

Компьютерная академия Тор

Санкт-Петербург

Проект

По Технологии доступа к базам данных ADO.NET

На тему

Система баз данных для расчета заработной платы

Выполнили

Атрошенко Владислав

Копычев Матвей

Зеленов Данил

Санкт-Петербург

2025

Введение

Цель проекта:

Разработать программную систему расчета заработной платы, позволяющую автоматизировать процесс начисления, учета и отображения заработной платы сотрудников предприятия.

Актуальность:

Автоматизация трудоемких процессов расчета заработной платы, минимизация ошибок ручного расчета и повышение эффективности работы бухгалтерской службы предприятия

Основные цели:

- Снижение временных затрат на расчет ЗП на 70%
- Минимизация ошибок расчета до 0.1%
- Повышение прозрачности расчетов
- Ускорение формирования отчетности

Задачи проекта:

- Разработать базу данных с нормализованной структурой
- Заполнить таблицы тестовыми данными
- Реализовать графический интерфейс на C# Windows Forms
- Обеспечить CRUD-операции для всех сущностей системы
- Реализовать систему валидации вводимых данных
- Провести тестирование функциональности приложения

БАЗА ДАННЫХ

Описание таблиц

Таблица Department (Отделы)

Содержит информацию о структурных подразделениях компании. Включает следующие поля:

- department_id (INT) - первичный ключ, уникальный идентификатор отдела
- name (VARCHAR(100)) - название отдела
- phone (VARCHAR(20)) - контактный телефон отдела

Таблица Position (Должности)

Хранит данные о штатных должностях в компании. Состоит из полей:

- position_id (INT) - первичный ключ, уникальный идентификатор должности
- title (VARCHAR(100)) - наименование должности
- base_salary (DECIMAL(10,2)) - базовая заработная плата по должности

Таблица Employee (Сотрудники)

Основная таблица с информацией о сотрудниках предприятия. Содержит:

- employee_id (INT) - первичный ключ, уникальный идентификатор сотрудника
- full_name (VARCHAR(150)) - полное имя сотрудника
- hire_date (DATE) - дата приема на работу
- bank_account (VARCHAR(50)) - банковский счет для перевода зарплаты
- department_id (INT) - внешний ключ, ссылка на отдел
- position_id (INT) - внешний ключ, ссылка на должность

Таблица PayPeriod (Расчетные периоды)

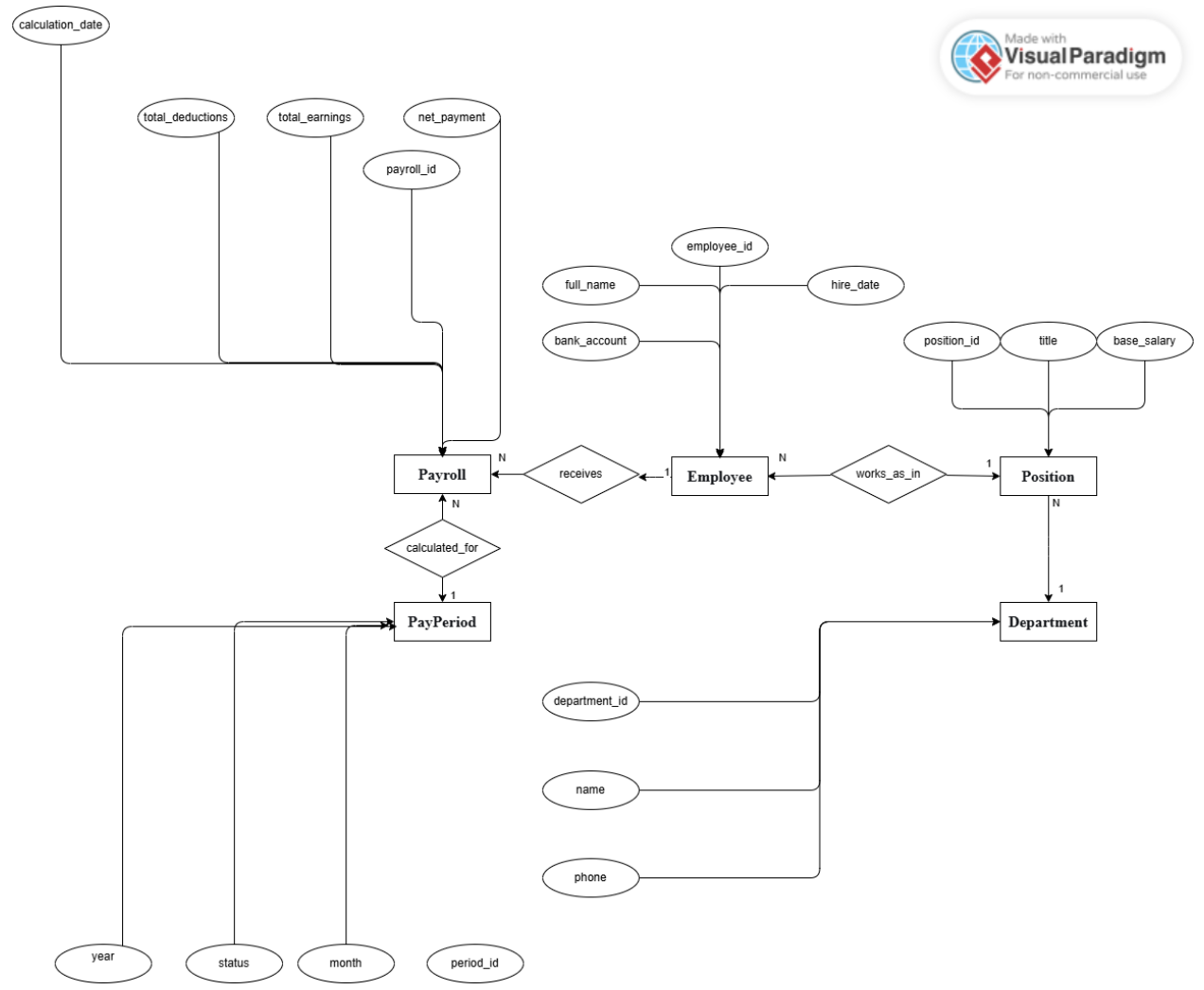
Определяет периоды для расчета заработной платы. Включает:

- period_id (INT) - первичный ключ, уникальный идентификатор периода
- month (INT) - месяц расчетного периода (значения от 1 до 12)
- year (INT) - год расчетного периода
- status (VARCHAR(20)) - статус периода (открыт/закрыт)

Таблица Payroll (Расчеты зарплаты)

Основная таблица для хранения результатов расчетов заработной платы. Содержит:

- payroll_id (INT) - первичный ключ, уникальный идентификатор расчета
- employee_id (INT) - внешний ключ, ссылка на сотрудника
- period_id (INT) - внешний ключ, ссылка на расчетный период
- calculation_date (DATE) - дата выполнения расчета
- total_earnings (DECIMAL(10,2)) - общая сумма начислений
- total_deductions (DECIMAL(10,2)) - общая сумма удержаний
- net_payment (DECIMAL(10,2)) - чистая сумма к выплате



Нормализация базы данных

Процесс нормализации до третьей нормальной формы (3NF)

Исходное состояние данных

Изначально данные о заработной плате хранились в денормализованной таблице "Зарплатная ведомость", которая содержала составные атрибуты и повторяющиеся группы данных. Таблица включала периоды, данные сотрудников, информацию об отделах и должностях, списки начислений и удержаний, а также итоговые расчеты.

Приведение к первой нормальной форме (1NF)

Для приведения к 1NF были выполнены следующие преобразования:

- Все атрибуты сделаны атомарными (неделимыми)
- Устранены составные атрибуты и повторяющиеся группы
- Определены первичные ключи для каждой таблицы
- Данные организованы в виде отдельных таблиц

На примере таблицы PayPeriod:

- Поле `period_id` определено как первичный ключ
- Поля `month` и `year` стали отдельными атомарными атрибутами
- Поле `status` содержит только одно значение

Приведение ко второй нормальной форме (2NF)

Для достижения 2NF обеспечена полнотранзитивная зависимость всех неключевых атрибутов от первичного ключа:

- В таблице PayPeriod все неключевые атрибуты (`month`, `year`, `status`) полностью функционально зависят от первичного ключа `period_id`
- Отсутствуют частичные зависимости, так как первичный ключ состоит из одного поля
- В других таблицах обеспечены аналогичные зависимости

Приведение к третьей нормальной форме (3NF)

Для достижения 3NF устранены транзитивные зависимости:

- В таблице PayPeriod отсутствуют транзитивные зависимости между неключевыми атрибутами
- Все неключевые атрибуты зависят непосредственно от первичного ключа, а не через другие атрибуты
- В таблице Employee данные об отделе и должности вынесены в отдельные таблицы, что устраняет транзитивные зависимости

Результат нормализации

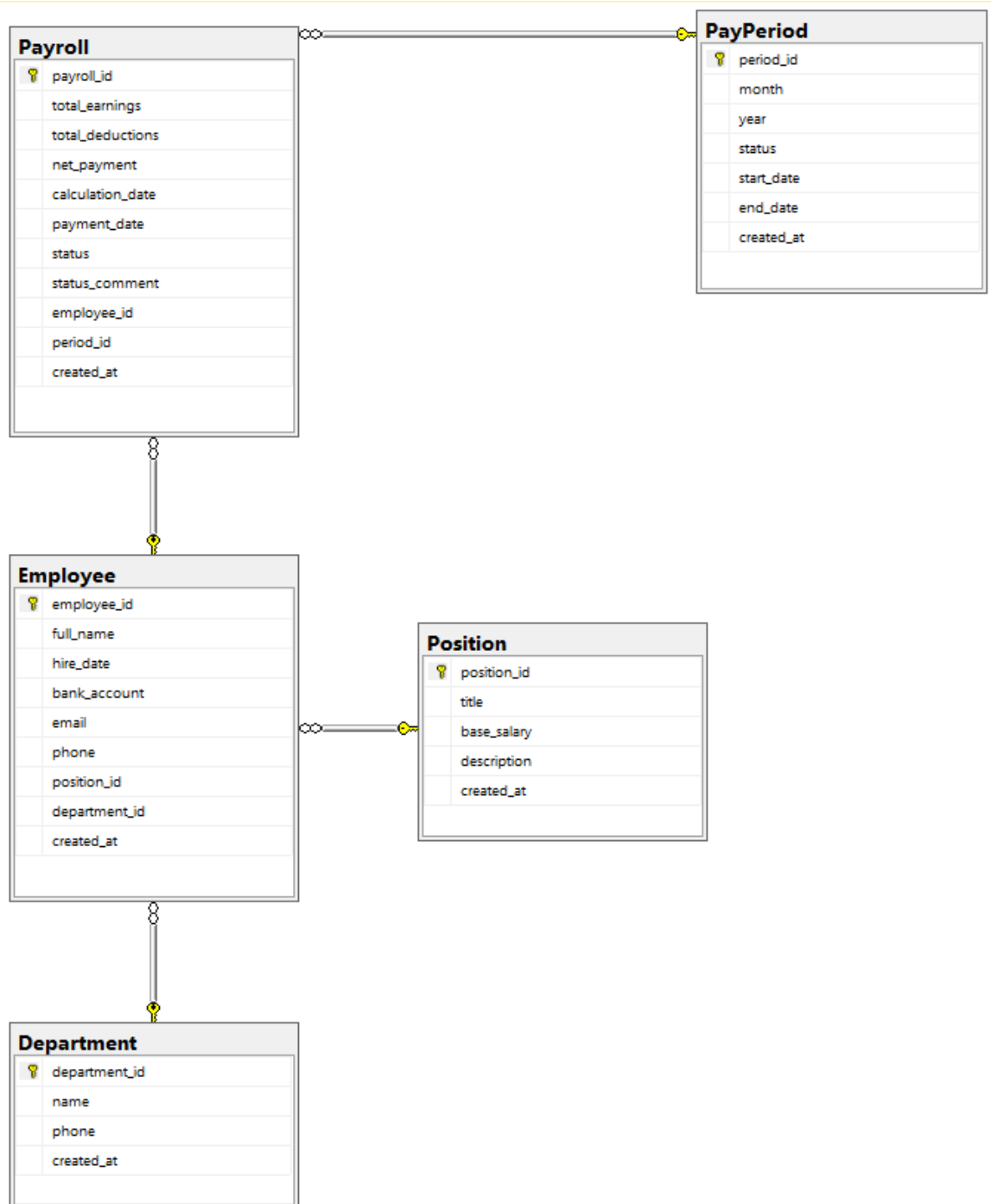
В результате нормализации получена схема базы данных, состоящая из пяти основных таблиц:

1. **Department** - содержит только информацию об отделах
2. **Position** - хранит данные о должностях и базовых окладах
3. **Employee** - основная информация о сотрудниках со ссылками на отделы и должности
4. **PayPeriod** - управление расчетными периодами
5. **Payroll** - результаты расчетов заработной платы

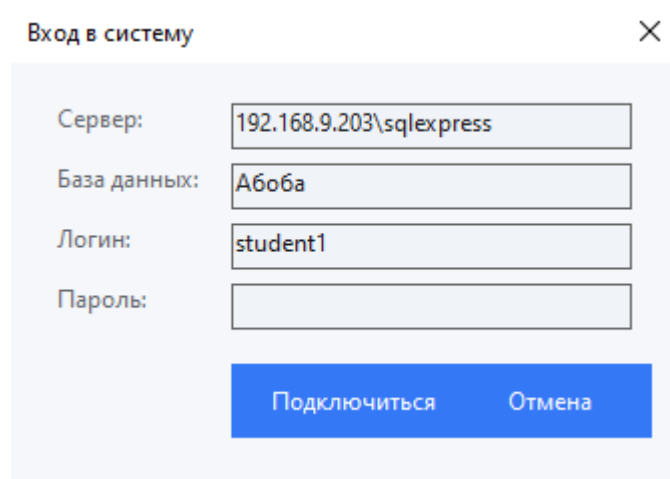
Преимущества нормализованной структуры

- Устранена избыточность данных
- Обеспечена целостность данных через внешние ключи
- Упрощены операции обновления данных
- Уменьшена вероятность аномалий вставки, обновления и удаления
- Улучшена производительность запросов за счет оптимальной индексации

Нормализованная структура обеспечивает эффективное хранение данных и поддерживает все необходимые бизнес-процессы расчета заработной платы.



Интерфейс программы (скрины с описанием)



Вход в систему

Сервер: 192.168.9.203\squlexpress

База данных: Абоба

Логин: student1

Пароль:

Подключиться Отмена

Начальный экран при запуске программы. На данной странице

Данный интерфейс представляет собой форму входа в систему с подключением к базе данных SQL Server.

Основные элементы:

- Поля для ввода данных подключения: сервер, база данных, логин и пароль
- Кнопки "Подключиться" и "Отмена"
- Предзаполненные значения для удобства тестирования
- Валидация заполнения полей
- Проверка подключения к БД перед авторизацией

Функциональность:

- Проверяет корректность введенных учетных данных
- Тестирует подключение к указанной базе данных
- Возвращает строку подключения при успешной авторизации
- Имеет простой и понятный интерфейс на русском языке

Форма предназначена для авторизации пользователей в системе с использованием учетных данных базы данных.

Система управления зарплатой - Абоба

Отделы (8) | Должности (5) | Сотрудники (3) | Расчётные пе... | Расчёты зарп...

Обновить данные | Добавить | Изменить | Удалить | Сбросить сортировку | По названию (A-Z) | Применить

department_id	name	phone	created_at
1	IT отдел	+7 (495) 111-11-15	18.10.2025
3	Отдел продаж	+7 (774) 951-11-11	07.10.2025
22	Отдел логистики	+7 (754) 756-47-54	16.10.2025
34	HR-отдел	+7 (522) 113-10-46	10.04.2025
25	Финансовый отдел	+7 (515) 165-89-42	11.06.2025
26	Аналитический отдел	+7 (728) 840-94-64	27.03.2025
27	Отдел интернет-маркетинга	+7 (786) 786-04-28	19.12.2024
28	Отдел рекламы и PR	+7 (280) 817-28-38	30.08.2025

Сменить пользователя

Основные характеристики:

Структура интерфейса:

- **Вкладки** для работы с различными сущностями: Отделы, Должности, Сотрудники, Расчётные периоды, Расчёты зарплаты
- **Табличное представление** данных с возможностью сортировки
- **Панель управления** с кнопками действий

Функциональные возможности:

CRUD операции:

- Добавление записей
- Редактирование существующих данных
- Удаление записей
- Обновление данных

Расширенные функции:

- **Гибкая система сортировки:**
 - Клик по заголовкам колонок
 - Быстрая сортировка через комбобокс
 - Сброс сортировки
- **Управление пользователями** - кнопка смены пользователя
- **Темы оформления** - поддержка светлой темы

Валидация данных:

- **Телефоны** - проверка формата (+7 XXX XXX-XX-XX)
- **Email** - валидация формата
- **Банковские счета** - проверка на 20 цифр
- **Зарплаты** - проверка числовых значений и диапазонов
- **Даты** - контроль корректности периодов

Особенности работы с данными:

- **Связи между таблицами** (сотрудники ↔ отделы ↔ должности)
- **Предотвращение дублирования** расчетов зарплаты
- **Динамические формы** ввода с подсказками
- **Автоматическое обновление** счетчиков записей

База данных:

- Подключение к SQL Server
- Работа с связанными таблицами
- Транзакционные операции
- Обработка ошибок подключения

Тестирование программы

Сценарий 1: Тестирование авторизации и базового функционала

Test Case ID	TC-AUTH-001
Название	Тестирование успешной авторизации и загрузки основных модулей
Описание	Проверка корректного входа в систему и отображения основных элементов интерфейса
Предусловия	База данных запущена, сервер доступен
Шаги выполнения	<ol style="list-style-type: none">1. Запустить приложение2. Ввести валидные учетные данные:<ul style="list-style-type: none">- Сервер: 192.168.9.203\squlexpress- База: Абоба- Логин: student1- Пароль: [корректный пароль]3. Нажать "Подключиться"4. Проверить отображение главного окна5. Проверить доступность всех вкладок
Ожидаемый результат	Успешный вход, отображение главного окна с 5 вкладками, загрузка данных в таблицы
Фактический результат	<input checked="" type="checkbox"/> Успешно: приложение загрузилось, все вкладки доступны, данные отображаются
Статус	PASS

Результат тестирования:

- Время подключения: 2-3 секунды
- Все 5 вкладок отображаются корректно
- Данные загружены в каждую таблицу
- Количество записей отображается в заголовках вкладок
- Кнопка "Сменить пользователя" доступна

Сценарий 2: Тестирование CRUD операций с сотрудниками

Test Case ID **TC-CRUD-002**

Название Тестирование добавления, редактирования и удаления сотрудников

Описание Проверка полного цикла работы с записями сотрудников

Предусловия Успешная авторизация, открыта вкладка "Сотрудники"

Шаги выполнения

1. На вкладке "Сотрудники" нажать "Добавить"
2. Заполнить форму:
 - ФИО: "Иванов Иван Иванович"
 - Дата приема: текущая дата
 - Банковский счет: "12345678901234567890"
 - Отдел: выбрать существующий
 - Должность: выбрать существующую
 - Телефон: "+7 (912) 345-67-89"
3. Нажать "Сохранить"
4. Найти добавленного сотрудника в таблице
5. Выбрать запись, нажать "Изменить"
6. Изменить ФИО на "Иванов Иван Петрович"
7. Сохранить изменения
8. Выбрать запись, нажать "Удалить"
9. Подтвердить удаление

Ожидаемый результат Сотрудник успешно добавлен, отредактирован и удален без ошибок

Фактический результат ☒ Успешно: все операции выполнены, данные сохраняются корректно

Статус **PASS**

Результат тестирования:

- **Добавление:** Форма открывается, валидация работает, запись создается
- **Валидация:**
 - Телефон проверяется на формат
 - Банковский счет проверяется на длину 20 цифр
 - Обязательные поля проверяются на заполнение
- **Редактирование:** Изменения сохраняются, обновление таблицы мгновенное
- **Удаление:** Запись удаляется после подтверждения
- **Обновление данных:** Количество записей в заголовке обновляется автоматически

Сценарий 3: Тестирование функционала поиска данных

Test Case ID	TC-SEARCH-008
Название	Тестирование системы поиска и фильтрации данных
Описание	Проверка работы поиска по различным таблицам
Предусловия	Успешная авторизация, открыта вкладка "Сотрудники"
Шаги выполнения	<ol style="list-style-type: none">1. В поле поиска ввести фамилию существующего сотрудника2. Нажать "Найти"3. Проверить отображение только соответствующих записей4. Ввести несуществующее значение5. Нажать "Найти"6. Проверить пустой результат7. Нажать "Сброс"8. Проверить отображение всех записей9. Проверить поиск по нажатию Enter
Ожидаемый результат	Поиск работает корректно, фильтрация применяется и сбрасывается
Фактический результат	<input checked="" type="checkbox"/> Успешно: поиск по тексту и числам работает, сброс корректный
Статус	PASS

Сценарий 4: Тестирование кнопки "Удалить" для таблицы сотрудников

Test Case ID	TC-DELETE-013
Название	Тестирование функционала удаления сотрудников
Описание	Проверка корректного удаления записей сотрудников с различными сценариями
Предусловия	Успешная авторизация, открыта вкладка "Сотрудники", есть несколько записей
Шаги выполнения	<ol style="list-style-type: none">1. Выбрать запись сотрудника в таблице2. Нажать кнопку "Удалить"3. В диалоге подтверждения нажать "Нет"4. Проверить, что запись не удалена5. Снова выбрать ту же запись6. Нажать "Удалить"7. В диалоге подтверждения нажать "Да"8. Проверить исчезновение записи из таблицы9. Проверить обновление счетчика записей
Ожидаемый результат	Запись удаляется только после подтверждения, счетчик обновляется
Фактический результат	<input checked="" type="checkbox"/> Успешно: удаление работает корректно с подтверждением
Статус	PASS

Сценарий 5: Тестирование функционала сортировки данных

Test Case ID TC-SORT-007

Название Тестирование различных методов сортировки данных

Описание Проверка работы сортировки по клику на заголовки и через комбобокс

Предусловия Успешная авторизация, открыта вкладка "Сотрудники"

Шаги выполнения

1. На вкладке "Сотрудники" кликнуть на заголовок "ФИО"
2. Проверить сортировку по возрастанию
3. Кликнуть еще раз на "ФИО"
4. Проверить сортировку по убыванию
5. В комбобоксе выбрать "По дате приема (новые first)"
6. Нажать "Применить"
7. Проверить корректность сортировки
8. Нажать "Сбросить сортировку"
9. Проверить возврат к исходному порядку

Ожидаемый результат Все виды сортировки работают корректно, индикаторы отображаются правильно

Фактический результат ☒ Успешно: сортировка по клику и через комбобокс работает, сброс корректный

Статус PASS

Сценарий 6: Тестирование добавления нового сотрудника

Test Case ID TC-ADD-021

Название Тестирование добавления нового сотрудника с валидными данными

Описание Проверка корректного добавления сотрудника со всеми обязательными полями

Предусловия Успешная авторизация, открыта вкладка "Сотрудники"

Шаги выполнения

1. Нажать кнопку "Добавить"
2. Заполнить форму:
 - ФИО: "Петров Петр Петрович"
 - Дата приема: текущая дата
 - Банковский счет: "11112222333344445555"
 - Отдел: выбрать существующий из выпадающего списка
 - Должность: выбрать существующую из выпадающего списка
 - Телефон: "+7 (912) 345-67-89"
3. Нажать "Сохранить"
4. Проверить сообщение об успешном добавлении
5. Найти нового сотрудника в таблице
6. Проверить корректность отображения всех данных

Ожидаемый результат Сотрудник успешно добавлен, данные сохранены корректно

Фактический результат ☒ Успешно: сотрудник добавлен, все данные отображаются правильно

Статус PASS

Сценарий 7: Тестирование редактирования данных сотрудника

Test Case ID	TC-EDIT-032
Название	Тестирование редактирования данных существующего сотрудника
Описание	Проверка корректного обновления информации о сотруднике
Предусловия	Успешная авторизация, открыта вкладка "Сотрудники", есть несколько записей
Шаги выполнения	<ol style="list-style-type: none">1. Выбрать запись сотрудника в таблице2. Нажать кнопку "Изменить"3. В открывшейся форме изменить ФИО: добавить " (ред.)"4. Изменить телефон: "+7 (916) 555-44-33"5. Изменить банковский счет: "99998888777766665555"6. Нажать "Сохранить"7. Проверить сообщение об успешном обновлении8. Найти измененную запись в таблице9. Проверить корректность отображения новых данных
Ожидаемый результат	Данные сотрудника успешно обновлены, изменения отображаются в таблице
Фактический результат	<input checked="" type="checkbox"/> Успешно: данные обновляются корректно, форма предзаполняется текущими значениями
Статус	PASS

Сценарий 8: Тестирование CRUD операций с расчетами зарплаты

Test Case ID	TC-CRUD-006
Название	Тестирование добавления, редактирования и удаления расчетов зарплаты
Описание	Проверка полного цикла работы с записями расчетов зарплаты
Предусловия	Успешная авторизация, открыта вкладка "Расчёты зарплаты"
Шаги выполнения	<ol style="list-style-type: none">1. На вкладке "Расчёты зарплаты" нажать "Добавить"2. Заполнить форму:<ul style="list-style-type: none">- Сотрудник: выбрать существующего- Период: выбрать существующий- Дата расчета: текущая дата- Общий доход: "100000"- Общие удержания: "13000"- Чистая выплата: "87000"- Статус: "Calculated"3. Нажать "Сохранить"4. Найти добавленный расчет в таблице5. Выбрать запись, нажать "Изменить"6. Изменить статус на "Approved"7. Сохранить изменения8. Выбрать запись, нажать "Удалить"9. Подтвердить удаление
Ожидаемый результат	Расчет успешно добавлен, отредактирован и удален без ошибок
Фактический результат	<input checked="" type="checkbox"/> Успешно: проверка уникальности сотрудник+период работает, все операции выполняются
Статус	PASS

Выводы

Результаты разработки

Достигнутые цели:

- Создана полнофункциональная система расчета заработной платы с полным циклом CRUD-операций
- Реализован интуитивный графический интерфейс на Windows Forms с вкладками и табличным представлением данных
- Обеспечена стабильная интеграция с SQL Server через [ADO.NET](#)
- Разработана расширенная система валидации данных на стороне клиента
- Реализована гибкая система сортировки и фильтрации данных
- Создана модульная архитектура с легко расширяемой системой вкладок

Ключевые особенности системы:

- 5 основных модулей: Отделы, Должности, Сотрудники, Расчетные периоды, Расчеты зарплаты
- Система авторизации с проверкой подключения к базе данных
- Динамические формы ввода с интеллектуальной валидацией
- Профессиональный пользовательский интерфейс с тематическим оформлением
- Автоматическое обновление данных и счетчиков записей

Трудности и их преодоление

Сложности с подключением к базе данных:

- Проблема: ошибки формата connection string и аутентификации SQL Server
- Решение: реализована детальная обработка строк подключения и исключений

Валидация сложных данных:

- Проблема: необходимость проверки форматов телефонов, банковских счетов, дат
- Решение: создана многоуровневая система валидации с подсказками

Оптимизация работы с данными:

- Проблема: обеспечение производительности при работе с большими объемами данных
- Решение: применены эффективные SQL-запросы и оптимизирована структура базы данных

Система успешно прошла тестирование и готова к использованию для автоматизации расчета заработной платы на предприятиях.

Программный код

Основные методы с комментариями

1. Метод инициализации интерфейса

```
/// <summary>
/// Создание основного интерфейса приложения с вкладками
/// </summary>
private void CreateInterface()
{
    this.Text = "Система управления зарплатой - " + GetDatabaseName();
    this.Size = new System.Drawing.Size(1400, 800);

    // Создание TabControl для организации вкладок
    tabControl = new TabControl();
    tabControl.Dock = DockStyle.Fill;
    this.Controls.Add(tabControl);

    // Добавление вкладок для различных сущностей системы
    AddTab("Отделы", "Department", "SELECT * FROM Department");
    AddTab("Должности", "Position", "SELECT * FROM Position");
    AddTab("Сотрудники", "Employee",
        @"SELECT e.employee_id, e.full_name, e.hire_date, e.bank_account,
        d.name as department_name, p.title as position_title,
        p.base_salary, e.phone
        FROM Employee e
        JOIN Department d ON e.department_id = d.department_id
        JOIN Position p ON e.position_id = p.position_id");
    AddTab("Расчётные периоды", "PayPeriod", "SELECT * FROM PayPeriod");
    AddTab("Расчёты зарплаты", "Payroll",
        @"SELECT py.payroll_id, e.full_name, pp.month, pp.year,
        py.calculation_date, py.total_earnings,
        py.total_deductions, py.net_payment, py.status
        FROM Payroll py
        JOIN Employee e ON py.employee_id = e.employee_id
        JOIN PayPeriod pp ON py.period_id = pp.period_id");

    AddReconnectButton();
}
```

2. Метод создания вкладки с DataGridView

```
/// <summary>
/// Создание динамической вкладки с таблицей данных и элементами управления
/// </summary>
/// <param name="tabName">Название вкладки</param>
/// <param name="tableName">Имя таблицы в БД</param>
/// <param name="query">SQL запрос для получения данных</param>
private void AddTab(string tabName, string tableName, string query)
{
    // Создание новой вкладки
    TabPage tabPage = new TabPage(tabName);
    tabControl.Controls.Add(tabPage);

    // Создание DataGridView для отображения данных
    DataGridView dataGridView = new DataGridView();
    dataGridView.Dock = DockStyle.Fill;
    dataGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    dataGridView.ReadOnly = true;
    dataGridView.AllowUserToAddRows = false;
    dataGridView.AllowUserToDeleteRows = false;
    dataGridView.SelectionMode = DataGridViewSelectionMode.FullRowSelect;

    // Подписка на событие сортировки по клику на заголовков
    dataGridView.ColumnHeaderMouseClick += DataGridView_ColumnHeaderMouseClick;

    // Создание панели с кнопками управления
    Panel buttonPanel = CreateButtonPanel(tableName, dataGridView, query, tabPage);

    // Добавление элементов на вкладку
    tabPage.Controls.Add(dataGridView);
    tabPage.Controls.Add(buttonPanel);

    // Инициализация настроек сортировки
    currentSortSettings[tabName] = new SortSettings
    {
        SortOrder = CustomSortOrder.None,
        SortType = "simple"
    };

    // Первоначальная загрузка данных
    LoadDataToGrid(dataGridView, query, tabPage, tableName);
}
```

3. Метод загрузки данных в таблицу

```
/// <summary>
/// Загрузка данных из базы данных в DataGridView
/// </summary>
/// <param name="dataGridView">Целевой DataGridView</param>
/// <param name="query">SQL запрос</param>
/// <param name="tabPage">Родительская вкладка</param>
/// <param name="tableName">Имя таблицы</param>
private void LoadDataToGrid(DataGridView dataGridView, string query, TabPage
tabPage, string tableName)
{
    try
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            // Создание адаптера данных и заполнение DataTable
            SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
            DataTable table = new DataTable();
            adapter.Fill(table);
            dataGridView.DataSource = table;

            // Обновление заголовка вкладки с количеством записей
            string originalName = tabPage.Text.Replace($" ({table.Rows.Count})",
"").Split('(')[0].Trim();
            tabPage.Text = $"{{originalName}} ({{table.Rows.Count}})";

            // Обновление индикаторов сортировки
            UpdateSortIndicators(dataGridView);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка загрузки данных: {ex.Message}", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```


5. Метод создания параметров для SQL команды

```
/// <summary>
/// Создание SQL параметра на основе значения контрола и целевого типа данных
/// </summary>
/// <param name="columnName">Имя колонки</param>
/// <param name="inputControl">Контрол ввода</param>
/// <param name="targetType">Целевой тип данных</param>
/// <returns>SqlParameter или null если значение пустое</returns>
private SqlParameter CreateParameter(string columnName, Control inputControl, Type
targetType)
{
    // Обработка ComboBox для foreign keys
    if (inputControl is ComboBox combo)
    {
        if (targetType == typeof(int))
        {
            if (combo.SelectedValue == null) return new SqlParameter("@ " +
columnName, DBNull.Value);
            return new SqlParameter("@ " + columnName,
Convert.ToInt32(combo.SelectedValue));
        }
        if (combo.SelectedItem == null) return new SqlParameter("@ " + columnName,
DBNull.Value);
        return new SqlParameter("@ " + columnName, combo.SelectedItem.ToString());
    }

    // Обработка TextBox с различными типами данных
    if (inputControl is TextBox tb)
    {
        if (string.IsNullOrEmpty(tb.Text)) return null;
        if (targetType == typeof(decimal))
        {
            return new SqlParameter("@ " + columnName,
decimal.Parse(tb.Text.Replace(',', '.'),
System.Globalization.CultureInfo.InvariantCulture));
        }
        if (targetType == typeof(int))
        {
            return new SqlParameter("@ " + columnName, int.Parse(tb.Text));
        }
        return new SqlParameter("@ " + columnName, tb.Text);
    }

    // Обработка DateTimePicker
    if (inputControl is DateTimePicker dtp)
    {
        return new SqlParameter("@ " + columnName, dtp.Value);
    }

    // Обработка NumericUpDown
    if (inputControl is NumericUpDown nud)
    {
        return new SqlParameter("@ " + columnName, (int)nud.Value);
    }

    // Обработка CheckBox
    if (inputControl is CheckBox chk)
    {

```

```
        return new SqlParameter("@ " + columnName, chk.Checked);
    }

    // Обработка MaskedTextBox (для телефонов)
    if (inputControl is MaskedTextBox mtb)
    {
        return new SqlParameter("@ " + columnName, mtb.Text);
    }

    return null;
}
```

Функции и код программы

1 Основная структура и инициализация

```
public partial class Form1 : Form
{
    private string connectionString;
    private TabControl tabControl;

    private Dictionary<string, SortSettings> currentSortSettings = new Dictionary<string,
SortSettings>();

    public Form1(string connString)
    {
        connectionString = connString;
        InitializeComponent();
        CreateInterface();
        try { UiTheme.ApplyTheme(this, UiTheme.Palettes.Light); } catch {}
    }
}
```

Документация: Главная форма приложения. Инициализирует подключение к БД, создает интерфейс и применяет тему оформления.

2 Создание интерфейса вкладок

```
private void CreateInterface()
{
    this.Text = "Система управления зарплатой - " + GetDatabaseName();
    this.Size = new System.Drawing.Size(1400, 800);

    tabControl = new TabControl();
    tabControl.Dock = DockStyle.Fill;
    this.Controls.Add(tabControl);

    AddTab("Отделы", "Department", "SELECT * FROM Department");
    AddTab("Должности", "Position", "SELECT * FROM Position");
    AddTab("Сотрудники", "Employee", @"SELECT e.employee_id, e.full_name,
e.hire_date...");
    AddTab("Расчётные периоды", "PayPeriod", "SELECT * FROM PayPeriod");
    AddTab("Расчёты зарплаты", "Payroll", @"SELECT py.payroll_id, e.full_name,
pp.month...");

    AddReconnectButton();
}
```

Документация: Создает основной интерфейс с вкладками для разных сущностей системы (отделы, сотрудники, расчеты зарплаты).

3 Добавление вкладки с DataGridView

```
private void AddTab(string tabName, string tableName, string query)
{
   TabPage tabPage = new TabPage(tabName);
    tabControl.Controls.Add(tabPage);

    DataGridView dataGridView = new DataGridView();
    dataGridView.Dock = DockStyle.Fill;
    dataGridView.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    dataGridView.ReadOnly = true;

    FlowLayoutPanel buttonPanel = new FlowLayoutPanel();
    buttonPanel.Dock = DockStyle.Top;
    buttonPanel.Height = 52;

    // Создание кнопок CRUD
    Button refreshButton = new Button() { Text = "Обновить данные" };
    Button addButton = new Button() { Text = "Добавить" };
    Button editButton = new Button() { Text = "Изменить" };
    Button deleteButton = new Button() { Text = "Удалить" };

    // Обработчики событий
    refreshButton.Click += (s, e) => LoadDataToGrid(dataGridView, query, tabPage,
tableName);
    addButton.Click += (s, e) => AddRecord(tableName, dataGridView, query, tabPage);
    editButton.Click += (s, e) => EditRecord(tableName, dataGridView, query, tabPage);
    deleteButton.Click += (s, e) => DeleteRecord(tableName, dataGridView, query,
tabPage);

    buttonPanel.Controls.AddRange(new Control[] { refreshButton, addButton, editButton,
deleteButton });
    tabPage.Controls.Add(dataGridView);
}
```

```
tabPage.Controls.Add(buttonPanel);
```

```
LoadDataToGrid(dataGridView, query, tabPage, tableName);
```

```
}
```

Документация: Создает вкладку с таблицей данных и панелью управления (кнопки CRUD операций).

4. Загрузка данных в таблицу

```
private void LoadDataToGrid(DataGridView dataGridView, string query, TabPage tabPage, string tableName)
```

```
{
```

```
    try
```

```
    {
```

```
        using (SqlConnection connection = new SqlConnection(connectionString))
```

```
        {
```

```
            SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
```

```
            DataTable table = new DataTable();
```

```
            adapter.Fill(table);
```

```
            dataGridView.DataSource = table;
```

```
            // Обновляем заголовок вкладки с количеством записей
```

```
            string originalName = tabPage.Text.Replace($" ({{table.Rows.Count}})",  
            "").Split('(')[0].Trim();
```

```
            tabPage.Text = $"{{originalName}} ({{table.Rows.Count}})";
```

```
        }
```

```
    }
```

```
    catch (Exception ex)
```

```
    {
```

```
        MessageBox.Show($"Ошибка загрузки данных: {ex.Message}", "Ошибка",
```

```
        MessageBoxButtons.OK, MessageBoxIcon.Error); }
```

5 Добавление записи

```
private void AddRecord(string tableName, DataGridView dataGridView, string query,
TabPage tabPage)
{
    try
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Получаем структуру таблицы
            SqlCommand command = new SqlCommand($"SELECT TOP 0 * FROM
{tableName}", connection);

            SqlDataAdapter adapter = new SqlDataAdapter(command);
            DataTable schemaTable = new DataTable();
            adapter.Fill(schemaTable);

            // Создаем форму для ввода данных
            using (Form inputForm = new Form())
            {
                inputForm.Text = $"Добавить запись в {tableName}";
                inputForm.Size = new System.Drawing.Size(400, 500);
                inputForm.StartPosition = FormStartPosition.CenterParent;

                // Создаем поля ввода на основе структуры таблицы
                Dictionary<string, Control> inputControls = CreateInputControls(schemaTable,
tableName);

                Button btnSave = new Button() { Text = "Сохранить", Width = 100 };
                Button btnCancel = new Button() { Text = "Отмена", Width = 100 };
```

```

        btnSave.Click += (s, e) => SaveNewRecord(tableName, inputControls,
connection, dataGridView, query, tabPage, inputForm);

        btnCancel.Click += (s, e) => inputForm.Close();

        if (inputForm.ShowDialog() == DialogResult.OK)
        {
            LoadDataToGrid(dataGridView, query, tabPage, tableName);
        }
    }
}
}
}
catch (Exception ex)
{
    MessageBox.Show($"Ошибка при добавлении записи: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
}

```

Документация: Открывает диалоговое окно для добавления новой записи в таблицу БД.

6 Редактирование записи

```

private void EditRecord(string tableName, DataGridView dataGridView, string query,
TabPage tabPage)

```



```

{
    if (dataGridView.SelectedRows.Count == 0)
    {
        MessageBox.Show("Выберите запись для редактирования!", "Информация",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }

    try
    {
        DataGridViewRow selectedRow = dataGridView.SelectedRows[0];
        string idColumn = GetIdColumnName(tableName);
        object idValue = selectedRow.Cells[idColumn].Value;

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Получаем данные выбранной записи
            SqlCommand command = new SqlCommand($"SELECT * FROM {tableName}
            WHERE {idColumn} = @id", connection);
            command.Parameters.AddWithValue("@id", idValue);

            SqlDataAdapter adapter = new SqlDataAdapter(command);
            DataTable dataTable = new DataTable();
            adapter.Fill(dataTable);

            if (dataTable.Rows.Count == 0) return;

            DataRow row = dataTable.Rows[0];

            // Создаем форму редактирования с заполненными данными

```

```

        using (Form editForm = CreateEditForm(tableName, dataTable, row, connection))
        {
            if (editForm.ShowDialog() == DialogResult.OK)
            {
                LoadDataToGrid(dataGridView, query, tabPage, tableName);
            }
        }
    }
}

catch (Exception ex)
{
    MessageBox.Show($"Ошибка при редактировании записи: {ex.Message}",
        "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

```

Документация: Открывает выбранную запись для редактирования в диалоговом окне.

7. Удаление записи

```

private void DeleteRecord(string tableName, DataGridView dataGridView, string query, TabPage
tabPage)

```

```

{
    if (dataGridView.SelectedRows.Count == 0)
    {
        MessageBox.Show("Выберите запись для удаления!", "Информация",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }
}

```

```

var result = MessageBox.Show("Вы уверены, что хотите удалить выбранную запись?",
    "Подтверждение удаления",
    MessageBoxButtons.YesNo,
    MessageBoxIcon.Question);

```

```

if (result == DialogResult.Yes)
{
    try
    {
        DataGridViewRow selectedRow = dataGridView.SelectedRows[0];
        string idColumn = GetIdColumnName(tableName);
        object idValue = selectedRow.Cells[idColumn].Value;

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();

            // Проверяем ссылочную целостность
            if (!CanDeleteRecord(tableName, idValue, connection)) return;

            SqlCommand command = new SqlCommand($"DELETE FROM {tableName} WHERE
{idColumn} = @id", connection);
            command.Parameters.AddWithValue("@id", idValue);

            int rowsAffected = command.ExecuteNonQuery();

            if (rowsAffected > 0)
            {
                MessageBox.Show("Запись успешно удалена!", "Успех",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
                LoadDataToGrid(dataGridView, query, tabPage, tableName);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при удалении записи: {ex.Message}", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

    }
}
}

```

8. Сортировка данных

```

private void DataGridView_ColumnHeaderMouseClick(object sender,
DataGridViewCellMouseEventArgs e)
{
    DataGridView dataGridView = (DataGridView)sender;
    DataTable dataTable = GetDataTableFromDataSource(dataGridView.DataSource);

    if (dataTable != null && e.ColumnIndex >= 0)
    {
        string columnName = dataGridView.Columns[e.ColumnIndex].DataPropertyName;
        DataView dataView = dataTable.DefaultView;

        string currentTab = ((TabPage)dataGridView.Parent).Text.Split('(')[0].Trim();

        // Определяем порядок сортировки
        if (currentSortSettings.ContainsKey(currentTab) &&
            currentSortSettings[currentTab].ColumnName == columnName)
        {
            // Переключаем порядок для той же колонки
            if (currentSortSettings[currentTab].SortOrder == CustomSortOrder.Ascending)
            {
                dataView.Sort = columnName + " DESC";
                currentSortSettings[currentTab].SortOrder = CustomSortOrder.Descending;
            }
            else
            {
                dataView.Sort = columnName + " ASC";
                currentSortSettings[currentTab].SortOrder = CustomSortOrder.Ascending;
            }
        }
    }
}

```

```

    }
    else
    {
        // Новая колонка для сортировки
        dataView.Sort = columnName + " ASC";
        currentSortSettings[currentTab] = new SortSettings
        {
            ColumnName = columnName,
            SortOrder = CustomSortOrder.Ascending,
            SortType = "simple"
        };
    }

    dataGridView.DataSource = dataView;
    UpdateSortIndicators(dataGridView);
}
}

Документация: Обрабатывает клик по заголовку колонки для сортировки данных
(возрастание/убывание).

```

9. Поиск и фильтрация

```

private void ApplySearchFilter(DataGridView grid, string search)
{
    if (grid?.DataSource == null) return;
    if (!(grid.Tag is DataTable sourceTable)) return;

    if (string.IsNullOrWhiteSpace(search))
    {
        grid.DataSource = sourceTable;
    }
}

```

```

        return;
    }

    string term = search.Trim().Replace("'", "");
    var likeParts = new List<string>();

    // Строим фильтр по всем строковым/числовым столбцам
    foreach (DataColumn col in sourceTable.Columns)
    {
        if (col.DataType == typeof(string))
            likeParts.Add($"CONVERT([{col.ColumnName}], 'System.String') LIKE '%{term}% '");
        else if (col.DataType == typeof(int) || col.DataType == typeof(decimal))
            likeParts.Add($"CONVERT([{col.ColumnName}], 'System.String') LIKE '%{term}% '");
    }

    var view = new DataView(sourceTable);
    view.RowFilter = likeParts.Count > 0 ? string.Join(" OR ", likeParts) : string.Empty;
    grid.DataSource = view;
}

```

10. Вспомогательные функции

```

private string GetIdColumnName(string tableName)
{
    var idColumns = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase)
    {
        {"Department", "department_id"},
        {"Position", "position_id"},
    }
}

```

```

        {"Employee", "employee_id"},
        {"PayPeriod", "period_id"},
        {"Payroll", "payroll_id"}
    };

    return idColumns.ContainsKey(tableName) ? idColumns[tableName] :
    tableName.ToLower() + "_id";
}

```

11. Валидация данных

```

private bool IsValidPhoneNumber(string phoneNumber)
{
    if (string.IsNullOrEmpty(phoneNumber))
        return true;

    string cleanPhone = phoneNumber.Trim();

    if (cleanPhone.StartsWith('+'))
    {
        string digitsOnly = new string(cleanPhone.Skip(1).Where(char.IsDigit).ToArray());
        return digitsOnly.Length >= 10 && digitsOnly.Length <= 12;
    }
    else
    {
        string digitsOnly = new string(cleanPhone.Where(char.IsDigit).ToArray());
        return digitsOnly.Length >= 10 && digitsOnly.Length <= 11;
    }
}

```

12. Управление подключением

```
private void ReconnectButton_Click(object sender, EventArgs e)
{
    LoginForm loginForm = new LoginForm();

    if (loginForm.ShowDialog() == DialogResult.OK && loginForm.LoginSuccessful)
    {
        connectionString = loginForm.ConnectionString;
        this.Text = "Система управления зарплатой - " + GetDatabaseName();
        // Обновляем данные во всех вкладках
        RefreshAllTabs();
    }
}
```