Приднестровский государственный университет им. Т.Г. Шевченко

Инженерно-технический институт

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**КУРСОВАЯ РАБОТА**

по дисциплине: Объектно-ориентированное программирование

Тема: Логическая игра “Крестики-нолики”

Работу выполнил

студент группы ИТ21ДР62ПИ

Кабанов Илья Андреевич

Проверил:

Ст. преподаватель:

Федорченко Г.С.

Тирасполь, 2023

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 3

1 Исследование предметной области 4

1.1 Описание поставленной задачи 4

1.2 Описание предметной области 5

1.3 Описание выявленных сущностей 6

1.4 Взаимодействие между сущностями 6

1.5 Особые ситуации 7

1.6 Тестирование 8

1.7 План работы 8

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 13

2.1 Диаграмма классов 13

2.2 Интерфейсные классы 14

2.3 Абстрактный класс 15

2.4 Классы-наследники 15

2.5 События 15

2.6 Исключительные ситуации 15

3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 17

3.1 Описание программной реализации 17

3.2 Тестирование всей программы 20

ЗАКЛЮЧЕНИЕ 30

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 31

ПРИЛОЖЕНИЕ – Листинги программы 32

Оглавление

[1 ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc135866875)

[**1.1** **Описание поставленной задачи** 6](#_Toc135866876)

[**1.2** **Описание предметной области** 7](#_Toc135866877)

[**1.3** **Описание выявленных сущностей** 8](#_Toc135866878)

[**1.4** **Взаимодействие между сущностями** 9](#_Toc135866879)

[**1.5** **Особые ситуации** 11](#_Toc135866880)

[**1.6** **Тестирование** 12](#_Toc135866881)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 14](#_Toc135866882)

[**2.1** **Диаграмма классов** 14](#_Toc135866883)

[**2.2** **Класс** 15](#_Toc135866884)

[**2.3** **События** 16](#_Toc135866885)

[СПИСок ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 32](#_Toc135866886)

[Приложение А. – Руководство пользователя 33](#_Toc135866887)

**ВВЕДЕНИЕ**

Логическая игра "Крестики-нолики" является одной из наиболее популярных и известных игр на протяжении многих лет. Эта игра привлекает людей всех возрастов и уровней игрового опыта благодаря своей простоте и увлекательности. В данной курсовой работе представлена реализация логической игры "Крестики-нолики" в виде приложения, которое позволяет двум игрокам соревноваться на неограниченном поле.

Цель игры состоит в том, чтобы построить непрерывную линию из пяти или более фишек (крестиков или ноликов) по горизонтали, вертикали или диагонали. Приложение предлагает два режима игры: первый - игрок, который первым построит линию, выигрывает, а второй - игрок, который наберет больше очков за определенное количество времени, побеждает. Во втором режиме количество построенных линий не ограничено, и каждая линия определенной длины приносит игроку одно очко.

Приложение обеспечивает возможность начать новую игру на чистом поле и проверяет соответствие действий игроков правилам игры, а также условия окончания игры. Для удобства пользователей, они могут выбрать, чтобы один из игроков был компьютером. Важным аспектом реализации игры является ограничение расстояния между фишками, чтобы избежать образования изолированных игр на поле и излишнего увеличения его размера. Это расстояние устанавливается на максимально допустимое значение, не превышающее пять.

В данной курсовой работе будет рассмотрена реализация приложения "Крестики-нолики", его основные функциональные и интерфейсные возможности, а также алгоритмы, используемые для определения победителя и обеспечения соблюдения правил игры. Более того, будут рассмотрены аспекты проектирования и реализации программного кода, обеспечивающие эффективное функционирование игры и удовлетворение потребностей пользователей.

Реализация логической игры "Крестики-нолики" является актуальной задачей, поскольку она позволяет применить различные алгоритмы и структуры данных, а также разработать понятный и удобный интерфейс для пользователей. Данное приложение может быть использовано как развлекательное средство, а также для развития логического мышления и стратегического мышления у игроков.

# ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

## **Описание поставленной задачи**

Приложение является реализацией известной логической игры “Крестики-нолики”. В данной реализации предусматривается игра двух игроков на неограниченном поле. Цель игры построить непрерывную линию из пяти или более фишек (крестиков или ноликов) по горизонтали, вертикали или диагонали. Возможны два режима игры, когда выигрывает тот, кто первым построит линию, или кто больше наберет очков за определенное количество времени. Во втором случае количество построенных линий не ограничено, каждая линия заданной длины приносит игроку одно очко. Приложение должно обеспечивать начало новой игры на чистом поле, а также проверку соответствия действий игроков правилам игры и условия окончания игры. Роль одного из игроков (по выбору пользователя) может выполнять компьютер. Для исключения возможности образования изолированных игр на одном поле и неоправданного увеличения размера поля следует установить максимально допустимое расстояние (не более пяти) от уже существующих фишек до новой, размещаемой игроком.

Результаты курсовой работы:

Программный продукт, решающий поставленные задачи.

Пояснительная записка, содержащая:

- задание на курсовую работу;

- анализ предметной области;

- постановку задачи;

- описание выделенных сущностей, их атрибутов и операций, определенных над ними;

- диаграмму классов;

- описание наиболее интересных фрагментов программы;

- результаты тестирования программы.

Программа должна содержать:

- интерфейсные классы;

- абстрактный класс;

- реальный класс или набор реальных классов, позволяющих решить поставленную задачу;

- события и обработчики событий;

- обработчики чрезвычайных ситуаций.

## **Описание предметной области**

Предметная область логической игры "Крестики-нолики" включает в себя игровое поле и правила, которые определяют способ игры и условия победы.

Игровое поле в данной реализации является неограниченным и представляет собой бесконечную сетку клеток. Каждая клетка может быть пустой или содержать символ "Х" (крестик) или "О" (нолик). Игроки ходят по очереди, ставя свои символы в свободные клетки поля.

Цель игры состоит в том, чтобы построить непрерывную линию из пяти или более своих символов (крестиков или ноликов) по горизонтали, вертикали или диагонали. Это отличается от стандартных "Крестиков-ноликов", где требуется построить линию из трех символов.

В реализации предусмотрены два режима игры. В первом режиме побеждает игрок, который первым построит линию из пяти или более символов. Во втором режиме игры количество построенных линий не ограничено, и каждая фишка в линии приносит игроку одно очко. Здесь игроки соревнуются за наибольшее количество очков за определенное количество времени.

Приложение должно обеспечивать следующие функции:

* Начало новой игры на чистом поле.
* Проверку действий игроков на соответствие правилам игры.
* Проверку условий окончания игры в соответствии с выбранным режимом (линия из пяти или более символов или истечение времени).
* Возможность выбора роли одного из игроков для компьютера.
* Установку максимально допустимого расстояния (не более пяти) от уже существующих фишек до новой, размещаемой игроком, чтобы избежать образования изолированных игр и неоправданного увеличения размера поля.

Весь функционал приложения должен соответствовать предложенным условиям игры и обеспечивать удовлетворительный игровой опыт для пользователей.

Игра "Крестики-нолики" требует от игроков логического мышления, стратегии и умения предвидеть ходы противника. Она часто используется в качестве развлекательной игры, а также в образовательных целях для развития логического мышления и принятия решений.

## **Описание выявленных сущностей**

Для реализации логической игры «Крестики-нолики» необходимо разработать следующие классы:

1. Классы для формирования игровой доски: *AbstractBoard*, *Board*:

* *AbstractBoard*: Абстрактный класс, определяющий общие свойства и методы для игровой доски. Этот класс может содержит атрибуты, такие как размер игрового поля, методы для размещения фишек и проверки статуса доски.
* *Board*: Класс, реализующий игровую доску. Он наследуется от абстрактного класса AbstractBoard и предоставляет конкретную реализацию методов для работы с игровым полем. Класс Board имеет внутреннее представление игровой доски, методы для размещения фишек на поле, проверки статуса доски (наличие победителя или ничья) и другие функции, необходимые для управления игровым процессом.

1. Классы для определения победителя:

* *IGameStatus*: Интерфейс, определяющий методы для определения статуса игры. Этот интерфейс включает методы, такие как проверка наличия победителя, определение ничьей и получение информации о текущем статусе игры.
* *GameStatus*: Класс, реализующий интерфейс IGameStatus. Он содержит логику для определения статуса игры, включая проверку наличия победителя в логической игре "Крестики-нолики". Класс GameStatus может использовать информацию о размещенных фишках на доске и правила игры для определения победы или ничьи.
* *GameStatusTime*: Класс, являющийся аналогом класса GameStatus и изменяющий логику для определения победителя на основе набранных очков за определенное количество времени. В этом классе реализованы методы для подсчета очков и определения победителя, основываясь на результате игры в заданное время.

1. Классы для реализации игры с компьютером:

* *IPlayer*: Интерфейс, определяющий методы для игрока в игре "Крестики-нолики". Этот интерфейс включает методы для совершения хода, выбора клетки на доске и получения информации о текущем состоянии игры.
* ComputerPlayer: Класс, реализующий интерфейс IPlayer и представляющий компьютерного игрока. Этот класс содержит логику для автоматического выбора ходов компьютером на основе заданных правил и алгоритмов.

1. Классы для проверки соблюдения правил игры:

* *FilledCellException*: Класс исключения, который может возникнуть при попытке размещения фишки в уже занятую клетку на игровой доске. Это исключение может быть выброшено для информирования о недопустимом ходе игрока.
* *OutOf5RangeCellsException*: Класс исключения, который может возникнуть при попытке разместить фишку на расстоянии более пяти клеток от уже существующих фишек на доске. Это исключение может быть использовано для предотвращения образования изолированных игр или неправильных ходов.

1. Классы для реализации интерфейса и взаимодействия с пользователем:

* *Program*: класс, который содержит точку входа в программу и управляет выполнением всех остальных классов.

## **Взаимодействие между сущностями**

В данной архитектуре классы взаимодействуют следующим образом:

1. Взаимодействие между классами для формирования игровой доски (*AbstractBoard*, *Board*):

* Класс *Board* создается на основе абстрактного класса *AbstractBoard*, наследуя его свойства и методы.
* *Board* может использовать методы *AbstractBoard* для размещения фишек на игровом поле, проверки статуса доски и других операций, связанных с игровой доской.

1. Взаимодействие между классами для определения победителя (*IGameStatus*, *GameStatus*, *GameStatusTime*):

* Класс *GameStatus* реализует интерфейс *IGameStatus* и предоставляет функционал для определения победителя или ничьей в игре "Крестики-нолики".
* Если используется режим игры по времени, класс *GameStatusTime* расширяет функционал *GameStatus*, добавляя логику для подсчета очков и определения победителя на основе результатов игры в заданное время.

1. Взаимодействие между классами для реализации игры с компьютером (*IPlayer*, *ComputerPlayer*):

* Класс *ComputerPlayer* реализует интерфейс *IPlayer* и представляет компьютерного игрока в игре "Крестики-нолики".
* *ComputerPlayer* может использовать методы интерфейса *IPlayer* для совершения ходов на основе заданных стратегий или алгоритмов, а также получения информации о текущем состоянии игры.

1. Взаимодействие с классами для проверки соблюдения правил игры (*FilledCellException*, *OutOf5RangeCellsException*):

* Классы FilledCellException и OutOf5RangeCellsException могут быть выброшены при возникновении ошибок в ходе игры.
* Например, при попытке размещения фишки в занятую клетку, класс FilledCellException может быть выброшен для информирования игрока о недопустимом ходе.
* Если игрок пытается разместить фишку на расстоянии более пяти клеток от уже существующих фишек, класс *OutOf5RangeCellsException* может быть выброшен для предотвращения неправильных ходов.

Взаимодействие между сущностями обеспечивает функциональность игры "Крестики-нолики", позволяя игрокам совершать ходы, размещать фишки на доске, проверять статус игры и определять победителя. Компьютерный игрок может использоваться для игры против компьютера, а исключения помогают контролировать соблюдение правил игры. Эти взаимодействия совместно обеспечивают полноценную и корректную реализацию игрового процесса "Крестики-нолики".

## **Особые ситуации**

В игре "Крестики-нолики" могут возникать различные особые ситуации, которые требуют специальной обработки. Ниже перечислены некоторые из них и описано, как на них можно реагировать:

1. Победа одного из игроков:

* Условие: если один из игроков построил непрерывную линию из пяти или более фишек (крестиков или ноликов) по горизонтали, вертикали или диагонали.
* Реакция: при обнаружении победителя необходимо остановить игру, отобразить сообщение о победе и, при необходимости, предложить начать новую игру.

1. Ничья:

* Условие: если на игровом поле заполнены все клетки, и ни один из игроков не смог построить непрерывную линию из пяти или более фишек.
* Реакция: при обнаружении ничьей необходимо остановить игру, отобразить сообщение о ничьей и, при необходимости, предложить начать новую игру.

1. Недопустимый ход игрока:

* Условие: игрок пытается разместить фишку в уже занятую клетку на игровой доске.
* Реакция: в этом случае может быть выброшено исключение FilledCellException, которое должно быть обработано для информирования игрока о недопустимом ходе. Игровой процесс должен оставаться неизменным, и игрок должен быть предупрежден о необходимости выбрать другую клетку.

1. Неправильный ход игрока:

* Условие: игрок пытается разместить фишку на расстоянии более пяти клеток от уже существующих фишек на доске.
* Реакция: в этом случае может быть выброшено исключение OutOf5RangeCellsException, которое должно быть обработано для предотвращения неправильного хода игрока. Игрок должен быть предупрежден о правилах игры и необходимости размещать фишку на допустимом расстоянии.

1. Некорректные параметры игры:

* Условие: при создании игры указаны некорректные параметры, такие как неверный размер игрового поля или неправильный режим игры.
* Реакция: в этом случае может быть выброшено соответствующее исключение или выведено сообщение об ошибке. Игра должна быть прекращена или переинициализирована с корректными параметрами.

В каждой из этих особых ситуаций необходимо предусмотреть соответствующую логику обработки, чтобы игровой процесс продолжался корректно, игрокам было показано информативное сообщение, и при необходимости предложено начать новую игру.

## **Тестирование**

* + 1. **Тестирование модуля настройки параметров игры**

Ниже приведен список проверок, который использовался для тестирования модуля авторизации:

* Попытка начала игры, не указав размер игрового поля – ожидаем получения сообщения об ошибке.
* Попытка начала игры, не выбрав используемый в игре символ - ожидаем получение сообщения об ошибке.
* Попытка указать длину собираемой линии больше, чем размер игрового поля, либо меньше нуля (при выборе 2 режима игры) – ожидаем сообщение об ошибке.
* Попытка указать время меньше нуля (при выборе 2 режима игры) - ожидаем получение сообщения об ошибке.
* Проверка перехода на главную форму после успешного запуска игры.
  + 1. **Тестирование формы игры**

Ниже приведен список проверок, который использовался для тестирования главной формы:

* Проверка попытки размещения фишки в уже занятую клетку на игровой доске.
* Проверка попытки разместить фишку на расстоянии более 5 клеток от уже существующих фишек на доске.

## **План работы**

* + 1. Изучение задания и требований к программе.
    2. Определение параметров модели и ее разработка.
    3. Реализация случайных событий.
    4. Разработка алгоритма обслуживания.
    5. Визуализация процесса.
    6. Тестирование работы модулей и классов по отдельности.
    7. Тестирование работы программы в целом.
    8. Исправление ошибок и недочетов, выявленных в процессе тестирования.
    9. Финальное тестирование программы.
    10. Написание отчета о выполненной работе и подготовка к защите курсовой работы.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

## **Диаграмма классов**

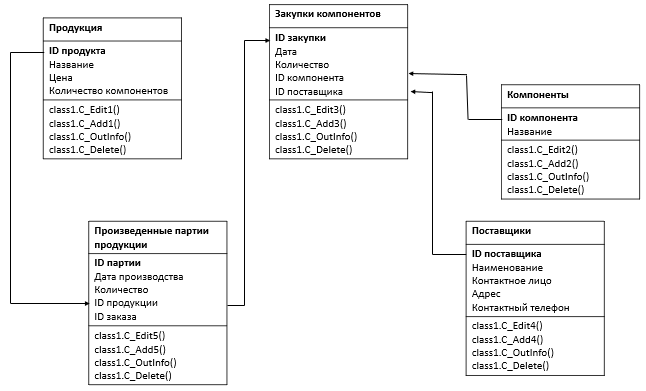


Рисунок 2.1 – Диаграмма классов

Классы "Продукция" и "Произведенные партии продукции" имеют связь "один ко многим", так как одна единица продукции может иметь несколько произведенных партий. Соответственно, у класса "Произведенные партии продукции" будет атрибут, связанный с классом "Продукция", обозначающий, какая именно продукция была произведена в данной партии.

Классы "Продукция" и "Закупки компонентов" также имеют связь "один ко многим", так как для производства одной единицы продукции может понадобиться несколько закупок компонентов. Аналогично предыдущей связи, у класса "Закупки компонентов" будет атрибут, связанный с классом "Продукция", обозначающий, какая именно продукция была произведена с использованием данных компонентов.

Классы "Закупки компонентов" и "Поставщики" имеют связь "многие ко многим", так как один поставщик может поставлять несколько компонентов, а компоненты могут закупаться у нескольких поставщиков. Для реализации этой связи потребуется создать дополнительную таблицу, которая будет связывать таблицы "Закупки компонентов" и "Поставщики".

Связь между классами "Компоненты" и "Произведенные партии продукции". Эта связь обозначает, что каждая произведенная партия продукции состоит из компонентов.

## **Интерфейсные классы**

## **Интерфейс *IGameStatus***

Интерфейс *IGameStatus* определяет методы, необходимые для проверки статуса игры.

Методы, определенные в интерфейсе *IGameStatus*, включают:

1. Метод *void CheckerRow(int i, string a)*: метод используется для проверки строки с заданным индексом *i* на наличие непрерывной линии фишек *a* (крестиков или ноликов). При обнаружении непрерывной линии должны применяться соответствующие действия для определения победителя или ничьи.
2. Метод *void CheckerColumn(int i, string a)*: метод используется для проверки столбца с заданным индексом *i* на наличие непрерывной линии фишек *a*. При обнаружении непрерывной линии должны применяться соответствующие действия для определения победителя или ничьи.
3. Метод *void CheckerDiagonalLeftToRight(string a)*: метод используется для проверки диагонали слева направо на наличие непрерывной линии фишек *a*. При обнаружении непрерывной линии должны применяться соответствующие действия для определения победителя или ничьи.
4. Метод *void CheckerDiagonalRightToLeft(string a)*: Метод используется для проверки диагонали справа налево на наличие непрерывной линии фишек *a*. При обнаружении непрерывной линии должны применяться соответствующие действия для определения победителя или ничьи.
5. Метод *void Checker()*: метод используется для общей проверки статуса игры и определения победителя или ничьи. Внутри этого метода должны вызываться другие методы *CheckerRow, CheckerColumn, CheckerDiagonalLeftToRight* и *CheckerDiagonalRightToLeft* для проверки всех возможных комбинаций фишек.
6. Метод *void Win(string a)*: Метод вызывается при обнаружении победителя игры. Он должен выполнить необходимые действия для завершения игры и отображения сообщения о победе игрока с фишками *a*.

Интерфейс *IGameStatus* определяет эти методы, которые должны быть реализованы в классе *GameStatus* и *GameStatusTime*, отвечающих за определение статуса игры в данной реализации игры "Крестики-нолики".

## **Интерфейс *IPlayer***

Интерфейс *IPlayer* определяет метод, необходимый игроку для осуществления хода.

Методы, определенные в интерфейсе *IPlayer*, включают:

1. Метод *void MakeAMove()*: Метод используется для выполнения хода игрока. Внутри этого метода игрок должен выбрать пустую клетку на игровой доске и разместить в ней свою фишку (крестик или нолик).

## **Абстрактный класс**

Абстрактный класс *AbstractBoard* представляет абстракцию игровой доски в игре. Класс содержит следующие поля:

1. Поле public Button[,] button: Двумерный массив кнопок, представляющих клетки игровой доски. Каждая кнопка может быть нажата игроком или выбрана компьютером для размещения фишки (крестика или нолика) на доске.
2. Поле *public int stepsDone*: целочисленная переменная, отслеживающая количество уже сделанных ходов на игровой доске. Используется для определения ничьей и контроля окончания игры.

## **Классы-наследники**

## **Класс Board**

Класс *Board* является наследником абстрактного класса *AbstractBoard* и представляет конкретную реализацию игровой доски в игре. Класс содержит следующие поля:

1. Поле *public event CheckEventHandler Check*: событие, которое срабатывает при необходимости проверки текущего состояния игры.
2. Поле *Form1 form*: Ссылка на объект главной формы игры.
3. Поле *ComputerPlayer player*: объект, представляющий игрока-компьютера, если режим игры с компьютером включен.
4. Поле *bool IsCompPlayer*: флаг, указывающий, включен ли режим игры с компьютером.
5. Поле *bool IsWinner*: флаг, указывающий, есть ли победитель в текущей игре.
6. Поле *int Mode*: режим игры: 0 - игра до победы по линиям, 1 - игра до набора определенного количества очков.
7. Поле int length: дополнительный параметр, который указывает количество очков, необходимое для победы в режиме игры с ограниченным временем (GameStatusTime).

Класс содержит следующие методы:

1. Метод public void OnCheck(): метод, вызываемый для выполнения проверки текущего состояния игры. В зависимости от выбранного режима игры (Mode), создается объект класса GameStatus или GameStatusTime для выполнения проверки. Если событие Check не равно null, вызывается событие для выполнения проверки.
2. Метод private void OnClick(object sender, EventArgs e): Обработчик события нажатия на кнопку игровой доски. Внутри метода происходит размещение фишки игрока в выбранной клетке и выполнение необходимых действий (смена хода, проверка состояния игры и др.). В случае возникновения исключения, выводится соответствующее сообщение об ошибке.
3. Метод private Tuple<int, int> FindButton(Button targetButton): Вспомогательный метод для поиска кнопки на игровой доске по ее объекту. Метод просматривает все кнопки на доске и возвращает координаты найденной кнопки в виде объекта Tuple<int, int>.
4. Метод public bool IsValidMove(int rowIndex, int columnIndex, string symbol): метод проверяет, является ли выбранный ход допустимым. Проверяются все направления от выбранной клетки и расстояние до других фишек с тем же символом. Возвращает true, если ход допустим, и false в противном случае.
5. public Board(int size, string Xor0Start, bool IsCompPlayer, Panel panel1, Form1 form, int gamemode, int length): конструктор класса, который инициализирует игровую доску и устанавливает необходимые параметры. Создает массив кнопок для представления игровой доски. Задает начальное значение символа (крестик или нолик) для первого хода. Устанавливает флаг IsCompPlayer для определения, играется ли с компьютером. Устанавливает режим игры и длину игры в режиме GameStatusTime. Создает объект GameStatusTime, если режим игры выбран как GameStatusTime. Инициализирует кнопки игровой доски и добавляет обработчик события нажатия OnClick для каждой кнопки.

Класс Board предоставляет реализацию игровой доски, включая взаимодействие с кнопками на доске, проверку состояния игры и управление игроком-компьютером (при наличии).

## **Класс ComputerPlayer**

Класс ComputerPlayer представляет компьютерного игрока в игре "Крестики-нолики". Он реализует интерфейс IPlayer, что означает, что компьютерный игрок обязан иметь метод MakeAMove() для совершения хода.

В методе MakeAMove() компьютерный игрок выбирает случайную свободную клетку на игровой доске. Сначала генерируются случайные значения для строки (row) и столбца (column). Затем происходит проверка, является ли выбранная клетка допустимым ходом в соответствии с правилами игры и текущим состоянием доски. Если выбранная клетка является допустимым ходом, то компьютерный игрок осуществляет свой ход, заполняя клетку символом текущего игрока (board.step), увеличивает счетчик выполненных ходов (board.stepsDone) и меняет символ текущего игрока для следующего хода. Затем происходит вызов метода OnCheck(), который инициирует проверку состояния игры. Если успешно совершен ход, переменная moveDone устанавливается в true и цикл завершается.

Конструктор класса ComputerPlayer принимает объект типа Board, представляющий игровую доску, и сохраняет его в поле board. Это позволяет компьютерному игроку взаимодействовать с игровой доской и совершать ходы на основе ее текущего состояния.

Таким образом, класс ComputerPlayer реализует логику хода компьютерного игрока в игре "Крестики-нолики".

## **Класс GameStatus**

Класс GameStatus реализует интерфейс IGameStatus и отвечает за отслеживание состояния игры и определение победителя. Он содержит приватные поля board, которое ссылается на объект Board, и form, которое ссылается на объект Form1, для взаимодействия с игровой доской и формой.

Конструктор GameStatus принимает объекты Board и Form1 и устанавливает связь между ними. Также он вызывает метод On(), чтобы подписаться на событие Check на игровой доске.

Методы On() и Off() отвечают за подписку и отписку от события Check на игровой доске соответственно.

Метод Checker() является главным методом класса, который выполняет проверку состояния игры. Он последовательно вызывает вспомогательные методы CheckerRow(), CheckerColumn(), CheckerDiagonalLeftToRight() и CheckerDiagonalRightToLeft() для проверки выигрышных комбинаций строк, столбцов и диагоналей. Если обнаруживается выигрышная комбинация или ничья, соответствующие действия выполняются, и в случае необходимости выводится сообщение для начала новой игры.

Метод Win() вызывается при выигрыше игрока. Он изменяет цвет клеток на игровой доске, выводит сообщение о победе игрока и предлагает начать новую игру.

Методы CheckerRow(), CheckerColumn(), CheckerDiagonalLeftToRight() и CheckerDiagonalRightToLeft() выполняют проверку выигрышных комбинаций в строках, столбцах и диагоналях.

## **Класс GameStatusTime**

Класс GameStatusTime также реализует интерфейс IGameStatus и расширяет функциональность класса GameStatus для подсчета очков и учета времени в игре.

У него есть дополнительные поля pointsX и points0, которые представляют счет очков для игрока X и O соответственно. Также есть поле length, которое определяет длину линии, необходимую для выигрыша.

Метод Win() переопределен для учета очков и вывода информации о победителе и текущем счете. Он также останавливает таймер игры и предлагает начать новую игру.

Методы CheckerRow(), CheckerColumn(), CheckerDiagonalLeftToRight() и CheckerDiagonalRightToLeft() были доработаны для подсчета очков при обнаружении выигрышной комбинации.

Метод Checker() вызывает соответствующие методы проверки, а затем определяет победителя на основе количества очков и вызывает метод Win() для вывода информации о победителе.

Методы On() и Off() остаются неизменными и отвечают за подписку и отписку от события Check на игровой доске.

Конструктор GameStatusTime принимает объекты Board, Form1 и lineLength, а также вызывает соответствующие методы для установки связей и подписки на события. Параметр lineLength определяет длину линии для выигрыша.

## **События**

Делегат – это тип данных, который определяет сигнатуру метода, который может вызываться через этот делегат. Он используется для передачи методов в качестве параметров других методов или для хранения ссылок на методы в коллекциях или переменных.

Событие – это механизм в C#, который позволяет классам и объектам сообщать о происходящих в них событиях и уведомлять другие классы и объекты об этих событиях. Событие может иметь несколько обработчиков, которые вызываются при возникновении события.

Взаимодействие между классами осуществляется с помощью события Check. Классы GameStatus и GameStatusTime подписываются на это событие и выполняют свои проверки при каждом вызове события.

В классе GameStatus метод Checker выполняет проверку наличия победителя в строках, столбцах и диагоналях доски. Если победитель найден, вызывается метод Win, который изменяет цвет кнопок на доске, выводит сообщение о победе и предлагает начать игру заново. Если все ходы сделаны и победитель не найден, вызывается метод Win с параметром "Ничья".

Класс GameStatusTime выполняет аналогичные проверки, но с учетом заданной длины линии, необходимой для победы. Если победитель найден, метод Win также увеличивает очки соответствующего игрока и выводит их на экран вместе с сообщением о победе. Если ничья, то в качестве победителя указывается "Ничья".

Оба класса GameStatus и GameStatusTime подписываются на событие Check доски в конструкторах классов. Когда событие Check вызывается, соответствующие методы проверки (Checker) классов GameStatus и GameStatusTime выполняются.

Класс Form1 отвечает за отображение игровой доски и обработку событий взаимодействия с пользователем. Он создает экземпляры классов Board, GameStatus и GameStatusTime, а также содержит методы для обработки нажатий на кнопки и таймера.

**3 РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ**

**3.1 Описание программной реализации**

Рассмотрим реализацию поиска данных в таблице базы данных по названию на листинге 3.1:

public void С\_Search(DataGridView DGV, string STRSQL)

{

string strSQL;

strSQL = STRSQL;

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

SqlCommand cmd = new SqlCommand(strSQL, cn);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

DataTable table = new DataTable();

adapter.Fill(table);

DGV.DataSource = table;

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

}

}

private void button3\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT \* FROM Продукция WHERE Название LIKE \'%" + textBox3.Text.Trim() + "%\'";

class1.С\_Search(dataGridView1, strSQL);

}

Листинг 3.1­ – Реализация запроса на поиск данных по названию

Рассмотрим реализацию общего поиска данных в таблице базы данных на листинге 3.2:

private void button2\_Click(object sender, EventArgs e)

{

string strSQL;

strSQL = "SELECT [ID продукта], Название, Цена, [Количество компонентов] FROM Продукция " +

"WHERE CONCAT([ID продукта], Название, Цена, [Количество компонентов]) like '%" + textBox1.Text + "%'";

class1.С\_Search(dataGridView1,strSQL);

}

Листинг 3.2­ – Реализация запроса на общий поиск данных

Рассмотрим реализацию редактирование данных в таблице базы данных на листинге 3.3:

public void C\_Edit1(DataGridView DGV, string NameTable, string Key, TextBox textBox2, TextBox textBox3, TextBox textBox4, string column1, string column2, string column3)

{

String strSQL1 = "UPDATE " + NameTable + " SET " + column1 + "=@1," + column2 + "=@2," + column3 + "=@3 WHERE " + Key + "="

+ DGV.CurrentRow.Cells[0].Value;

using (cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

SqlCommand sqlCommand = new SqlCommand(strSQL1, cn);

sqlCommand.Parameters.AddWithValue("@1", textBox2.Text);

sqlCommand.Parameters.AddWithValue("@2", Convert.ToInt32(textBox3.Text));

sqlCommand.Parameters.AddWithValue("@3", Convert.ToInt32(textBox4.Text));

sqlCommand.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Редактирование прошло успешно!");

}

}

private void button3\_Click(object sender, EventArgs e) // редактирование

{

class1.C\_Edit1(dataGridView1, "Продукция", "[ID продукта]", textBox2, textBox3, textBox4, "Название", "Цена", "[Количество компонентов]");

}

Листинг 3.3 – Реализация запроса по редактированию выбранной строки

Рассмотрим реализацию добавления данных в таблице базы данных на листинге 3.4:

public void C\_Add(DataGridView DGV, string NameTable, string Key, TextBox textBox2, TextBox textBox3, TextBox textBox4, string column2, string column3, string column4) // добавление

{

using (SqlConnection cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

string sqlLastID = "SELECT MAX(" + Key + ") FROM " + NameTable;

SqlCommand cmd = new SqlCommand(sqlLastID, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

int ID = 0;

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string a = reader[0].ToString();

ID = (int)reader[0];

}

reader.Close();

string sqlInsert = "INSERT INTO " + NameTable + " (" + Key + ", " + column2 + ", " + column3 + ", " + column4 + ") VALUES (@ID, @Column2, @Column3, @Column4)";

cmd = new SqlCommand(sqlInsert, cn);

cmd.Parameters.AddWithValue("@ID", ID + 1);

cmd.Parameters.AddWithValue("@Column2", textBox2.Text);

cmd.Parameters.AddWithValue("@Column3", Convert.ToInt32(textBox3.Text));

cmd.Parameters.AddWithValue("@Column4", Convert.ToInt32(textBox4.Text));

cmd.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Добавление прошло успешно!");

}

}

private void button12\_Click(object sender, EventArgs e)

{

class1.C\_Add(dataGridView1, "Продукция", "[ID продукта]", textBox2, textBox3, textBox4, "Название", "Цена", "[Количество компонентов]");

}

Листинг 3.4 – Реализация запроса на добавление строки в таблицу

Рассмотрим реализацию удаления данных из базы данных на листинге 3.5:

public void C\_Delete(string NameTable, string Key, TextBox textBox) // удаление

{

if (!string.IsNullOrEmpty(textBox.Text))

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

string strSQL1 = "SELECT" + Key + "FROM" + NameTable + "WHERE" + Key + " = " + Convert.ToInt32(textBox.Text);

SqlCommand cmd = new SqlCommand(strSQL1, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string strSQL = "DELETE FROM" + NameTable + "WHERE" + Key + "=@1";

cmd = new SqlCommand(strSQL, cn);

cmd.Parameters.AddWithValue("@1", Convert.ToInt32(textBox.Text));

DialogResult dialog = MessageBox.Show("Вы уверены, что хотите удалить запись?", "", MessageBoxButtons.OKCancel);

if (dialog == DialogResult.OK)

{

reader.Close();

cmd.ExecuteNonQuery();

MessageBox.Show("Запись успешно удалена");

}

}

else

{

MessageBox.Show("Такого вызова не существует");

}

reader.Close();

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}}

else

{

MessageBox.Show("Заполните все поля!");

}}

private void button2\_Click(object sender, EventArgs e) // удаление

{

class1.C\_Delete(" Продукция ", " [ID продукта] ", textBox24);

}

Листинг 3.5 – Реализация запроса на удаление выбранной строки из таблицы

Рассмотрим реализацию просмотра данных из базы данных на листинге 3.6:

public void C\_OutInfo(DataGridView DGV, string nameTable) //просмотр

{

strSQL = "SELECT \* FROM " + nameTable;

using (cn = new SqlConnection(bldr.ConnectionString))

{try

{

cn.Open();

SqlCommand cmd = new SqlCommand(strSQL, cn);

SqlDataReader rdr = cmd.ExecuteReader();

DataTable t = new DataTable();

t.Load(rdr);

DGV.DataSource = t.DefaultView;

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}}}

private void button1\_Click(object sender, EventArgs e)

{

string nameTable = "";

switch (comboBox1.SelectedIndex)

{

case 0: nameTable = "Продукция"; break;

case 1: nameTable = "Компоненты"; break;

case 2: nameTable = "[Закупки компонентов]"; break;

case 3: nameTable = "[Произведенные партии продукции]"; break;

case 4: nameTable = "Поставщики"; break;

}

class1.C\_OutInfo(dataGridView1, nameTable);

}

Листинг 3.6 – Реализация запроса на просмотр таблицы

**3.2 Тестирование отдельных модулей программы**

На рисунке 3.2.1 представлена форма «Просмотр данных». После входа на форму пользователь имеет право просмотреть информацию выбранной таблицы (листинг 3.7).

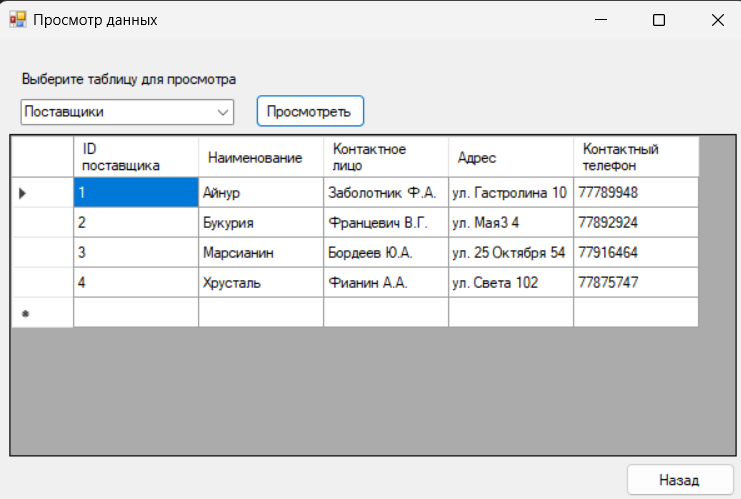


Рисунок 3.2.1 – Форма «Просмотр данных»

public void C\_OutInfo(DataGridView DGV, string nameTable) //просмотр

{

strSQL = "SELECT \* FROM " + nameTable;

using (cn = new SqlConnection(bldr.ConnectionString))

{try

{

cn.Open();

SqlCommand cmd = new SqlCommand(strSQL, cn);

SqlDataReader rdr = cmd.ExecuteReader();

DataTable t = new DataTable();

t.Load(rdr);

DGV.DataSource = t.DefaultView;

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}}}

private void button1\_Click(object sender, EventArgs e)

{

string nameTable = "";

switch (comboBox1.SelectedIndex)

{

case 0: nameTable = "Продукция"; break;

case 1: nameTable = "Компоненты"; break;

case 2: nameTable = "[Закупки компонентов]"; break;

case 3: nameTable = "[Произведенные партии продукции]"; break;

case 4: nameTable = "Поставщики"; break;

}

class1.C\_OutInfo(dataGridView1, nameTable);

}

Листинг 3.7 – Реализация просмотра

На рисунке 3.2.2 представлена форма «Удалить, изменить, добавить». После входа на форму пользователь имеет право добавить информацию в таблицу (листинг 3.8).

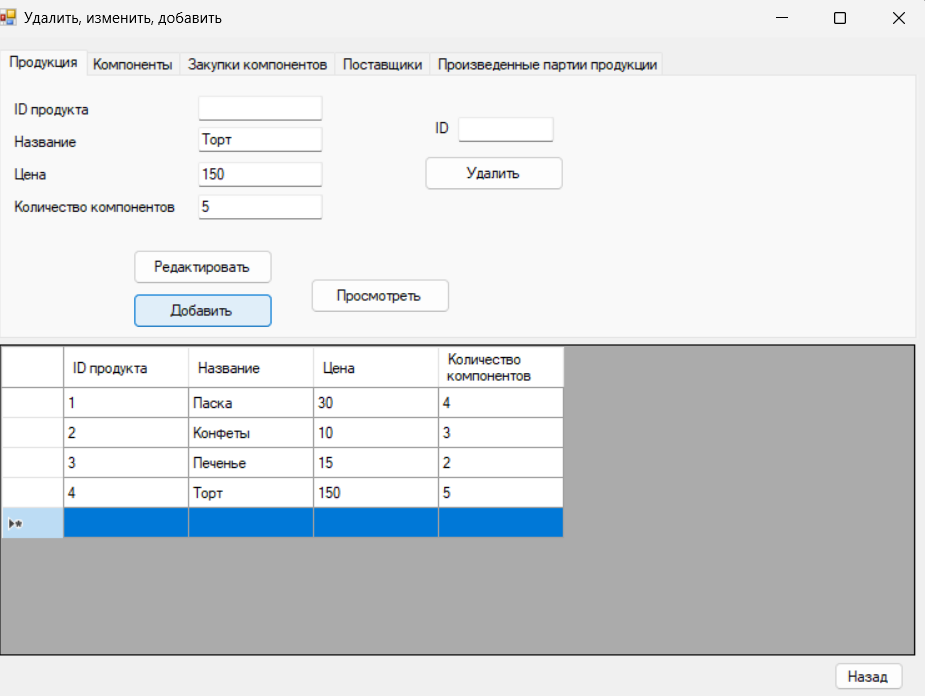


Рисунок 3.2.2 – Форма «Удалить, изменить, добавить»

public void C\_Add(DataGridView DGV, string NameTable, string Key, TextBox textBox2, TextBox textBox3, TextBox textBox4, string column2, string column3, string column4) // добавление

{

using (SqlConnection cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

string sqlLastID = "SELECT MAX(" + Key + ") FROM " + NameTable;

SqlCommand cmd = new SqlCommand(sqlLastID, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

int ID = 0;

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string a = reader[0].ToString();

ID = (int)reader[0];

}

reader.Close();

string sqlInsert = "INSERT INTO " + NameTable + " (" + Key + ", " + column2 + ", " + column3 + ", " + column4 + ") VALUES (@ID, @Column2, @Column3, @Column4)";

cmd = new SqlCommand(sqlInsert, cn);

cmd.Parameters.AddWithValue("@ID", ID + 1);

cmd.Parameters.AddWithValue("@Column2", textBox2.Text);

cmd.Parameters.AddWithValue("@Column3", Convert.ToInt32(textBox3.Text));

cmd.Parameters.AddWithValue("@Column4", Convert.ToInt32(textBox4.Text));

cmd.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Добавление прошло успешно!");

}

}

private void button12\_Click(object sender, EventArgs e)

{

class1.C\_Add(dataGridView1, "Продукция", "[ID продукта]", textBox2, textBox3, textBox4, "Название", "Цена", "[Количество компонентов]");

}

Листинг 3.8 – Реализация добавления строки

На рисунке 3.2.3 представлена форма «Удалить, изменить, добавить». После входа на форму пользователь имеет право редактировать информацию в таблицу (листинг 3.9).

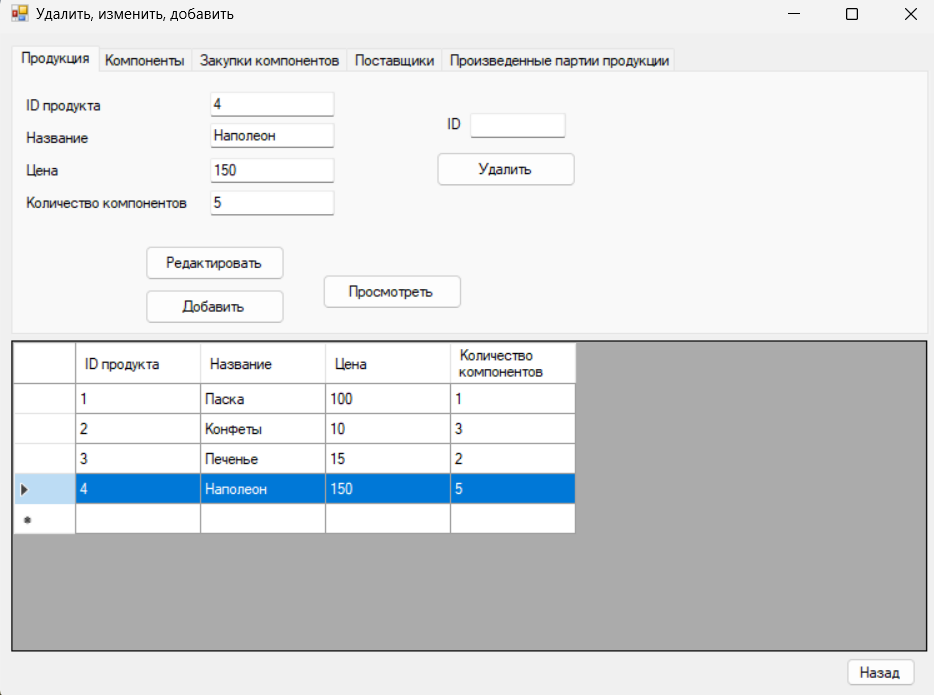


Рисунок 3.2.3 – Форма «Удалить, изменить, добавить»

public void C\_Edit1(DataGridView DGV, string NameTable, string Key, TextBox textBox2, TextBox textBox3, TextBox textBox4, string column1, string column2, string column3)

{

String strSQL1 = "UPDATE " + NameTable + " SET " + column1 + "=@1," + column2 + "=@2," + column3 + "=@3 WHERE " + Key + "="

+ DGV.CurrentRow.Cells[0].Value;

using (cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

SqlCommand sqlCommand = new SqlCommand(strSQL1, cn);

sqlCommand.Parameters.AddWithValue("@1", textBox2.Text);

sqlCommand.Parameters.AddWithValue("@2", Convert.ToInt32(textBox3.Text));

sqlCommand.Parameters.AddWithValue("@3", Convert.ToInt32(textBox4.Text));

sqlCommand.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Редактирование прошло успешно!");

}

}

private void button3\_Click(object sender, EventArgs e) // редактирование

{

class1.C\_Edit1(dataGridView1, "Продукция", "[ID продукта]", textBox2, textBox3, textBox4, "Название", "Цена", "[Количество компонентов]");

}

Листинг 3.9 – Реализация редактирования

На рисунке 3.2.4 представлена форма «Удалить, изменить, добавить». После входа на форму пользователь имеет право удалять информацию в таблицу (листинг 3.10).

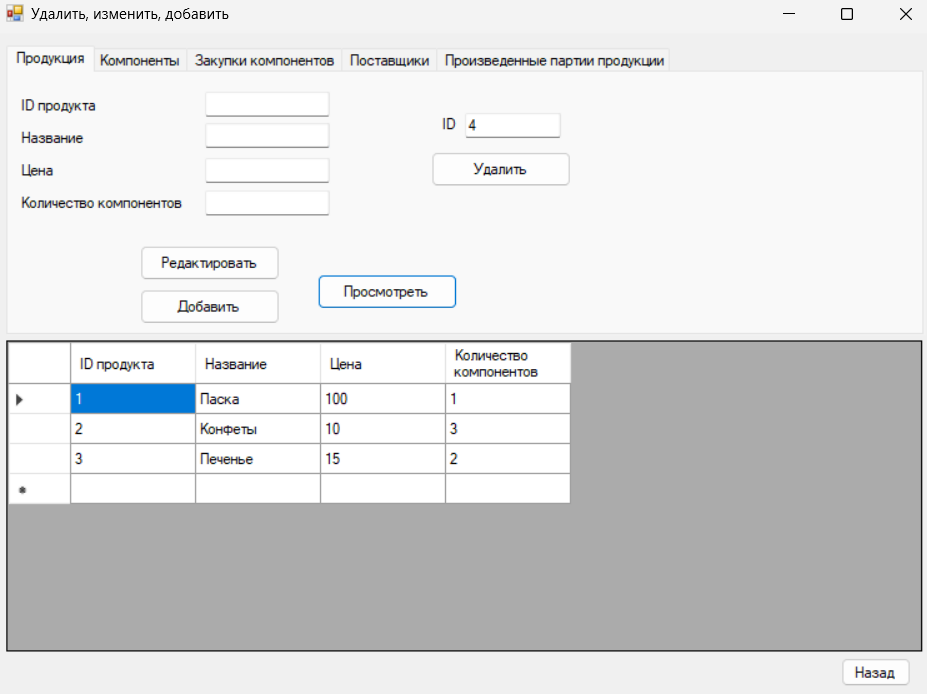


Рисунок 3.2.4 – Форма «Удалить, изменить, добавить»

public void C\_Delete(string NameTable, string Key, TextBox textBox) // удаление

{

if (!string.IsNullOrEmpty(textBox.Text))

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

string strSQL1 = "SELECT" + Key + "FROM" + NameTable + "WHERE" + Key + " = " + Convert.ToInt32(textBox.Text);

SqlCommand cmd = new SqlCommand(strSQL1, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string strSQL = "DELETE FROM" + NameTable + "WHERE" + Key + "=@1";

cmd = new SqlCommand(strSQL, cn);

cmd.Parameters.AddWithValue("@1", Convert.ToInt32(textBox.Text));

DialogResult dialog = MessageBox.Show("Вы уверены, что хотите удалить запись?", "", MessageBoxButtons.OKCancel);

if (dialog == DialogResult.OK)

{

reader.Close();

cmd.ExecuteNonQuery();

MessageBox.Show("Запись успешно удалена");

}}

else

{

MessageBox.Show("Такого вызова не существует");

}

reader.Close();

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}}

else

{ MessageBox.Show("Заполните все поля!");

}}

private void button2\_Click(object sender, EventArgs e) // удаление

{

class1.C\_Delete(" Продукция ", " [ID продукта] ", textBox24); }

Листинг 3.10 – Реализация удаления

На рисунке 3.2.5 представлена форма «Поиск по продукции». После входа на форму пользователь имеет право производить общий поиск (листинг 3.11).

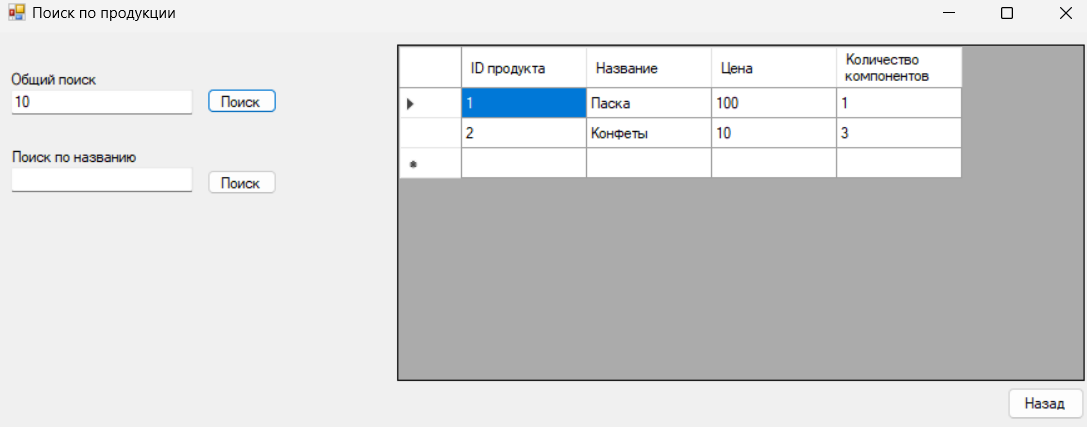


Рисунок 3.2.5 – Форма «Поиск по продукции»

private void button2\_Click(object sender, EventArgs e)

{

string strSQL;

strSQL = "SELECT [ID продукта], Название, Цена, [Количество компонентов] FROM Продукция " +

"WHERE CONCAT([ID продукта], Название, Цена, [Количество компонентов]) like '%" + textBox1.Text + "%'";

class1.С\_Search(dataGridView1,strSQL);

}

Листинг 3.11 – Реализация общего поиска

На рисунке 3.2.6 представлена форма «Поиск по продукции». После входа на форму пользователь имеет право производить поиск по названию (листинг3.12).

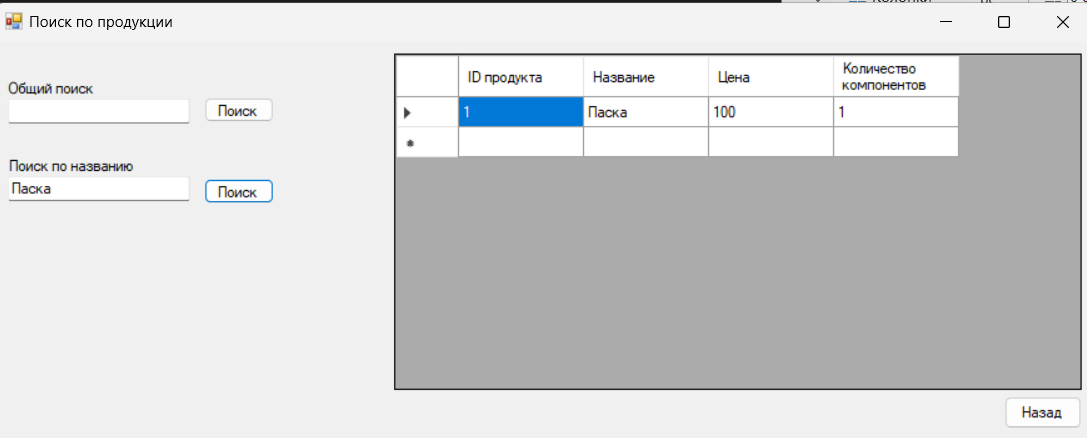


Рисунок 3.2.6 – Форма «Поиск по продукции»

public void С\_Search(DataGridView DGV, string STRSQL)

{

string strSQL;

strSQL = STRSQL;

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

SqlCommand cmd = new SqlCommand(strSQL, cn);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

DataTable table = new DataTable();

adapter.Fill(table);

DGV.DataSource = table;

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

}

}

private void button3\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT \* FROM Продукция WHERE Название LIKE \'%" + textBox3.Text.Trim() + "%\'";

class1.С\_Search(dataGridView1, strSQL);

}

Листинг 3.12 – Реализация поиска по названию

На рисунке 3.2.7 представлена форма «Запросы». После входа на форму пользователь имеет право запрашивать суммы заказов за определенный период времени (листинг 3.13).

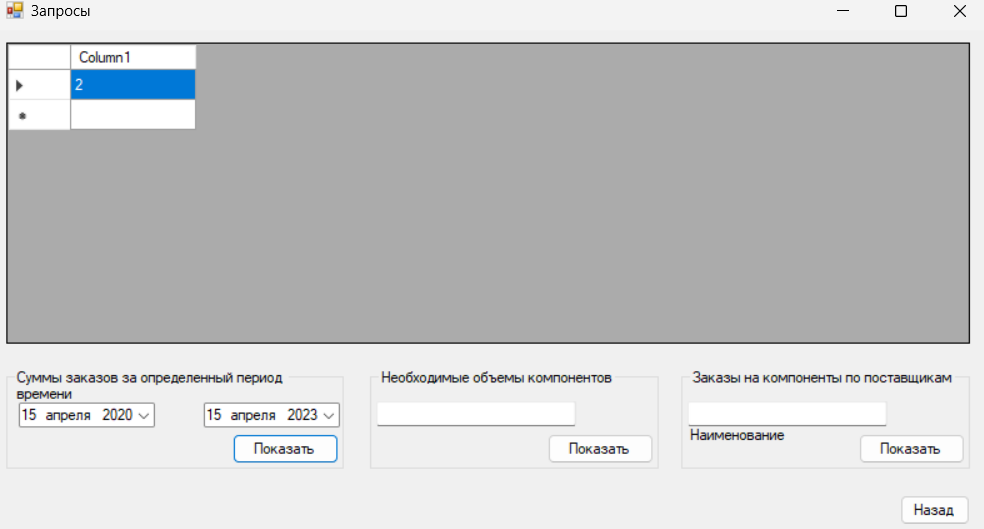


Рисунок 3.2.7 – Форма «Запросы»

public void С\_Search(DataGridView DGV, string STRSQL)

{

string strSQL;

strSQL = STRSQL;

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

SqlCommand cmd = new SqlCommand(strSQL, cn);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

DataTable table = new DataTable();

adapter.Fill(table);

DGV.DataSource = table;

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

}

}

private void button2\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT COUNT(\*) FROM [Закупки компонентов] WHERE [Дата] BETWEEN '" + dateTimePicker1.Value.Date + "' AND '" + dateTimePicker2.Value.Date + "'";

class1.С\_Search(dataGridView1, strSQL);

}

Листинг 3.13 – Реализация запроса

На рисунке 3.2.8 представлена форма «Запросы». После входа на форму пользователь имеет право запрашивать необходимые объемы компонентов (листинг3.14).

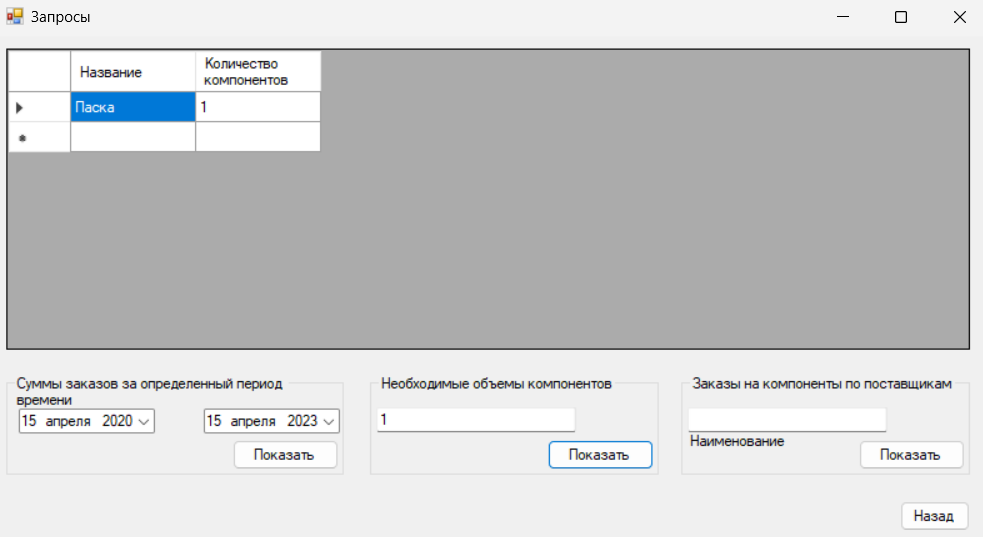


Рисунок 3.2.8 – Форма «Запросы»

private void button3\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT Название, [Количество компонентов] FROM Продукция WHERE [Количество компонентов] = '" + textBox1.Text.Trim() + "'";

class1.С\_Search(dataGridView1, strSQL);

}

Листинг 3.14 – Реализация запроса

На рисунке 3.2.9 представлена форма «Запросы». После входа на форму пользователь имеет право запрашивать заказы на компоненты по поставщикам (листинг3.15).

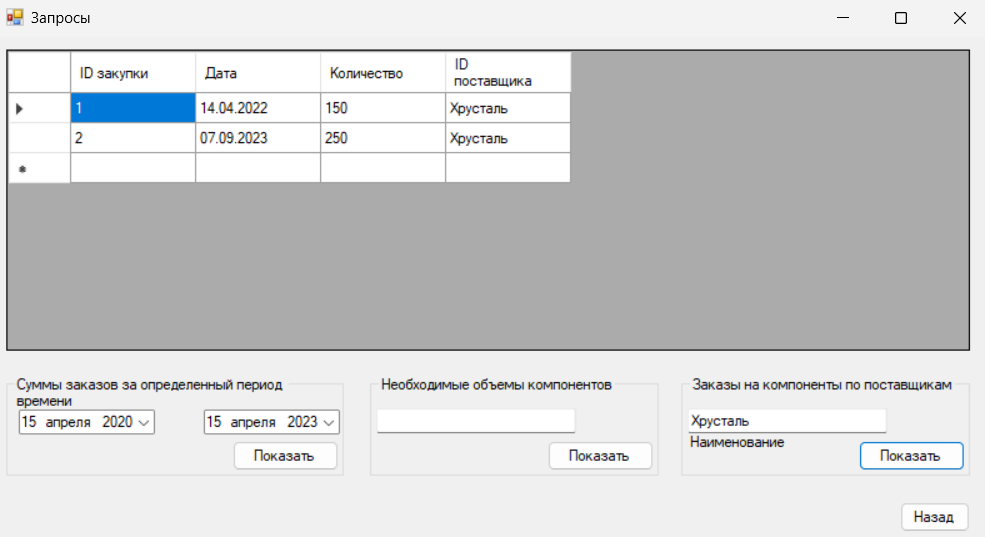


Рисунок 3.2.9 – Форма «Запросы»

private void button4\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT [ID закупки], Дата, Количество, [ID поставщика] FROM [Закупки компонентов] WHERE [ID поставщика] = '" + textBox2.Text.Trim() + "'";

class1.С\_Search(dataGridView1, strSQL);

}

Листинг 3.15 – Реализация запроса

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы была спроектирована и средствами MS SQL реализована база для ведения учета кондитерской фабрики.

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 и логическое проектирование БД средствами ER-диаграммы. Получена итоговая диаграмма БД.

Во второй главе курсовой работы была проведена разработка БД средствам MS SQL на основе полученной концептуальной 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 ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Пирогов, В.Ю. Информационные системы и базы данных: организация и проектирование: Учебное пособие / В.Ю. Пирогов. - СПб.: БХВ-Петербург, 2009. - 528 c.
2. Петцольд Ч. Программирование с использованием Microsoft Windows Forms. Мастер-класс / Пер. с англ. – М.: Русская редакция; Спб.: Питер, 2006. – 432 стр.
3. Глушаков С.В., Ломотько Д.В. Базы данных: Учебный курс. - Киев: Абрис, 2000.
4. Карпова Т.С. Базы данных: модели, разработка, реализация. - СПб.: Питер, 2001.
5. Шилдт Г. Полный справочник по С#.: Пер. с англ. — М.: Издательский дом "Вильямс", 2004. – Дата доступа: 06.12.2019.
6. Пирогов, В.Ю. Информационные системы и базы данных: организация и проектирование: Учебное пособие / В.Ю. Пирогов. - СПб.: БХВ-Петербург, 2009. - 528 c.
7. Бойко В.В.Проектирование баз данных информационных систем / Бойко В.В., Савинков В.М. - 2-е изд. - М.: Финансы и статистика, 1989. - 350с.
8. Баканов М.В., Романова В.В., Крюкова Т.П. Базы данных. Системы управления базами данных: учебное пособие / М.В. Баканов, В.В. Романова, Т.П. Крюкова; Кемеровский технологический институт пищевой промышленности. - Кемерово, 2010. – 166 с.
9. Баранчинков А.И. Алгоритмы и модели ограничения доступа к записям баз данных. – М.: Горячая линия – Телеком, 2011. – 181 с.: ил.
10. Бураков П.В., Петров В.Ю. Введение в системы баз данных: Учебное пособ. - Изд-во: СПбГУ ИТМО, 2012. – 129 с.

# Приложение А. – Руководство пользователя

**Form1.cs**

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

this.Hide();

Functions f = new Functions();

f.Show();

}

}

**Functions.cs**

public partial class Functions : Form

{

public Functions()

{

InitializeComponent();

}

private void button3\_Click(object sender, EventArgs e)

{

this.Close();

Ud\_Izm\_Dob f = new Ud\_Izm\_Dob();

f.Show();

}

private void button4\_Click(object sender, EventArgs e)

{

this.Close();

Prosmotr f= new Prosmotr();

f.Show();

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

Poisk f = new Poisk();

f.Show();

}

private void button5\_Click(object sender, EventArgs e)

{

this.Close();

Zapros f = new Zapros();

f.Show();

}

}

class Empl //класс функций

{

// Внутренние закрытые данные класса

private SqlConnectionStringBuilder bldr;

private SqlConnection cn;

private string strSQL;

private string[,] STR;

// Конструкторы

public Empl() { }

public Empl(string[,] STR)

{

this.STR = STR;

}

// Методы

public void C\_Connection()

{

bldr = new SqlConnectionStringBuilder();

bldr.DataSource = @"SASHA-PC";

bldr.IntegratedSecurity = true;

bldr.InitialCatalog = @"Кондитерская";

}

public void С\_Combo(DataGridView DGV, string NameTable, ComboBox comboBox, string ID)

{

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

string strSQL1 = "SELECT DISTINCT " + ID + " FROM " + NameTable;

SqlCommand cmd1 = new SqlCommand(strSQL1, cn);

SqlDataReader rdr1 = cmd1.ExecuteReader();

int i = 0;

while (rdr1.Read())

{

comboBox.Items.Insert(i, rdr1[ID]);

i++;

}

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

}

}

public void C\_OutInfo(DataGridView DGV, string nameTable) //просмотр

{

strSQL = "SELECT \* FROM " + nameTable;

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

SqlCommand cmd = new SqlCommand(strSQL, cn);

SqlDataReader rdr = cmd.ExecuteReader();

DataTable t = new DataTable();

t.Load(rdr);

DGV.DataSource = t.DefaultView;

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

}

}

public void C\_Delete(string NameTable, string Key, TextBox textBox) // удаление

{

if (!string.IsNullOrEmpty(textBox.Text))

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

string strSQL1 = "SELECT" + Key + "FROM" + NameTable + "WHERE" + Key + " = " + Convert.ToInt32(textBox.Text);

SqlCommand cmd = new SqlCommand(strSQL1, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string strSQL = "DELETE FROM" + NameTable + "WHERE" + Key + "=@1";

cmd = new SqlCommand(strSQL, cn);

cmd.Parameters.AddWithValue("@1", Convert.ToInt32(textBox.Text));

DialogResult dialog = MessageBox.Show("Вы уверены, что хотите удалить запись?", "", MessageBoxButtons.OKCancel);

if (dialog == DialogResult.OK)

{

reader.Close();

cmd.ExecuteNonQuery();

MessageBox.Show("Запись успешно удалена");

}

}

else

{

MessageBox.Show("Такого вызова не существует");

}

reader.Close();

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

}

else

{

MessageBox.Show("Заполните все поля!");

}

}

public void C\_Edit1(DataGridView DGV, string NameTable, string Key, TextBox textBox2, TextBox textBox3, TextBox textBox4, string column1, string column2, string column3)

{

String strSQL1 = "UPDATE " + NameTable + " SET " + column1 + "=@1," + column2 + "=@2," + column3 + "=@3 WHERE " + Key + "="

+ DGV.CurrentRow.Cells[0].Value;

using (cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

SqlCommand sqlCommand = new SqlCommand(strSQL1, cn);

sqlCommand.Parameters.AddWithValue("@1", textBox2.Text);

sqlCommand.Parameters.AddWithValue("@2", Convert.ToInt32(textBox3.Text));

sqlCommand.Parameters.AddWithValue("@3", Convert.ToInt32(textBox4.Text));

sqlCommand.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Редактирование прошло успешно!");

}

}

public void C\_Edit2(DataGridView DGV, string NameTable, string Key, TextBox textBox6, string column1)

{

String strSQL1 = "UPDATE " + NameTable + " SET " + column1 + "=@1 WHERE " + Key + "="

+ DGV.CurrentRow.Cells[0].Value;

using (cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

SqlCommand sqlCommand = new SqlCommand(strSQL1, cn);

sqlCommand.Parameters.AddWithValue("@1", textBox6.Text);

sqlCommand.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Редактирование прошло успешно!");

}

}

public void C\_Edit3(DataGridView DGV, string NameTable, string Key, DateTime date, TextBox textBox8, ComboBox comboBox1, ComboBox comboBox2, string column1, string column2, string column3, string column4)

{

String strSQL1 = "UPDATE " + NameTable + " SET " + column1 + "=@1, " + column2 + "=@2, " + column3 + "=@3, " + column4 + "=@4 WHERE " + Key + "=" + DGV.CurrentRow.Cells[0].Value;

using (cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

SqlCommand sqlCommand = new SqlCommand(strSQL1, cn);

sqlCommand.Parameters.AddWithValue("@1", date);

sqlCommand.Parameters.AddWithValue("@2", Convert.ToInt32(textBox8.Text));

sqlCommand.Parameters.AddWithValue("@3", comboBox1.SelectedItem);

sqlCommand.Parameters.AddWithValue("@4", comboBox2.SelectedItem);

sqlCommand.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Редактирование прошло успешно!");

}

}

public void C\_Edit4(DataGridView DGV, string NameTable, string Key, TextBox textBox13, TextBox textBox16, TextBox textBox15, TextBox textBox14, string column1, string column2, string column3, string column4)

{

String strSQL1 = "UPDATE " + NameTable + " SET " + column1 + "=@1," + column2 + "=@2," + column3 + "=@3,"+ column4 + "=@4 WHERE " + Key + "="

+ DGV.CurrentRow.Cells[0].Value;

using (cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

SqlCommand sqlCommand = new SqlCommand(strSQL1, cn);

sqlCommand.Parameters.AddWithValue("@1", textBox13.Text);

sqlCommand.Parameters.AddWithValue("@2", textBox16.Text);

sqlCommand.Parameters.AddWithValue("@3", textBox15.Text);

sqlCommand.Parameters.AddWithValue("@4", textBox14.Text);

sqlCommand.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Редактирование прошло успешно!");

}

}

public void C\_Edit5(DataGridView DGV, string NameTable, string Key, DateTime date, TextBox textBox22, ComboBox comboBox3, TextBox textBox9, string column1, string column2, string column3, string column4)

{

String strSQL1 = "UPDATE " + NameTable + " SET " + column2 + "=@2, " + column3 + "=@3, " + column4 + "=@4 WHERE " + Key + "=" + DGV.CurrentRow.Cells[0].Value;

using (cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

SqlCommand sqlCommand = new SqlCommand(strSQL1, cn);

sqlCommand.Parameters.AddWithValue("@1", date);

sqlCommand.Parameters.AddWithValue("@2", Convert.ToInt32(textBox22.Text));

sqlCommand.Parameters.AddWithValue("@3", comboBox3.SelectedItem);

sqlCommand.Parameters.AddWithValue("@4", Convert.ToInt32(textBox9.Text));

sqlCommand.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Редактирование прошло успешно!");

}

}

public void C\_Add(DataGridView DGV, string NameTable, string Key, TextBox textBox2, TextBox textBox3, TextBox textBox4, string column2, string column3, string column4) // добавление

{

using (SqlConnection cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

string sqlLastID = "SELECT MAX(" + Key + ") FROM " + NameTable;

SqlCommand cmd = new SqlCommand(sqlLastID, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

int ID = 0;

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string a = reader[0].ToString();

ID = (int)reader[0];

}

reader.Close();

string sqlInsert = "INSERT INTO " + NameTable + " (" + Key + ", " + column2 + ", " + column3 + ", " + column4 + ") VALUES (@ID, @Column2, @Column3, @Column4)";

cmd = new SqlCommand(sqlInsert, cn);

cmd.Parameters.AddWithValue("@ID", ID + 1);

cmd.Parameters.AddWithValue("@Column2", textBox2.Text);

cmd.Parameters.AddWithValue("@Column3", Convert.ToInt32(textBox3.Text));

cmd.Parameters.AddWithValue("@Column4", Convert.ToInt32(textBox4.Text));

cmd.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Добавление прошло успешно!");

}

}

public void C\_Add2(DataGridView DGV, string NameTable, string Key, TextBox textBox6, string column1) // добавление

{

using (SqlConnection cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

string sqlLastID = "SELECT MAX(" + Key + ") FROM " + NameTable;

SqlCommand cmd = new SqlCommand(sqlLastID, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

int ID = 0;

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string a = reader[0].ToString();

ID = (int)reader[0];

}

reader.Close();

string sqlInsert = "INSERT INTO " + NameTable + " (" + Key + ", " + column1 + ") VALUES (@ID, @1)";

cmd = new SqlCommand(sqlInsert, cn);

cmd.Parameters.AddWithValue("@ID", ID + 1);

cmd.Parameters.AddWithValue("@1", textBox6.Text);

cmd.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Добавление прошло успешно!");

}

}

public void C\_Add3(DataGridView DGV, string NameTable, string Key, TextBox textBox12, DateTime date, TextBox textBox8, ComboBox comboBox1, ComboBox comboBox2, string column1, string column2, string column3, string column4) // добавление

{

using (SqlConnection cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

string sqlLastID = "SELECT MAX(" + Key + ") FROM " + NameTable;

SqlCommand cmd = new SqlCommand(sqlLastID, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

int ID = 0;

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string a = reader[0].ToString();

ID = (int)reader[0];

}

reader.Close();

string sqlInsert = "INSERT INTO " + NameTable + " (" + Key + ", " + column1 + ", " + column2 + ", " + column3 + ", " + column4 + ") VALUES (@ID, @1, @2, @3, @4)";

cmd = new SqlCommand(sqlInsert, cn);

cmd.Parameters.AddWithValue("@ID", ID + 1);

cmd.Parameters.AddWithValue("@1", date);

cmd.Parameters.AddWithValue("@2", Convert.ToInt32(textBox8.Text));

cmd.Parameters.AddWithValue("@3", comboBox1.SelectedItem);

cmd.Parameters.AddWithValue("@4", comboBox2.SelectedItem);

cmd.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Добавление прошло успешно!");

}

}

public void C\_Add4(DataGridView DGV, string NameTable, string Key, TextBox textBox13, TextBox textBox16, TextBox textBox15, TextBox textBox14, string column1, string column2, string column3, string column4) // добавление

{

using (SqlConnection cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

string sqlLastID = "SELECT MAX(" + Key + ") FROM " + NameTable;

SqlCommand cmd = new SqlCommand(sqlLastID, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

int ID = 0;

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string a = reader[0].ToString();

ID = (int)reader[0];

}

reader.Close();

string sqlInsert = "INSERT INTO " + NameTable + " (" + Key + ", " + column1 + ", " + column2 + ", " + column3 + ", " + column4 + ") VALUES (@ID, @1, @2, @3, @4)";

cmd = new SqlCommand(sqlInsert, cn);

cmd.Parameters.AddWithValue("@ID", ID + 1);

cmd.Parameters.AddWithValue("@1", textBox13.Text);

cmd.Parameters.AddWithValue("@2", textBox16.Text);

cmd.Parameters.AddWithValue("@3", textBox15.Text);

cmd.Parameters.AddWithValue("@4", textBox14.Text);

cmd.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Добавление прошло успешно!");

}

}

public void C\_Add5(DataGridView DGV, string NameTable, string Key, DateTime date, TextBox textBox22, ComboBox comboBox3, TextBox textBox9, string column1, string column2, string column3, string column4) // добавление

{

using (SqlConnection cn = new SqlConnection(bldr.ConnectionString))

{

cn.Open();

string sqlLastID = "SELECT MAX(" + Key + ") FROM " + NameTable;

SqlCommand cmd = new SqlCommand(sqlLastID, cn);

SqlDataReader reader = cmd.ExecuteReader();

reader.Read();

int ID = 0;

if (reader.HasRows && !string.IsNullOrWhiteSpace(reader[0].ToString()))

{

string a = reader[0].ToString();

ID = (int)reader[0];

}

reader.Close();

string sqlInsert = "INSERT INTO " + NameTable + " (" + Key + ", " + column1 + ", " + column2 + ", " + column3 + ", " + column4 + ") VALUES (@ID, @1, @2, @3, @4)";

cmd = new SqlCommand(sqlInsert, cn);

cmd.Parameters.AddWithValue("@ID", ID + 1);

cmd.Parameters.AddWithValue("@1", date);

cmd.Parameters.AddWithValue("@2", Convert.ToInt32(textBox22.Text));

cmd.Parameters.AddWithValue("@3", Convert.ToString(comboBox3.SelectedItem));

cmd.Parameters.AddWithValue("@4", Convert.ToInt32(textBox9.Text));

cmd.ExecuteNonQuery();

cn.Close();

MessageBox.Show("Добавление прошло успешно!");

}

}

public void С\_Search(DataGridView DGV, string STRSQL)

{

string strSQL;

strSQL = STRSQL;

using (cn = new SqlConnection(bldr.ConnectionString))

{

try

{

cn.Open();

SqlCommand cmd = new SqlCommand(strSQL, cn);

SqlDataAdapter adapter = new SqlDataAdapter(cmd);

DataTable table = new DataTable();

adapter.Fill(table);

DGV.DataSource = table;

cn.Close();

}

catch (SqlException ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

**Poisk.cs**

public partial class Poisk : Form

{

Empl class1 = new Empl(); // вызов класса

public Poisk()

{

InitializeComponent();

class1.C\_Connection(); //путь к БД

}

private void button2\_Click(object sender, EventArgs e)

{

string strSQL;

strSQL = "SELECT [ID продукта], Название, Цена, [Количество компонентов] FROM Продукция " +

"WHERE CONCAT([ID продукта], Название, Цена, [Количество компонентов]) like '%" + textBox1.Text + "%'";

class1.С\_Search(dataGridView1,strSQL);

}

private void button3\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT \* FROM Продукция WHERE Название LIKE \'%" + textBox3.Text.Trim() + "%\'";

class1.С\_Search(dataGridView1, strSQL);

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

Functions f= new Functions();

f.Show();

}

}

**Prosmotr.cs**

public partial class Prosmotr : Form

{

Empl class1 = new Empl(); // вызов класса

public Prosmotr()

{

InitializeComponent();

class1.C\_Connection(); //путь к БД

}

private void button1\_Click(object sender, EventArgs e)

{

string nameTable = "";

switch (comboBox1.SelectedIndex)

{

case 0: nameTable = "Продукция"; break;

case 1: nameTable = "Компоненты"; break;

case 2: nameTable = "[Закупки компонентов]"; break;

case 3: nameTable = "[Произведенные партии продукции]"; break;

case 4: nameTable = "Поставщики"; break;

}

class1.C\_OutInfo(dataGridView1, nameTable);

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

Functions f = new Functions();

f.Show();

}

}

**Ud\_Izm\_Dob.cs**

public partial class Ud\_Izm\_Dob : Form

{

Empl class1 = new Empl(); // вызов класса

public Ud\_Izm\_Dob()

{

InitializeComponent();

class1.C\_Connection(); //путь к БД

class1.С\_Combo(dataGridView1, "Компоненты", comboBox1, "Название");

class1.С\_Combo(dataGridView1, "Поставщики", comboBox2, "Наименование");

class1.С\_Combo(dataGridView1, "Продукция", comboBox3, "Название");

}

private void button2\_Click(object sender, EventArgs e) // удаление

{

class1.C\_Delete(" Продукция ", " [ID продукта] ", textBox24);

}

private void button5\_Click(object sender, EventArgs e)

{

class1.C\_Delete(" Компоненты ", " [ID компонента] ", textBox5);

}

private void button6\_Click(object sender, EventArgs e)

{

class1.C\_Delete(" [Закупки компонентов] ", " [ID закупки] ", textBox11);

}

private void button8\_Click(object sender, EventArgs e)

{

class1.C\_Delete(" Поставщики ", " [ID поставщика] ", textBox18);

}

private void button10\_Click(object sender, EventArgs e)

{

class1.C\_Delete(" [Произведенные партии продукции] ", " [ID партии] ", textBox19);

}

private void button3\_Click(object sender, EventArgs e) // редактирование

{

class1.C\_Edit1(dataGridView1, "Продукция", "[ID продукта]", textBox2, textBox3, textBox4, "Название", "Цена", "[Количество компонентов]");

}

private void button4\_Click(object sender, EventArgs e)

{

class1.C\_Edit2(dataGridView1, " Компоненты ", " [ID компонента] ", textBox6, " Название ");

}

private void button7\_Click(object sender, EventArgs e)

{

class1.C\_Edit3(dataGridView1, " [Закупки компонентов] ", " [ID закупки] ", DateTime.Parse(dateTimePicker1.Text), textBox8, comboBox1, comboBox2, " Дата ", " Количество ", " [ID компонента] ", " [ID поставщика] ");

}

private void button9\_Click(object sender, EventArgs e)

{

class1.C\_Edit4(dataGridView1, " Поставщики ", " [ID поставщика] ", textBox13, textBox16, textBox15, textBox14, " Наименование ", " [Контактное лицо] ", " Адрес ", " [Контактный телефон] ");

}

private void button11\_Click(object sender, EventArgs e)

{

class1.C\_Edit5(dataGridView1, " [Произведенные партии продукции] ", " [ID партии] ", DateTime.Parse(dateTimePicker2.Text), textBox22, comboBox3, textBox9, " [Дата производства] ", " Количество ", " [ID продукции] ", " [ID заказа] ");

}

private void button12\_Click(object sender, EventArgs e)

{

class1.C\_Add(dataGridView1, "Продукция", "[ID продукта]", textBox2, textBox3, textBox4, "Название", "Цена", "[Количество компонентов]");

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

Functions f= new Functions();

f.Show();

}

private void button13\_Click(object sender, EventArgs e)

{

class1.C\_Add2(dataGridView1, " Компоненты ", "[ID компонента]", textBox6, "Название");

}

private void button14\_Click(object sender, EventArgs e)

{

class1.C\_Add3(dataGridView1, " [Закупки компонентов] ", "[ID закупки]", textBox12, DateTime.Parse(dateTimePicker1.Text), textBox8, comboBox1, comboBox2, "Дата", "Количество", "[ID компонента]", " [ID поставщика] ");

}

private void button15\_Click(object sender, EventArgs e)

{

class1.C\_Add4(dataGridView1, " Поставщики ", "[ID поставщика]", textBox13, textBox16, textBox15, textBox14, "Наименование", "[Контактное лицо]", "Адрес", "[Контактный телефон]");

}

private void button16\_Click(object sender, EventArgs e)

{

class1.C\_Add5(dataGridView1, " [Произведенные партии продукции] ", "[ID партии]", DateTime.Parse(dateTimePicker2.Text), textBox22, comboBox3, textBox9, "[Дата производства]", "Количество", "[ID продукции]", "[ID заказа]");

}

private void button17\_Click\_1(object sender, EventArgs e)

{

string nameTable = "Продукция";

string strSQL = "SELECT \* FROM " + nameTable;

class1.C\_OutInfo(dataGridView1, nameTable);

}

private void button18\_Click(object sender, EventArgs e)

{

string nameTable = "Компоненты";

string strSQL = "SELECT \* FROM " + nameTable;

class1.C\_OutInfo(dataGridView1, nameTable);

}

private void button19\_Click(object sender, EventArgs e)

{

string nameTable = "[Закупки компонентов]";

string strSQL = "SELECT \* FROM " + nameTable;

class1.C\_OutInfo(dataGridView1, nameTable);

}

private void button20\_Click(object sender, EventArgs e)

{

string nameTable = "Поставщики";

string strSQL = "SELECT \* FROM " + nameTable;

class1.C\_OutInfo(dataGridView1, nameTable);

}

private void button21\_Click(object sender, EventArgs e)

{

string nameTable = "[Произведенные партии продукции]";

string strSQL = "SELECT \* FROM " + nameTable;

class1.C\_OutInfo(dataGridView1, nameTable);

}

}

**Ud\_Izm\_Dob.cs**

public partial class Zapros : Form

{

Empl class1 = new Empl();

public Zapros()

{

InitializeComponent();

class1.C\_Connection();

}

private void button2\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT COUNT(\*) FROM [Закупки компонентов] WHERE [Дата] BETWEEN '" + dateTimePicker1.Value.Date + "' AND '" + dateTimePicker2.Value.Date + "'";

class1.С\_Search(dataGridView1, strSQL);

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

Functions f= new Functions();

f.Show();

}

private void button3\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT Название, [Количество компонентов] FROM Продукция WHERE [Количество компонентов] = '" + textBox1.Text.Trim() + "'";

class1.С\_Search(dataGridView1, strSQL);

}

private void button4\_Click(object sender, EventArgs e)

{

string strSQL = "SELECT [ID закупки], Дата, Количество, [ID поставщика] FROM [Закупки компонентов] WHERE [ID поставщика] = '" + textBox2.Text.Trim() + "'";

class1.С\_Search(dataGridView1, strSQL);

}

}