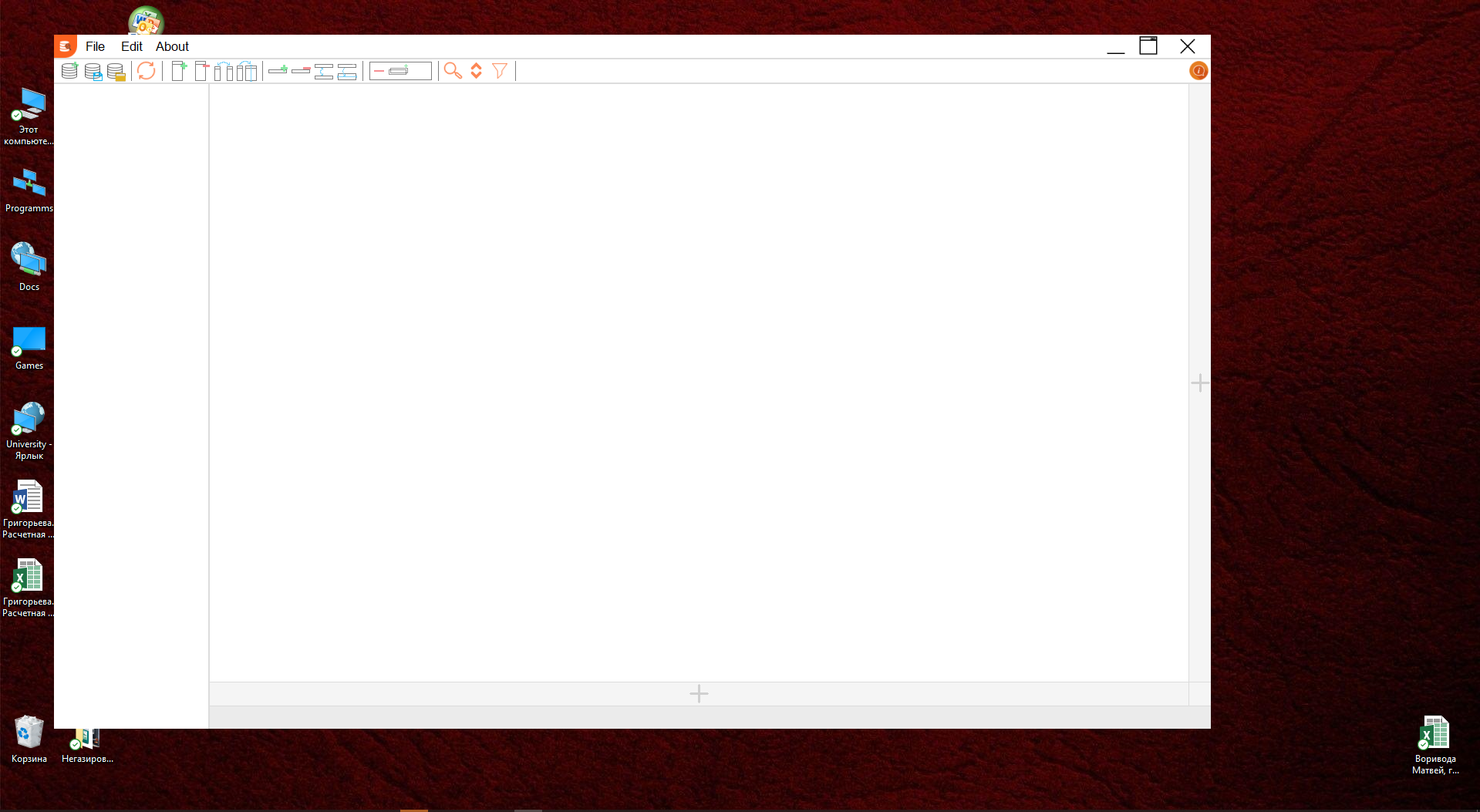


Рисунок 4.4 – положение окна до перетаскивания.



# Рисунок 4.5 – положение окна после перетаскивания.

# Сортировка. По клику должен выбраться соответствующий столбец и по нему должна отсортироваться вся таблца. При отмене операции таблица вернётся в прежнее состояние.

До:

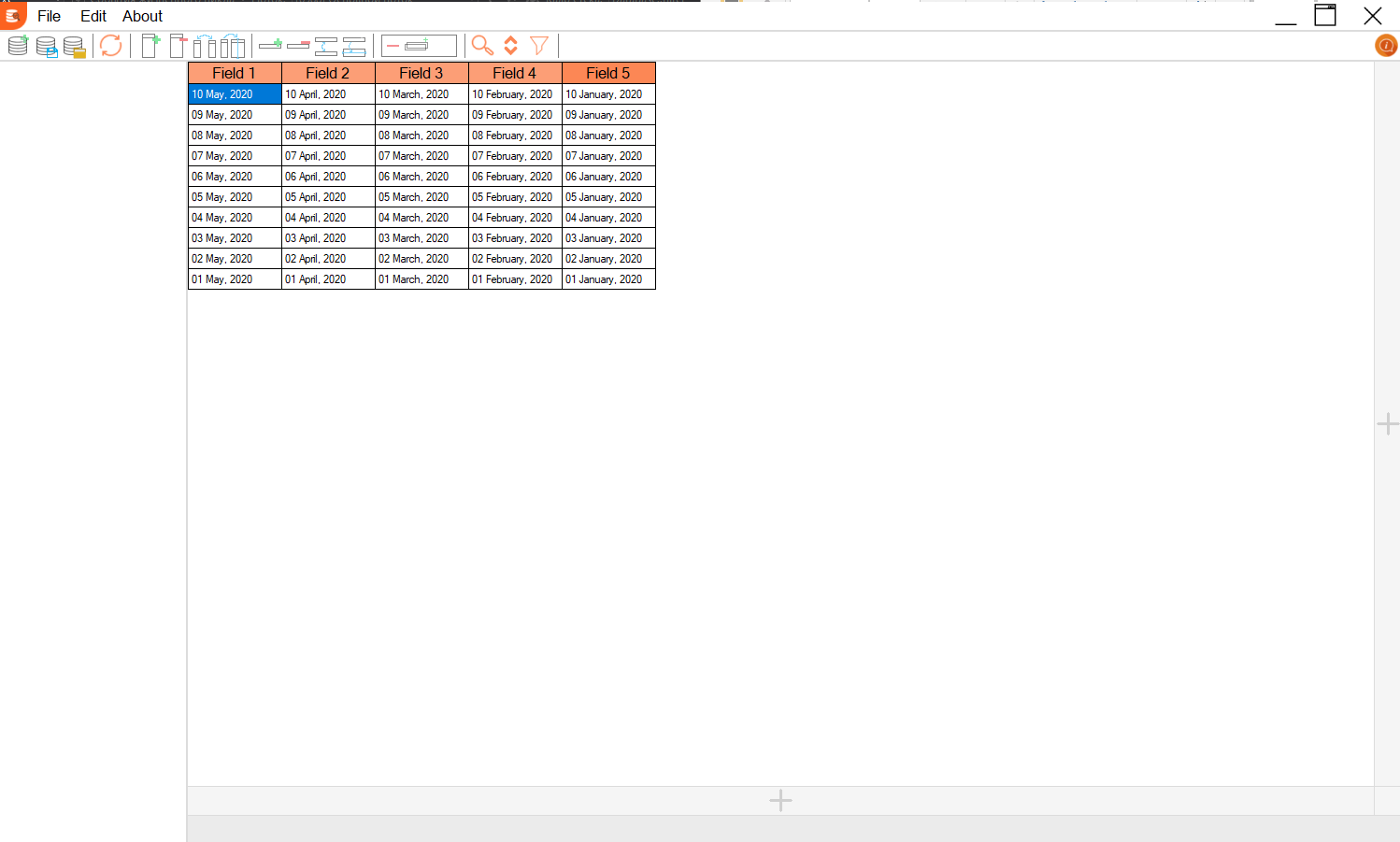


Рисунок 4.6 – таблица до сортировки

Результат:

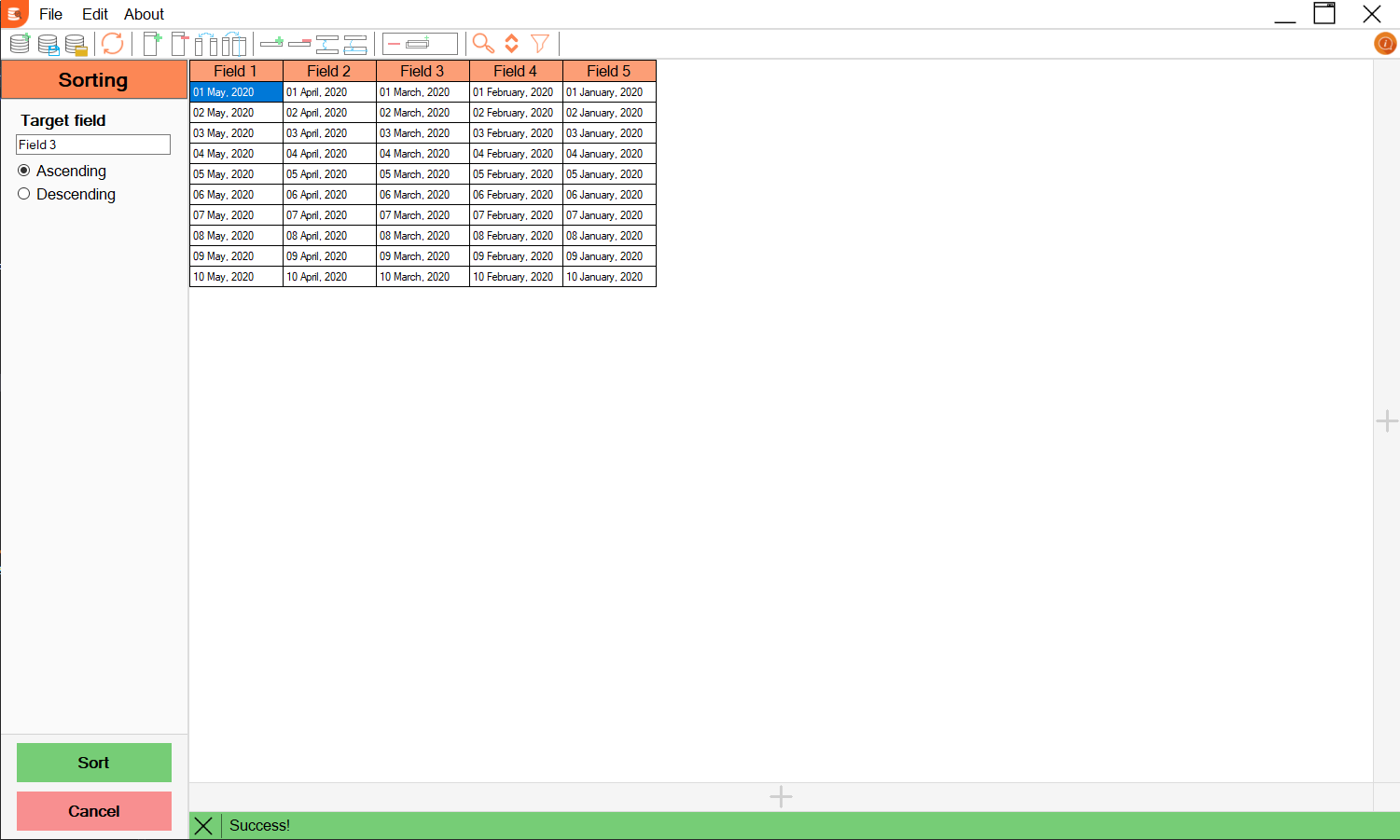
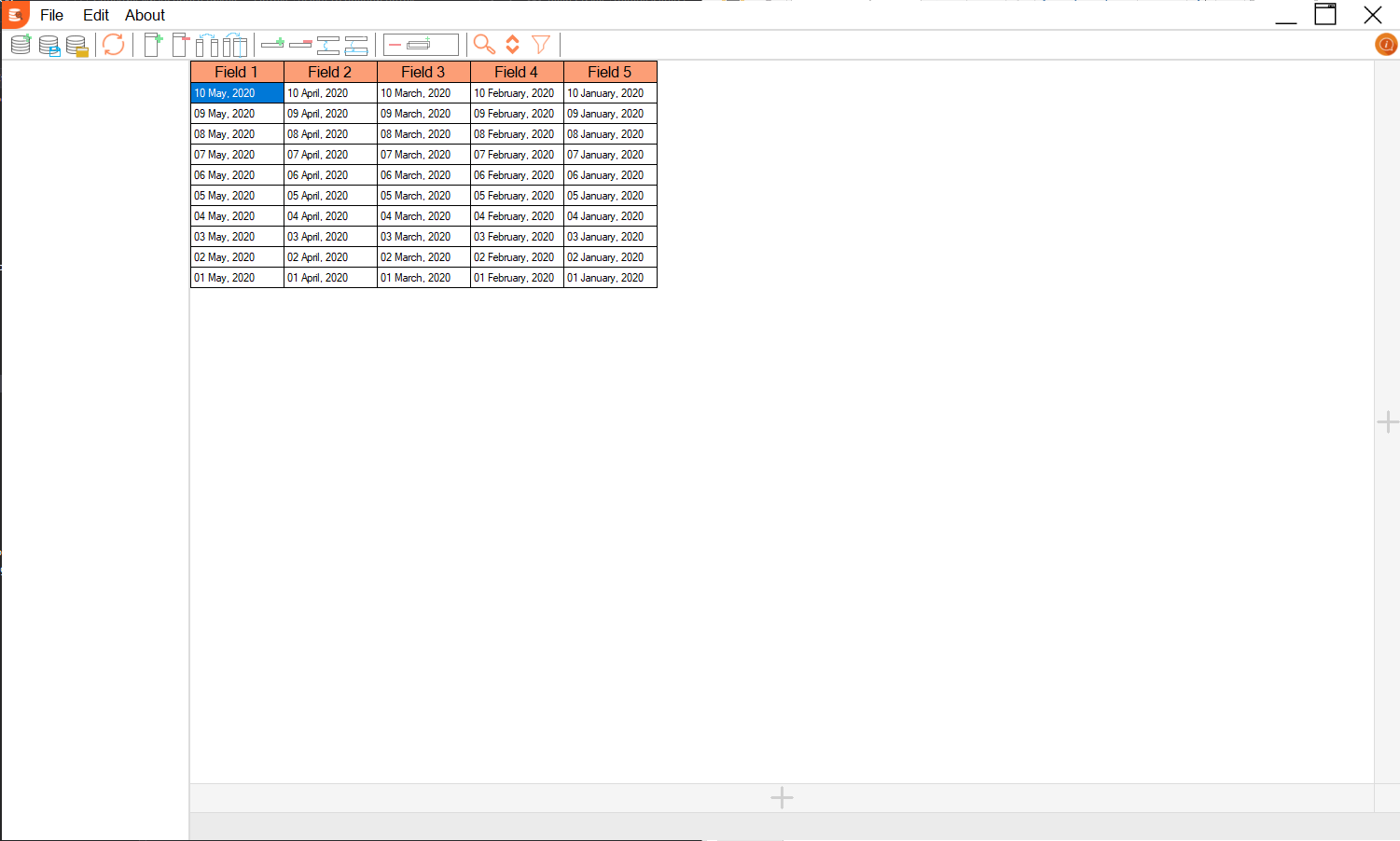


Рисунок 4.7 – таблица после сортировки

# Отмена:



# Рисунок 4.8 – таблица при отмене операции

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При старте программы открывается окно с несколькими панелями: в центре таблицы представляющая базу данных, слева от неё панель которая будет необходима для операций над базой данных, сверху панель команд. В самой верхней части окна находится главное меню, в котором дублируются все команды. Сверху слева для пользователя будет доступ к помощи и описанию каждой функции.

Добавление поля: необходимо ввести уникальное имя поля и выбрать тип данных в этом поле.

Удаление поля: ввести имя поля или кликнуть по нему, чтобы выбрать, а затем удалить.

Перемещение полей: для того чтобы переместить поля, можно потянуть за заголовок столбца и перетащить на нужную позицию или же воспользоваться встроенными функциями смены полей местами.

Для записей используются все те же функции и режим прямой работы с записями, в котором можно быстро удалить или вставить копию строки.

Поиск: поиск производится по всей базе данных по заданному условию и начиная с выделенной клетки.

Сортировка: представлены две сортировки (по возрастанию и по убыванию). Для того чтобы отсортировать базу данных необходимо выбрать целевой столбец и выбрать тип сортировки. Таблица представится в отсортированном виде и при закрытии режима сортировки таблица вернётся в прежнее состояние.

Фильтрация: необходимо выбрать режим, после чего в появившейся дополнительной строке, ввести условия фильтрации. Как и с сортировкой, при закрытии режима фильтрации, таблица вернётся в прежнее состояние.

# ЗАКЛЮЧЕНИЕ

Существенным фактором информационного века является структурированное представление данных. Поэтому было сформулирована концепция базы данных предназначенной для хранения большого объёма информации.

В результате выполнения данной курсовой работы разработано программное средство для работы с базами данных, которое выполняет следующие функции:

* создание пустой базы данных;
* открытие и сохранение базы данных;
* резервное копирование;
* преобразование в различные типы данных;
* полноценная работа с полями базы данных;
* редактирование записей;
* поиск, сортировка и фильтрация.

База данных необходима для работы с большим объёмом информации. Программное средство, разработанное в данной курсовой работе может быть полезно как для отдельно взятого человека, так и для компании.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Язык Программирования С/ Брайан Керниган, Деннис Ритчи. – Санкт-Петербург : Диалектика, 2019. – 288 с.

[2] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. – СПб.: Питер, 2018. – 464 с.: ил.

[3] Глухова, Л. А. Основы алгоритмизации и программирования: Лаб. практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обуч. В 4 ч. / Л. А. Глухова, Е. Е. Фадеева, Е. П. Фадеева. – Минск: БГУИР, 2007. – Ч. 3. – 51 с.

[4] Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003.

[5] Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003.

# ПРИЛОЖЕНИЕ А

# Исходный код программы

#ifndef DATABASE\_CPP

#define DATABASE\_CPP

#define DATABASE\_IS\_EMPTY "{ Database is empty. }"

#include "DBRecord.cpp"

#include "DBFilter.cpp"

class CellLocation {

public:

int row, col;

CellLocation(int row, int col) {

this->row = row;

this->col = col;

}

};

class DataBase {

private:

typedef DBRecord<DBType\*>\* R;

typedef struct SLink {

R data;

SLink\* next, \* prev;

} \*Link;

Link begin, end;

int size;

Link getLink(int index) {

if (index < 0) return nullptr;

Link current = begin;

while (current != nullptr && index > 0) {

current = current->next;

index--;

}

return current;

}

DBRecord<DBString\*> titles;

DBRecord<DBNumber\*> types;

public:

DataBase() : types(), titles() {

begin = end = nullptr;

size = 0;

}

bool isCorrectRecord(R record) {

for (int i = 0; i < types.getSize(); i++) {

if (record->getType(i) != (int)types.get(i)->get()) return false;

}

return true;

}

static DBType\* newStandardElement(DBType::types type) {

switch (type) {

case DBType::tDate:

return new DBDate();

case DBType::tDay:

return new DBDay();

case DBType::tNumber:

return new DBNumber();

case DBType::tText:

return new DBText();

case DBType::tTime:

return new DBTime();

default:

return nullptr;

}

}

static DBType\* newElementFromString(DBType::types type, DBString example) {

switch (type) {

case DBType::tDate:

return new DBDate(example);

case DBType::tDay:

return new DBDay(example);

case DBType::tNumber:

return new DBNumber(example);

case DBType::tText:

return new DBText(example);

case DBType::tTime:

return new DBTime(example);

default:

return nullptr;

}

}

bool filter(bool mode, DBRecord<DBString\*>\* rules) {

DBFilter filter(mode, rules, &types);

DBEnum\* rec = this->getEnumerator();

int i = 0;

while (rec->hasNext()) {

if (!filter.matches(rec->next())) {

removeRecord(i);

i--;

}

i++;

}

return true;

}

bool addField(int type, DBString\* title) {

if (type < DBType::tDate || type > DBType::tTime) return false;

if (titles.indexOf(title) > -1) return false;

titles.add(title);

types.add(new DBNumber(type));

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

getLink(i)->data->add(newStandardElement(DBType::types(type)));

}

return true;

}

bool addRecord() {

DBRecord<DBType\*>\* record = new DBRecord<DBType\*>();

for (int i = 0; i < getFieldCount(); i++)

{

record->add(newStandardElement(DBType::types(types.get(i)->get())));

}

return addRecord(record);

}

bool addRecord(R data) {

if (!isCorrectRecord(data)) return false;

if (begin == nullptr) {

begin = end = (Link)malloc(sizeof(struct SLink));

begin->data = data->copy();

begin->next = nullptr;

begin->prev = nullptr;

}

else {

end->next = (Link)malloc(sizeof(struct SLink));

end->next->prev = end;

end = end->next;

end->data = data->copy();

end->next = nullptr;

}

size++;

return true;

}

bool removeField(int index)

{

if (!types.remove(index)) return false;

if (!titles.remove(index)) return false;

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

{

if (!getLink(i)->data->remove(index)) return false;

}

return true;

}

bool removeRecord(int index) {

Link removable = getLink(index);

if (removable == nullptr) return false;

if (removable->prev == nullptr) {

begin = removable->next;

}

else {

removable->prev->next = removable->next;

}

if (removable->next == nullptr) {

end = removable->prev;

}

else {

removable->next->prev = removable->prev;

}

size--;

return true;

}

bool swapFields(int firstIndex, int secondIndex)

{

if (!types.swap(firstIndex, secondIndex)) return false;

if (!titles.swap(firstIndex, secondIndex)) return false;

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

{

if (!getLink(i)->data->swap(firstIndex, secondIndex)) return false;

}

return true;

}

bool swapRecords(int firstIndex, int secondIndex) {

if (firstIndex == secondIndex) return true;

if (firstIndex > secondIndex) {

int temp = firstIndex;

firstIndex = secondIndex;

secondIndex = temp;

}

Link first = getLink(firstIndex);

Link second = getLink(secondIndex);

if (first == nullptr || second == nullptr) return false;

if (first->prev == nullptr) {

begin = second;

}

else {

first->prev->next = second;

}

if (second->next == nullptr) {

end = first;

}

else {

second->next->prev = first;

}

if (first->next == second) {

first->next = second->next;

second->next = first;

}

else {

first->next->prev = second;

Link temp = first->next;

first->next = second->next;

second->next = temp;

}

if (second->prev == first) {

second->prev = first->prev;

first->prev = second;

}

else {

second->prev->next = first;

Link temp = second->prev;

second->prev = first->prev;

first->prev = temp;

}

return true;

}

bool moveFieldBefore(int movableIndex, int beforeIndex)

{

if (!types.moveBefore(movableIndex, beforeIndex)) return false;

if (!titles.moveBefore(movableIndex, beforeIndex)) return false;

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

{

if (!getLink(i)->data->moveBefore(movableIndex, beforeIndex)) return false;

}

return true;

}

bool moveRecordBefore(Link movable, Link before) {

if (movable == nullptr) return false;

if (movable->prev == nullptr)

{

begin = movable->next;

}

else {

movable->prev->next = movable->next;

}

if (movable->next == nullptr)

{

end = movable->prev;

}

else {

movable->next->prev = movable->prev;

}

if (before == nullptr)

{

movable->prev = end;

end->next = movable;

end = movable;

}

else {

if (before == nullptr) return false;

movable->prev = before->prev;

before->prev = movable;

movable->next = before;

if (movable->prev == nullptr)

{

begin = movable;

}

else {

movable->prev->next = movable;

}

}

return true;

}

bool moveRecordBefore(int movableIndex, int beforeIndex)

{

if (beforeIndex - 1 == movableIndex || beforeIndex == movableIndex) return true;

Link movable = getLink(movableIndex);

Link before = getLink(beforeIndex);

if (movable == nullptr) return false;

if (movable->prev == nullptr)

{

begin = movable->next;

}

else {

movable->prev->next = movable->next;

}

if (movable->next == nullptr)

{

end = movable->prev;

}

else {

movable->next->prev = movable->prev;

}

if (beforeIndex == size)

{

movable->prev = end;

end->next = movable;

end = movable;

}

else {

if (before == nullptr) return false;

movable->prev = before->prev;

before->prev = movable;

movable->next = before;

if (movable->prev == nullptr)

{

begin = movable;

}

else {

movable->prev->next = movable;

}

}

return true;

}

bool copyRecord(int index) {

return addRecord(get(index))

&& moveRecordBefore(size - 1, index);

}

bool ascendingSort(int field) {

if (field < 0 || field > size - 1) return false;

if (size < 2) return true;

Link minStart = begin;

bool isChanged;

do {

Link current = minStart;

DBType\* min = current->data->get(field);

Link minLink = current;

isChanged = false;

current = current->next;

while (current != nullptr)

{

DBType\* data = current->data->get(field);

if (data->lesserThan(min)) {

min = current->data->get(field);

minLink = current;

isChanged = true;

}

current = current->next;

}

if (isChanged) {

moveRecordBefore(minLink, minStart);

}

} while (minStart != nullptr && isChanged);

return true;

}

bool descendingSort(int field) {

if (field < 0 || field > size - 1) return false;

if (size < 2) return true;

Link minStart = begin;

bool isChanged;

do {

Link current = minStart;

DBType\* min = current->data->get(field);

Link minLink = current;

isChanged = false;

current = current->next;

while (current != nullptr)

{

DBType\* data = current->data->get(field);

if (data->biggerThan(min)) {

min = current->data->get(field);

minLink = current;

isChanged = true;

}

current = current->next;

}

if (isChanged) {

moveRecordBefore(minLink, minStart);

}

} while (minStart != nullptr && isChanged);

return true;

}

R get(int index) {

Link current = getLink(index);

if (current == nullptr) return nullptr;

return current->data->copy();

}

DBString getTitle(int index) {

return \*titles.get(index);

}

bool isCorrectInitializer(DBString init, int index) {

return newElementFromString(DBType::types(getType(index)), init)->isCorrect();

}

int indexOfTitle(DBString\* title)

{

return titles.indexOf(title);

}

int getFieldCount()

{

return types.getSize();

}

bool setTitle(int titleIndex, DBString\* title)

{

return titles.set(titleIndex, title);

}

bool set(int recIndex, int index, DBType\* data) {

if (index < 0 || index > types.getSize() - 1) return false;

if ((int)types.get(index)->get() != data->getType()) return false;

Link current = getLink(recIndex);

if (current == nullptr) return false;

current->data->set(index, data);

return true;

}

int getSize() {

return size;

}

int getType(int index) {

return (int)types.get(index)->get();

}

CellLocation findCell(DBString findable, int startCol, int startRow) {

Link current = getLink(startRow);

if (current == nullptr) return CellLocation(-1, -1);

while (current != nullptr) {

if ((startCol = current->data->

find(newElementFromString(DBType::types(getType(startCol)), findable), startCol)) > -1) return CellLocation(startRow, startCol);

startCol = 0;

current = current->next;

startRow++;

}

return CellLocation(-1, -1);

}

DataBase\* copy()

{

DataBase\* db = new DataBase();

for (int i = 0; i < getFieldCount(); i++)

{

db->addField((int)types.get(i)->get(), titles.get(i));

}

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

{

db->addRecord(get(i));

}

return db;

}

DBString toString() {

if (begin == nullptr && types.getSize() == 0) {

return { DATABASE\_IS\_EMPTY };

}

else {

DBString ans("");

ans = ans + "Titles" + ": ";

for (int i = 0; i < types.getSize(); i++) {

ans = ans + \*titles.get(i) + ";";

}

ans = ans + "\nTypes" + "; ";

for (int i = 0; i < types.getSize(); i++) {

ans = ans + DBType::typeToString((int)types.get(i)->get()) + "; ";

}

ans = ans + "\n";

Link current = begin;

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

{

ans = ans + DBString(i) + ": " + current->data->toString() + "\n";

current = current->next;

}

return ans;

}

}

DBString toString(const char\* indexSeparator, const char\* separator, const char\* beginner,

const char\* finalist) {

if (begin == nullptr && types.getSize() == 0) {

return { DATABASE\_IS\_EMPTY };

}

else {

DBString ans("");

ans = ans + "Titles" + indexSeparator;

for (int i = 0; i < types.getSize(); i++) {

ans = ans + \*titles.get(i) + separator;

}

ans = ans + "\nTypes" + indexSeparator;

for (int i = 0; i < types.getSize(); i++) {

ans = ans + DBType::typeToString((int)types.get(i)->get()) +

separator;

}

ans = ans + "\n";

Link current = begin;

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

{

ans = ans + DBString(i) + indexSeparator + current->data->

toString(separator, beginner, finalist) + "\n";

current = current->next;

}

return ans;

}

}

DBString toStringForFile() {

if (begin == nullptr && types.getSize() == 0) {

return { "" };

}

else {

DBString ans("");

for (int i = 0; i < titles.getSize(); i++) {

ans = ans + getTitle(i) + "\n" + (DBString(getType(i))) + "\n";

}

ans = ans + "\n" + DBString(getSize()) + "\n";

Link current = begin;

while (current != nullptr) {

ans = ans + current->data->toStringForFile() + "\n\n";

current = current->next;

}

return ans;

}

}

class DBEnum {

private:

Link current;

public:

DBEnum(Link begin)

{

current = begin;

}

bool hasNext()

{

return current != nullptr;

}

R next()

{

if (hasNext()) {

R copy = current->data->copy();

current = current->next;

return copy;

}

return nullptr;

}

};

DBEnum\* getEnumerator() {

return new DBEnum(begin);

}

};

#endif

# ПРИЛОЖЕНИЕ Б

# Блок-схема методов