Титульник

Содержание

Введение

Целью курсового проекта является проектирование и разработка информационной системы поликлиники.

Актуальность ИС заключается в необходимости структуризации и упорядоченности информации, хранящейся в больших объемах, которые сложно обрабатывать без использования информационной системы.

Данная подсистема должна хранить и обрабатывать информацию о пациентах поликлиники.

Задачами курсовой работы являются:

1. анализ предметной области;
2. разработка структуры базы данных;
3. создание базы данных средствами выбранной СУБД;

1. Проектирование базы данных

1.1. Анализ предметной области

Предметной областью данной курсовой работы являются данные о пациентах поликлиники.

База данных должна хранить данные о каждой книге, о читателях и сотрудниках, а так же о выдаче и возврате книг в библиотеку.

В базе данных должны быть данные о каждого пациенте, докторе, а так же о кабинетах, в которых они принимают. Также необходимо хранить сведения о приемах и выписанных диагнозах.

Сведения о пациенте должны включать ФИО, улицу и дом проживания, а так же телефон для связи.

О докторе необходимо хранить его ФИО и номер телефона.

Так же необходимо сведения о приемах пациентов. В эти сведения входят данные о пациенте, докторе, кабинете приема, дате и времени приема и стоимости. Так же необходимо хранить сведение о выписанном диагнозе.

В базе данных должны быть сведения о кабинетах, их название и типе.

В сведения о болезни входит название болезни.

Операторы данной программы ведут учет книг в библиотеки и операций с ними.

1.2. Постановка задачи

Таким образом, необходимо разработать структуру базы данных, структуру запросов, сформировать схему данных с указанием первичных и внешних ключей. БД должна содержать следующую информацию:

- сведения о пациентах;
- сведения о врачах;
- сведения о кабинетах;
- сведения о болезнях;
- сведения о приемах;

- сведения о типах кабинетов;
- сведения об улицах;

База данных должна быть разработана средствами среды MYSQL.

Необходимо разработать концептуальную, логическую и физическую модели базы данных.

1.3. Концептуальное проектирование базы данных

Проектирование концептуальной модели подразумевает выделение не только основных информационных объектов предметной области и атрибутов, но и определение связей между ними.

На основании выполненного анализа предметной области для концептуальной модели базы данных библиотекb можно выделить следующие информационные объекты:

- Пациенты – справочник о званиях пациентах;
- Врачи – справочник о врачах;
- Болезни – справочник о болезнях;
- Кабинеты – справочник о кабинетах;
- Типы кабинетов – справочник о типах кабинетов;
- Приемы – справочник о приемах;
- Улицы – справочник о улицах района подведомственному поликлиники;

Для объекта «*Пациенты*» можно выделить такие атрибуты, как: код читателя, ФИО, номер телефона, код улицы, номер дома. Атрибут «Код пациента» является первичным ключом.

Для объекта «*Доктора*» можно выделить такие атрибуты, как: Код доктора, ФИО, номер телефона. Атрибут «Код доктора» является первичным ключом.

Для объекта «*Болезни*» можно выделить такие атрибуты, как: код болезни, название болезни. Атрибут «Код болезни» является первичным ключом.

Для объекта «*Кабинеты*» можно выделить такие атрибуты, как: код кабинета, название кабинета, код типа кабинета. Атрибут «Код кабинета» является первичным ключом. Атрибуты «Код типа кабинета» являются внешними ключами соответственно из объектов «Типы кабинетов».

Для объекта «*Типы кабинетов*» можно выделить такие атрибуты, как: код типа, название типа. Атрибут «Код типа» является первичным ключом.

Для объекта «*Улицы*» можно выделить такие атрибуты, как: код улицы, название улицы. Атрибут «Код улицы» является первичным ключом.

Для объекта «*Приемы*» можно выделить следующие атрибуты: код записи, код клиента, код доктора, код болезни, код кабинета, дата приема, стоимость приема.. Атрибут «Код записи» является первичным ключом. Атрибуты «Код клиента», «Код доктора», «Код болезни», «Код кабинета» являются внешними ключами соответственно из объектов «Клиенты», «Доктора», «Болезни», «Кабинеты».

Между объектами были определены связи следующих типов:

- Одного пациент может приходит несколько раз на приемы. Следовательно, тип связи между объектами «Пациенты» и «Приемы» «один-ко-многим»;
- Один доктор может принимать разных пациентов в разное время. Следовательно, тип связи между объектами «Доктора» и «Приемы» «один-ко-многим»;
- В одном кабинете может быть выполнено несколько приемов. Следовательно, тип связи между объектами «Кабинеты» и «Приемы» «один-ко-многим»;
- Один и тот же диагноз может быть выписан разным пациентам. Следовательно, тип связи между объектами «Болезни» и «Приемы» «один-ко-многим»;
- В поликлинике может быть несколько кабинетов с одним типом. Следовательно, тип связи между объектами «Типы кабинетов» и «Кабинеты» «один-ко-многим»;

- На одной улице может жить несколько пациентов. Следовательно, тип связи между объектами «Улицы» и «Пациенты» «один-ко-многим»;

На основании выделенных информационных объектов и связей между ними, средствами Microsoft Visio была построена схема концептуальной модели БД, которая приведена на рисунке 1.

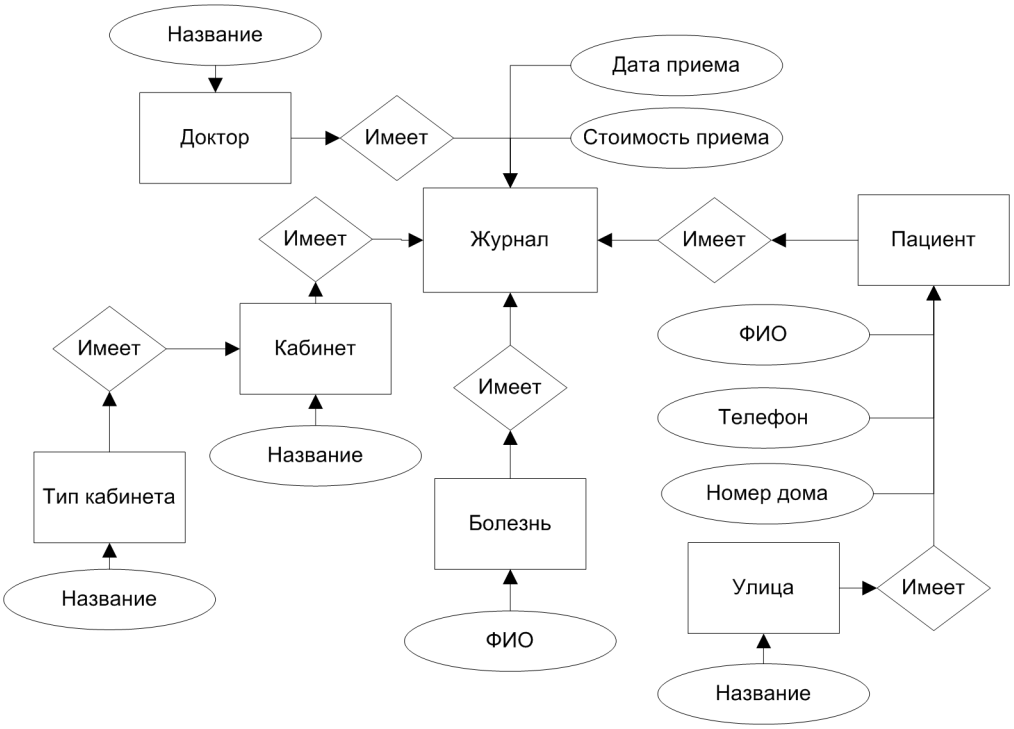

Рисунок 1 – Концептуальная модель БД

Таким образом, было произведено концептуальное проектирование БД.

1.4. Логическое проектирование базы данных

Для логической модели БД характерно определение типов данных, внешних ключей. Проектирование данной модели должно производиться независимо от того, какая система управления базами данных (СУБД) выбрана для конечной реализации БД.

Также, необходимо привести все отношения к третьей нормальной форме (3НФ).

Характеристика отношений логической модели БД представлена в таблицах 1 – 7.

Таблица 1 – Характеристика отношения «Тип кабинета»

Имя атрибута	Тип данных	Ключ
Код типа	Счетчик	PK
Название типа	Текстовый	

Таблица 2 – Характеристика отношения «Улицы»

Имя атрибута	Тип данных	Ключ
Код улицы	Счетчик	PK
Название улицы	Текстовый	

Таблица 3 – Характеристика отношения «Болезни»

Имя атрибута	Тип данных	Ключ
Код болезни	Счетчик	PK
Название болезни	Текстовый	

Таблица 4 – Характеристика отношения «Пациенты»

Имя атрибута	Тип данных	Ключ
Код пациента	Счетчик	PK
ФИО	Текстовый	
Телефон	Текстовый	
Код улицы	Числовой	
Номер дома	Числовой	FK

Таблица 5 – Характеристика отношения «Доктор»

Имя атрибута	Тип данных	Ключ
Код доктора	Числовой	PK
ФИО	Текстовый	
Телефон	Текстовый	

Таблица 6 – Характеристика отношения «Кабинеты»

Имя атрибута	Тип данных	Ключ
Код кабинета	Числовой	PK
Название кабинета	Текстовый	
Код типа кабинета	Числовой	FK

Таблица 7 – Характеристика отношения «Приемы»

Имя атрибута	Тип данных	Ключ
Код приема	Счетчик	PK
Код пациента	Текстовый	FK
Код кабинета	Числовой	FK
Код доктора	Числовой	FK
Код болезни	Числовой	FK
Дата приема	Числовой	
Стоимость приема	Числовой	

1.5. Физическое проектирование базы данных

Физическое проектирование БД выполняется с учетом выбранной СУБД. В данной работе в качестве СУБД используется MYSQL.

Схема физической модели данных представлена на рисунке 3.

Средствами MS Visio2007 была построена схема физической модели БД, которая представлена на рисунке 3.

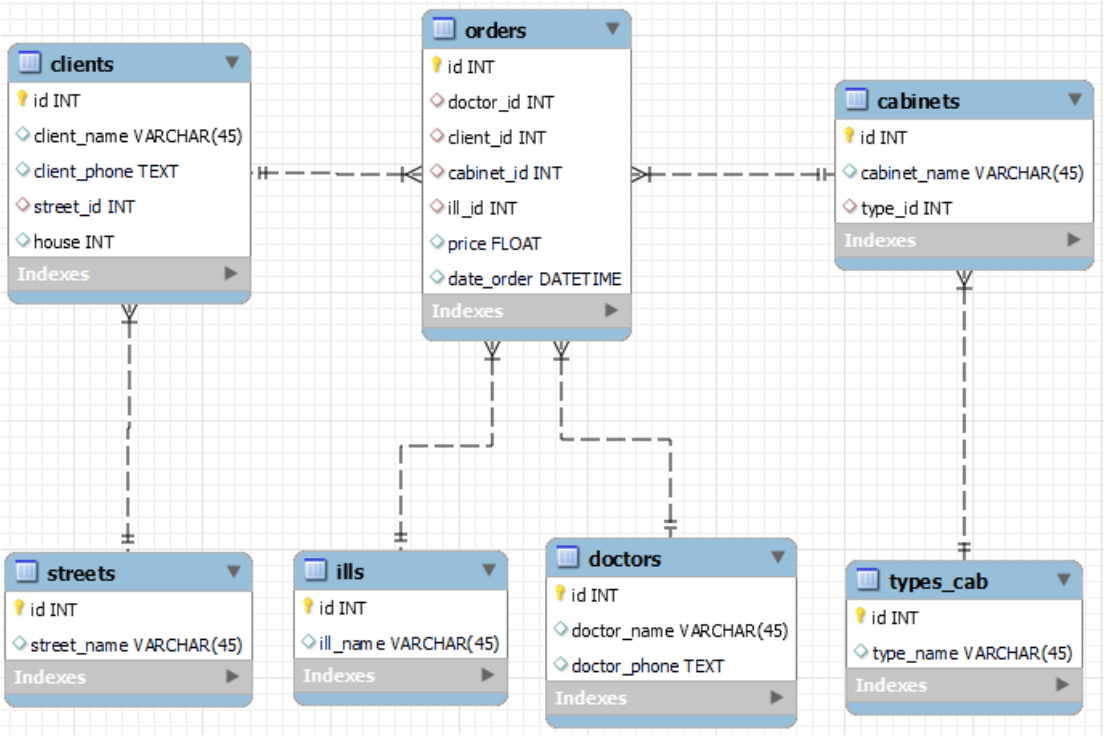

Рисунок 2 - Схема физической модели БД

Таким образом, было произведено физическое проектирование БД.

2. Реализация базы данных

2.1. Состав и структура базовых таблиц

База данных реализована средствами СУБД MYSQL. Запросы для создания таблиц приведены ниже.

```
CREATE TABLE IF NOT exists ills(
      id integer not null key auto_increment,
      ill_name varchar(45) unique
);
CREATE TABLE IF NOT exists streets(
      id integer not null key auto_increment,
      street_name varchar(45) unique
);
CREATE TABLE IF NOT exists clients(
      id integer not null key auto_increment,
      client_name varchar(45) unique,
    client_phone text,
    street_id int,
    house int,
      FOREIGN KEY (street_id)  REFERENCES streets (id)
);
CREATE TABLE IF NOT exists doctors(
      id integer not null key auto_increment,
      doctor_name varchar(45) unique,
    doctor_phone text
);
CREATE TABLE IF NOT exists types_cab(
      id integer not null key auto_increment,
      type_name varchar(45) unique
);
CREATE TABLE IF NOT exists cabinets(
      id integer not null key auto_increment,
      cabinet_name varchar(45) unique,
    type_id int,
      FOREIGN KEY (type_id)  REFERENCES types_cab (id)
);
CREATE TABLE IF NOT exists orders(
      id integer not null key auto_increment,
    doctor_id int,
      client_id int,
      cabinet_id int,
    ill_id int,
    price float,
    date_order datetime,
    FOREIGN KEY (doctor_id)  REFERENCES doctors (id),
      FOREIGN KEY (cabinet_id)  REFERENCES cabinets (id),
    FOREIGN KEY (ill_id)  REFERENCES ills (id),
      FOREIGN KEY (client_id)  REFERENCES clients (id)
);
```

Листинг 1. Создание таблиц

2.2. Разработка запросов

Обращение к базе данных осуществляется с помощью запросов. Для упрощения реализации программы все операции по добавлению, удалению, изменению данных в таблице, а так же запросы выполнены в виде хранимых процедур. Приведем пример реализованных процедур для одной таблице.

```
-- Добавление
CREATE PROCEDURE add_to_ills (note_name text)
INSERT INTO ills (ill_name) VALUES (note_name);


-- Вывод
CREATE PROCEDURE select_ill ()
SELECT id, ill_name as 'Жанр' from ills order by id;


-- Удаление
CREATE PROCEDURE kill_ills (note_id int)
DELETE FROM ills WHERE id=note_id;


-- Изменение
CREATE PROCEDURE update_ills (note_id int, note_name text)
UPDATE ills SET ill_name=note_name where id=note_id;
```

Полный код всех хранимых процедур приведен в приложение.

Листинг 2. Хранимые процедуры для работы с таблицей Ills (Болезни).

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

3.1 Диаграмма классов

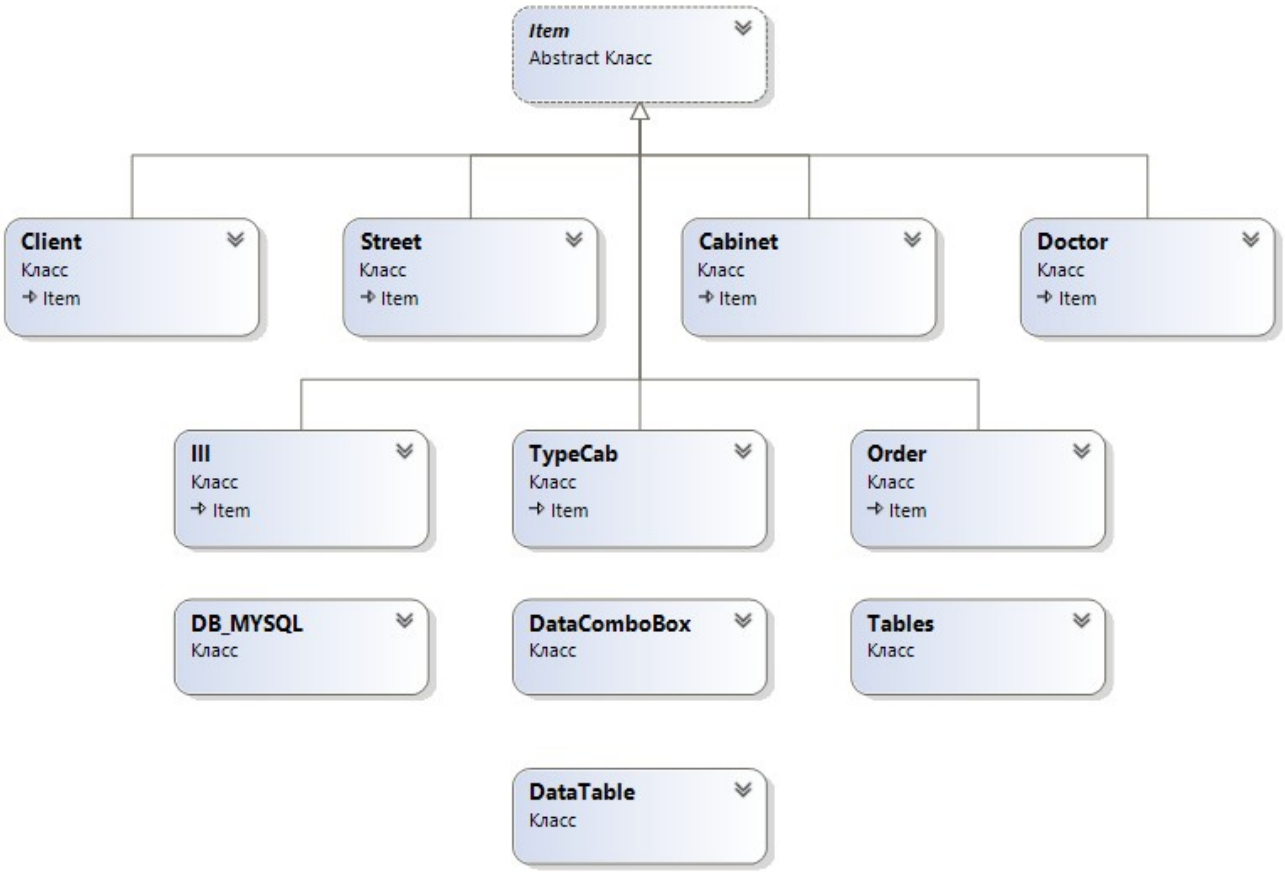

Рисунок 3. Диаграмма классов

В данной работе реализована иерархия классов для работы с таблицами, которые наследуются от класса Item, а так же класс для работы с базой данных DB_MYSQL и класс DataComboBox для добавления элементов в ComboBox на форме.

В данной работе имеется один абстрактный класс Item, от которого наследуются все классы для работы с таблицами. Данный класс имеет поле id, одинаковое для всех таблиц и метод для проверки, что все поля класс заполнены. Данный метод нужен для проверки перед добавлением в базу данных новой записи, что заполнены все поля. Данная проверка необходима для запрета добавления пустых данных в базу данных. Так же в классе реализованы методы для вызова хранимых процедур для работы с базой данных. Вызов данных процедур аналогичен для всех классов и отличается только набором полей для добавления или изменения, а так же названием

таблицы. Функция получения списка полей реализована как абстрактная и перегружается в каждом классе отдельно.

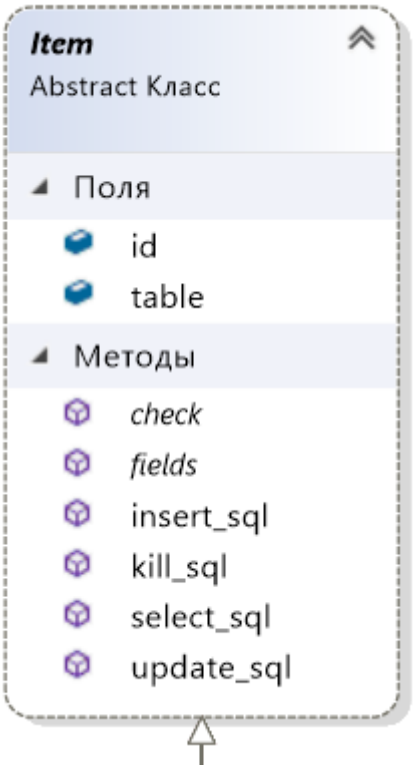

Рисунок 4. Класс Item

Классами наследниками являются все классы для работы с таблицами. Они наследуются от класса Item. В каждом классе имеется enum с перечислением всех полей в данной таблице. Так же в классе перегружается абстрактный метод базового класса для формирования запросов.

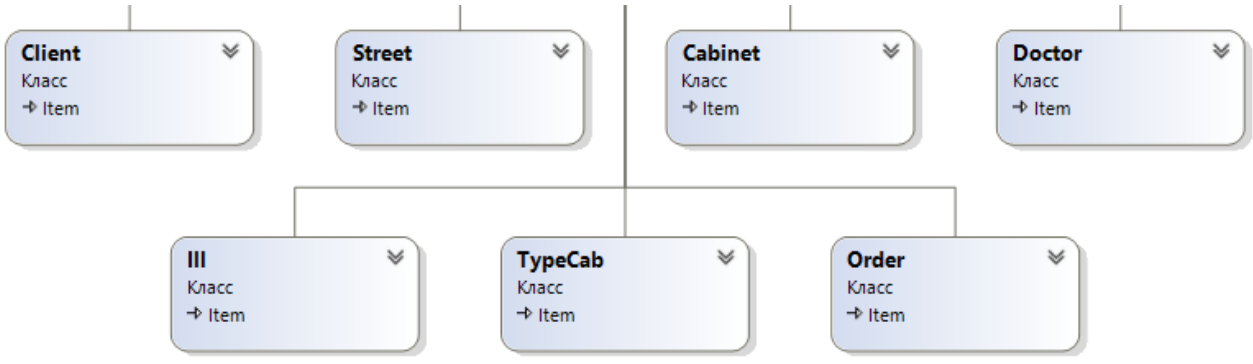

Рисунок 5. Наследуемые классы

Так же в программы реализованы и другие, не наследуемые классы.

Класс DataTable хранит названия таблицы, название главного поля в таблице, которое отображается в элементе ComboBox, ссылку на меню

GroupBox для работ с данной таблицей, ссылку на поле «Выбрать» в этом меню. Данный класс позволяет оптимизировать операции в программе. Так с его помощью достаточно в начале программы создать список из классов DataTable, и затем нужный элемент получать по индексы из списка, так как операции с разными таблицами однотипные. Это резко уменьшает число потенциальных ошибок в коде.

Так же в функции имеется делегат. Данный делегат необходим для хранения ссылки на функцию по считыванию данных с формы. В делегате два аргумента – считывать ли данные с номером записи id и ссылка на элемент «Выбрать» в данном меню.

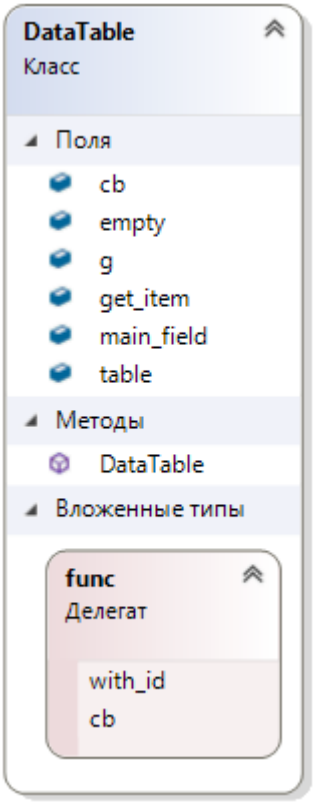

Рисунок 6. Класс DataTable

Так же в программе имеется класс для работы с базой данных DB_MYSQL. В данном классе реализованы все необходимые функции для работы с СУБД. Данный класс позволяет разделить интерфейс и работу с СУБД. В последствие данный класс можно заменить на другой класс для работы с другой СУДБ.

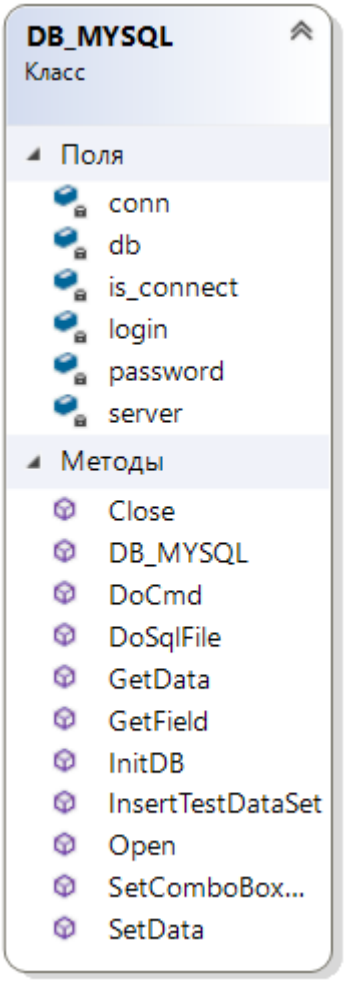

Рисунок 7. Класс DB_MYSQL

4 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММЫ

4.1 Описание программной реализации

Разработка графической части

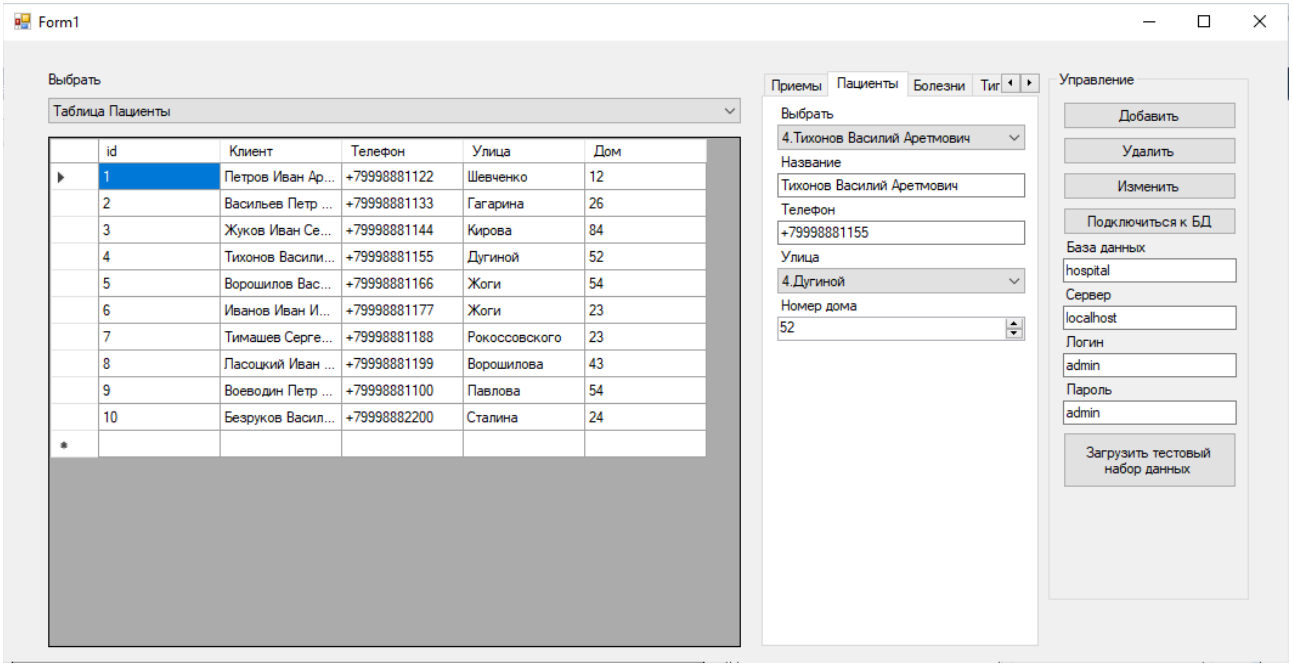

Рисунок 8. Окно программы

Графический интерфейс состоит из следующих элементов. Слева вверху выбор таблицы, с которой пользователь работает в данный момент. Ниже поле для вывода таблицы. Справа от таблицы меню для работы с каждой таблицей. Вкладки активируются при выборе соответствующей таблицей. Все остальные вкладки в этот момент становятся неактивными. Это сделано для минимизации ошибок и упрощение работы пользователю. Так он сразу видит, куда можно заносить данные, а куда нет, какая таблицы и какое меню для нее сейчас активно.

В правой части расположено меню для работы с программой. С его помощью можно добавлять, удалять, изменять данные в таблице. Так же с его помощью можно подключиться к нужной базе данных. Да тестирования имеется кнопка для добавления тестовых значений.

Программа построена на принципах ООП. В программе имеется основной класс формы приложения и серия классов для работы с данными для каждой таблицы.

Опишем программную часть. Для работы с формой имеется класс Form1. В данном классе содержатся все элементы формы и все функции обработчики кнопок и элементов выбора, а так же следующие поля

Таблица 1. «Переменные класса Form1»

Тип	Название	Описание
List<DataTable>	data	Список для упрощения работы программы
DB_MYSQL	DB	Соединение с базой данных.

В классе все функции разбиты на блоки.

Блок функций заполнения данных на форме

Заполнение данных при выборе записи в элементе выбора. Данные функции получают из БД данные согласно выбранной пользователем записи и заполняют все необходимые поля формы

Блок функций обработчиков нажатий кнопок

private void b_add_Click(object sender, EventArgs e) – функция добавляет запись в таблицу. В функции также имеется блок switch для определения из какая конкретно сейчас выбрана таблица и какие данные требуется считать с формы. В функции формируется запрос и выполняется проверка на полное заполнение всех полей. Если все поля заполнены программа выполняет запрос и добавляет данные в БД.

```
private void b_add_Click(object sender, EventArgs e)
{
	Item item = get_item(false);
	if (item == null)
	{
		MessageBox.Show("Выберете таблицу для добавления", "Ошибка");
		return;
	}
	if (item.check(false))
	{
		MessageBox.Show("Заполните все поля", "Ошибка");
		return;
	}
	if (!DB.DoCmd(item.insert_sql()))
		return;
	cb_tables_SelectedIndexChanged();
	MessageBox.Show("Запись добавлена", "Сообщение");
}
```

Листинг 3. Функция добавления записи в таблицу.

private void b_kill_Click(object sender, EventArgs e) – функция удаляет запись из БД. В функции также имеется блок switch для определения из какая конкретно сейчас выбрана таблица и следовательно из какой таблицы надо удалить запись. Затем функция считывает номер записи для удаления и выполняет запрос в БД

```
private void b_kill_Click(object sender, EventArgs e)
{
    int ind = cb_tables.SelectedIndex;
    if (!Tables.is_table(ind))
    {
        MessageBox.Show("Выберете таблицу для удаления", "Ошибка");
        return;
    }
    Item item = data[ind].empty;
    item.id = get_id(data[ind].cb);
    if (item.id == -1)
    {
        MessageBox.Show("Выберите запись для удаления", "Ошибка");
        return;
    }
    if (!DB.DoCmd(item.kill_sql()))
        return;
    cb_tables_SelectedIndexChanged();
    MessageBox.Show("Запись удалена", "Сообщение");
}
```

Листинг 4. Функция добавления записи в таблицу.

private void b_change_Click(object sender, EventArgs e) – функция работает аналогично добавлению, но только выполняет изменение данных выбранной записи

private void b_connect_Click(object sender, EventArgs e) – функция осуществляет подключение к выбранной базе данных

4.2 Тестирование программы

Для проведения тестирования добавим тестовые данные с помощью кнопки «Загрузить тестовый набор данных».

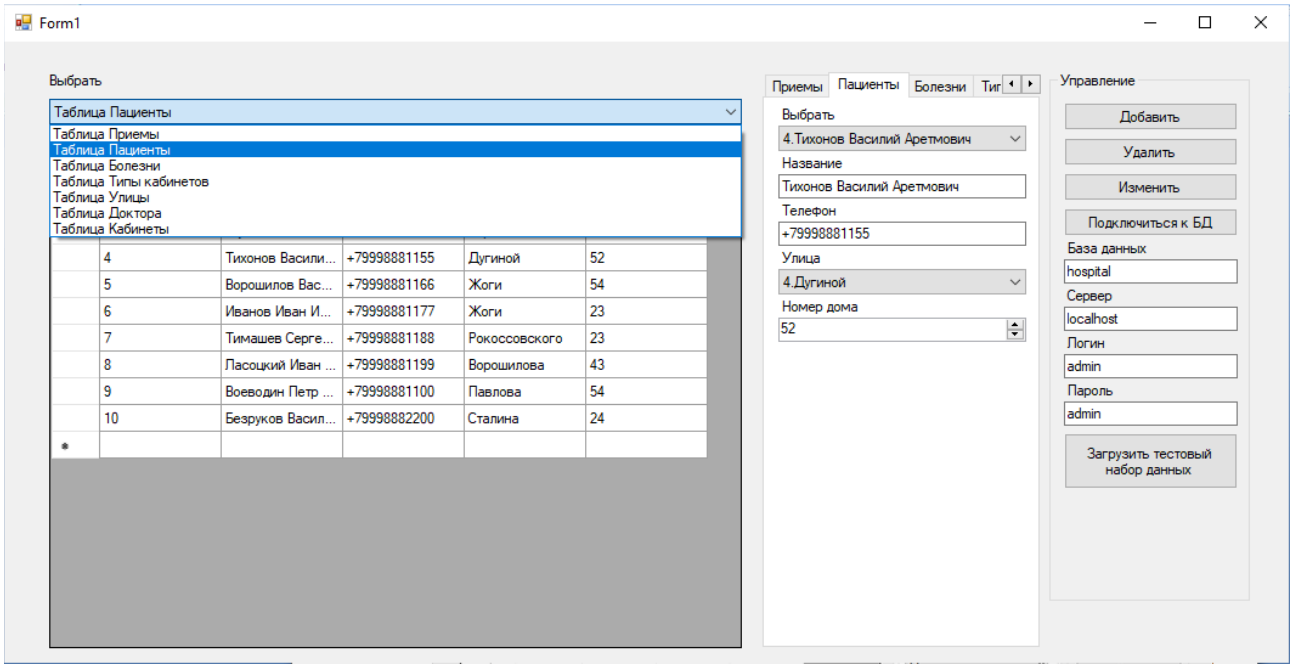

Рисунок 9. Выбор таблицы

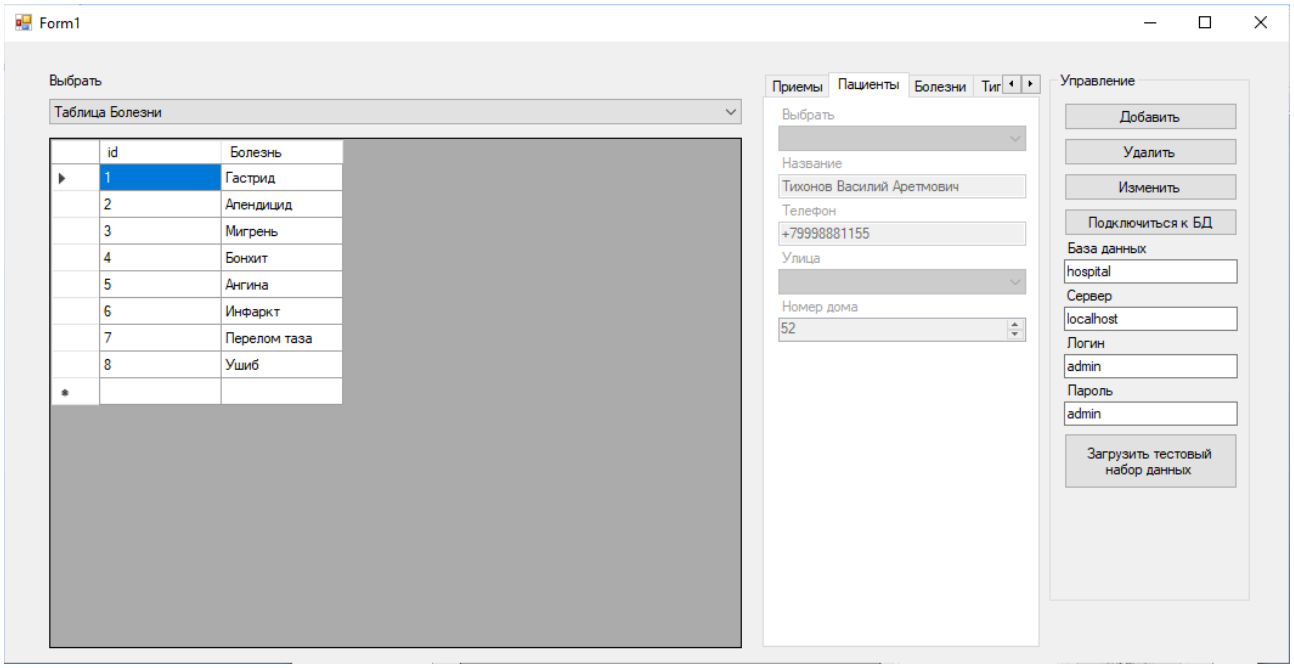

Рисунок 10 – Таблица Болезни

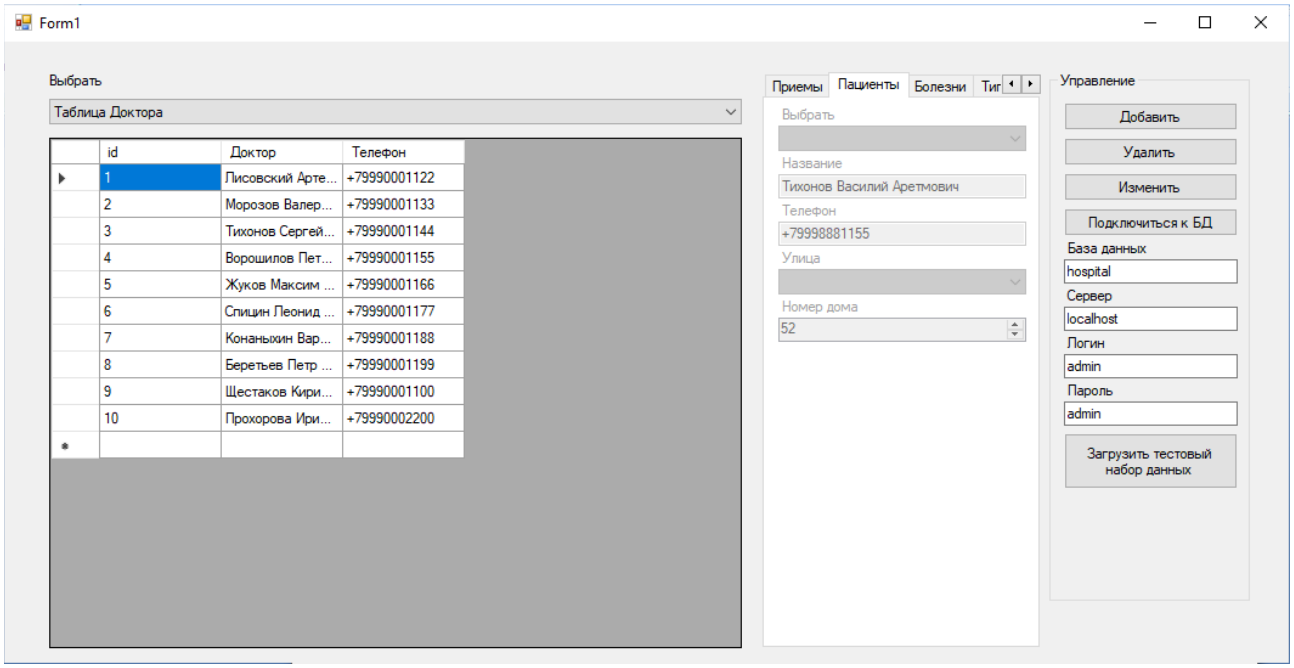

Рисунок 11 – Таблица Доктора

Заключение

В результате выполнения курсовой работы была спроектирована и средствами MYSQL реализована база данных поликлиники.

В ходе работы над первой главой курсового проекта проведено исследование предметной области, в результате чего, была сформулирована постановка задачи. Также, в рамках этой же главы было выполнено концептуальное, логическое и физическое проектирование БД средствами MS Visio. Получена итоговая диаграмма БД.

Во второй главе курсовой работы была проведена разработка БД средствам MYSQL на основе полученной физической модели. Для созданной БД разработаны необходимые пользовательские запросы. В конце главы приведена краткая инструкция по использованию базы данных.

Все поставленные в начале работы задачи выполнены. Цель курсовой работы достигнута.

Список литературы

1) Microsoft Visio. Официальный сайт разработчика. Электронный ресурс. Режим доступа: https://www.microsoft.com/ru-ru/microsoft-365/visio/flowchart-software

2) Волков, Д. А. Базы данных: учебно-методическое пособие / Д. А. Волков. — Москва: МИСИ-МГСУ, Ай Пи Эр Медиа, ЭБС АСВ, 2018. — 77 с. — ISBN 978-5-7264-1883-4. — Текст: электронный // Электронно-библиотечная система IPR BOOKS. Электронный ресурс. Режим доступа: http://www.iprbookshop.ru/79883.html

3) Жилинский, А. Самоучитель Misrosoft MySQL 2008 / А. Жилинский. - М.: БХВ-Петербург, 2018. - 240 с.

4) Кригель, А. MySQL. Библия пользователя / А. Кригель. - М.: Диалектика / Вильямс, 2019. - 318 с.

5) Кузнецов, С. Д. Основы баз данных / С.Д. Кузнецов. - М.: Бином. Лаборатория знаний, Интернет-университет информационных технологий, 2017. - 488 с.

6) Латыпова, Р. Р. Базы данных. Курс лекций / Р.Р. Латыпова. - Москва: Высшая школа, 2016. - 177 с.

7) Проектирование баз данных. Википедия. Электронный ресурс. Режим доступа: https://ru.wikipedia.org/wiki/Проектирование_баз_данных

Приложение № 1. Листинг программы

Form.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Program.Classes;

namespace Program
{
    public delegate Item func(bool with_id, DataTable dt);

    public partial class Form1 : Form
    {
        DB_MYSQL DB = null;
        List<DataComboBox> list_cb = new List<DataComboBox>();
        List<DataTable> tables = new List<DataTable>();

        string file_create = "create_table.sql";
        string file_test_data = "init.sql";
        const string format = "yyyy-MM-dd hh:mm";

        public Form1()
        {
            InitializeComponent();
            btns_enable(false);
            menu.Enabled = false;
            hide_items();
            init();
        }

        void init()
        {
            // таблицы
                  tables.Add(new DataTable(new Order(), get_order, cb_order,
tab_order));
                tables.Add(new DataTable(new Client(), get_client, cb_client,
tab_client, t_client_name, new ComboBox[] { cb_sel_client }));
             tables.Add(new DataTable(new Ill(), get_default, cb_ill, tab_ill,
t_ill_name, new ComboBox[] { cb_sel_ill }));
             tables.Add(new DataTable(new TypeCab(), get_default, cb_type_cab,
tab_type_cab, t_type_cab_name, new ComboBox[] { cb_sel_type_cab }));
               tables.Add(new DataTable(new Street(), get_default, cb_street,
tab_street, t_street_name, new ComboBox[] { cb_sel_street }));
                tables.Add(new DataTable(new Doctor(), get_doctor, cb_doctor,
tab_doctor, t_doctor_name, new ComboBox[] { cb_sel_doctor }));
              tables.Add(new DataTable(new Cabinet(), get_cabinet, cb_cabinet,
tab_cabinet, t_cabinet_name, new ComboBox[] { cb_sel_cabinet }));

            foreach (var data in tables)
                add_cb(data);
        }

        void Program_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (DB != null)
                DB.Close();
        }

        // заполнение полей при выборе в combobox
          private void cb_order_SelectedIndexChanged(object sender, EventArgs
e)
```

```
        {
            DataTable tb = tables[menu.SelectedIndex];
            ComboBox cb = tb.cb;
            int id = get_id(cb, tb.table.rus_name);

             set_cb(id, Order.Fileds.client_id, tb.table.name, cb_sel_client,
Client.Fileds.main, Tables.clients.name);
                  set_cb(id, Order.Fileds.ill_id, tb.table.name, cb_sel_ill,
Ill.Fileds.main, Tables.ills.name);
                        set_cb(id, Order.Fileds.cabinet_id, tb.table.name,
cb_sel_cabinet, Cabinet.Fileds.main, Tables.cabinets.name);
             set_cb(id, Order.Fileds.doctor_id, tb.table.name, cb_sel_doctor,
Doctor.Fileds.main, Tables.doctors.name);

                DB.SetData(Order.Fileds.date_order, tb.table.name, cb, null,
null, t_date_order);
            DB.SetData(Order.Fileds.price, tb.table.name, cb, null, t_price);
        }

         private void cb_cabinet_SelectedIndexChanged(object sender, EventArgs
e)
        {
            DataTable tb = tables[menu.SelectedIndex];
            ComboBox cb = tb.cb;
            int id = get_id(cb, tb.table.rus_name);

                         set_cb(id, Cabinet.Fileds.type_id, tb.table.name,
cb_sel_type_cab, TypeCab.Fileds.main, Tables.types_cab.name);

                  DB.SetData(Cabinet.Fileds.cabinet_name, tb.table.name, cb,
tb.t_name);
        }

         private void cb_doctor_SelectedIndexChanged(object sender, EventArgs
e)
        {
            DataTable tb = tables[menu.SelectedIndex];
            ComboBox cb = tb.cb;
            int id = get_id(cb, tb.table.rus_name);

                     DB.SetData(Doctor.Fileds.doctor_name, tb.table.name, cb,
t_doctor_name);
                  DB.SetData(Doctor.Fileds.doctor_phone, tb.table.name, cb,
t_doctor_phone);
        }

         private void cb_client_SelectedIndexChanged(object sender, EventArgs
e)
        {
            DataTable tb = tables[menu.SelectedIndex];
            ComboBox cb = tb.cb;
            int id = get_id(cb, tb.table.rus_name);

             set_cb(id, Client.Fileds.street_id, tb.table.name, cb_sel_street,
Street.Fileds.main, Tables.streets.name);

                     DB.SetData(Client.Fileds.client_name, tb.table.name, cb,
t_client_name);
                  DB.SetData(Client.Fileds.client_phone, tb.table.name, cb,
t_client_phone);
                   DB.SetData(Client.Fileds.house, tb.table.name, cb, null,
t_house);
        }
```

```
// считывание данных с формы
Item get_item(bool with_id)
{
    int ind = menu.SelectedIndex;
    return tables[ind].get_item(with_id, tables[ind]);
}

// считывание для простых таблиц с одним полем
Item get_default(bool with_id, DataTable dt)
{
    var item = dt.item;
    if (with_id)
        item.id = get_id(dt.cb, dt.table.rus_name);
    item.main_name = dt.t_name.Text;
    return item;
}

// считывание для таблиц с несколькими полями
Item get_client(bool with_id, DataTable dt)
{
    var item = new Client();
    if (with_id)
        item.id = get_id(dt.cb, dt.table.rus_name);
    item.client_name = t_client_name.Text;
    item.client_phone = t_client_phone.Text;
    item.street_id = get_id(cb_sel_street, Street.t.rus_name);
    item.house = (int)t_house.Value;
    return item;
}
Item get_order(bool with_id, DataTable dt)
{
    var item = new Order();
    if (with_id)
        item.id = get_id(dt.cb, dt.table.rus_name);
    item.client_id = get_id(cb_sel_client, Client.t.rus_name);
    item.ill_id = get_id(cb_sel_ill, Ill.t.rus_name);
    item.doctor_id = get_id(cb_sel_doctor, Doctor.t.rus_name);
    item.cabinet_id = get_id(cb_sel_cabinet, Cabinet.t.rus_name);
    item.date_order = get_date(t_date_order);
    item.price = (float)t_price.Value;
    return item;
}
Item get_doctor(bool with_id, DataTable dt)
{
    var item = new Doctor();
    if (with_id)
        item.id = get_id(dt.cb, dt.table.rus_name);
    item.doctor_name = t_doctor_name.Text;
    item.doctor_phone = t_doctor_phone.Text;
    return item;
}
Item get_cabinet(bool with_id, DataTable dt)
{
    var item = new Cabinet();
    if (with_id)
        item.id = get_id(dt.cb, dt.table.rus_name);
    item.cabinet_name = t_cabinet_name.Text;
    item.type_cab_id = get_id(cb_sel_type_cab, TypeCab.t.rus_name);
    return item;
}

//
void update_cb()
{
```

```
            DB.SetComboBoxData(ref list_cb);
        }
         void set_cb(int id, string field_id, string cur_table, ComboBox cb,
string field_name, string target_table)
        {
            try
            {
                        int  note_id  =  int.Parse(DB.GetField(id,  field_id,
cur_table));
                  string text = $"{note_id}.{DB.GetField(note_id, field_name,
target_table)}";
                cb.Text = text;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "Ошибка");
            }
        }
           int get_id(ComboBox cb, string field, string mess = "Не выбрано
значение в элементе выбора: ")
        {
            try
            {
                return int.Parse(cb.Text.Split('.')[0]);
            }
            catch
            {
                MessageBox.Show(mess + field, "Ошибка");
                return -1;
            }
        }
        string get_date(DateTimePicker dt, string format = format)
        {
            return dt.Value.ToString(format); ;
        }

        // отображение выборок из базы данных
        void select_menu(object sender = null, EventArgs e = null)
        {
            hide_items();
            int ind = menu.SelectedIndex;

            if (!Tables.is_table(ind))
            {
                tables[ind].item = get_item(true);
                if (tables[ind].item.not_full(false))
                {
                            MessageBox.Show("Выберите условие в меню Запрос",
"Ошибка");
                    return;
                }
            }
                              if   (DB.GetData(tables[ind].item.select_sql(),
tables[ind].table.name, ref table))
                update_cb();
        }

        // кнопки
        void b_add_Click(object sender, EventArgs e)
        {
            Item item = get_item(false);
            if (item == null)
            {
                MessageBox.Show("Выберете таблицу для добавления", "Ошибка");
```

```
                return;
            }
            if (item.not_full(false))
            {
                MessageBox.Show("Одно из полей незаполненно", "Ошибка");
                return;
            }
            if (!DB.DoCmd(item.insert_sql()))
                return;
            select_menu();
            MessageBox.Show("Запись добавлена", "Сообщение");

        }
        void b_kill_Click(object sender, EventArgs e)
        {
            int ind = menu.SelectedIndex;
            if (!Tables.is_table(ind))
            {
                     MessageBox.Show("Выберете в меню таблицу для удаления",
"Ошибка");
                return;
            }
            Item item = tables[ind].item;
            item.id = get_id(tables[ind].cb, item.table.rus_name);
            if (item.id == -1)
                return;

            if (!DB.DoCmd(item.kill_sql()))
                return;
            select_menu();
            MessageBox.Show("Запись удалена", "Сообщение");
        }
        void b_load_Click(object sender, EventArgs e)
        {
            DB.InsertTestDataSet(file_test_data);
            menu.SelectedIndex = 1;
            hide_items();
            select_menu();
        }
        void b_change_Click(object sender, EventArgs e)
        {
            Item item = get_item(true);
            if (item == null)
            {
                MessageBox.Show("Выберете таблицу для изменения", "Ошибка");
                return;
            }
            if (item.not_full(true))
            {
                    MessageBox.Show("Заполните все поля и выберете запись для
изменения", "Ошибка");
                return;
            }
            if (!DB.DoCmd(item.update_sql()))
                return;
            select_menu();
            MessageBox.Show("Запись изменена", "Сообщение");
        }
        void b_connect_Click(object sender, EventArgs e)
        {
            string db = t_shema.Text;
            string server = t_server.Text;
            string login = t_login.Text;
            string password = t_password.Text;
```

```
            DB = new DB_MYSQL(db, server, login, password);

            if (!DB.Connect(file_create))
                return;

            btns_enable(true);
            menu.Enabled = true;
            menu.SelectedIndex = 0;
            MessageBox.Show("База данных успешно подключена", "Сообщение");
        }

        // работа с окном
        void hide_items()
        {
            int ind = menu.SelectedIndex;

            foreach (var note in tables)
                if (note.tab != null)
                    note.tab.Enabled = false;

            if (!Tables.is_table(ind))
                return;

            if (menu.SelectedIndex != -1)
                tables[menu.SelectedIndex].tab.Enabled = true;
        }
        void btns_enable(bool state)
        {
                foreach (var btn in new Button[] { b_add, b_kill, b_change,
b_load })
                btn.Enabled = state;
        }
        void set_default(object sender = null, EventArgs e = null)
        {
            DataTable tb = tables[menu.SelectedIndex];
            ComboBox cb = tb.cb;
            TextBox t_name = tb.t_name;

            DB.SetData(tb.table.main, tb.table.name, cb, t_name);
        }
        void add_cb(DataTable tb)
        {
            list_cb.Add(new DataComboBox(tb.cb, tb.table));
            if (tb.cbs != null)
                foreach (var cb in tb.cbs)
                    list_cb.Add(new DataComboBox(cb, tb.table));
        }
    }
}
```

create_table.sql

```
CREATE TABLE IF NOT exists ills(
      id integer not null key auto_increment,
      ill_name varchar(45) unique
);

CREATE TABLE IF NOT exists streets(
      id integer not null key auto_increment,
      street_name varchar(45) unique
);
```

```
CREATE TABLE IF NOT exists clients(
      id integer not null key auto_increment,
      client_name varchar(45) unique,
    client_phone text,
    street_id int,
    house int,
      FOREIGN KEY (street_id)  REFERENCES streets (id)
);

CREATE TABLE IF NOT exists doctors(
      id integer not null key auto_increment,
      doctor_name varchar(45) unique,
    doctor_phone text
);

CREATE TABLE IF NOT exists types_cab(
      id integer not null key auto_increment,
      type_name varchar(45) unique
);

CREATE TABLE IF NOT exists cabinets(
      id integer not null key auto_increment,
      cabinet_name varchar(45) unique,
    type_id int,
      FOREIGN KEY (type_id)  REFERENCES types_cab (id)
);

CREATE TABLE IF NOT exists orders(
      id integer not null key auto_increment,
    doctor_id int,
      client_id int,
      cabinet_id int,
    ill_id int,
    price float,
    date_order datetime,
    FOREIGN KEY (doctor_id)  REFERENCES doctors (id),
      FOREIGN KEY (cabinet_id)  REFERENCES cabinets (id),
    FOREIGN KEY (ill_id)  REFERENCES ills (id),
      FOREIGN KEY (client_id)  REFERENCES clients (id)
);

-- Добавление
CREATE PROCEDURE add_to_ills (ill_name varchar(45))
INSERT INTO ills (ill_name) VALUES (ill_name);

CREATE PROCEDURE add_to_clients (client_name varchar(45), client_phone text,
street_id int, house int)
INSERT INTO clients (client_name, client_phone, street_id, house) VALUES
(client_name, client_phone, street_id, house);

CREATE PROCEDURE add_to_cabinets (cabinet_name varchar(45), type_id int)
INSERT INTO cabinets (cabinet_name, type_id) VALUES (cabinet_name, type_id);

CREATE PROCEDURE add_to_doctors (doctor_name varchar(45), doctor_phone text)
INSERT INTO doctors (doctor_name, doctor_phone) VALUES (doctor_name,
doctor_phone);

CREATE PROCEDURE add_to_streets (street_name varchar(45))
INSERT INTO streets (street_name) VALUES (street_name);

CREATE PROCEDURE add_to_types_cab (type_name varchar(45))
INSERT INTO types_cab (type_name) VALUES (type_name);
```

```
CREATE PROCEDURE add_to_orders (doctor_id int, ill_id int,  cabinet_id int,
client_id int, date_order datetime, price float)
INSERT INTO orders (doctor_id, ill_id, cabinet_id, client_id, date_order,
price) VALUES
(doctor_id, ill_id, cabinet_id, client_id, date_order, price);

-- Удаление
CREATE PROCEDURE kill_ills (note_id int)
DELETE FROM ills WHERE id=note_id;

CREATE PROCEDURE kill_clients (note_id int)
DELETE FROM clients WHERE id=note_id;

CREATE PROCEDURE kill_doctors (note_id int)
DELETE FROM doctors WHERE id=note_id;

CREATE PROCEDURE kill_cabinets (note_id int)
DELETE FROM cabinets WHERE id=note_id;

CREATE PROCEDURE kill_streets (note_id int)
DELETE FROM streets WHERE id=note_id;

CREATE PROCEDURE kill_types_cab (note_id int)
DELETE FROM types_cab WHERE id=note_id;

CREATE PROCEDURE kill_orders (note_id int)
DELETE FROM orders WHERE id=note_id;

-- Изменение
CREATE PROCEDURE update_ills (note_id int, ill_name varchar(45))
UPDATE ills SET ill_name=ill_name where id=note_id;

CREATE  PROCEDURE  update_clients  (note_id  int,  client_name  varchar(45),
client_phone text, street_id int, house int)
UPDATE  clients  SET  client_name=client_name,  client_phone=client_phone,
street_id=street_id, house=house where id=note_id;

CREATE  PROCEDURE  update_doctors  (note_id  int,  doctor_name  varchar(45),
doctor_phone text)
UPDATE doctors SET doctor_name=doctor_name, doctor_phone=doctor_phone where
id=note_id;

CREATE  PROCEDURE  update_cabinets  (note_id  int,  cabinet_name  varchar(45),
type_id int)
UPDATE  cabinets  SET  cabinet_name=cabinet_name,  type_id=type_id  where
id=note_id;

CREATE PROCEDURE update_streets (note_id int, street_name varchar(45))
UPDATE streets SET street_name=street_name where id=note_id;

CREATE PROCEDURE update_types_cab (note_id int, type_name varchar(45))
UPDATE types_cab SET type_name=type_name where id=note_id;

CREATE  PROCEDURE  update_orders  (note_id  int,  doctor_id  int,  ill_id  int,
cabinet_id int, client_id int, date_order datetime, price float)
UPDATE orders SET
doctor_id=doctor_id,
ill_id=ill_id,
cabinet_id=cabinet_id,
client_id=client_id,
date_order=date_order,
price=price
where id=note_id;
```

```
-- Вывод
CREATE PROCEDURE select_ills ()
SELECT id, ill_name as 'Болезнь' from ills order by id;

CREATE PROCEDURE select_clients ()
SELECT cl.id, client_name as 'Клиент', client_phone as 'Телефон',
s.street_name as 'Улица', house as 'Дом' from clients as cl
join streets as s ON s.id=cl.street_id
order by cl.id;

CREATE PROCEDURE select_doctors ()
SELECT id, doctor_name as 'Доктор', doctor_phone as 'Телефон' from doctors
order by id;

CREATE PROCEDURE select_cabinets ()
SELECT c.id, cabinet_name as 'Кабинет', t.type_name as 'Тип кабинета' from
cabinets as c
join types_cab as t ON t.id=c.type_id
order by c.id;

CREATE PROCEDURE select_types_cab ()
SELECT id, type_name as 'Тип кабинета' from types_cab order by id;

CREATE PROCEDURE select_streets ()
SELECT id, street_name as 'Улица' from streets order by id;

CREATE PROCEDURE select_orders ()
SELECT o.id,
d.doctor_name as 'Доктор',
cl.client_name as 'Пациент',
cab.cabinet_name as 'Кабинет',
i.ill_name as 'Диагноз',
date_order as 'Дата приема',
price as 'Стоимость приема'
from orders as o
join doctors as d ON d.id=o.doctor_id
join clients as cl ON cl.id=o.client_id
join cabinets as cab ON cab.id=o.cabinet_id
join ills as i ON i.id=o.ill_id
order by o.id;
```