Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

СОЗДАНИЕ ПРИЛОЖЕНИЯ В MICROSOFT VISUAL STUDIO

ДЛЯ РАБОТЫ СО СВЯЗАННЫМИ ТАБЛИЦАМИ БАЗЫ ДАННЫХ

Отчет по лабораторной работе по дисциплине «Основы разработки баз данных»

Студент гр. 573-3

\_\_\_\_\_\_\_\_\_\_\_\_ Е. А. Макаров

дата

\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Руководитель:

Преподаватель:

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ Р. О. Остапенко

подпись

оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

Томск, 2025

**Введение**

В современном мире базы данных играют важную роль в работе различных организаций. Чтобы упростить и автоматизировать такую работу, создаются специальные приложения, которые позволяют удобно работать с данными.

Microsoft Visual Studio — это удобная среда для создания программ с графическим интерфейсом. В ней можно разрабатывать приложения, которые работают с базами данных и отображают связанные таблицы. Это помогает видеть всю информацию сразу и легко её изменять.

Цель этой лабораторной работы — научиться создавать приложение для работы с базой данных школы. В нём будет использоваться форма, где можно просматривать и редактировать связанные таблицы. Также будет добавлен шаблон «Одиночка» для правильной организации кода, реализована обработка ошибок, поиск и фильтрация данных. Кроме того, будут использованы различные элементы управления и созданы вычисляемые и подстановочные поля.

В результате будет создано простое, но полезное приложение для работы с базой данных, которое поможет лучше понять, как устроены программы, обрабатывающие связанную информацию.

**Задачи:**

Для своей БД необходимо выполнить:

* создать форму/формы проекта для работы с данными по связанным таблицам. Обязательно 1:М на одной форме, М:М через 1:М и М:1 на одной форме;
* реализовать шаблон проектирования «Одиночка» (англ. Singleton);
* обработать исключения в программе, возникающие при некорректной работе с данными (не менее 2-3 разных ошибок);
* реализовать поиск, фильтрацию данных;
* создать вычисляемые колонки (не менее 1-2 разных);
* создать подстановочные колонки (не менее 1-2 разных);

1. **ОСНОВНАЯ ЧАСТЬ**
   1. **Работа с таблицами М:М**

В нашей лабораторной работе встречается сущность Учитель-Предмет, которая хранит связь М:М между этими таблицами. Для работы с данной связью мы модифицируем окно для предмета. Оно будет позволять добавлять одному предмету несколько преподавателей (рисунок 1.1) и разным преподавателем один предмет (рисунок 1.2).

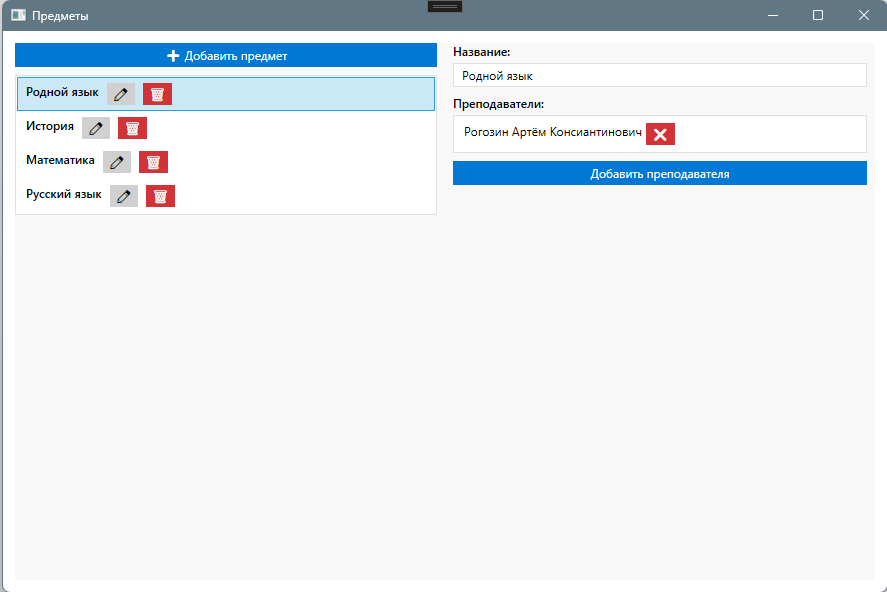


Рисунок 1.1 – Окно с предметами

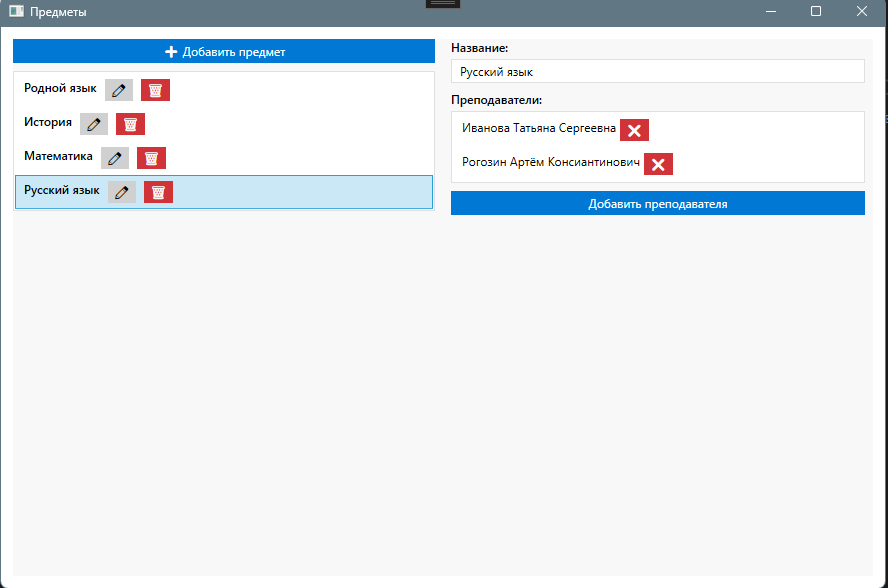


Рисунок 1.2 - Окно с предметами

Как мы видим на рисунках, один преподаватель может вести разные предметы, как и у одного предмета могут быть разные преподаватели.

* 1. **Обработка исключений**

В базе данных у нас встречаются столбцы с ограничениями, которые мы будем проверять на корректность ввода. Для этого достаточно использовать конструкцию try – catch и обрабатывать исключения, получаемые от СУБД.

Добавим обработчик исключения (листинг 1.1) на создание школьника, чтобы не позволять добавлять некорректные данные в БД. Также сделаем уведомление об этом в приложении (рисунок 1.3), чтобы пользователь получал ошибку.

Листинг 1.1 – Обработка исключения

try

{

var newClass = new Class

{

Number = number,

Letter = letterBox.Text,

TeacherId = teacherComboBox.SelectedItem != null ? ((Teacher)teacherComboBox.SelectedItem).TeacherId : (int?)null

};

\_classRepository.Create(newClass);

if (newClass.TeacherId.HasValue)

{

var teacher = \_teacherRepository.GetById(newClass.TeacherId.Value);

teacher.ClassId = newClass.ClassId;

\_teacherRepository.Update(teacher);

}

\_dataChangeNotifier.NotifyClassChanged();

MessageBox.Show($"Класс {newClass.FullName} успешно добавлен.", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

window.Close();

}

catch (System.Exception ex)

{

MessageBox.Show($"Ошибка при добавлении: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

};

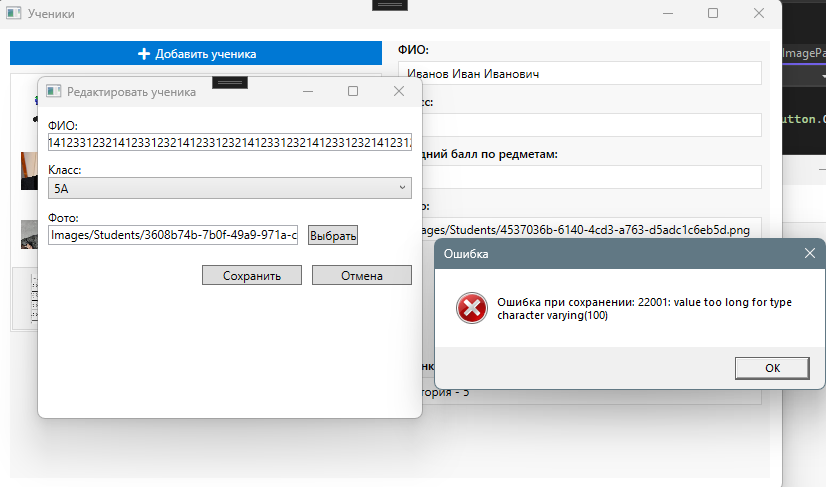


Рисунок 1.3 – Обработка ошибки

Также пример обработки для окна с предметами (рисунок 1.4).

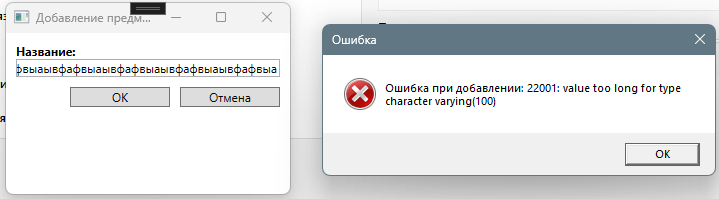


Рисунок 1.4 – Обработка ошибки

Как мы видим, в обоих случая возникает ошибка из-за ограничения на длину, программа ловит его, обрабатывает и выводит сообщение с ошибкой.

* 1. **Фильтрация**

Окно расписания подразумевает, что в нём будет очень много данных, которые не всегда будет удобно смотреть полностью. Добавим фильтрацию по четверти и классу (рисунок 1.5 и 1.6), чтобы смотреть его расписание в конкретную четверть.

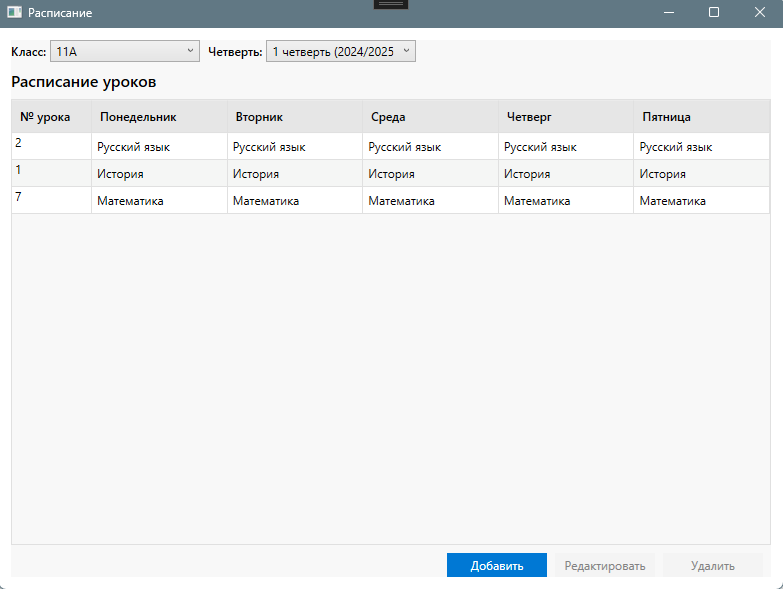


Рисунок 1.5 – Фильтрация расписания

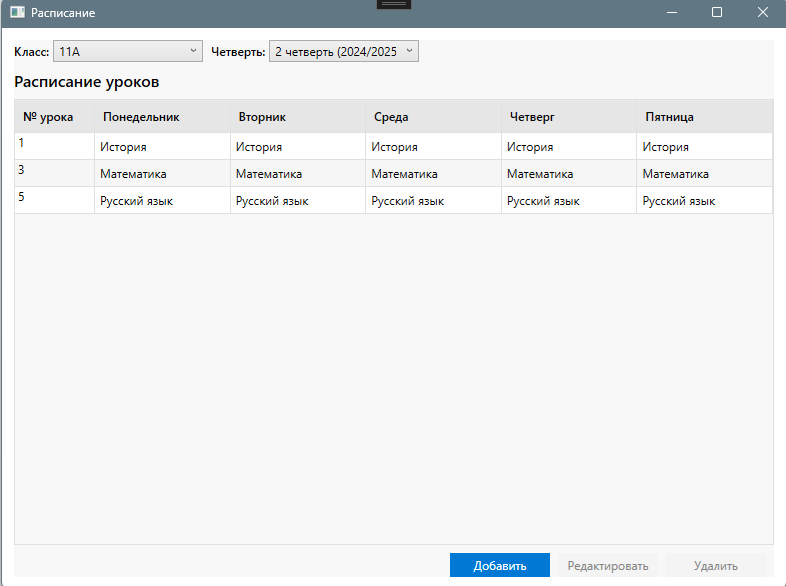


Рисунок 1.6 – Фильтрация расписания

* 1. **Вычисляемы колонки**

Не всегда нужно хранить все данные в базе данных. Может быть намного удобнее вычислять их по ходу работы приложения.

Создадим для ученика функцию подсчёта его среднего балла (листинг 1.2) и выведем его в нашу форму (рисунок 1.7).

Листинг 1.2 – Функция вычисления среднего балла

private double? CalculateAverageGrade(int studentId, NpgsqlConnection connection)

{

using var command = new NpgsqlCommand(

"SELECT AVG(grade\_value) FROM grade WHERE student\_id = @student\_id", connection);

command.Parameters.AddWithValue("@student\_id", studentId);

var result = command.ExecuteScalar();

if (result == DBNull.Value)

{

return null;

}

return Convert.ToDouble(result);

}

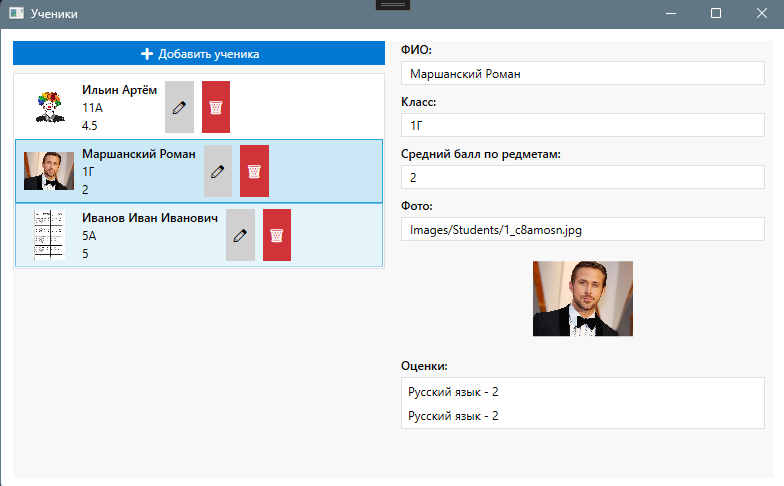


Рисунок 1.7 – Пример отображения среднего балла

* 1. **Подстановочные колонки**

Когда мы создаём какой либо объект, содержащий внешние ключи, можно вместо ID эти предметов добавлять сразу эти значения с помощью команды JOIN.

Сделаем так, что при выставлении оценки отображались только существующие преподаватели, предметы и ученики. Обновим репозиторий для этого (листинг 1.3) и добавим все изменения в окно (рисунок 1.8).

Листинг 1.3 – Код репозитория для оценок

using Application.Interfaces;

using Npgsql;

using SchoolDB.Application.Interfaces;

using SchoolDB.Domain.Entities;

using System;

using System.Collections.Generic;

namespace Infrastructure.Repositories

{

public class GradeRepository : IGradeRepository

{

private readonly string \_connectionString;

private readonly IDataChangeNotifier \_notifier;

public GradeRepository(string connectionString, IDataChangeNotifier notifier)

{

\_connectionString = connectionString;

\_notifier = notifier;

}

public void Create(Grade entity)

{

using var connection = new NpgsqlConnection(\_connectionString);

using var command = new NpgsqlCommand(

"INSERT INTO grade (student\_id, subject\_id, quarter\_id, grade\_value) VALUES (@student\_id, @subject\_id, @quarter\_id, @grade\_value) RETURNING grade\_id", connection);

command.Parameters.AddWithValue("@student\_id", entity.StudentId);

command.Parameters.AddWithValue("@subject\_id", entity.SubjectId);

command.Parameters.AddWithValue("@quarter\_id", entity.QuarterId);

command.Parameters.AddWithValue("@grade\_value", entity.GradeValue);

connection.Open();

entity.GradeId = (int)command.ExecuteScalar();

\_notifier.NotifyGradeChanged();

}

public Grade GetById(int id)

{

using var connection = new NpgsqlConnection(\_connectionString);

using var command = new NpgsqlCommand(

@"SELECT g.grade\_id, g.student\_id, g.subject\_id, g.quarter\_id, g.grade\_value,

s.student\_id, s.full\_name,

sub.subject\_id, sub.subject\_name,

q.quarter\_id, q.quarter\_number, q.academic\_year,

t.teacher\_id, t.full\_name

FROM grade g

JOIN student s ON g.student\_id = s.student\_id

JOIN subject sub ON g.subject\_id = sub.subject\_id

JOIN quarter q ON g.quarter\_id = q.quarter\_id

JOIN teacher\_subject ts ON ts.subject\_id = sub.subject\_id

JOIN teacher t ON ts.teacher\_id = t.teacher\_id

WHERE g.grade\_id = @id", connection);

command.Parameters.AddWithValue("@id", id);

connection.Open();

using var reader = command.ExecuteReader();

if (reader.Read())

{

var grade = new Grade

{

GradeId = reader.GetInt32(0),

StudentId = reader.GetInt32(1),

SubjectId = reader.GetInt32(2),

QuarterId = reader.GetInt32(3),

GradeValue = reader.GetInt32(4),

Student = new Student { StudentId = reader.GetInt32(5), FullName = reader.GetString(6) },

Subject = new Subject { SubjectId = reader.GetInt32(7), SubjectName = reader.GetString(8) },

Quarter = new Quarter { QuarterId = reader.GetInt32(9), QuarterNumber = reader.GetInt32(10), AcademicYear = reader.GetString(11) }

};

grade.Subject.TeacherSubjects = new List<TeacherSubject>

{

new TeacherSubject { TeacherId = reader.GetInt32(12), SubjectId = grade.SubjectId }

};

return grade;

}

return null;

}

public IEnumerable<Grade> GetAll()

{

var grades = new List<Grade>();

using var connection = new NpgsqlConnection(\_connectionString);

using var command = new NpgsqlCommand(

@"SELECT g.grade\_id, g.student\_id, g.subject\_id, g.quarter\_id, g.grade\_value,

s.student\_id, s.full\_name,

sub.subject\_id, sub.subject\_name,

q.quarter\_id, q.quarter\_number, q.academic\_year,

t.teacher\_id, t.full\_name

FROM grade g

JOIN student s ON g.student\_id = s.student\_id

JOIN subject sub ON g.subject\_id = sub.subject\_id

JOIN quarter q ON g.quarter\_id = q.quarter\_id

JOIN teacher\_subject ts ON ts.subject\_id = sub.subject\_id

JOIN teacher t ON ts.teacher\_id = t.teacher\_id", connection);

connection.Open();

using var reader = command.ExecuteReader();

while (reader.Read())

{

var grade = new Grade

{

GradeId = reader.GetInt32(0),

StudentId = reader.GetInt32(1),

SubjectId = reader.GetInt32(2),

QuarterId = reader.GetInt32(3),

GradeValue = reader.GetInt32(4),

Student = new Student { StudentId = reader.GetInt32(5), FullName = reader.GetString(6) },

Subject = new Subject { SubjectId = reader.GetInt32(7), SubjectName = reader.GetString(8) },

Quarter = new Quarter { QuarterId = reader.GetInt32(9), QuarterNumber = reader.GetInt32(10), AcademicYear = reader.GetString(11) }

};

grade.Subject.TeacherSubjects = new List<TeacherSubject>

{

new TeacherSubject { TeacherId = reader.GetInt32(12), SubjectId = grade.SubjectId }

};

grades.Add(grade);

}

return grades;

}

public IEnumerable<Grade> GetByStudentId(int studentId)

{

var grades = new List<Grade>();

using var connection = new NpgsqlConnection(\_connectionString);

using var command = new NpgsqlCommand(

@"SELECT g.grade\_id, g.student\_id, g.subject\_id, g.quarter\_id, g.grade\_value,

sub.subject\_id, sub.subject\_name,

q.quarter\_id, q.quarter\_number, q.academic\_year,

t.teacher\_id, t.full\_name

FROM grade g

JOIN subject sub ON g.subject\_id = sub.subject\_id

JOIN quarter q ON g.quarter\_id = q.quarter\_id

JOIN teacher\_subject ts ON ts.subject\_id = sub.subject\_id

JOIN teacher t ON ts.teacher\_id = t.teacher\_id

WHERE g.student\_id = @student\_id", connection);

command.Parameters.AddWithValue("@student\_id", studentId);

connection.Open();

using var reader = command.ExecuteReader();

while (reader.Read())

{

var grade = new Grade

{

GradeId = reader.GetInt32(0),

StudentId = reader.GetInt32(1),

SubjectId = reader.GetInt32(2),

QuarterId = reader.GetInt32(3),

GradeValue = reader.GetInt32(4),

Subject = new Subject { SubjectId = reader.GetInt32(5), SubjectName = reader.GetString(6) },

Quarter = new Quarter { QuarterId = reader.GetInt32(7), QuarterNumber = reader.GetInt32(8), AcademicYear = reader.GetString(9) }

};

grade.Subject.TeacherSubjects = new List<TeacherSubject>

{

new TeacherSubject { TeacherId = reader.GetInt32(10), SubjectId = grade.SubjectId }

};

grades.Add(grade);

}

return grades;

}

public void Update(Grade entity)

{

using var connection = new NpgsqlConnection(\_connectionString);

using var command = new NpgsqlCommand(

"UPDATE grade SET student\_id = @student\_id, subject\_id = @subject\_id, quarter\_id = @quarter\_id, grade\_value = @grade\_value WHERE grade\_id = @id", connection);

command.Parameters.AddWithValue("@student\_id", entity.StudentId);

command.Parameters.AddWithValue("@subject\_id", entity.SubjectId);

command.Parameters.AddWithValue("@quarter\_id", entity.QuarterId);

command.Parameters.AddWithValue("@grade\_value", entity.GradeValue);

command.Parameters.AddWithValue("@id", entity.GradeId);

connection.Open();

command.ExecuteNonQuery();

\_notifier.NotifyGradeChanged();

}

public void Delete(int id)

{

using var connection = new NpgsqlConnection(\_connectionString);

using var command = new NpgsqlCommand(

"DELETE FROM grade WHERE grade\_id = @id", connection);

command.Parameters.AddWithValue("@id", id);

connection.Open();

command.ExecuteNonQuery();

\_notifier.NotifyGradeChanged();

}

}

}

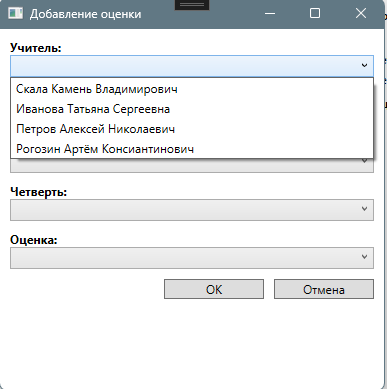


Рисунок 1.8 – Пример работы окна добавления

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы было разработано приложение для работы с базой данных школы с использованием Microsoft Visual Studio. В приложении были созданы формы для отображения и редактирования связанных таблиц, в том числе реализованы связи типа «один ко многим», «многие ко многим» и «многие к одному».

Реализована основная логика работы с данными: добавление, редактирование, удаление и просмотр информации. Применён шаблон проектирования «Одиночка» для организации доступа к базе данных. Также была выполнена обработка ошибок при неправильной работе с данными, добавлены функция фильтрации, созданы вычисляемые и подстановочные колонки.

В процессе работы были изучены возможности библиотеки ADO.NET для взаимодействия с базой данных и освоены основные элементы управления, такие как DataGridView, TextBox, Label и другие.