Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Рязанский государственный радиотехнический университет имени В.Ф. Уткина

Кафедра ЭВМ

К защите

Руководитель работы:

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

дата подпись

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

по дисциплине

**«Клиент-серверные приложения баз данных»**

На тему

«Создание клиент-серверного приложения сервиса для доставки еды»

Выполнил студент группы 245

Евдокимов А.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

дата сдачи на проверку подпись

Руководитель работы

ассистент каф. ЭВМ

Баранова С. Н. \_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

оценка дата защиты подпись

Рязань 2025

Оглавление

[Оглавление 2](#_Toc201109593)

[Введение 4](#_Toc201109594)

[Постановка задачи 4](#_Toc201109595)

[Описание автоматизируемой области 4](#_Toc201109596)

[Выделение основных функций и бизнес-правил автоматизируемой области 5](#_Toc201109597)

[Разработка архитектуры ИС 6](#_Toc201109598)

[Разработка серверной части информационной системы 7](#_Toc201109599)

[Инфологическое проектирование БД 7](#_Toc201109600)

[Даталогическое проектирование 14](#_Toc201109601)

[Разработка триггеров, ХП и представлений 20](#_Toc201109602)

[Разработка триггеров 20](#_Toc201109603)

[Разработка ХП 21](#_Toc201109604)

[Разработка представлений 26](#_Toc201109605)

[Разработка клиентского части информационной системы 27](#_Toc201109606)

[Разработка прототипа интерфейса пользователя 27](#_Toc201109607)

[Реализация клиентской части 31](#_Toc201109608)

[Создание форм Windows Forms 32](#_Toc201109609)

[Тестирование основных функций приложения 36](#_Toc201109610)

[Тестирование белым ящиком 36](#_Toc201109611)

[Тестирование черным ящиком 71](#_Toc201109612)

[Анализ классов эквивалентности 71](#_Toc201109613)

[Анализ граничных значений 74](#_Toc201109614)

[Анализ причинно-следственных связей 75](#_Toc201109615)

[ЗАКЛЮЧЕНИЕ 76](#_Toc201109616)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 77](#_Toc201109617)

[ПРИЛОЖЕНИЕ А: сценарий создания БД 78](#_Toc201109618)

[ПРИЛОЖЕНИЕ Б: сценарий создания триггеров БД 83](#_Toc201109619)

[ПРИЛОЖЕНИЕ В: сценарий создания представлений БД 87](#_Toc201109620)

[ПРИЛОЖЕНИЕ Г: сценарий создания хранимых процедур БД 88](#_Toc201109621)

[ПРИЛОЖЕНИЕ Д: сценарий заполнения таблиц БД 97](#_Toc201109622)

[ПРИЛОЖЕНИЕ Ж: сценарий тестирования хранимых процедур БД 99](#_Toc201109623)

[ПРИЛОЖЕНИЕ З: сценарий заполнения таблиц БД изображениями 102](#_Toc201109624)

[ПРИЛОЖЕНИЕ К: исходный текст клиентского приложения – класс Database 103](#_Toc201109625)

[ПРИЛОЖЕНИЕ Л: исходный текст клиентского приложения – класс ClientActions 109](#_Toc201109626)

[ПРИЛОЖЕНИЕ М: исходный текст клиентского приложения – класс Program 118](#_Toc201109627)

[ПРИЛОЖЕНИЕ Н: исходный текст клиентского приложения – класс User 119](#_Toc201109628)

[ПРИЛОЖЕНИЕ О: исходный текст клиентского приложения – класс DeliveryAppContext 120](#_Toc201109629)

**Введение**

Клиент-серверные приложения являются удобным средством взаимодействия бизнеса и клиента. Современные информационные системы позволяют совершать сложные операции с данными быстро и эффективно.

С помощью информационных систем осуществляется много видов деятельности. В данной работе будет разработана система заказов еды с помощью доставки.

В настоящее время доставка – самый удобный и быстрый способ получения желаемого товара. Всё, что нужно клиенту, желающему сделать заказ, – выбрать нужное в приложении и совершить заказ.

Приложения для доставки пищи обычно имеют информацию о меню, список заказов пользователя и информацию о пользователе.

Пользователь может сделать заказ на свой адрес и получить его от курьера.

Цель работы заключается в получение навыков разработки клиент-серверного приложения.

Задачей работы является анализ выбранной предметной области, выявление требований и бизнес-правил предметной области и создание клиент-серверного приложения по предметной области.

**Постановка задачи**

**Описание автоматизируемой области**

Информационная система «Доставка еды» предназначена для совершения пользователем заказов: комплектации (выбора желаемых блюд) и выбора места для доставки. Помимо этого, пользователь может просматривать информацию о себе, своих заказах и все товары, существующие в информационной системе.

Система позволяет взаимодействовать бизнесу и пользователю. Взаимодействие происходит при помощи совершения пользователем заказов пищи. Пользователь может совершить заказ, выбрав блюда из списка всех блюд. При совершении заказа пользователь должен ввести данные, такие как адрес, куда будет доставлен заказ. Пока заказ не отправлен производителю, пользователь может редактировать его, добавляя и удаляя оттуда товары. После отправки заказа, производитель начинает выполнение заказа, уведомляя клиента о состоянии готовности заказа. После выполнения заказа пользователь получает доставку с помощью курьера предприятия, которое получило заказ.

## **Выделение основных функций и бизнес-правил автоматизируемой области**

Информационная система выполняет следующие **функции**:

1. Идентификация пользователя
2. Представление пользователю информации
3. Совершение заказа
4. Уведомление пользователя о статусе заказа

Для выбранной предметной области были получены **бизнес-правила**:

1. Обслуживание будет проводиться только в городах на территории РФ
2. У пользователя должны быть обязательно известны логин, пароль, имя пользователя и телефон
3. Логин пользователя должен быть уникален
4. Логин должен быть от 5 символов до 30 символов.
5. Пароль должен быть больше 5 символов, но меньше 16
6. Пароли хранятся в hash кодах
7. Имя должно быть от 3 до 19 символов
8. Телефон пользователя должен быть уникален
9. Телефон пользователя должен быть представлен в формате номеров РФ (+7/8)
10. Пользователь может добавить электронную почту для отправки чеков. Почта должна иметь формат \*@\*.\*, где \* - любой символ
11. При изменении пароля новый пароль должен отличаться от старого.
12. Пользователь имеет аккаунт, в который ему требуется войти для совершения заказа
13. В одном заказе могут быть блюда только одного производителя
14. Клиент может иметь много адресов
15. Выбор адреса для доставки осуществляется при добавлении первого блюда в заказ
16. Адреса не могут повторяться у одного пользователя, но могут у разных
17. Пользователь может собирать только один заказ
18. Пока заказ формируется, курьер неизвестен
19. Одни и те же блюда можно включать в заказ несколько раз
20. Логин производителя уникален и имеет от 5 до 30 символов
21. Наименование производителя имеет меньше или ровно 30 символов
22. Пока заказ собирается, нельзя удалить адрес формируемого заказа

## **Разработка архитектуры ИС**

Архитектурой ИС будет клиент-серверная двухзвенная архитектура, представленная в виде сервера и клиентского приложения.

Моделью ИС будет являться распределённая модель. Из-за отсутствия прослойки между клиентом и базой данных, часть функций будет возложено на клиент, а часть на СУБД по средствам хранимых процедур.

# 

# **Разработка серверной части информационной системы**

## **Инфологическое проектирование БД**

При проектировании базы данных был использован ER метод, включающий в себя:

1. **Выделение сущностей и связей между ними**

Для выбранной предметной области были выделены следующие сущности: Клиент, Адрес Клиента, Заказ, Курьер, Производитель, Блюдо, Ингредиенты, Уведомления

1. **Построение ER диаграммы классов, включающий все выделенные сущности и определения характеристик этих связей.**

Связь Клиент-Уведомление

Клиенту направляется личное сообщение, которое прочитать может только он. Пользователю может приходить много сообщений с указанием, какому пользователю. Пользователь необязательно должен получить хотя бы одно сообщение, но если сообщение отправлено, то известно кому оно отправлено. Отношение 1:N с классом принадлежности Необязательный –Обязательный. Отношение приведено на рисунке 1.



Рисунок 1 - Отношение Клиент-Уведомление

Связь Клиент-Адрес клиента

Клиент может иметь много адресов (бизнес-правило). Адрес не может существовать без привязки к Клиенту. Адрес принадлежит конкретному клиенту (при повторении адресов они считаются разными атрибутами, поскольку принадлежат разным клиентам). Клиент необязательно может иметь адрес. Отношение 1:N с классом принадлежности Необязательный – Обязательный. Отношение приведено на рисунке 2.



Рисунок 2 - Отношение Клиент-Адрес клиента

Связь Заказ – Адрес Клиента

Заказ совершается на конкретный адрес клиента. На один и тот же адрес может быть совершено несколько заказов. Заказ обязательно имеет адрес клиента. Также на адрес может быть не совершено заказов вообще. Отношение N:1 с классом принадлежности Обязательный – Необязательный. Отношение приведено на рисунке 3.



Рисунок 3 - Отношение Заказ – Адрес Клиента

Связь Курьер – Заказ

Каждый отдельный курьер может доставлять много заказов. Один заказ доставляется одним курьером. Курьер необязательно доставляет хотя бы один заказ. Курьер не известен при создании заказа (бизнес-правило). Отношение 1:N с классом принадлежности Необязательный – Необязательный. Отношение приведено на рисунке 4.



Рисунок 4 - Отношение Курьер – Заказ

Связь Заказ-Блюдо

Заказ содержит много блюд. Много блюд входят в заказ. Каждое блюдо необязательно содержатся в заказе, но заказ не существует без любого блюда. Отношение N:N с классом принадлежности Обязательный – Необязательный. Отношение приведено на рисунке 5.



Рисунок 5 - Отношение Заказ-Блюдо

Связь Производитель – Блюдо

Производитель производит много блюд. Каждое блюдо принадлежит конкретному производителю. Производитель необязательно производит блюда, но блюдо обязательно имеет производителя. Отношение 1:N с классом принадлежности Необязательный – Обязательный. Отношение приведено на рисунке 6.



Рисунок 6 - Отношение Производитель – Блюдо

Связь Блюдо-Ингредиенты

Каждое блюдо необязательно имеет ингредиенты. Ингредиенты могут не быть в блюде вообще (если не указано производителем). Отношение N:N с классом принадлежности Необязательный – Необязательный. Отношение приведено на рисунке 7.



Рисунок 7 - Отношение Блюдо-Ингредиенты

Общая ER диаграмма приведена на рисунке 8.



Рисунок 8 – ER диаграмма

1. **Формирование набора предварительных отношений по ER диаграмме с указанием первичного ключа для каждого отношения**

По связям из ER диаграммы выявлены следующие отношения:

**Связь Клиент-Уведомление**

Данная связь будет отображена в базе данных, по правилу 4, как два отношения: по одному для каждой сущности, где ключом является ключ соответствующей сущности и в отношении для N-связной сущности добавляется ключ односвязной сущности.

В отношение Уведомление войдёт поле Логин Клиента и станет частью первичного ключа.

**Связь Клиент-Адрес клиента**

Данная связь будет отображена в базе данных, по правилу 4, как два отношения: по одному для каждой сущности, где ключом является ключ соответствующей сущности и в отношении для N-связной сущности добавляется ключ односвязной сущности.

В отношение Адрес Клиента войдёт поле Логин Клиента и станет частью первичного ключа.

**Связь Заказ – Адрес Клиента**

Данная связь будет отображена в базе данных, по правилу 4, как два отношения: по одному для каждой сущности, где ключом является ключ соответствующей сущности и в отношении для N-связной сущности добавляется ключ односвязной сущности.

В отношение Заказ войдёт поле Адрес Клиента.

**Связь Курьер – Заказ**

Данная связь будет отображена в базе данных, по правилу 5, как 3 отношения: по одному для каждой сущности, где ключом является ключ соответствующей сущности, и одно для связи. Последнее содержит ключи обеих сущностей; первичным ключом является ключ N связной сущности.

Появится дополнительное отношение **Курьер-Заказ**, в котором первичным ключом будет выступать первичный ключ отношения Заказ.

**Связь Заказ-Блюдо**

Данная связь будет отображена в базе данных, по правилу 6, как 3 отношения: по одному на каждую сущности, где ключом является ключ соответствующей сущности, и одно для связи, в котором предварительный ключ содержит ключи обеих сущностей.

Появится дополнительное отношение **Заказ-Блюдо**, в котором первичным ключом будет выступать первичные ключи отношений Заказ и Блюдо.

**Связь Производитель-Блюдо**

Данная связь будет отображена в базе данных, по правилу 4, как два отношения: по одному для каждой сущности, где ключом является ключами будут ключи соответствующей сущности и в отношении для N-связной сущности добавляется ключ односвязной сущности.

В отношение Блюдо войдёт поле Логин Производителя.

**Связь Блюдо-Ингредиенты**

Данная связь будет отображена в базе данных, по правилу 6, как 3 отношения: по одному на каждую сущности, где ключом является ключ соответствующей сущности, и одно для связи, в котором предварительный ключ содержит ключи обеих сущностей.

Появится дополнительное отношение **Блюдо-Ингредиенты**, в котором первичным ключом будет выступать первичные ключи отношений Ингредиенты и Блюдо.

В таблице 1 приведены первичные ключи для всех сущностей.

Таблица 1 – Первичные ключи

|  |  |
| --- | --- |
| **Отношение** | **Первичный ключ** |
| Клиент | Логин Клиента |
| Адрес Клиента | Логин Клиента, Город, Район, Улица, Здание, Помещение |
| Заказ | Номер Заказа |
| Курьер | Номер-Серия Паспорта |
| Производитель | Логин Производителя |
| Блюдо | Название, Логин Производителя |
| Ингредиенты | Название Ингредиента |
| Уведомления | Логин Клиента, Дата/Время, Сообщение |
| Курьер-Заказ | Номер Заказа |
| Заказ-Блюдо | Название Блюда, Логин Производителя, Номер заказа |
| Блюдо-Ингредиенты | Название Блюда, Логин Производителя, Название Ингредиента |

Пояснение к отношению Адрес Клиента: адреса разных клиентов могут повторяться, но у одного клиента не может быть двух одинаковых адресов.

Пояснение к отношению Курьер: и Серия-номер паспорта, и Номер трудовой книги – уникальные идентификаторы, то есть потенциальные ключи. Был выбран первичный ключ Серия-номер паспорта, так как номер паспорта содержит больше символов (10), чем номер трудовой книги (7). Значит его уникальность выше.

Пояснение к отношению Блюдо: блюда с одним и тем же названием могут встречаться у производителей, но не могут встречаться у одного производителя.

Пояснение к отношению Уведомления: в одно и то же время многим пользователям могут прийти одинаковые сообщения (например, при массовой рассылке об акциях). Также пользователю в одно время могут прийти уведомления о разных заказах.

1. **Подготовка списка всех представляющих интерес атрибутов, кроме тех, которые уже были выделены. Распределение атрибутов по отношениям, где каждый атрибут может попасть только в одно отношение**

Для выделенных сущностей были выбраны и распределены атрибуты:

Для отношения Клиент: Пароль, Имя, Телефон, Электронная почта, Активность

Для отношения Адрес Клиента: Активность

Для отношения Заказ: Время Заказа, Статус Заказа, Оценка Заказа, Время Завершения Заказа

Для отношения Курьер: Фамилия, Имя, Отчество, Телефон, Трудовая книга

Для отношения Блюдо: Изображение, Цена, Описание, Калории, Масса, Видимость

Для отношения Производитель: Пароль, Наименование, Оценка, Регион, Город, Район, Улица, Здание, Этаж, Помещение

Для отношения Уведомления: Просмотрено

Для отношения Блюдо-Заказ: Количество блюд в заказе

1. Проверка отношений на БКНФ

Все отношения соответствуют 1 НФ, так как имеют ровно одно значение в атрибуте.

Для проверки на 2 НФ были получены зависимости:

**Отношение Клиент**

**Отношение Адрес Клиента**

**Отношение Заказ**

**Отношение Курьер**

**Отношение Блюдо**

{Название, Производитель} → Изображение, Цена, Описание, Калории, Масса, Видимость

**Отношение Производитель**

**Отношение Уведомление**

**Отношение Курьер-Заказ**

**Отношение Блюдо-Заказ**

Не упомянутые отношения не имеют зависимых от первичного ключа атрибутов.

В отношениях нет зависимости не ключевых атрибутов от части составного ключа, то есть они соответствуют 2НФ.

В отношениях нет транзитивных зависимостей и соответствуют 3НФ, то есть ни одни не ключевой атрибут не определяет другой не ключевой атрибут.

В отношениях детерминанты всех функциональных зависимостей являются потенциальными ключами, то есть они находятся в БКНФ.

**Даталогическое проектирование**

По полученным отношениям была построена схема базы данных. Результат приведён на рисунке 9.



Рисунок 9 – Схема базы данных

Для оптимизации работы были получены суррогатные ключи в таблицах Клиент, Адрес Клиента, Курьер, Блюдо, Логин Производителя. Ключи в зависимых таблицах также были изменены.

Все первичные ключи получили ограничение UNIQUE.

Для оптимизации хранимых значений поле Статус отношения Заказ будет вынесено в отдельную таблицу Статус.

Для ускорения выборки данных в отношение Заказ был добавлен атрибут ID клиента.

Схема базы данных приведена на рисунке 10.



Рисунок 10 – Финальная схема базы данных

**Разработка ограничений БД**

**Поля таблицы Клиент**

Id клиента будет использовать тип данных int, предоставляющий значений, то есть 4 294 967 296 значений.

Из бизнес-правил известно, что логин должен быть меньше 30 символов, но начальное значение 5 символов. Для данного поля будет использоваться тип nvarchar(30). Выбор данного типа обоснован наличием в строках кириллицы, поэтому тип varchar не подходит.

Из бизнес-правила известно, что пароль будет в hash кодах. В качестве алгоритма хеширования будет использоваться SHA256. Он имеет длину выходного значения 32 байта. Для хранения пароля будет использоваться тип данных binary(32).

Из бизнес-правила известно, что имя содержит от 3 до 20 символов. Для хранения имени будет использоваться nvarchar(20). Выбор данного типа обоснован наличием в строках кириллицы.

Из бизнес-правила известно, что телефон представлен в формате номеров РФ, то есть начинается с +7 или 8 и имеет 12/11 символов.

Для реализации данного поля будет использоваться пользовательский тип данных, наследующийся от типа char(12), на который будет наложена проверка на соответствие формату номера +7\*\*\*\*\*\*\*\*\*\*, где \* - цифра 0-9. Реализация номера через 8 будет производиться на клиентском приложении.

Электронная почта будет использовать varchar(50), так как в электронной почте используются только латинские символы, что соответствует кодировке ASCII. Также поле электронная почта будет проверяться на соответствие формату \*@\*.\*,

Активность будет использовать tinyint для передачи логического значения (правда-ложь).

На поле Логин и Телефон будут наложены ограничения UNIQUE(по бизнес-правилу).

**Поля таблицы Адрес Клиента**

Номер адреса будет использовать тип int.

ID Клиента будет иметь тип int, в соответсвии с полем ID из таблицы Клиент.

Поля Город, Район, Улица будут импеть тип nvarchar(50).

Поле Здание/Дом будут иметь тип nvarchar(6).

Поле Помещение/Квартира будут иметь тип nvarchar(5).

Поле активность будет иметь тип данных tinyint.

На поля ID клиента, Город, Район, Улица, Здание, Помещение будет наложено общее ограничение UNIQUE (потенциальный ключ).

На поле ID Клиента будет возложено ограничение внешнего ключа (поле ID таблицы Клиенты) со стратегией поддержания ссылочной целостности каскадирование.

**Поля таблицы Производитель**

Поле ID производителя будет использовать тип int.

Из бизнес-правил известно, что логин производителя, так же, как и логин пользователя, должен быть меньше 30 символов, но начальное значение 5 символов. Для данного поля будет использоваться тип nvarchar(30).

Пароль, хранящийся в hash-кодах будет использовать тип binary(32).

Из бизнес-правил известно, что наименование производителя будет иметь 30 символов, для данного поля будет использоваться тип nvarchar(30).

Оценка будет использовать пользовательский тип данных, наследуемый из tinyint, на который будет наложено ограничение от 0 до 5 (в соответствии со звёздами в подобных приложениях).

Поля Город, Район, Улица будут импеть тип nvarchar(50).

Поле Здание/Дом будут иметь тип nvarchar(6).

Поле Помещение/Квартира будут иметь тип nvarchar(5).

На поле Логин будет наложено ограничение UNIQUE (по бизнес-правилу).

**Поля таблицы Статус**

Поле ID будет иметь тип данных smallint, так как набор данных будет ограничиваться небольшим константным значение количества статусов.

Поле значение будет иметь тип данных nvarchar(50).

**Поля таблицы Курьер**

Поле ID будет иметь тип данных int.

Поля Имя, Отчество и Фамилия будет иметь тип данных nvarchar(30).

Поле Телефон будет иметь ранее созданный для телефона клиента тип данных.

Поле Номер-серия паспорта будет иметь тип данных char(10).

Поле Номер трудовой книги будет иметь тип данных char(7).

На поле Номер-серия паспорта и Номер трудовой книги будет наложено ограничение UNIQUE.

**Поля таблицы Блюдо**

Поле ID будет иметь тип int.

Поле ID Производителя будет иметь тип, в соответствии с первичным ключом таблицы Производитель.

Поле Название будет иметь тип nvarchar(50).

Поле Изображение будет иметь тип varbinary(MAX).

Поле Цена будет иметь тип данных money.

Поле Описание будет иметь тип nvarchar(100).

Поле Калорийность будет иметь данных int.

Поле Масса будет иметь тип данных int.

Поле Видимость будет иметь тип tinyint.

На поля Название и ID Производителя будут иметь общее ограничение UNIQUE.

На поле ID Производителя будет возложено ограничение внешнего ключа (поле ID таблицы Производитель) со стратегией поддержания ссылочной целостности каскадирование.

**Поля таблицы Заказ**

Поле Номер Заказа будет иметь тип int.

Поле ID Клиента будет иметь тип int, в соответствии с полем ID из таблицы Клиент.

Поле ID Адреса будет иметь тип int, в соответствии с полем ID из таблицы Адрес Клиента.

Поля Время заказа и Время завершения заказа будут иметь тип datetime.

Поле Статус будет иметь тип данных smallint, в соответствии с полем ID из таблицы Статус.

Поле Оценка заказа будет иметь пользовательский тип, наследуемый из tinyint, на который будет наложено ограничение от 0 до 5.

На поле Статус будет возложено ограничение внешнего ключа (поле ID таблицы Статус) со запертом совершения действий.

На поле ID Адреса будет возложено ограничение внешнего ключа (поле ID таблицы Адреса Клиента) со стратегией поддержания ссылочной целостности каскадирование.

На поле Клиент будет возложено ограничение внешнего ключа (поле ID таблицы Клиент) со запертом совершения действий.

**Поля таблицы Курьер-Заказ**

Поле Номер Заказа будет использовать тип данных int, в соответствие с полем Номер Заказа из таблицы Заказ.

Поле ID Курьера будет использовать тип int, в соответствие с полем ID из таблицы Курьер.

На поле Номер Заказа будет возложено ограничение внешнего ключа (поле Номер Заказа таблицы Заказ) со стратегией поддержания ссылочной целостности каскадирование.

На поле ID Курьера будет возложено ограничение внешнего ключа (поле ID таблицы Курьер) со стратегией поддержания ссылочной целостности каскадирование.

**Поля таблицы Блюдо-Заказ**

Поле Номер Заказа будет использовать тип данных int, в соответствие с полем Номер Заказа из таблицы Заказ.

Поле ID Блюда будет использовать тип данных int, в соответствие с полем ID из таблицы Блюдо.

Полю Количество будет использовать тип int.

На поле Номер Заказа будет возложено ограничение внешнего ключа (поле Номер Заказа таблицы Заказ) со стратегией поддержания ссылочной целостности каскадирование.

На поле ID Блюда будет возложено ограничение внешнего ключа (поле ID таблицы Блюдо) со стратегией поддержания ссылочной целостности каскадирование.

**Поля таблицы Ингредиенты**

Поле ID будет использовать тип int.

Поле Наименование будет использовать тип данных nvarchar(30).

**Поля таблицы Блюдо-Ингредиенты**

Поле ID Ингредиента будет использовать тип int, в соответствие с полем ID из таблицы Ингредиенты.

Поле ID Блюда будет иметь тип данных int, в соответствие с полем ID из таблицы Блюдо.

На поле ID Ингредиента будет возложено ограничение внешнего ключа (поле ID таблицы Ингредиенты) со стратегией поддержания ссылочной целостности каскадирование.

На поле ID Блюда будет возложено ограничение внешнего ключа (поле ID таблицы Блюдо) со стратегией поддержания ссылочной целостности каскадирование.

**Поля таблицы Уведомления**

Поле ID Клиента будет иметь тип данных int, в соответствии с полем ID из таблицы Клиент.

Поле Сообщение будет иметь тип данных nvarchar(50).

Поле Дата Отправки будет иметь тип datetime.

Поле Прочитано будет иметь тип smallint как логическое значение.

На поле ID Клиента будет возложено ограничение внешнего ключа (поле ID таблицы Клиенты) со стратегией поддержания ссылочной целостности каскадирование.

## **Разработка триггеров, ХП и представлений**

**Разработка триггеров**

Для поддержания целостности данных были разработаны триггеры:

1. **Триггер проверки принадлежности адреса клиенту**

After триггер на таблицу Заказ при обновлении и добавлении, который проверяет, что адрес принадлежит пользователю, который инициировал заказ. Для работы использует курсор на вставленные записи. Если адрес не принадлежит клиенту, вызывается исключение 50020. Триггер разрешает циклическую связь между таблицами Заказ, Адрес клиента и Клиент.

1. **Триггер проверки принадлежности продуктов только одному производителю**

After триггер на таблицу Блюдо-Заказ при вставке новых записей. Использует курсор на вставленные записи. Перебирая заказы, проверяет, что все блюда в нём принадлежит одному производителю. Иначе вызывается исключение 50010.

1. **Триггер проверки наличия только одного собираемого заказа**

After триггер на таблицу Заказ при вставке новых записей. Проверяет записи на наличии двух записей у вставленных клиентов со статусом «Собирается». Если такие записи обнаружены, выбрасывается исключение 50011.

1. **Триггер уничтожения пустого заказа**

After триггер на таблицу Блюдо-Заказ при удалении записей.

Перебирает заказы, из которых были удалены блюда, и, если в заказе не осталось блюд, то удаляет его.

1. **Триггер автоматической отправки сообщения о состоянии заказа**

After триггер на таблицу Заказ при обновлении записей. Просматривает каждый изменённый заказ и отправляет сообщение о его статусе клиенту.

1. **Триггер проверки текущего заказа при скрытии (удалении) адреса**

After триггер на таблицу Адреса Клиентов на обновление. Использует вложенный курсор. Внешний курсор просматривает адреса, которые были скрыты. Внутренний курсор, проходит по заказам, имеющий скрываемый адрес. Если статус скрываемого заказа «Собирается», то вызывается исключение 50013.

**Общий подход к разработке триггеров**

В триггерах, где используются курсоры, при вызове исключений внутри курсора, добавлен обработчик, закрывающий курсор (курсоры) и освобождающий память.

### **Разработка ХП**

Для модификации данных были разработаны хранимые процедуры:

1. **Регистрация клиента**

Данная процедура предназначена для создания нового пользователя. Данная процедура получает на вход логин, пароль, имя, телефон, почту.

Эти данные валидируются:

При неправильной длине логина (от 5 до 30) вызывается исключение 50000.

При существовании логина, который был передан, в базе данных вызывается исключение 50001.

При неправильной длине имени (от 3 до 20) вызывается исключение 50002.

При существовании телефона в базе данных вызывается исключение 50003.

При вводе телефона неправильного формата вызывается исключение 50004.

При вводе почты неправильного формата вызывается исключение 50005.

Если валидация прошла успешно, то начинается транзакция и происходит вставка данных в таблицу Клиент. Транзакция успешно завершается.

1. **Удаление клиента**

Данная процедура предназначена для удаления пользователей. Она принимает логин и пароль в качестве параметров.

Происходи проверка, существует ли пользователь с заданными параметрами. Если нет, то вызывается исключение 50006. Если пользователь найден, то открывается транзакция, происходит обновление и пользователь становится неактивным. Транзакция завершается.

1. **Смена логина клиента**

Данная процедура предназначена для смены логина пользователем. Она принимает старый логин, пароль, новый логин.

Происходит проверка на наличие пользователя с введённым старым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Также проверяется наличие пользователя с введённым пользователем новым логином. Если такой пользователь уже существует, то вызывается исключение 50001.

Далее происходит проверка на соответствие длине нового логина требуемой длине (от 5 до 30 символов). Если проверка неуспешна, то вызывается исключение 50000.

При прохождении всех валидаций, запускается транзакция и выполняется обновление поля логин новым логином. Транзакция завершается.

1. **Смена имени клиента**

Данная процедура предназначена для смены имени пользователя. Она принимает логин, пароль, новое имя.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Далее происходит проверка на соответствие требуемой длине нового имени (от 3 до 20). Если длина не соответствует требованиям, то вызывается исключение 50002.

Если валидация пройдена, открывается транзакция и происходит изменение имени клиента. Транзакция закрывается.

1. **Смена телефона клиента**

Данная процедура предназначена для смены телефона клиента. Она принимает логин, пароль и новый телефон.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Далее проверяется наличие телефона в базе данных. Если телефон найден, то вызывается исключение 50003.

Также проверяется соответствие телефона формату телефонов РФ.

При вводе телефона неправильного формата вызывается исключение 50004.

Если данные прошли валидацию, то открывается транзакция, изменяется номер телефона. Транзакция завершается.

1. **Смена электронной почты**

Данная процедура предназначена для смены почты пользователя и принимает логин, пароль и новую почту.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

При вводе почты неправильного формата вызывается исключение 50005.

Если валидация проедена, то открывается транзакция и происходит смена почты клиента. Транзакция успешно завершается.

1. **Смена пароля клиента**

Данная процедура предназначена для смены пароля пользователя. Она принимает логин, старый пароль и новый пароль.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Открывается транзакция и происходит смена пароля, транзакция закрывается.

1. **Добавление адреса клиента**

Данная процедура предназначена для создания новой записи об адресе пользователя. Она принимает логин, пароль и данные об адресе: город, район, улицу, здание, помещение.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Выделяется ID пользователя по логину. Проверяется существование введённого адреса в базе данных. Если он уже существует у данного пользователя, вызывается исключение 50014.

Если проверка пройдена открывается транзакция. Проверяется наличие адреса в скрытых. Если адрес скрыт, то запись обновляется и становится видимой. Если запись не обнаружена в скрытых, то создаётся новая запись. Транзакция завершается.

1. **Изменение адреса клиента**

Данная процедура предназначена для изменения существующей записи об адресе. Она принимает ID изменяемого адреса, логин пользователя, пароль и данные о новом адресе: город, район, улицу, здание, помещение.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Выделяется ID пользователя по логину. Проверяется существование адреса с ID в базе данных. Если он не существует у данного пользователя, вызывается исключение 50007.

Если проверки пройдены, открывается транзакция и происходит обновление адреса пользователя. Транзакция завершается.

1. **Удаление адреса клиента**

Данная процедура вызывает скрытие адреса. Она принимает ID адреса, логин и пароль.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Выделяется ID пользователя по логину. Проверяется существование адреса с ID в базе данных. Если он не существует у данного пользователя, вызывается исключение 50007.

Если проверки пройдены, открывается транзакция и происходит скрытие адреса пользователя. Транзакция завершается.

1. **Инициирование заказа**

Данная процедура инициирует создание заказа. Она принимает логин и ID адреса, на который будет осуществляться заказ.

Происходит проверка на наличие пользователя с введённым логином и паролем. Если пользователь не найден, вызывается исключение 50006.

Выделяется ID пользователя по логину. Открывается транзакция и создаётся новая запись в таблице Заказ. Транзакция завершается.

1. **Добавление в заказ блюда**

Данная процедура добавляет блюдо в заказ. Она принимает ID заказа, ID блюда и количество блюд.

Происходит проверка на наличие заказа. Если заказ не найден, вызывается исключение 50009.

Транзакция открывается. Если блюдо ещё не в заказе, добавляется новая запись. Иначе существующая запись модифицируется (инкрементируется количество блюд). Транзакция завершается.

1. **Удаление блюда из заказа**

Данная процедура удаляет блюдо из заказа. Она принимает ID заказа и ID блюда.

Происходит проверка на наличие блюда в заказе. Если блюдо не найдено, вызывается исключение 50012.

Транзакция открывается. Если количество блюд, которые должны быть удалены, равно 1, то запись удаляется из таблицы Блюдо-Заказ. Иначе количество декрементируется. Транзакция закрывается.

1. **Оформление заказа**

Данная процедура меняет статус заказа с «Собираемый» на «В обработке». Она принимает ID заказа.

Происходит проверка статуса заказа. Если статус заказа отличается от «Собирается», то вызывается исключение 50008.

Открывается транзакция. Изменяется статус заказа и записывается дата и время заказа. Транзакция закрывается.

1. **Смена адреса заказа**

Данная процедура предназначена для смены адреса в собираемом заказе. Она принимает ID заказа и ID адреса.

Происходит проверка существования адреса. Если адрес не существует, то вызывается исключение 50007.

Далее проверяется существование заказа. Если заказ не существует, то вызывается исключение 50009.

Также проверяется статус заказа. Если его статус не является «Собирается», то вызывается исключение 50012.

Если проверки пройдены, то открывается транзакция и происходит смена адреса.

**Общий подход к разработке хранимых процедур**

Все вышеописанные процедуры имеют обработку исключений, выполняющую проверку на наличие открытой транзакции. Если транзакция открыта, то происходит откат транзакции.

### **Разработка представлений**

Для ускорения выборки данных на клиентском приложении были созданы представления:

1. Таблица статусов заказов
2. Адреса с логином
3. Набор блюд в заказах
4. Все блюда
5. Заказ и производитель
6. Информация о блюдах

**Разработка клиентского части информационной системы**

**Разработка прототипа интерфейса пользователя**

При разработке прототипа интерфейса были созданы концепции интерфейса, который положили основу реализации визуальной части приложения.

Первое, что должен увидеть при входе пользователь – окно входа и регистрации. Концептуальное изображение приведено на рисунке 11.

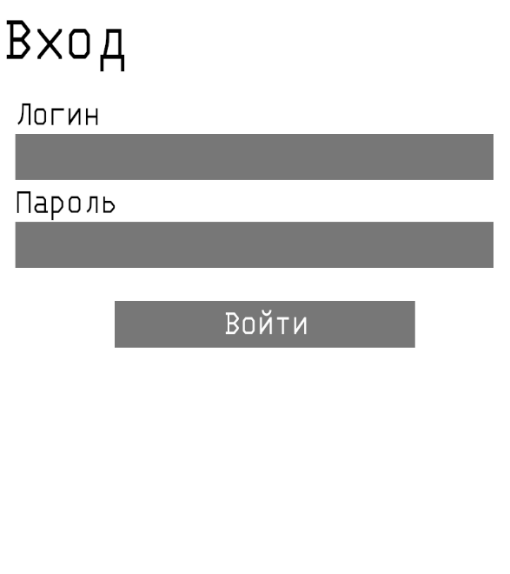


Рисунок 11 – Концепция формы входа

Ещё не зарегистрированный пользователь должен иметь возможность зарегистрироваться. Для этого была введена новая форма – форма регистрации. Концептуальное изображение приведено на рисунке 12.



Рисунок 12 – Концепция формы регистрации

После регистрации/входа пользователь попадает на основную страницу, на которой должны находится возможность создания заказа, возможность посмотреть прошлые и текущие заказы и возможность посмотреть данные пользователя.

Данные функции были расположены на вкладках главной страницы.

Концептуальное изображение вкладок приведено на рисунках 13-15.

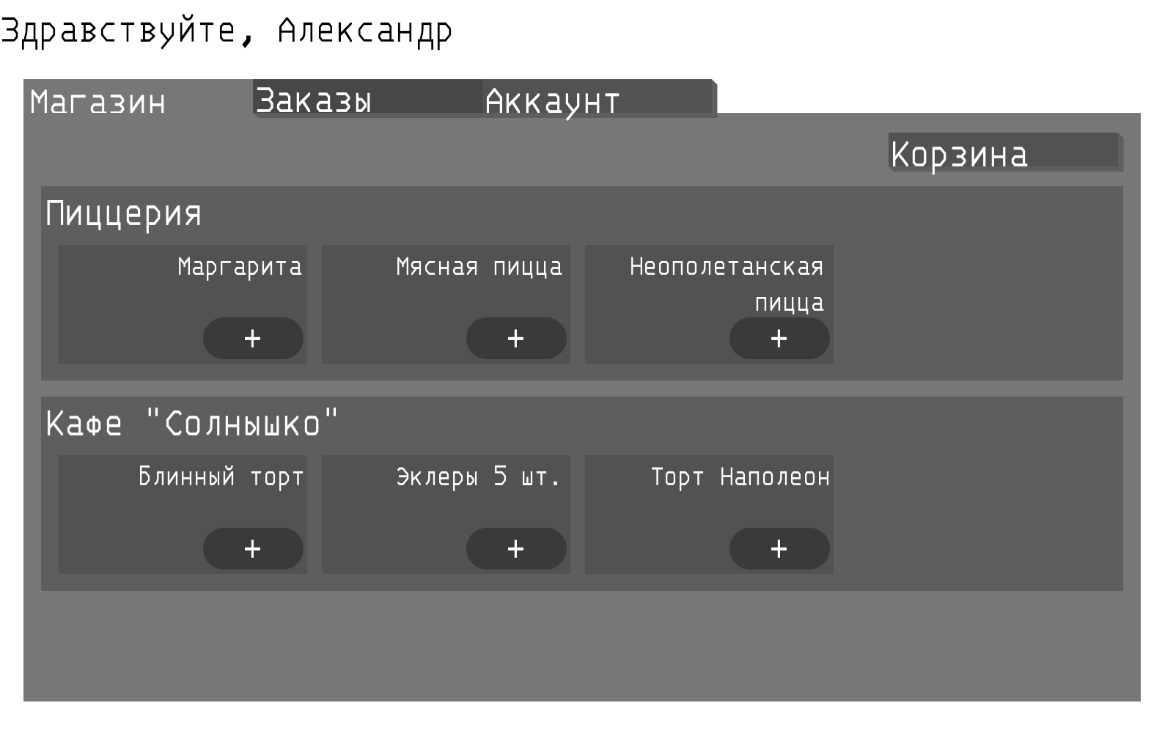


Рисунок 13 – Концепция главной формы (вкладка Магазин)

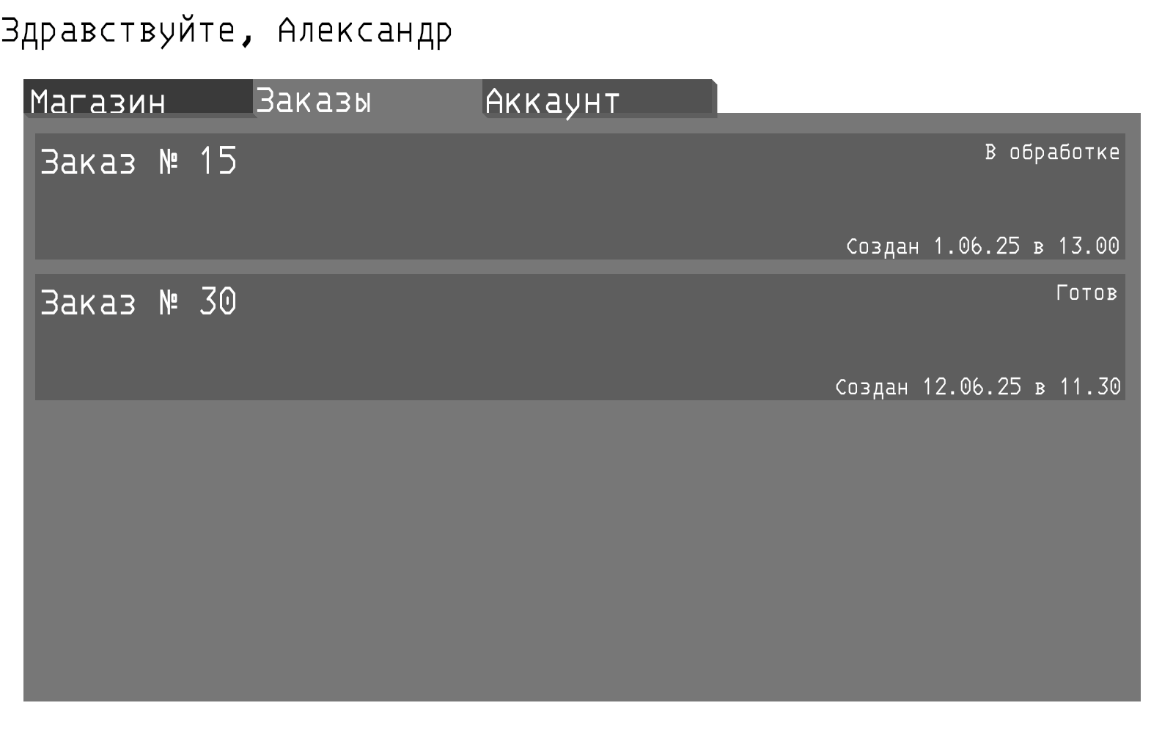


Рисунок 14 – Концепция главной формы (вкладка Заказы)

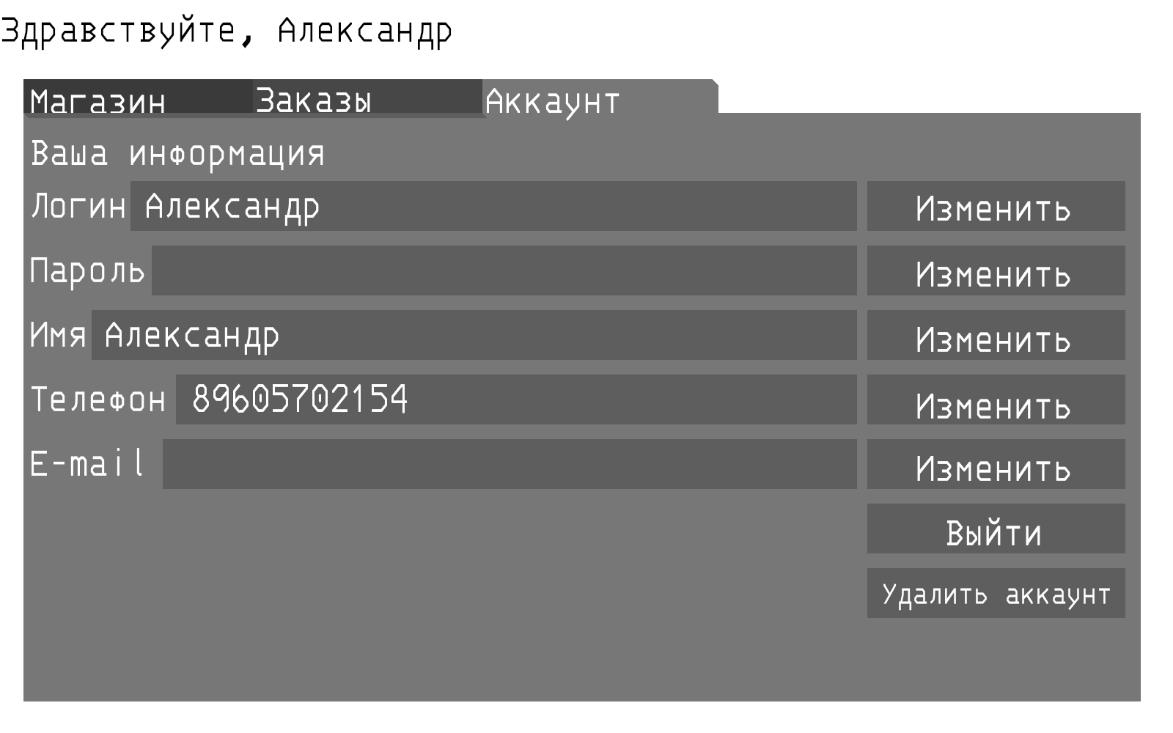


Рисунок 15 – Концепция главной формы (вкладка Аккаунт)

Во вкладке Магазин пользователь может посмотреть информацию о блюде детально. Концептуальное изображение детальной информации о блюде приведено на рисунке 16.

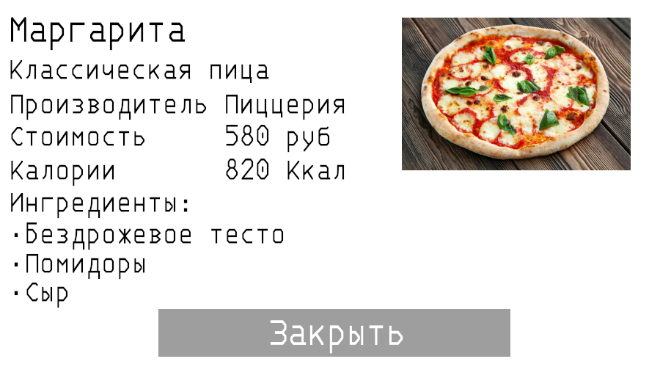


Рисунок 16 – Концепция формы информации о блюде

При добавлении блюда в заказ, появляется форма с выбором адреса, на который будет доставлен заказ. Концептуальное изображение формы приведено на рисунке 17.

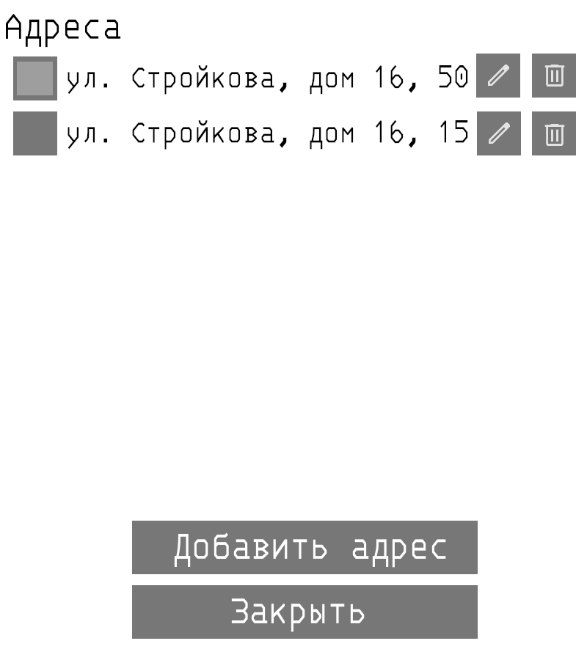


Рисунок 17 – Концепция формы выбора адреса

На этой форме пользователь может изменить или удалить адрес. Изменение адреса будет иметь вид, отражённый на рисунке #.

После выбора адреса, можно закрыть окно и нажать на кнопку Корзина, которая откроет новую форму Заказ. Концептуальное изображение формы приведено на рисунке 18.

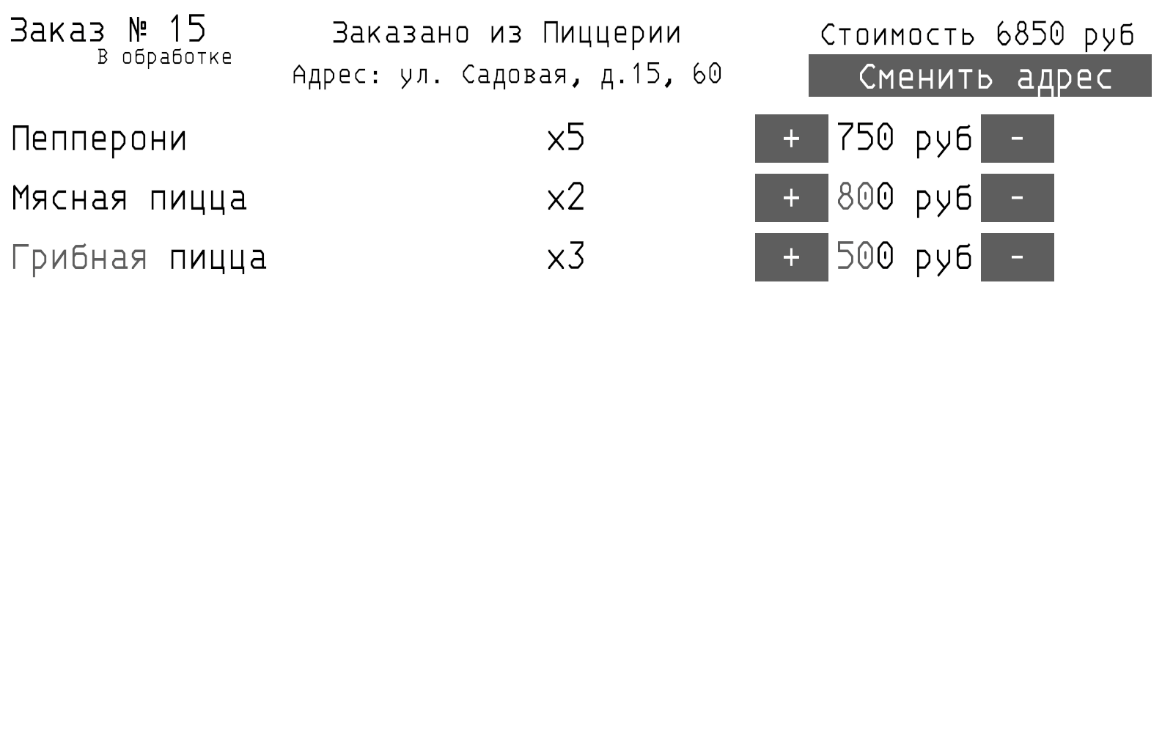


Рисунок 18 – Концепция формы Заказ

Для редактирования информации пользователя на вкладке Аккаунт, нужно перейти на вкладку Аккаунт и выбрать соответствующий пункт. Та же форма используется для адресов. Концептуальное изображение формы приведено на рисунке 19.

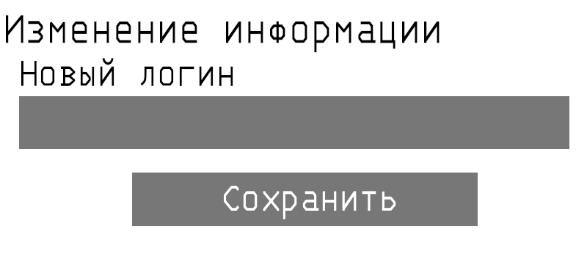


Рисунок 18 – Концепция формы Изменение данных

**Реализация клиентской части**

Клиентская часть реализована с помощью C# .NET Framework 4.7.2. Для взаимодействия приложения с СУБД будет использован Entity Framework 3.1.32. В качестве способа представления будет использоваться Windows Forms.

Для реализации будет использоваться IDE Microsoft Visual Studio 2022 17.13.1.

При разработке клиентского приложения было решено разделить функционал приложения на несколько логических уровней, реализованных с помощью статических классов. Деление было сделано с целью разграничения функционала и упрощения работы.

1. **Уровень базы данных**

Уровень базы данных отвечает за функционал, взаимодействующий с СУБД. Для работы с СУБД был использован Entity Framework 3.1.32. Главной причиной выбора данной технологии служит объектно-ориентированный подход к представлению данных и дальнейшая простота работы с ними через LINQ.

Все вызовы хранимых процедур и выборок через представления будут производится через класс Database.

Уровень представлен классом Database, DeliveryAppContext (контекст Entity Framework) и классами Entity Framework, представляющими собой таблицы и представления базы данных.

1. **Уровень клиента**

Уровень клиента отвечает за передачу данных и формирование из кодов исключений осмысленных сообщений.

Также данный уровень отвечает за функционал, который не мог быть реализован на СУБД: хеширование паролей и форматирование номера телефона.

Данный уровень представлен классами ClientActions, осуществляющим вышеизложенные действия, DatabaseImage, отвечающий за обработку изображений, получаемых из базы данных, и классом User.

Класс User – синглтон класс, который хранит в себе данные о пользователе и предоставляет их для заполнения.

1. **Уровень представления**

Уровень представления отвечает за отображение приложения на компьютере пользователя. Данный уровень представлен набором форм.

1. **Точка входа в программу**

Класс Program – точка входа программы, запускающая контекст Windows Forms и открывающая окно входа в систему.

**Создание форм Windows Forms**

Были созданы формы в соответствии с концептуальными изображениями.

Форма входа и регистрации была объедена в одну для удобства перехода между ними. Для этого были созданы вкладки. Форма отображена на рисунках 19-20.

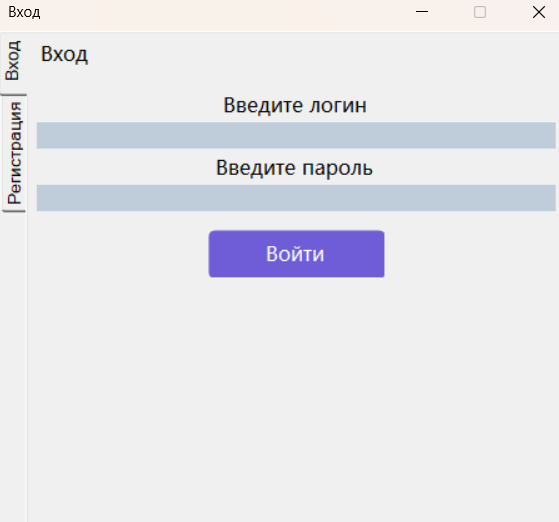


Рисунок 19 – Форма входа

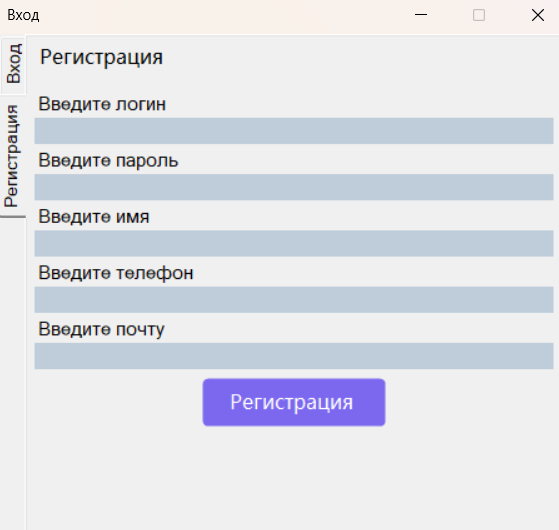


Рисунок 20 – Форма регистрации

После ввода данных или регистрации нового пользователя, открывается вкладка магазина главная форма. Оно показано на рисунке 21.

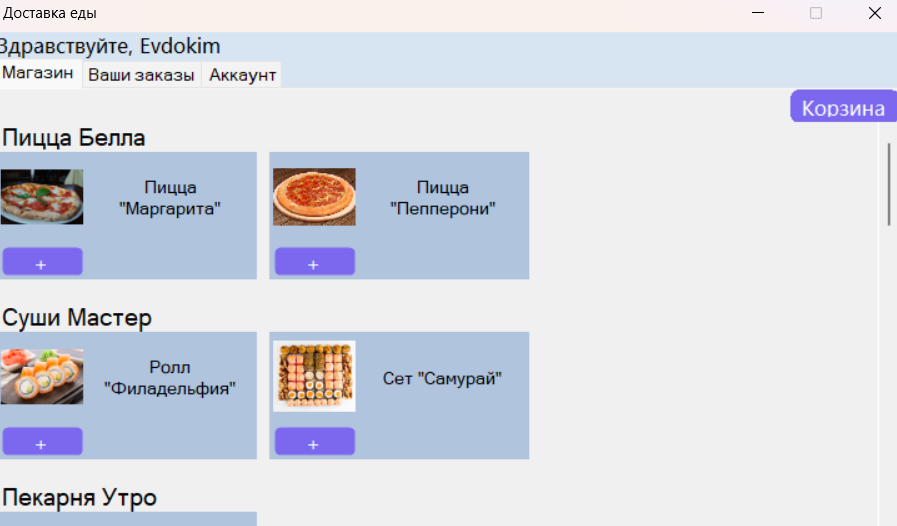


Рисунок 21 – Главная форма (вкладка Магазин)

Также доступны вкладки Ваши заказы (рисунок 22) и Аккаунт (рисунок 23).

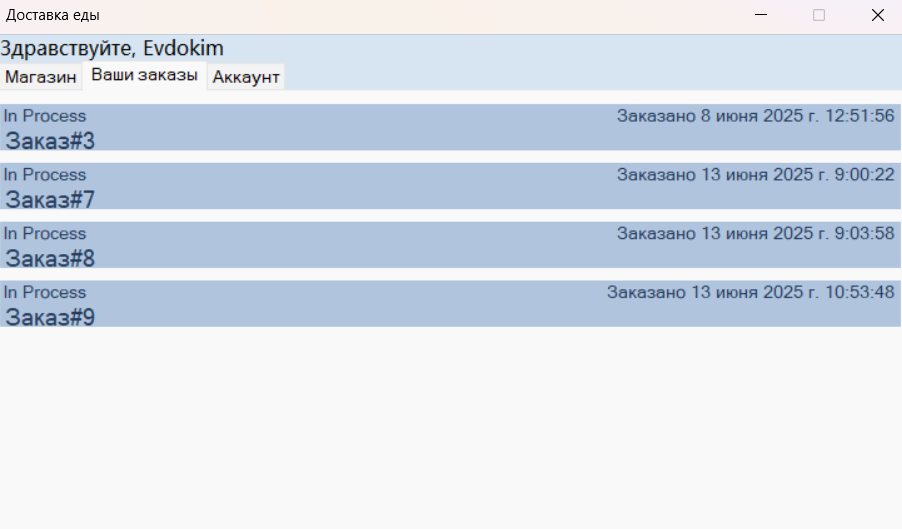


Рисунок 22 – Главная форма (вкладка Ваши Заказы)



Рисунок 23 – Главная форма (вкладка Аккаунт)

Для получения детальной информации о блюде нужно перейти во вклдку Магазин и нажать на нужное Блюдо. Форма отображения блюда приведена на рисунке 24.

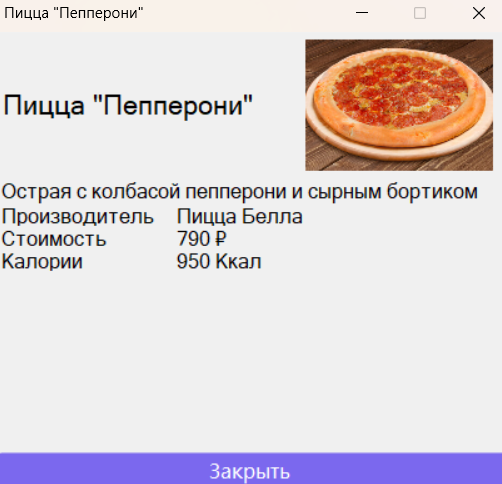


Рисунок 24 – Форма информации о блюде

Форма выбора адреса представлена на рисунке 25.

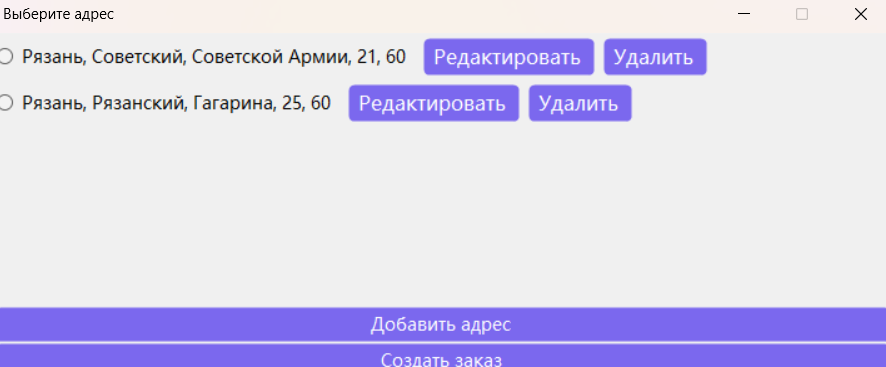


Рисунок 25 – Форма выбора адреса

При выборе Заказа на вкладке Ваши Заказы или при выборе Корзины появляется форма Заказ. Форма Заказ представлена на рисунке 26.

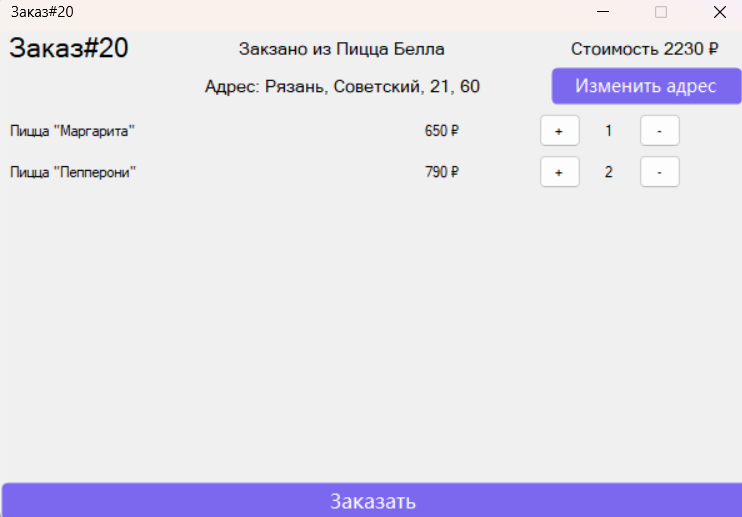


Рисунок 26 – Форма Заказ

Формы изменения данных представлены на рисунках 27-32.

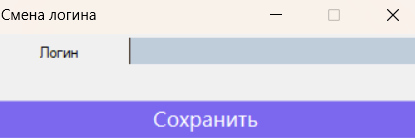


Рисунок 27 – Форма изменения Логина

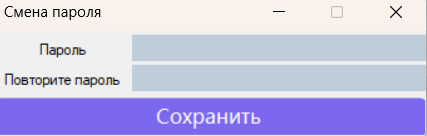


Рисунок 28 – Форма изменения Пароля

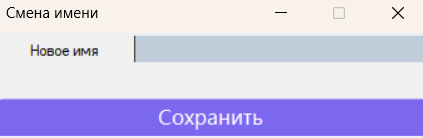


Рисунок 29 – Форма изменения Имени

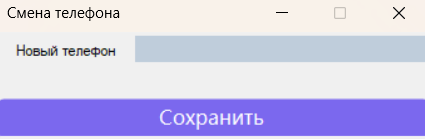


Рисунок 30 – Форма изменения Телефона

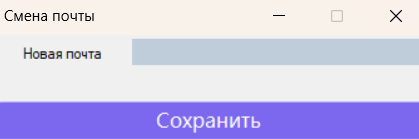


Рисунок 31 – Форма изменения Почты

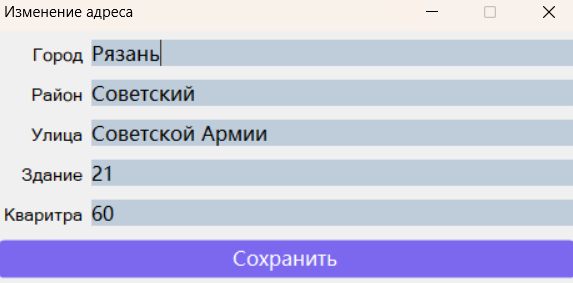


Рисунок 32 – Форма изменения Адреса

**Тестирование основных функций приложения**

Было проведено тестирование основных функций приложения как на СУБД, так и в клиентском приложении.

**Тестирование белым ящиком**

Тестирование белым ящиком подразумевает построение псевдокода, графа, основываясь на псевдокоде, и тестовых вариантов.

**Регистрация клиента**

1 если (длина(Логина) < 5 ИЛИ длина(Логина) > 30), то вызов Исключения

2 если (Логин уже есть в таблице Клиенты), то вызов Исключения

3 если (длина(Имя) < 3 ИЛИ длина(Имя) > 20), то вызов Исключения

4 если (Телефон уже есть в таблице Клиент), то вызов Исключения

5 если (Формат(Телефон) не соответствует паттерну), то вызов Исключения

6 если (Формат(Почта) не соответствует паттерну), то вызов Исключения

7 Открытие транзакции

7 Внесение данных

7 Закрытие транзакции

8 Конец

9 Обработка исключения

9 Закрытие открытой транзакции

9 Передача исключения

Граф теста приведён на рисунке 33.



Рисунок 33 – Граф теста «Регистрация клиента»

Цикломатическое число равно 7 (6 предикатных узлов + 1).

Тестовые пути:

1. 1-9
2. 1-2-9
3. 1-2-3-9
4. 1-2-3-4-9
5. 1-2-3-4-5-9-8
6. 1-2-3-4-5-6-9-8
7. 1-2-3-4-5-6-7-8

Тестовые варианты:

Тестовый вариант 1:

ИД: длина(Логина) < 5 ИЛИ длина(Логина) > 30

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 34. Тестирование на клиентском приложении представлено на рисунке 35.



Рисунок 34 –Тестирование тестового варианта 1 «Регистрация клиента» на СУБД

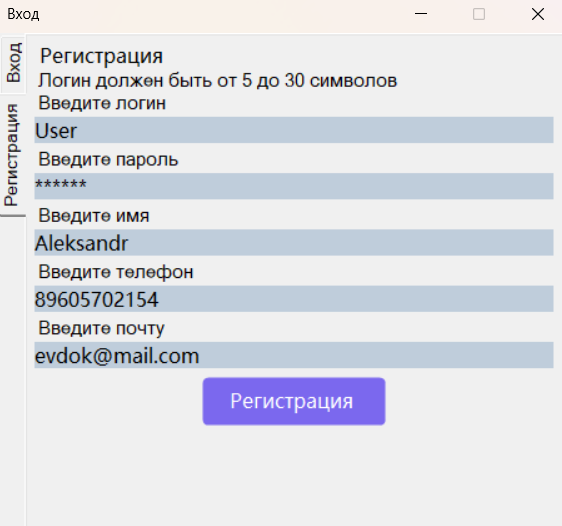


Рисунок 35 –Тестирование тестового варианта 1 «Регистрация клиента» на клиентском приложении

Тестовый вариант 2:

ИД: (длина(Логина) >= 5 И длина(Логина) <= 30) И

Логин уже есть в таблице Клиенты

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 36. Тестирование на клиентском приложении представлено на рисунке 37.



Рисунок 36 –Тестирование тестового варианта 2 «Регистрация клиента» на СУБД

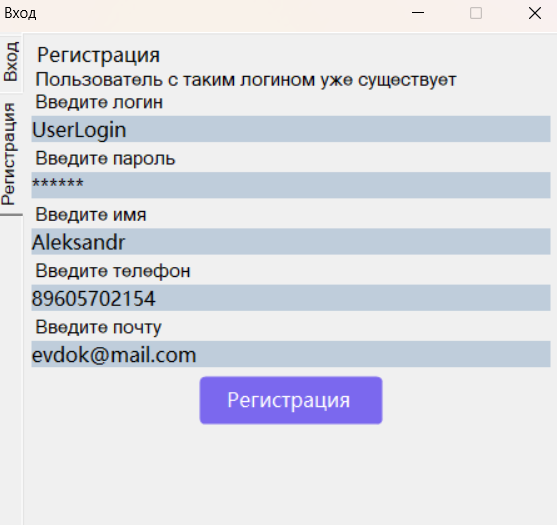


Рисунок 37 –Тестирование тестового варианта 2 «Регистрация клиента» на клиентском приложении

Тестовый вариант 3:

ИД: (длина(Логина) >= 5 И длина(Логина) < =30) И

Логина нет в таблице Клиенты И

(длина(Имени) < 3 ИЛИ длина(Имени) > 20)

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 38. Тестирование на клиентском приложении представлено на рисунке 39.



Рисунок 38 –Тестирование тестового варианта 3 «Регистрация клиента» на СУБД

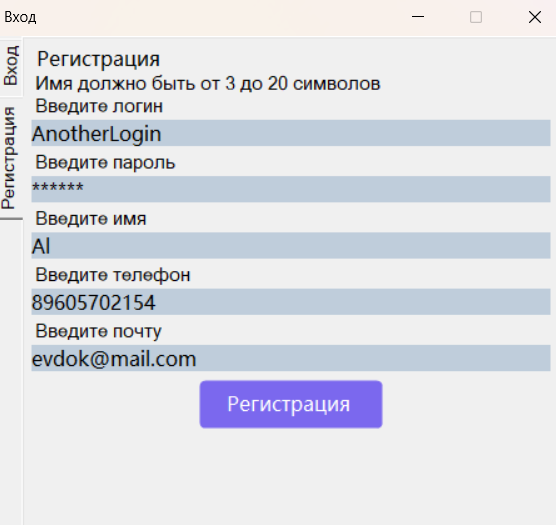


Рисунок 39 –Тестирование тестового варианта 3 «Регистрация клиента» на клиентском приложении

Тестовый вариант 4:

ИД: (длина(Логина) >= 5 И длина(Логина) <= 30) И

Логина нет в таблице Клиенты И

(длина(Имени) >= 3 И длина(Имени) <= 20) И

Телефон уже есть в таблице Клиент

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 40. Тестирование на клиентском приложении представлено на рисунке 41.



Рисунок 40 –Тестирование тестового варианта 4 «Регистрация клиента» на СУБД

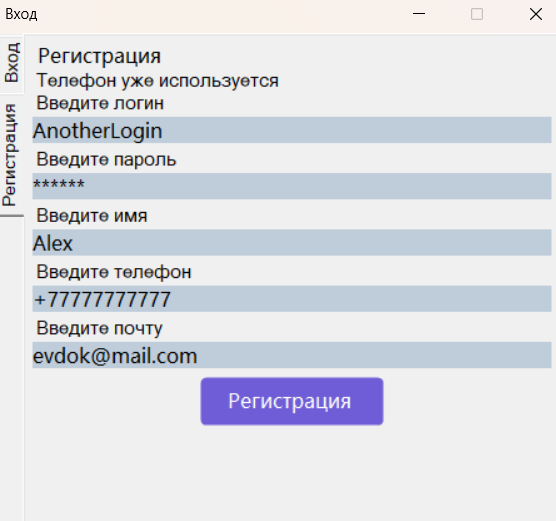


Рисунок 41 –Тестирование тестового варианта 4 «Регистрация клиента» на клиентском приложении

Тестовый вариант 5:

ИД: (длина(Логина) >= 5 И длина(Логина) <= 30) И

Логина нет в таблице Клиенты И

(длина(Имени) >= 3 И длина(Имени) <= 20) И

Телефона нет в таблице Клиент И

Формат(Телефона) не соответствует паттерну

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 42. Тестирование на клиентском приложении представлено на рисунке 43.



Рисунок 42 –Тестирование тестового варианта 5 «Регистрация клиента» на СУБД

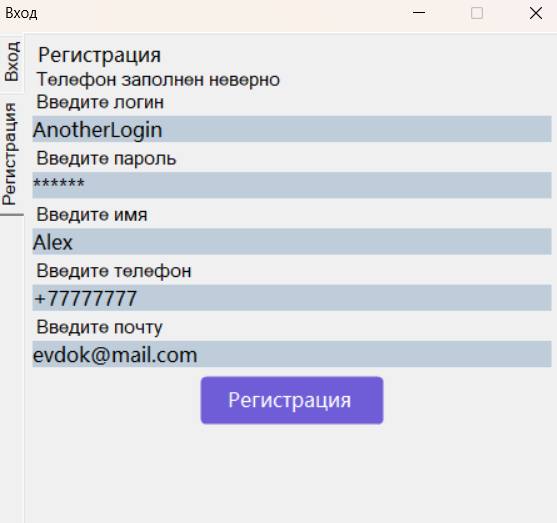


Рисунок 43 – Тестирование тестового варианта 5 «Регистрация клиента» на клиентском приложении

Тестовый вариант 6:

ИД: (длина(Логина) >= 5 И длина(Логина) <= 30) И

Логина нет в таблице Клиенты И

(длина(Имени) >= 3 И длина(Имени) <= 20) И

Телефона нет в таблице Клиент И

Формат(Телефона) соответствует паттерну И

Формат(Почта) не соответствует паттерну

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 44. Тестирование на клиентском приложении представлено на рисунке 45.



Рисунок 44 – Тестирование тестового варианта 6 «Регистрация клиента» на СУБД

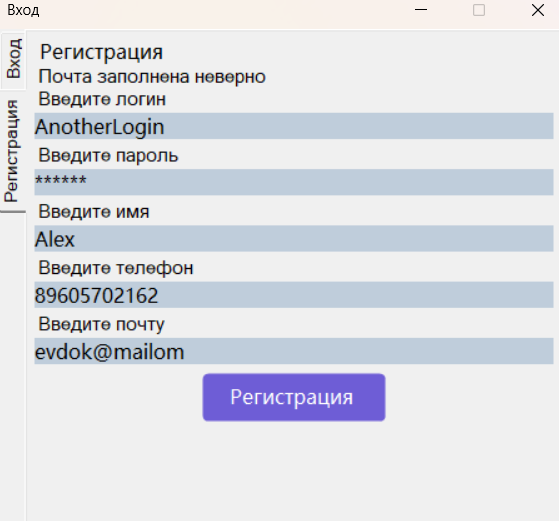


Рисунок 45 – Тестирование тестового варианта 6 «Регистрация клиента» на клиентском приложении

Тестовый вариант 7:

ИД: (длина(Логина) >= 5 И длина(Логина) <= 30) И

Логина нет в таблице Клиенты И

(длина(Имени) >= 3 И длина(Имени) <= 20) И

Телефона нет в таблице Клиент И

Формат(Телефона) соответствует паттерну И

Формат(Почта) соответствует паттерну

ОР: Сохранение данных и завершение

Тестирование в СУБД представлено на рисунке 46. Клиентское приложение при успешной регистрации открывает главную форму вкладку Магазин.



Рисунок 46 –Тестирование тестового варианта 7 «Регистрация клиента» на СУБД

Степень покрытия 7/7 \* 100% = 100%

**Удаление клиента**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 Открытие транзакции

2 Изменение данных

2 Закрытие транзакции

3 Конец

4 Обработка исключения

4 Закрытие открытой транзакции

4 Передача исключения

Граф теста приведён на рисунке 47.



Рисунок 47 – Граф теста «Удаление клиента»

Цикломатическое число равно 2 (1 предикатный узел + 1).

Тестовые пути:

1. 1-4-3
2. 1-2-3

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователь не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 48. Тестирование на клиентском приложении представлено на рисунке 49.



Рисунок 48 – Тестирование тестового варианта 1 «Удаление клиента» на СУБД

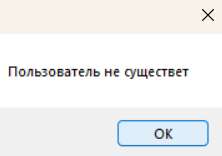


Рисунок 49 – Тестирование тестового варианта 1 «Удаление клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

ОР: Пользователь скрывается (аккаунт удаляется)

Тестирование в СУБД представлено на рисунке 48. Клиентское приложение при успешному удалении аккаунта открывает форму Входа.



Рисунок 48 –Тестирование тестового варианта 2 «Удаление клиента» на СУБД

Степень покрытия 2/2 \* 100% = 100%

**Смена логина клиента**

1 если (Старого Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 если (Новый Логин есть в таблице Клиенты), то вызов Исключения

3 если (длина(Логина) < 5 ИЛИ длина(Логина) > 30), то вызов Исключения

4 Открытие транзакции

4 Изменение данных

4 Закрытие транзакции

5 Конец

6 Обработка исключения

6 Закрытие открытой транзакции

6 Передача исключения

Граф теста приведён на рисунке 49.



Рисунок 49 – Граф теста «Смена логина клиента»

Цикломатическое число равно 4 (3 предикатных узла + 1).

Тестовые пути:

1. 1-6-5
2. 1-2-6-5
3. 1-2-3-6-5
4. 1-2-3-4-5

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 50. Тестирование на клиентском приложении представлено на рисунке 51.



Рисунок 50 – Тестирование тестового варианта 1 «Смена логина клиента» на СУБД

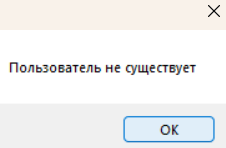


Рисунок 51 – Тестирование тестового варианта 1 «Смена логина клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

И Новый логин уже есть в таблице Клиенты

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 52. Тестирование на клиентском приложении представлено на рисунке 53.



Рисунок 52 –Тестирование тестового варианта 2 «Смена логина клиента» на СУБД

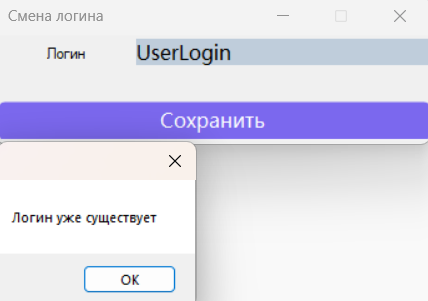


Рисунок 53 –Тестирование тестового варианта 2 «Смена логина клиента» на клиентском приложении

Тестовый вариант 3:

ИД: Пользователь существует

И Нового логина нет в таблице Клиенты

И длина(Логина) < 5 ИЛИ длина(Логина) > 30

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 54. Тестирование на клиентском приложении представлено на рисунке 55.



Рисунок 54 – Тестирование тестового варианта 3 «Смена логина клиента» на СУБД

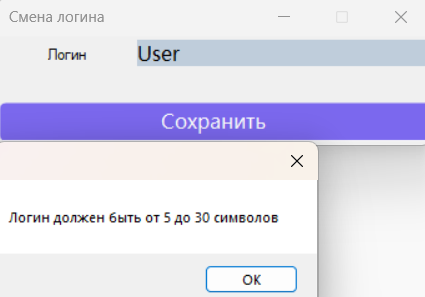


Рисунок 55 – Тестирование тестового варианта 3 «Смена логина клиента» на клиентском приложении

Тестовый вариант 4:

ИД: Пользователь существует

И Нового логина нет в таблице Клиенты

И (длина(Логина) >= 5 И длина(Логина) <= 30)

ОР: Смена логина и завершение работы

Тестирование в СУБД представлено на рисунке 56. При смене логина пользователь автоматически возвращается на вкладку Аккаунт.



Рисунок 56 – Тестирование тестового варианта 4 «Смена логина клиента» на СУБД

Степень покрытия 4/4 \* 100% = 100%

**Смена имени клиента**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 если (длина(Имя) < 3 ИЛИ длина(Имя) > 20), то вызов Исключения

3 Открытие транзакции

3 Изменение данных

3 Закрытие транзакции

4 Конец

5 Обработка исключения

5 Закрытие открытой транзакции

5 Передача исключения

Граф теста приведён на рисунке 57.



Рисунок 57 – Граф теста «Смена имени клиента»

Цикломатическое число равно 3 (2 предикатных узла + 1).

Тестовые пути:

1. 1-5-4
2. 1-2-5-4
3. 1-2-3-4

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 58. Тестирование на клиентском приложении представлено на рисунке 59.



Рисунок 58 – Тестирование тестового варианта 1 «Смена имени клиента» на СУБД

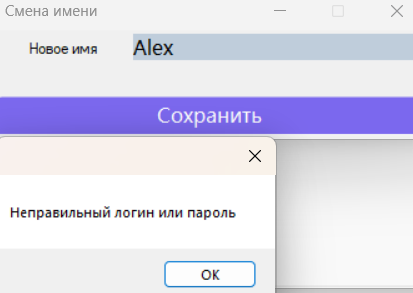


Рисунок 59 – Тестирование тестового варианта 1 «Смена имени клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

И (длина(Имя) < 3 ИЛИ длина(Имя) > 20)

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 60. Тестирование на клиентском приложении представлено на рисунке 61.



Рисунок 60 – Тестирование тестового варианта 2 «Смена имени клиента» на СУБД

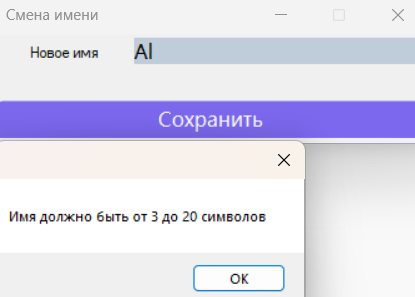


Рисунок 61 – Тестирование тестового варианта 2 «Смена имени клиента» на клиентском приложении

Тестовый вариант 3:

ИД: Пользователь существует

И (длина(Имя) >= 3 И длина(Имя) <= 20)

ОР: Изменение записи и завершение работы

Тестирование в СУБД представлено на рисунке 62. Успешное изменение имени на клиентском приложении возвращает пользователя на вкладку Аккаунт.



Рисунок 62 – Тестирование тестового варианта 3 «Смена имени клиента» на СУБД

Степень покрытия 3/3 \* 100% = 100%

**Смена телефона клиента**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 если (Телефон уже есть в таблице Клиент), то вызов Исключения

3 если (Формат(Телефон) не соответствует паттерну), то вызов Исключения

4 Открытие транзакции

4 Изменение данных

4 Закрытие транзакции

5 Конец

6 Обработка исключения

6 Закрытие открытой транзакции

6 Передача исключения

Граф теста приведён на рисунке 63.



Рисунок 63 – Граф теста «Смена телефона клиента»

Цикломатическое число равно 4 (3 предикатных узла + 1).

Тестовые пути:

1. 1-6-5
2. 1-2-6-5
3. 1-2-3-6-5
4. 1-2-3-4-5

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 64. Тестирование на клиентском приложении представлено на рисунке 65.



Рисунок 64 – Тестирование тестового варианта 1 «Смена телефона клиента» на СУБД

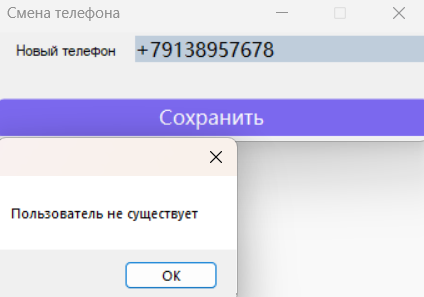


Рисунок 65 – Тестирование тестового варианта 1 «Смена телефона клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

И Телефон уже есть в таблице Клиент

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 66. Тестирование на клиентском приложении представлено на рисунке 67.



Рисунок 66 – Тестирование тестового варианта 2 «Смена телефона клиента» на СУБД

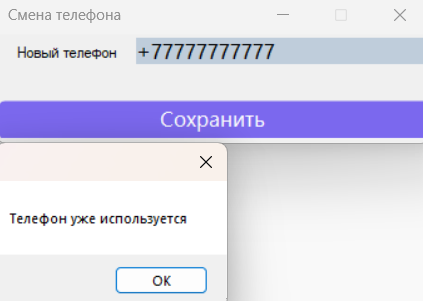


Рисунок 67 – Тестирование тестового варианта 2 «Смена телефона клиента» на клиентском приложении

Тестовый вариант 3:

ИД: Пользователь существует

И Телефон нет в таблице Клиент

И формат Телефона не соответствует паттерну

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 68. Тестирование на клиентском приложении представлено на рисунке 69.



Рисунок 68 – Тестирование тестового варианта 3 «Смена телефона клиента» на СУБД

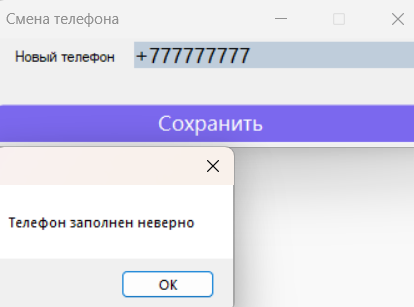


Рисунок 69 – Тестирование тестового варианта 3 «Смена телефона клиента» на клиентском приложении

Тестовый вариант 4:

ИД: Пользователь существует

И Телефон нет в таблице Клиент

И формат Телефона соответствует паттерну

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 70. Изменение телефона на клиентском приложении открывает главную форму на вкладке Аккаунт.



Рисунок 70 – Тестирование тестового варианта 4 «Смена телефона клиента» на СУБД

Степень покрытия 4/4 \* 100% = 100%

**Смена электронной почты**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 если (Формат(Почта) не соответствует паттерну), то вызов Исключения

3 Открытие транзакции

3 Изменение данных

3 Закрытие транзакции

4 Конец

5 Обработка исключения

5 Закрытие открытой транзакции

5 Передача исключения

Граф теста приведён на рисунке 71.



Рисунок 71 – Граф теста «Смена электронной почты»

Цикломатическое число равно 3 (2 предикатных узла + 1).

Тестовые пути:

1. 1-5-4
2. 1-2-5-4
3. 1-2-3-4

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 72. Тестирование на клиентском приложении представлено на рисунке 73.



Рисунок 72 – Тестирование тестового варианта 1 «Смена электронной почты» на СУБД

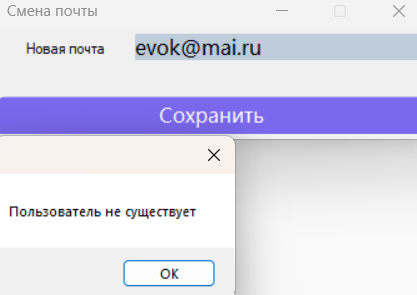


Рисунок 73 – Тестирование тестового варианта 1 «Смена электронной почты» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

И Формат Почты не соответствует паттерну

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 74. Тестирование на клиентском приложении представлено на рисунке 75.



Рисунок 74 – Тестирование тестового варианта 2 «Смена электронной почты» на СУБД

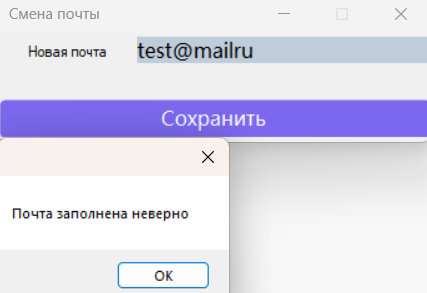


Рисунок 75 – Тестирование тестового варианта 2 «Смена электронной почты» на клиентском приложении

Тестовый вариант 3:

ИД: Пользователь существует

И Формат Почты соответствует паттерну

ОР: Изменение почты и завершение работы

Тестирование в СУБД представлено на рисунке 76. Изменение электронной почты на клиентском приложении открывает главную форму на вкладке Аккаунт.



Рисунок 76 – Тестирование тестового варианта 3 «Смена электронной почты» на СУБД

Степень покрытия 3/3 \* 100% = 100%

**Смена пароля клиента**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 Открытие транзакции

2 Изменение данных

2 Закрытие транзакции

3 Конец

4 Обработка исключения

4 Закрытие открытой транзакции

4 Передача исключения

Граф теста приведён на рисунке 77.



Рисунок 77 – Граф теста «Смена пароля клиента»

Цикломатическое число равно 2 (1 предикатный узел + 1).

Тестовые пути:

1. 1-4-3
2. 1-2-3

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 78. Тестирование на клиентском приложении представлено на рисунке 79.



Рисунок 78 – Тестирование тестового варианта 1 «Смена пароля клиента» на СУБД

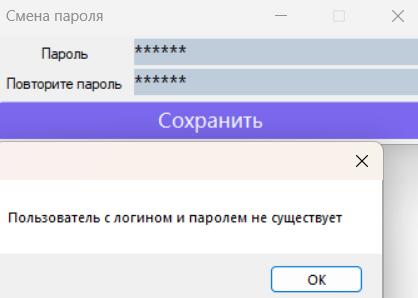


Рисунок 79 – Тестирование тестового варианта 1 «Смена пароля клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

ОР: Изменение пароля и завершение работы

Тестирование в СУБД представлено на рисунке 80. Изменение пароля клиента на клиентском приложении открывает главную форму на вкладке Аккаунт.



Рисунок 80 – Тестирование тестового варианта 2 «Смена пароля клиента» на СУБД

Степень покрытия 2/2 \* 100% = 100%

**Добавление адреса клиента**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 Получение ID пользователя по Логину

3 если (Адрес есть в базе данных у пользователя с Логином), то вызов Исключения

4 Открытие транзакции

5 если (Адрес есть в скрытых),

6 то Запись обновляется (адрес открывается)

7 иначе Создаётся новая запись

8 Закрытие транзакции

9 Конец

10 Обработка исключения

10 Закрытие открытой транзакции

10 Передача исключения

Граф теста приведён на рисунке 81.



Рисунок 81 – Граф теста «Добавление адреса клиента»

Цикломатическое число равно 4 (3 предикатных узла + 1).

Тестовые пути:

1. 1-10-9
2. 1-2-3-10-9
3. 1-2-3-4-5-6-8-9
4. 1-2-3-4-5-7-8-9

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 82. Тестирование на клиентском приложении представлено на рисунке 83.



Рисунок 82 – Тестирование тестового варианта 1 «Добавление адреса клиента» на СУБД

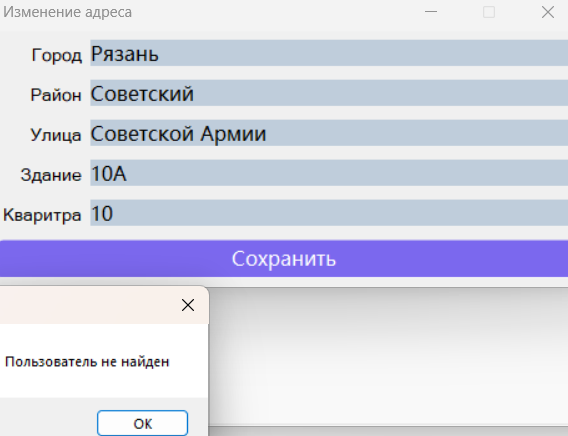


Рисунок 83 – Тестирование тестового варианта 1 «Добавление адреса клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

И Адрес уже есть у пользователя

И он открыт

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 84. Тестирование на клиентском приложении представлено на рисунке 85.



Рисунок 84 – Тестирование тестового варианта 2 «Добавление адреса клиента» на СУБД

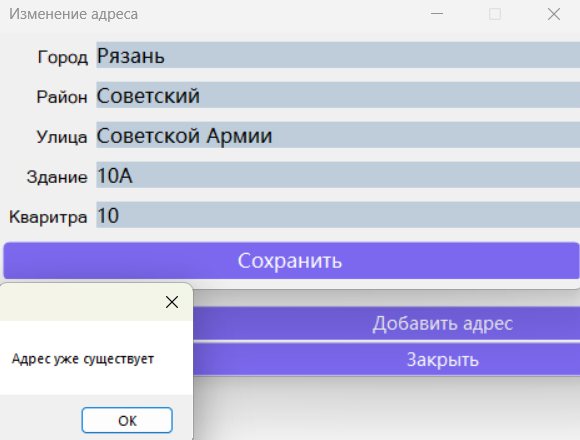


Рисунок 85 – Тестирование тестового варианта 2 «Добавление адреса клиента» на клиентском приложении

Тестовый вариант 3:

ИД: Пользователь существует

И Адрес уже есть у пользователя

И Адрес есть у скрыт

ОР: Открытие адреса и завершение работы

Тестирование в СУБД представлено на рисунке 86. При открытии адреса он появляется на вкладке с адресами.



Рисунок 86 – Тестирование тестового варианта 3 «Добавление адреса клиента» на СУБД

Тестовый вариант 4:

ИД: Пользователь существует

И Адрес не существует у пользователя

ОР: Создание записи и завершение работы

Тестирование в СУБД представлено на рисунке 87. При создании адреса он появляется на вкладке с адресами.



Рисунок 87 – Тестирование тестового варианта 4 «Добавление адреса клиента» на СУБД

Степень покрытия 4/4 \* 100% = 100%

**Изменение адреса клиента**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 Получение ID пользователя по Логину

3 если (Адрес не принадлежит пользователю с Логином), то вызов Исключения

4 Открытие транзакции

4 Данные обновляются

4 Закрытие транзакции

5 Конец

6 Обработка исключения

6 Закрытие открытой транзакции

6 Передача исключения

Граф теста приведён на рисунке 88.



Рисунок 88 – Граф теста «Изменение адреса клиента»

Цикломатическое число равно 3 (2 предикатных узла + 1).

Тестовые пути:

1. 1-6-5
2. 1-2-3-6-5
3. 1-2-3-4-5

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 89. Тестирование на клиентском приложении представлено на рисунке 90.



Рисунок 89 – Тестирование тестового варианта 1 «Изменение адреса клиента» на СУБД

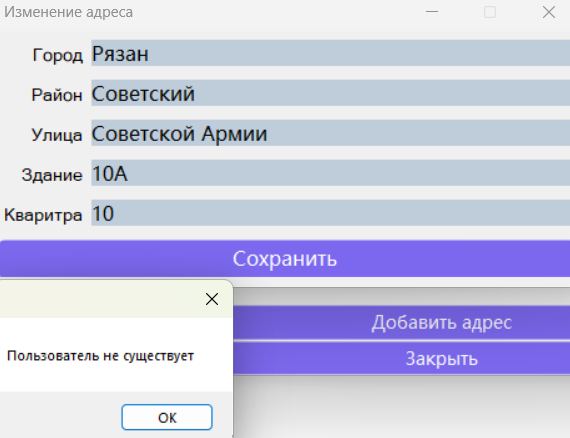


Рисунок 90 – Тестирование тестового варианта 1 «Изменение адреса клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

И Адрес не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 91. Тестирование на клиентском приложении представлено на рисунке 92.



Рисунок 91 – Тестирование тестового варианта 2 «Изменение адреса клиента» на СУБД

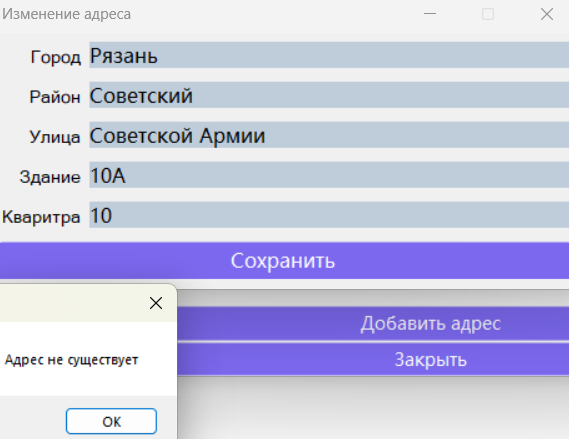


Рисунок 92 – Тестирование тестового варианта 2 «Изменение адреса клиента» на клиентском приложении

Тестовый вариант 3:

ИД: Пользователь существует

И Адрес существует

ОР: Изменение данных и завершение работы

Тестирование в СУБД представлено на рисунке 93. При изменении адреса он изменяется на вкладке с адресами.



Рисунок 93 – Тестирование тестового варианта 3 «Изменение адреса клиента» на СУБД

Степень покрытия 3/3 \* 100% = 100%

**Удаление адреса клиента**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 Выделение ID пользователя по логину

3 если (Адрес не существует у пользователя с Логином), то вызов Исключения

4 Открытие транзакции

4 Данные обновляются

4 Закрытие транзакции

5 Конец

6 Обработка исключения

6 Закрытие открытой транзакции

6 Передача исключения

Граф теста приведён на рисунке 94.



Рисунок 94 – Граф теста «Удаление адреса клиента»

Цикломатическое число равно 3 (2 предикатных узла + 1).

Тестовые пути:

1. 1-6-5
2. 1-2-3-6-5
3. 1-2-3-4-5

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 95. Тестирование на клиентском приложении представлено на рисунке 96.



Рисунок 95 – Тестирование тестового варианта 1 «Удаление адреса клиента» на СУБД

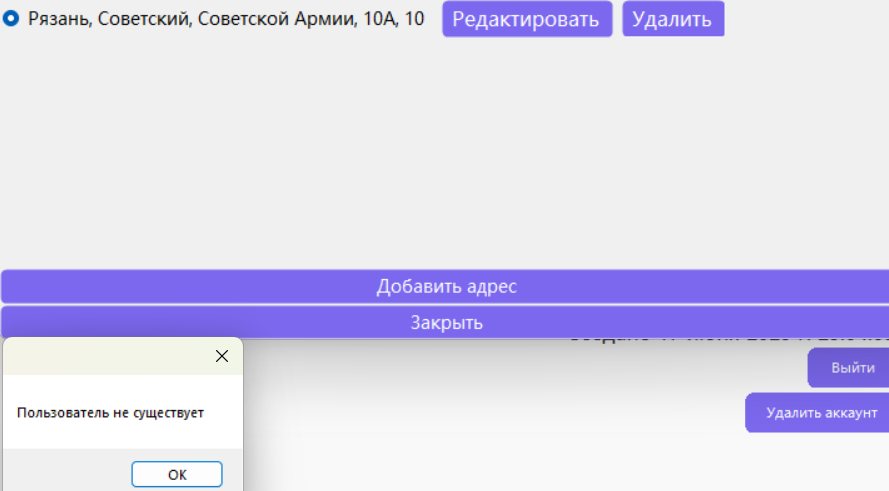


Рисунок 96 – Тестирование тестового варианта 1 «Удаление адреса клиента» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

И Адрес не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 97. Тестирование на клиентском приложении представлено на рисунке 98.



Рисунок 97 – Тестирование тестового варианта 2 «Удаление адреса клиента» на СУБД

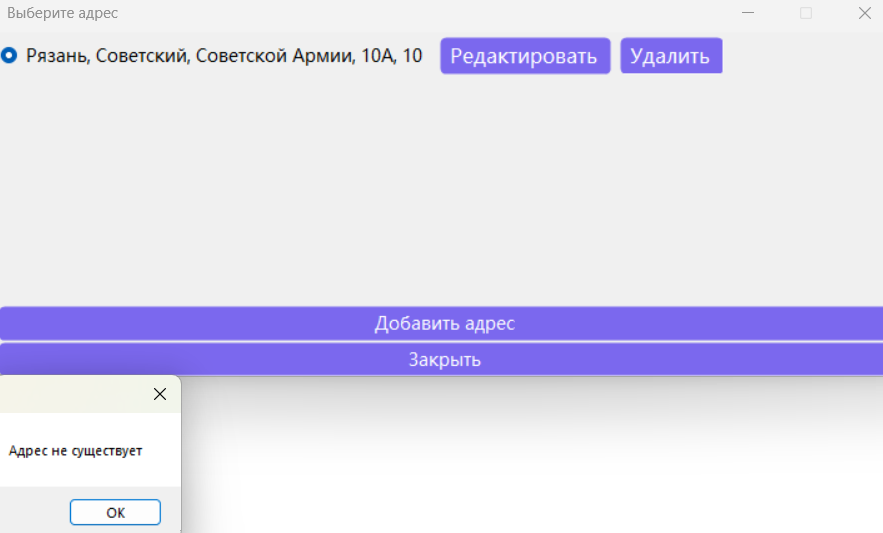


Рисунок 98 – Тестирование тестового варианта 2 «Удаление адреса клиента» на клиентском приложении

Тестовый вариант 3:

ИД: Пользователь существует

И Адрес существует

ОР: Адрес скрывается и завершение работы

Тестирование в СУБД представлено на рисунке 99. При удалении адреса он исчезает со вкладки с адресами.



Рисунок 99 – Тестирование тестового варианта 3 «Удаление адреса клиента» на СУБД

Степень покрытия 3/3 \* 100% = 100%

**Инициирование заказа**

1 если (Логина и Пароля нет в таблице Клиенты), то вызов Исключения

2 Выделение ID пользователя по логину

3 Открытие транзакции

3 Создание записи

3 Закрытие транзакции

4 Конец

5 Обработка исключения

5 Закрытие открытой транзакции

5 Передача исключения

Граф теста приведён на рисунке 100.



Рисунок 100 – Граф теста «Инициирование заказа»

Цикломатическое число равно 2 (1 предикатный узел + 1).

Тестовые пути:

1. 1-5-4
2. 1-2-3-4

Тестовые варианты:

Тестовый вариант 1:

ИД: Пользователя не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 101. Тестирование на клиентском приложении представлено на рисунке 102.



Рисунок 101 – Тестирование тестового варианта 1 «Инициирование заказа» на СУБД

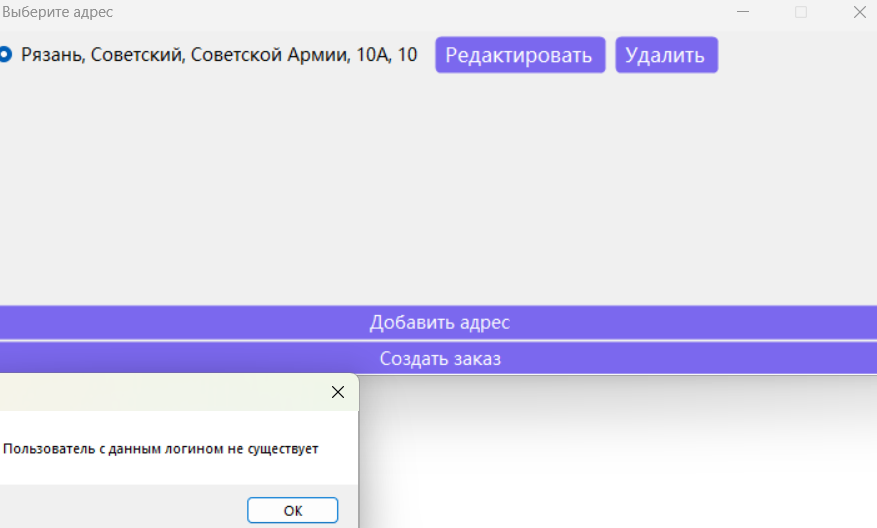


Рисунок 102 – Тестирование тестового варианта 1 «Инициирование заказа» на клиентском приложении

Тестовый вариант 2:

ИД: Пользователь существует

ОР: Создание заказа и завершение работы

Тестирование в СУБД представлено на рисунке 103. При создании заказа вкладка закрывается и становится возможным открыть корзину.



Рисунок 103 – Тестирование тестового варианта 2 «Инициирование заказа» на СУБД

Степень покрытия 2/2 \* 100% = 100%

**Добавление в заказ блюда**

1 если (Заказа с ID нет в таблице Заказы), то вызов Исключения

2 Открытие транзакции

3 если (Блюдо ещё не в Заказе с ID),

4 то Создаётся запись

5 иначе Существующая запись модифицируется

6 Закрытие транзакции

7 Конец

8 Обработка исключения

8 Закрытие открытой транзакции

8 Передача исключения

Граф теста приведён на рисунке 30.



Рисунок 30 – Граф теста «Добавление в заказ блюда»

Цикломатическое число равно 3 (2 предикатных узла + 1).

Тестовые пути:

1. 1-8-7
2. 1-2-3-4-6-7
3. 1-2-3-5-6-7

Тестовые варианты:

Тестовый вариант 1:

ИД: Заказа не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 103. Данный тестовый вариант невозможно воспроизвести в клиентском приложении, так при добавлении на клиенте проверяется наличие открытого заказа.



Рисунок 103 – Тестирование тестового варианта 1 «Добавление в заказ блюда» на СУБД

Тестовый вариант 2:

ИД: Заказ существует

И Блюда нет в заказе

ОР: Создание записи и завершение работы

Тестирование в СУБД представлено на рисунке 104. На клиентском приложении данный тестовый вариант отразится в корзине.



Рисунок 104 – Тестирование тестового варианта 2 «Добавление в заказ блюда» на СУБД

Тестовый вариант 3:

ИД: Заказ существует

И Блюдо уже есть в заказе

ОР: Изменение записи и завершение работы

Тестирование в СУБД представлено на рисунке 105. На клиентском приложении данный тестовый вариант отразится в корзине.



Рисунок 105 – Тестирование тестового варианта 3 «Добавление в заказ блюда» на СУБД

Степень покрытия 3/3 \* 100% = 100%

**Удаление блюда из заказа**

1 если (Блюдо не в Заказе с ID), то вызов Исключения

2 Открытие транзакции

3 если (только одно Блюдо в Заказе с ID),

4 то Удаляется запись

5 иначе Существующая запись модифицируется

6 Закрытие транзакции

7 Конец

8 Обработка исключения

8 Закрытие открытой транзакции

8 Передача исключения

Граф теста приведён на рисунке 106.



Рисунок 106 – Граф теста «Удаление блюда из заказа»

Цикломатическое число равно 3 (2 предикатных узла + 1).

Тестовые пути:

1. 1-8-7
2. 1-2-3-4-6-7
3. 1-2-3-5-6-7

Тестовые варианты:

Тестовый вариант 1:

ИД: Заказа не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 107. Данный тестовый вариант невозможно воспроизвести в клиентском приложении, так при удалении на клиенте действие совершается на вкладке Заказ.



Рисунок 107 – Тестирование тестового варианта 1 «Удаление блюда из заказ» на СУБД

Тестовый вариант 2:

ИД: Заказ существует

И одно блюдо в заказе

ОР: Уделение записи и завершение работы

Тестирование в СУБД представлено на рисунке 108. Тестовый вариант отразится на клиентском приложении на вкладке Заказ.



Рисунок 108 – Тестирование тестового варианта 2 «Удаление блюда из заказ» на СУБД

Тестовый вариант 3:

ИД: Заказ существует

И больше одного блюда в заказе

ОР: Изменение записи и завершение работы

Тестирование в СУБД представлено на рисунке 109. Тестовый вариант отразится на клиентском приложении на вкладке Заказ.



Рисунок 109 – Тестирование тестового варианта 3 «Удаление блюда из заказ» на СУБД

Степень покрытия 3/3 \* 100% = 100%

**Оформление заказа**

1 если (статус Заказа не «Собирается»), то вызов Исключения

2 Открытие транзакции

2 Изменение статуса

2 Закрытие транзакции

3 Конец

4 Обработка исключения

4 Закрытие открытой транзакции

4 Передача исключения

Граф теста приведён на рисунке 110.



Рисунок 110 – Граф теста «Оформление заказа»

Цикломатическое число равно 2 (1 предикатный узел + 1).

Тестовые пути:

1. 1-4-3
2. 1-2-3

Тестовые варианты:

Тестовый вариант 1:

ИД: Заказа со статусом «Собирается» нет

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 111. Данный тестовый вариант невозможно воспроизвести в клиентском приложении, так как оформление на клиенте совершается на вкладке Заказ.



Рисунок 111 – Тестирование тестового варианта 1 «Оформление заказа» на СУБД

Тестовый вариант 2:

ИД: Есть заказ со статусом «Собирается»

ОР: Смена статуса заказа и завершение работы

Тестирование в СУБД представлено на рисунке 112. После удачного заказа нужный можно увидеть во вкладке Ваши Заказы.



Рисунок 112 – Тестирование тестового варианта 2 «Оформление заказа» на СУБД

Степень покрытия 2/2 \* 100% = 100%

**Смена адреса заказа**

1 если (Адрес не существует), то вызов Исключения

2 если (Заказ не существует), то вызов Исключения

3 если (статус Заказа не «Собирается»), то вызов Исключения

4 Открытие транзакции

4 Изменение статуса

4 Закрытие транзакции

5 Конец

6 Обработка исключения

6 Закрытие открытой транзакции

6 Передача исключения

Граф теста приведён на рисунке 113.



Рисунок 113 – Граф теста «Смена адреса заказа»

Цикломатическое число равно 4 (3 предикатных узла + 1).

Тестовые пути:

1. 1-6-5
2. 1-2-6-5
3. 1-2-3-6-5
4. 1-2-3-4-5

Тестовые варианты:

Тестовый вариант 1:

ИД: Адрес не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 114. На клиентском приложении невозможно воссоздать данный тестовый вариант, так как пользователь выбирает из адресов.



Рисунок 114 – Тестирование тестового варианта 1 «Смена адреса заказа» на СУБД

Тестовый вариант 2:

ИД: Адрес существует

И Заказ не существует

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 115. На клиентском приложении невозможно воссоздать данный тестовый вариант, так как форму Адреса инициирует форма Заказ из адресов.



Рисунок 115 – Тестирование тестового варианта 2 «Смена адреса заказа» на СУБД

Тестовый вариант 3:

ИД: Адрес существует

И Заказ существует

И Статус заказа не «Собирается»

ОР: Вызов исключения и завершение работы

Тестирование в СУБД представлено на рисунке 116. На клиентском приложении невозможно воссоздать данный тестовый вариант, так как кнопка смены адреса в собранном заказе становится неактивной.



Рисунок 116 – Тестирование тестового варианта 3 «Смена адреса заказа» на СУБД

Тестовый вариант 4:

ИД: Адрес существует

И Заказ существует

И Статус заказа «Собирается»

ОР: Изменение данных и завершение работы

Тестирование в СУБД представлено на рисунке 117. На клиентском приложении данный тестовый вариант отобразится изменение адреса на вкладке Заказ.



Рисунок 117 – Тестирование тестового варианта 4 «Смена адреса заказа» на СУБД

Степень покрытия 4/4 \* 100% = 100%

**Тестирование черным ящиком**

**Анализ классов эквивалентности**

Не все процедуры из тех, что были разработаны, можно проверить методом анализа классов эквивалентности. Тестовые варианты были протестированы в СУБД.

**Регистрация клиента**

Таблица 2 – Тестирование чёрным ящиком процедуры Регистрация клиента

|  |  |  |
| --- | --- | --- |
| **Описание** | **Допустимый класс** | **Недопустимый класс** |
| Длина логина | Длина логина ∈ [5;30] | Длина логина < 5  Длина логина > 30 |
| Длина имени | Длина имени ∈ [3;20] | Длина имени < 3  Длина имени > 20 |
| Телефон | Телефон соответствует паттерну | Телефон не соответствует паттерну |
| Почта | Почта соответствует паттерну | Почта не соответствует паттерну |

Количество тестовых вариантов в данном случае 4\*6 = 24. По тестовым вариантам было построено дерево разбиений. Оно представлено на рисунке 118.



Рисунок 118 – Дерево разбиений процедуры Регистрация клиента

По результатам получаются 7 тестовых вариантов. Результаты тестирования приведены на рисунках 119-125.

Тестовый вариант 1:

Исходные данные (ИД): Длина логина = 2

Длина имени = 4

Телефон соответствует паттерну

Почта соответствует паттерну

Ожидаемый результат (ОР): Ошибка



Рисунок 119 – Тестирование процедуры Регистрация клиента (ТВ1)

Тестовый вариант 2:

Исходные данные (ИД): Длина логина = 50

Длина имени = 4

Телефон соответствует паттерну

Почта соответствует паттерну

Ожидаемый результат (ОР): Ошибка



Рисунок 120 – Тестирование процедуры Регистрация клиента (ТВ2)

Тестовый вариант 3:

Исходные данные (ИД): Длина логина = 20

Длина имени = 2

Телефон соответствует паттерну

Почта соответствует паттерну

Ожидаемый результат (ОР): Ошибка



Рисунок 121 – Тестирование процедуры Регистрация клиента (ТВ3)

Тестовый вариант 4:

Исходные данные (ИД): Длина логина = 20

Длина имени = 30

Телефон соответствует паттерну

Почта соответствует паттерну

Ожидаемый результат (ОР): Ошибка



Рисунок 122 – Тестирование процедуры Регистрация клиента (ТВ4)

Тестовый вариант 5:

Исходные данные (ИД): Длина логина = 20

Длина имени = 4

Телефон не соответствует паттерну

Почта соответствует паттерну

Ожидаемый результат (ОР): Ошибка



Рисунок 123 – Тестирование процедуры Регистрация клиента (ТВ5)

Тестовый вариант 6:

Исходные данные (ИД): Длина логина = 20

Длина имени = 4

Телефон соответствует паттерну

Почта не соответствует паттерну

Ожидаемый результат (ОР): Ошибка



Рисунок 124 – Тестирование процедуры Регистрация клиента (ТВ6)

Тестовый вариант 7:

Исходные данные (ИД): Длина логина = 20

Длина имени = 4

Телефон соответствует паттерну

Почта соответствует паттерну

Ожидаемый результат (ОР): Сохранение данных



Рисунок 125 – Тестирование процедуры Регистрация клиента (ТВ7)

### **Анализ граничных значений**

**Регистрация клиента**

Длина логина имеет граничные значения [5;30]. Следовательно, следует проверить значения 4 (вне диапазона), 5 (граница), 6 (внутри диапазона), 29 (внутри диапазона), 30 (граница) и 31 (вне диапазона). Получилось 6 тестовых вариантов.

Длина имени имеет граничные значение [3;20]. Следовательно, следует проверить значения 2 (вне диапазона), 3 (граница), 4 (внутри диапазона), 19 (внутри диапазона), 20 (граница) и 21 (вне диапазона). Получилось 6 тестовых вариантов.

Паттерн телефона не имеет граничных значений.

Паттерн почты не имеет граничных значений.

Тестирование граничных значений приведено на рисунках 126-



Рисунок 126 – Тестирование процедуры Регистрация клиента (Длина логина 4)



Рисунок 127 – Тестирование процедуры Регистрация клиента (Длина логина 5)



Рисунок 128 – Тестирование процедуры Регистрация клиента (Длина логина 6)



Рисунок 129 – Тестирование процедуры Регистрация клиента (Длина логина 29)



Рисунок 130 – Тестирование процедуры Регистрация клиента (Длина логина 30)



Рисунок 131 – Тестирование процедуры Регистрация клиента (Длина логина 31)



Рисунок 132 – Тестирование процедуры Регистрация клиента (Длина имени 2)



Рисунок 133 – Тестирование процедуры Регистрация клиента (Длина имени 3)



Рисунок 134 – Тестирование процедуры Регистрация клиента (Длина имени 4)



Рисунок 135 – Тестирование процедуры Регистрация клиента (Длина имени 19)



Рисунок 136 – Тестирование процедуры Регистрация клиента (Длина имени 20)



Рисунок 131 – Тестирование процедуры Регистрация клиента (Длина имени 21)

### **Анализ причинно-следственных связей**

**Регистрация клиента**

Метод причинно-следственного анализа не применим к данной процедуре, так как данные заносятся только в случае отсутствия ошибки. Никакие другие факторы не влияют на работу процедуры.

# **ЗАКЛЮЧЕНИЕ**

В ходе данной работы были получены навыки создания клиент-серверного приложения на C#. Был произведён анализ выбранной предметной области, выявление требований и бизнес-правил предметной области и создание клиент-серверного приложения по предметной области «Доставка еды».

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Клиент-серверная архитектура в картинках // Хабр URL: https://habr.com/ru/articles/495698/ (дата обращения: 17.06.25)
2. Обучающие ресурсы по SQL // Microsoft Learn URL: https://learn.microsoft.com/ru-ru/sql/sql-server/educational-sql-resources?view=sql-server-ver17 (дата обращения: 17.06.25)

# **ПРИЛОЖЕНИЕ А: сценарий создания БД**

USE master

CREATE DATABASE Delivery

ON

(NAME = 'Delivery',FILENAME = 'C:\Databases\Delivery.mdf',SIZE = 1,MAXSIZE = 10,FILEGROWTH = 1)

LOG ON

(NAME = 'Delivery\_log',FILENAME = 'C:\Databases\Delivery\_log.ldf',SIZE = 1,MAXSIZE = 5,FILEGROWTH = 1)

GO

USE Delivery

GO

CREATE TYPE PhoneNumber

FROM char(12)

GO

CREATE RULE PhoneTamplate

AS @value LIKE '+7[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'

GO

EXEC sp\_bindrule 'PhoneTamplate', 'PhoneNumber';

GO

CREATE TYPE GradeValue

FROM smallint

GO

CREATE RULE GradeRange

AS @value BETWEEN 0 AND 5

GO

EXEC sp\_bindrule 'GradeRange', 'GradeValue';

GO

CREATE RULE EmailTemplate

AS @value LIKE '%[\_a-zA-Z0-9.-]%@%[\_a-zA-Z0-9.-]%.[a-zA-Z][a-zA-Z]%'

GO

CREATE DEFAULT RightNow

AS GETDATE()

GO

CREATE DEFAULT DefaultOneValue

AS 1

GO

CREATE DEFAULT DefaultZeroValue

AS 0

GO

-- Client

CREATE TABLE "Client"

(

ID int IDENTITY(1,1) PRIMARY KEY,

Login nvarchar(30) NOT NULL,

Password binary(32) NOT NULL,

Name nvarchar(20) NOT NULL,

Phone PhoneNumber NOT NULL,

Email nvarchar(50),

Created datetime NOT NULL,

"Active Account" tinyint NOT NULL,

CONSTRAINT "C\_Unique\_Phones" UNIQUE ("Phone"),

CONSTRAINT "C\_Unique\_Login" UNIQUE ("Login")

);

EXEC sp\_bindefault 'RightNow', 'Client.Created'

EXEC sp\_bindefault 'DefaultOneValue', 'Client.Active Account'

GO

-- Client Address

CREATE TABLE "Client Address"

(

[ID] int IDENTITY(1, 1) NOT NULL,

[Client ID] int NOT NULL,

City nvarchar(50) NOT NULL,

District nvarchar(50) NOT NULL,

Street nvarchar(50) NOT NULL,

Building nvarchar(6) NOT NULL,

Room nvarchar(5) NOT NULL,

Active tinyint,

CONSTRAINT "C\_PK\_ClientAddress\_Address" PRIMARY KEY ("ID"),

CONSTRAINT "C\_Unique\_ClientAdress" UNIQUE ("Client ID", "City", "District", "Street", "Building", "Room"),

CONSTRAINT "C\_FK\_Login" FOREIGN KEY ("Client ID")

REFERENCES "Client" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE

);

EXEC sp\_bindefault 'DefaultOneValue', 'Client Address.Active'

-- Producer

CREATE TABLE "Producer"

(

"ID" int IDENTITY(1, 1) NOT NULL,

"Login" nvarchar(30) NOT NULL,

"Password" binary(32) NOT NULL,

"Name" nvarchar(30) NOT NULL,

"Grade" GradeValue,

"City" nvarchar(50) NOT NULL,

"District" nvarchar(50) NOT NULL,

"Street" nvarchar(50) NOT NULL,

"Building" nvarchar(6) NOT NULL,

"Room" nvarchar(5) NOT NULL,

CONSTRAINT "C\_PK\_ProducerID" PRIMARY KEY ("ID"),

CONSTRAINT "C\_U\_Login" UNIQUE ("Login")

);

-- Status

CREATE TABLE "Status"

(

"ID" smallint NOT NULL,

"Value" nvarchar(50) NOT NULL,

CONSTRAINT "C\_PK\_StatusID" PRIMARY KEY ("ID")

);

-- Courier

CREATE TABLE "Courier"

(

"ID" int IDENTITY(1, 1) NOT NULL,

"First Name" nvarchar(30) NOT NULL,

"Second Name" nvarchar(30) NOT NULL,

"Last Name" nvarchar(30) NOT NULL,

"Phone" PhoneNumber NOT NULL,

"Passport Number" char(10) NOT NULL,

"Work Book" char(7) NOT NULL,

CONSTRAINT "C\_CourierID" PRIMARY KEY ("ID"),

CONSTRAINT "C\_U\_Passport\_Number" UNIQUE ("Passport Number"),

CONSTRAINT "C\_U\_Work\_Book" UNIQUE ("Work Book")

);

-- ProducerDishes

CREATE TABLE "Dish"

(

"ID" int IDENTITY(1, 1) NOT NULL,

"Producer ID" int NOT NULL,

"Name" nvarchar(50) NOT NULL,

"Image" varbinary(MAX),

"Cost" money NOT NULL,

"Description" nvarchar(100),

"Calories" int NOT NULL,

"Mass" int NOT NULL,

"Visible" tinyint,

CONSTRAINT "C\_PK\_DishID" PRIMARY KEY ("ID"),

CONSTRAINT "C\_U\_Dish" UNIQUE ("Name", "Producer ID"),

CONSTRAINT "C\_FK\_ProducerID\_Producer ID" FOREIGN KEY ("Producer ID")

REFERENCES Producer ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE

);

EXEC sp\_bindefault 'DefaultOneValue', 'Dish.Visible'

-- Order

CREATE TABLE "Order"

(

"ID" int IDENTITY(1, 1) NOT NULL,

"Client ID" int NOT NULL,

"Client Address ID" int NOT NULL,

"Ordered At" datetime,

"Complited At" datetime,

"Status" smallint NOT NULL,

"Order Grade" GradeValue,

CONSTRAINT "C\_PK\_Order" PRIMARY KEY ("ID"),

CONSTRAINT "C\_FK\_OrderStatus" FOREIGN KEY ("Status")

REFERENCES "Status" ("ID")

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT "C\_FK\_Client\_Address" FOREIGN KEY ("Client Address ID")

REFERENCES "Client Address" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE,

CONSTRAINT "C\_FK\_Client" FOREIGN KEY ("Client ID")

REFERENCES "Client" ("ID")

ON UPDATE NO ACTION

ON DELETE NO ACTION

);

EXEC sp\_bindefault 'DefaultZeroValue', 'Order.Status'

GO

CREATE TABLE "Order Courier"

(

"Order ID" int NOT NULL,

"Courier ID" int NOT NULL,

CONSTRAINT "C\_PK\_Order\_Courier" PRIMARY KEY ("Order ID"),

CONSTRAINT "C\_FK\_Order" FOREIGN KEY ("Order ID")

REFERENCES "Order" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE,

CONSTRAINT "C\_FK\_Courier" FOREIGN KEY ("Courier ID")

REFERENCES "Courier" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE,

)

-- Dishes-Order

CREATE TABLE "Dishes Order"

(

"Order ID" int NOT NULL,

"Dish ID" int NOT NULL,

"Count" integer NOT NULL,

CONSTRAINT "C\_PK\_DishID\_OrderID" PRIMARY KEY ("Dish ID", "Order ID"),

CONSTRAINT "C\_FK\_Dishes" FOREIGN KEY ("Dish ID")

REFERENCES "Dish" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE,

CONSTRAINT "C\_FK\_OrderID" FOREIGN KEY ("Order ID")

REFERENCES "Order" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE,

);

EXEC sp\_bindefault 'DefaultOneValue', 'Dishes Order.Count'

CREATE TABLE "Ingredient"

(

"Ingredient ID" int IDENTITY(1, 1) NOT NULL,

"Ingredient Name" nvarchar(30) NOT NULL,

CONSTRAINT "C\_PK\_Ingredient" PRIMARY KEY ("Ingredient ID"),

CONSTRAINT "C\_U\_IngredientName" UNIQUE ("Ingredient Name")

);

CREATE TABLE "Dish Ingredients"

(

"Ingredient ID" int NOT NULL,

"Dish ID" int NOT NULL,

CONSTRAINT "C\_PK\_Dish\_Ingredient" PRIMARY KEY ("Ingredient ID", "Dish ID"),

CONSTRAINT "C\_FK\_Dish" FOREIGN KEY ("Dish ID")

REFERENCES "Dish" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE,

CONSTRAINT "C\_FK\_Ingredient" FOREIGN KEY ("Ingredient ID")

REFERENCES "Ingredient" ("Ingredient ID")

ON UPDATE CASCADE

ON DELETE CASCADE

);

GO

CREATE TABLE Notifications

(

"Client ID" int NOT NULL,

"Value" nvarchar(50) NOT NULL,

"Send" datetime NOT NULL,

"Read" smallint NOT NULL,

CONSTRAINT "C\_PK\_Notification" PRIMARY KEY ("Client ID", "Send", "Value"),

CONSTRAINT "C\_FK\_NotifyClient" FOREIGN KEY ("Client ID")

REFERENCES "Client" ("ID")

ON UPDATE CASCADE

ON DELETE CASCADE

)

EXEC sp\_bindefault 'RightNow', 'Notifications.Send'

EXEC sp\_bindefault 'DefaultZeroValue', 'Notifications.Read'

GO

INSERT INTO "Status" ("ID", "Value")

VALUES

(0,'Собирается'),

(1,'Обрабатывается'),

(2,'Готовится'),

(3,'Доставляется'),

(4,'Завершён'),

(5,'Отменён');

GO

CREATE TRIGGER StatusBlock ON Status

AFTER INSERT, UPDATE, DELETE

AS ROLLBACK TRAN;

# **ПРИЛОЖЕНИЕ Б: сценарий создания триггеров БД**

USE Delivery

GO

CREATE OR ALTER TRIGGER CheckIfAddressBelongsToClient ON "Order"

AFTER INSERT, UPDATE

AS

BEGIN

BEGIN TRY

DECLARE @Client int, @Address int;

DECLARE OrderAddressCursor CURSOR

SCROLL

FOR SELECT [Client ID], [Client Address ID] FROM inserted;

OPEN OrderAddressCursor

FETCH FIRST FROM OrderAddressCursor

INTO @Client, @Address

WHILE(@@FETCH\_STATUS = 0)

BEGIN

IF NOT EXISTS (SELECT \* FROM [Client Address] WHERE ID = @Address AND [Client ID] = @Client)

THROW 50020, 'Address doesn''t belong to client', 1

FETCH NEXT FROM OrderAddressCursor

INTO @Client, @Address

END

CLOSE OrderAddressCursor

DEALLOCATE OrderAddressCursor

END TRY

BEGIN CATCH

IF CURSOR\_STATUS('variable', '@OrderAddressCursor') >= 0

CLOSE OrderAddressCursor;

DEALLOCATE OrderAddressCursor;

THROW;

END CATCH

END

GO

CREATE OR ALTER TRIGGER OneProducerInSingleOreder ON "Dishes Order"

AFTER INSERT

AS

BEGIN

DECLARE @Order int;

DECLARE @FoundDifferentProds tinyint;

SET @FoundDifferentProds = 0

DECLARE OrderCursorVar CURSOR

SCROLL

FOR SELECT DISTINCT [Order ID] FROM inserted;

OPEN OrderCursorVar

FETCH FIRST FROM OrderCursorVar

INTO @Order

WHILE(@@FETCH\_STATUS = 0 AND @FoundDifferentProds = 0)

BEGIN

IF((

SELECT COUNT(DISTINCT [Producer ID])

FROM Dish RIGHT JOIN (

SELECT [Dish ID]

FROM [Dishes Order]

WHERE [Order ID] = @Order

) AS OrderDishes

ON Dish.ID = OrderDishes.[Dish ID]

) != 1)

SET @FoundDifferentProds = 1

FETCH NEXT FROM OrderCursorVar

INTO @Order

END

CLOSE OrderCursorVar

DEALLOCATE OrderCursorVar

IF (@FoundDifferentProds != 0) THROW 50010, 'Different producers in the same order', 1;

END

GO

CREATE OR ALTER TRIGGER OnlyOneOpenedOrder ON [Order]

AFTER INSERT

AS

BEGIN

IF ((

SELECT COUNT(\*)

FROM [Order] JOIN inserted ON [Order].[Client ID] = inserted.[Client ID]

WHERE [Order].[Ordered At] IS NULL AND [Order].[Status] = 0

) > 1) THROW 50011, 'More than one opened order', 1;

END

GO

CREATE OR ALTER TRIGGER ClearOrderOnEmpty ON "Dishes Order"

AFTER DELETE

AS

BEGIN

DECLARE @OrderID int;

DECLARE DeltedOrdersCursorVar CURSOR

SCROLL

FOR SELECT DISTINCT [Order ID] FROM deleted

OPEN DeltedOrdersCursorVar

FETCH FIRST FROM DeltedOrdersCursorVar

INTO @OrderID

WHILE(@@FETCH\_STATUS = 0)

BEGIN

IF((SELECT COUNT(\*) FROM [Dishes Order] WHERE [Order ID] = @OrderID) = 0)

BEGIN

DELETE FROM [Order]

WHERE [Order].ID = @OrderID

END

FETCH NEXT FROM DeltedOrdersCursorVar

INTO @OrderID

END

CLOSE DeltedOrdersCursorVar

DEALLOCATE DeltedOrdersCursorVar

END

GO

CREATE OR ALTER TRIGGER UpdateStatusMessage ON [Order]

AFTER UPDATE

AS

BEGIN

DECLARE @Order int, @Status int, @Client int;

DECLARE MessageOnOrderCursor CURSOR

SCROLL

FOR SELECT inserted.[ID], inserted.[Status], inserted.[Client ID] FROM inserted JOIN deleted ON inserted.[Status] != deleted.[Status]

OPEN MessageOnOrderCursor

FETCH FIRST FROM MessageOnOrderCursor

INTO @Order, @Status, @Client

WHILE(@@FETCH\_STATUS = 0)

BEGIN

IF(@Status BETWEEN 1 AND 5)

BEGIN

DECLARE @Message nvarchar(50);

SELECT @Message = CASE @Status

WHEN 1 THEN 'Order#' + CONVERT(varchar(10), @Order) + ' is processing'

WHEN 2 THEN 'Order#' + CONVERT(varchar(10), @Order) + ' is beeing prepared'

WHEN 3 THEN 'Order#' + CONVERT(varchar(10), @Order) + ' is beeing delivered'

WHEN 4 THEN 'Order#' + CONVERT(varchar(10), @Order) + ' is complited'

WHEN 5 THEN 'Order#' + CONVERT(varchar(10), @Order) + ' is canceled'

END

INSERT INTO Notifications([Client ID],[Value]) VALUES(@Client, @Message);

END

FETCH NEXT FROM MessageOnOrderCursor

INTO @Order, @Status, @Client

END

CLOSE MessageOnOrderCursor

DEALLOCATE MessageOnOrderCursor

END

GO

CREATE OR ALTER TRIGGER DestroyOrderAfterAddressHide ON [Client Address]

AFTER UPDATE

AS

BEGIN

BEGIN TRY

DECLARE @Address int;

DECLARE AddressCursor CURSOR

SCROLL

FOR SELECT [ID] FROM inserted WHERE Active = 0

OPEN AddressCursor

FETCH FIRST FROM AddressCursor

INTO @Address

WHILE(@@FETCH\_STATUS = 0)

BEGIN

DECLARE @Order int;

DECLARE @Status int;

DECLARE OrdersCursor CURSOR

SCROLL

FOR SELECT ID, [Status] FROM [Order] WHERE [Client Address ID] = @Address

OPEN OrdersCursor

FETCH FIRST FROM OrdersCursor

INTO @Order, @Status

WHILE(@@FETCH\_STATUS = 0)

BEGIN

IF (@Status = 0)

THROW 50013, 'Current order address can''not be deleted', 1

FETCH NEXT FROM OrdersCursor

INTO @Order, @Status

END

CLOSE OrdersCursor

DEALLOCATE OrdersCursor

FETCH NEXT FROM AddressCursor

INTO @Address

END

CLOSE AddressCursor

DEALLOCATE AddressCursor

END TRY

BEGIN CATCH

IF CURSOR\_STATUS('variable', '@OrdersCursor') >= 0

CLOSE OrdersCursor;

DEALLOCATE OrdersCursor;

IF CURSOR\_STATUS('variable', '@AddressCursor') >= 0

CLOSE AddressCursor;

DEALLOCATE AddressCursor;

THROW;

END CATCH

END

GO

# **ПРИЛОЖЕНИЕ В: сценарий создания представлений БД**

USE Delivery

GO

CREATE OR ALTER VIEW [Order Status Table]

AS

SELECT [Order].ID, Client.[Login] AS [Client Login],[Client Address ID],[Ordered At],[Complited At], [Status].ID AS [Status ID], [Status].Value AS [Status Value],[Order Grade], City, District, Street, Building, Room

FROM (([Order] JOIN Status ON [Order].Status = Status.ID) JOIN Client ON [Order].[Client ID] = Client.ID) JOIN [Client Address] ON [Order].[Client Address ID] = [Client Address].ID

GO

CREATE OR ALTER VIEW [Address By Login]

AS

SELECT Client.[Login] AS [Client Login], [Password], [Active Account], [Client Address].ID AS [Address ID], [City],[District],

[Street],[Building],[Room],[Active] AS [Active Address]

FROM Client JOIN [Client Address] ON Client.ID = [Client Address].[Client ID]

GO

CREATE OR ALTER VIEW [Order Set]

AS

SELECT [Order].ID AS [Order ID], Client.[Login] AS [Client Login], [Status] AS [Order Status],[Status].[Value] AS [Order Status Value], [Dish ID], [Count], Dish.[Name] As [Dish Name], Producer.[Name] AS [Producer Name], [Image], Cost, Mass, Calories, ([Count]\*Cost) AS [Sum]

FROM (((([Order] RIGHT JOIN [Status] ON [Order].[Status] = [Status].ID)

RIGHT JOIN [Dishes Order] ON [Order].ID = [Dishes Order].[Order ID])

LEFT JOIN Dish ON [Dishes Order].[Dish ID] = Dish.ID)

LEFT JOIN Client ON [Order].[Client ID] = Client.ID)

LEFT JOIN Producer ON Dish.[Producer ID] = Producer.ID

GO

CREATE OR ALTER VIEW [All Dishes]

AS

SELECT Dish.ID AS [Dish ID], Producer.ID AS [Producer ID], Dish.[Name] AS [Dish Name], [Image], Cost, [Description], Calories, Mass, Producer.[Name] AS [Producer Name]

FROM Dish LEFT JOIN Producer ON Dish.[Producer ID] = Producer.ID

WHERE Dish.Visible = 1

WITH CHECK OPTION

GO

CREATE OR ALTER VIEW [Order On Producer]

AS

SELECT DISTINCT [Order ID], [Producer ID], Producer.Name AS [Producer Name]

FROM ([Dishes Order] LEFT JOIN Dish ON [Dishes Order].[Dish ID]=Dish.ID) LEFT JOIN Producer ON Dish.[Producer ID] = Producer.ID

GO

CREATE OR ALTER VIEW [Dish All Info]

AS

SELECT Dish.ID, [Producer ID], Producer.Name AS [Producer Name], Dish.Name AS [Dish Name], Dish.Image, Dish.Cost, Dish.Calories, Dish.Description, Ingredient.[Ingredient Name]

FROM (Dish LEFT JOIN Producer ON Dish.[Producer ID] = Producer.ID) LEFT JOIN([Dish Ingredients] LEFT JOIN Ingredient ON [Dish Ingredients].[Ingredient ID] = Ingredient.[Ingredient ID])

ON [Dish Ingredients].[Dish ID] = Dish.Id

GO

# **ПРИЛОЖЕНИЕ Г: сценарий создания хранимых процедур БД**

USE Delivery

GO

CREATE OR ALTER PROCEDURE RegisterClient(@login nvarchar(30), @password binary(32), @name nvarchar(20), @phone PhoneNumber, @email nvarchar(50))

AS

BEGIN

BEGIN TRY

IF NOT(LEN(@login) BETWEEN 5 AND 30)

THROW 50000, 'Login should be more than 5 and less than 30 letters', 1;

IF EXISTS (SELECT \* FROM Client WHERE Login = @login)

THROW 50001, 'Login already exists', 1;

IF NOT(LEN(@name) BETWEEN 3 AND 20)

THROW 50002, 'Name should be more than 2 and less than 20 letters', 1;

IF EXISTS (SELECT \* FROM Client WHERE Phone = @phone)

THROW 50003, 'Phone already used', 1;

IF NOT (@phone LIKE '+7[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

THROW 50004, 'Phone doesn''t match pattern', 1;

IF NOT (@email LIKE '%[\_a-zA-Z0-9.-]%@%[\_a-zA-Z0-9.-]%.[a-zA-Z][a-zA-Z]%')

THROW 50005, 'Email doesn''t match pattern', 1;

BEGIN TRAN

INSERT INTO Client(Login,Password,Name,Phone,Email)

VALUES (@login, @password, @name, @phone, @email);

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE DeleteClient(@login nvarchar(30), @password binary(32))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'Client with login doesn''t exists', 1;

BEGIN TRAN

UPDATE Client

SET [Active Account] = 0

WHERE Login = @login AND Password = @password AND [Active Account] = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE ChangeClientLogin(@login nvarchar(30), @password binary(32), @newLogin nvarchar(30))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'User doesn''t exist', 1;

IF EXISTS (SELECT \* FROM Client WHERE Login = @newLogin)

THROW 50001, 'User with this login already exist', 1;

IF NOT(LEN(@newLogin) BETWEEN 5 AND 30)

THROW 50000, 'Login should be more than 5 and less than 30 letters', 1;

BEGIN TRAN

UPDATE Client

SET Login = @newLogin

WHERE Login = @login AND Password = @password AND [Active Account] = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE ChangeClientName(@login nvarchar(30), @password binary(32), @newName nvarchar(20))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'User doesn''t exist', 1;

IF NOT(LEN(@newName) BETWEEN 3 AND 20)

THROW 50002, 'Name should be more than 2 and less than 20 letters', 1;

BEGIN TRAN

UPDATE Client

SET Name = @newName

WHERE Login = @login AND Password = @password AND [Active Account] = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE ChangeClientPhone(@login nvarchar(30), @password binary(32), @newPhone PhoneNumber)

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'User doesn''t exist', 1;

IF EXISTS (SELECT \* FROM Client WHERE Phone = @newPhone)

THROW 50003, 'Phone already used', 1;

IF NOT (@newPhone LIKE '+7[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

THROW 50004, 'Phone doesn''t match pattern', 1;

BEGIN TRAN

UPDATE Client

SET Phone = @newPhone

WHERE Login = @login AND Password = @password AND [Active Account] = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE ChangeClientEmail(@login nvarchar(30), @password binary(32), @newEmail nvarchar(50))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'User doesn''t exist', 1;

IF NOT (@newEmail LIKE '%[\_a-zA-Z0-9.-]%@%[\_a-zA-Z0-9.-]%.[a-zA-Z][a-zA-Z]%')

THROW 50005, 'Email doesn''t match pattern', 1;

BEGIN TRAN

UPDATE Client

SET Email = @newEmail

WHERE Login = @login AND Password = @password AND [Active Account] = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE ChangeClientPassword(@login nvarchar(30), @oldPassword binary(32), @newPassword binary(32))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @oldPassword AND [Active Account] = 1)

THROW 50006, 'User dosn''t exist', 1;

BEGIN TRAN

UPDATE Client

SET Password = @newPassword

WHERE Login = @login AND Password = @oldPassword AND [Active Account] = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE AddClientAddress(@login nvarchar(30), @password binary(32), @city nvarchar(50), @district nvarchar(50),

@street nvarchar(50), @building nvarchar(10), @room nvarchar(10))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'Client with login doesn''t exists', 1;

DECLARE @clientID int;

SELECT @clientID = ID FROM Client

WHERE Login = @login AND Password = @password AND [Active Account] = 1

IF EXISTS (SELECT \* FROM [Client Address] WHERE [Client ID] = @clientID AND City = @City AND

District = @district AND Street = @street AND Building = @building AND Room = @room AND Active = 1)

THROW 50014, 'Address already exists', 1;

BEGIN TRAN

IF EXISTS (SELECT \* FROM [Client Address] WHERE [Client ID] = @clientID AND City = @City AND

District = @district AND Street = @street AND Building = @building AND Room = @room AND Active = 0)

BEGIN

UPDATE [Client Address]

SET Active = 1

WHERE [Client ID] = @clientID AND City = @City AND District = @district

AND Street = @street AND Building = @building AND Room = @room AND Active = 0

END

ELSE

BEGIN

INSERT INTO [Client Address] ([Client ID],City, District, Street, Building, Room)

VALUES (@clientID, @city, @district, @street, @building, @room)

END

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE EditClientAddress(@address int, @login nvarchar(30), @password binary(32), @city nvarchar(50),

@district nvarchar(50), @street nvarchar(50), @building nvarchar(10), @room nvarchar(10))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'Client with login doesn''t exists', 1;

DECLARE @clientID int;

SELECT @clientID = ID FROM Client

WHERE Login = @login AND Password = @password AND [Active Account] = 1

IF NOT EXISTS (SELECT \* FROM [Client Address] WHERE [Client ID] = @clientID AND ID = @address AND Active = 1)

THROW 50007, 'Address doesn''t exists', 1;

IF EXISTS (SELECT \* FROM [Client Address] WHERE [Client ID] = @clientID AND City = @City AND

District = @district AND Street = @street AND Building = @building

AND Room = @room AND Active = 1 AND ID != @address)

THROW 50014, 'Address already exists', 1;

BEGIN TRAN

UPDATE [Client Address]

SET City = @city, District = @district, Street = @street, Building = @building, Room = @room

WHERE ID = @address AND [Client ID] = @clientID AND [Active] = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE DeleteClientAddress(@addressID int, @login nvarchar(30), @password binary(32))

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM Client WHERE Login = @login AND Password = @password AND [Active Account] = 1)

THROW 50006, 'Client with login doesn''t exists', 1;

DECLARE @clientID int;

SELECT @clientID = ID FROM Client

WHERE Login = @login AND Password = @password AND [Active Account] = 1

IF NOT EXISTS (SELECT \* FROM [Client Address] WHERE [Client ID] = @clientID AND ID = @addressID AND Active = 1)

THROW 50007, 'Address doesn''t exist', 1;

BEGIN TRAN

UPDATE [Client Address]

SET Active = 0

WHERE [Client ID] = @clientID AND ID = @addressID AND Active = 1

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE InitOrder(@login nvarchar(30), @addressID int)

AS

BEGIN

BEGIN TRY

IF NOT EXISTS(SELECT \* FROM Client WHERE [Login] = @login AND [Active Account] = 1)

THROW 50006, 'Client with login doesn''t exists', 1;

DECLARE @clientID int;

SELECT @clientID = ID FROM Client

WHERE [Login] = @login AND [Active Account] = 1

BEGIN TRAN

INSERT INTO [Order] ([Client ID], [Client Address ID])

VALUES (@ClientID, @AddressID);

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE AddToOrder(@OrderID int, @dishID int, @Count int)

AS

BEGIN

BEGIN TRY

IF NOT EXISTS(SELECT \* FROM [Order] WHERE ID = @OrderID)

THROW 50009, 'Order doesn''t exists', 1;

BEGIN TRAN

IF NOT EXISTS(SELECT \* FROM [Dishes Order] WHERE [Dish ID] = @DishID AND [Order ID] = @OrderID)

BEGIN

INSERT INTO [Dishes Order] ([Dish ID],[Order ID],[Count])

VALUES (@DishID, @OrderID, @Count)

END

ELSE

BEGIN

UPDATE [Dishes Order]

SET [Count] = [Count] + @Count

WHERE [Dish ID] = @DishID AND [Order ID] = @OrderID

END

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

GO

CREATE OR ALTER PROCEDURE DeleteFromOrder(@Order int, @dishID int)

AS

BEGIN

BEGIN TRY

IF NOT EXISTS(SELECT \* FROM [Dishes Order] WHERE [Order ID] = @Order AND [Dish ID] = @dishID)

THROW 50012, 'Dish doesn''t found in order', 1;

BEGIN TRAN

IF((SELECT [Count] FROM [Dishes Order] WHERE [Order ID] = @Order AND [Dish ID] = @dishID) = 1)

BEGIN

DELETE FROM [Dishes Order]

WHERE [Order ID] = @Order AND [Dish ID] = @DishID

END

ELSE

BEGIN

UPDATE [Dishes Order]

SET [Count] = [Count] - 1

WHERE [Order ID] = @Order AND [Dish ID] = @DishID

END

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROCEDURE ApplyOrder(@Order int)

AS

BEGIN

BEGIN TRY

IF NOT EXISTS(SELECT \* FROM [Order] WHERE ID = @Order AND [Ordered At] IS NULL AND [Status] = 0)

THROW 50008, 'Order can''t be assembled', 1;

BEGIN TRAN

UPDATE [Order]

SET [Ordered At] = GETDATE(), [Status] = 1

WHERE ID = @Order

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

CREATE OR ALTER PROC ChangeOrderAddress(@Order int, @Address int)

AS

BEGIN

BEGIN TRY

IF NOT EXISTS (SELECT \* FROM [Client Address] WHERE ID = @Address AND Active = 1)

THROW 50007, 'Address doesn''t exists', 1;

IF NOT EXISTS(SELECT \* FROM [Order] WHERE ID = @Order)

THROW 50009, 'Order doesn''t exists', 1;

IF EXISTS(SELECT \* FROM [Order] WHERE ID = @Order AND Status != 0)

THROW 50012, 'Order already applied; Address unchangeable', 1;

BEGIN TRAN

UPDATE [Order]

SET [Client Address ID] = @Address

WHERE ID = @Order

COMMIT TRAN

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRAN;

THROW;

END CATCH

END

GO

# 

# **ПРИЛОЖЕНИЕ Д: сценарий заполнения таблиц БД**

USE Delivery

GO

INSERT INTO "Producer"

("Login", "Password", "Name", "Grade", "City", "District", "Street", "Building", "Room")

VALUES

('bella\_pizza', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Пицца Белла', 5, 'Москва', 'Центральный', 'Тверская', '15', '201'),

('sushi\_master', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Суши Мастер', 4, 'Санкт-Петербург', 'Центральный', 'Невский проспект', '22А', '305'),

('utro\_bakery', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Пекарня Утро', NULL, 'Москва', 'Западный', 'Кутузовский проспект', '12Б', '101'),

('vegan\_house', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Веган Хаус', 4, 'Казань', 'Приволжский', 'Баумана', '7', 'A5'),

('wok\_empire', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Wok Империя', 5, 'Екатеринбург', 'Верх-Исетский', 'Малышева', '34В', '412'),

('coffee\_bean', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Кофейное Зерно', NULL, 'Краснодар', 'Центральный', 'Красная', '56', '12'),

('dumpling\_king', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Король Пельменей', 3, 'Новосибирск', 'Октябрьский', 'Ленина', '11/2', '204'),

('grill\_hero', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Гриль Герой', 4, 'Самара', 'Промышленный', 'Гагарина', '77', '501'),

('fresh\_salad', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Свежий Салат', 5, 'Ростов-на-Дону', 'Советский', 'Большая Садовая', '25', '1А'),

('donut\_world', CONVERT(BINARY(32),'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'), 'Мир Пончиков', 4, 'Челябинск', 'Металлургический', 'Свободы', '8К1', '317');

INSERT INTO "Courier"

("First Name", "Second Name", "Last Name", "Phone", "Passport Number", "Work Book")

VALUES

(N'Андрей', N'Викторович', N'Иванов', '+79161234567', '4512123456', 'ТК-1001'),

(N'Ольга', N'Сергеевна', N'Петрова', '+79031234568', '4017123457', 'ТК-1002'),

(N'Дмитрий', N'Игоревич', N'Соколов', '+79261234569', '4500123458', 'ТК-1003'),

(N'Екатерина', N'Андреевна', N'Козлова', '+79111234570', '4623123459', 'ТК-1004'),

(N'Илья', N'Петрович', N'Новиков', '+79091234571', '4800123460', 'ТК-1005'),

(N'Анна', N'Владимировна', N'Морозова', '+79251234572', '4900123461', 'ТК-1006'),

(N'Сергей', N'Николаевич', N'Волков', '+79131234573', '4312123462', 'ТК-1007'),

(N'Мария', N'Дмитриевна', N'Лебедева', '+79021234574', '4100123463', 'ТК-1008'),

(N'Павел', N'Олегович', N'Кузнецов', '+79081234575', '4455123464', 'ТК-1009'),

(N'Наталья', N'Игоревна', N'Васнецова', '+79191234576', '4600123465', 'ТК-1010');

INSERT INTO "Dish"

("Producer ID", "Name", "Image", "Cost", "Description", "Calories", "Mass", "Visible")

VALUES

(1, N'Пицца "Маргарита"', NULL, 650.00, N'Классическая итальянская пицца с моцареллой и томатами', 820, 550, 1),

(1, N'Пицца "Пепперони"', NULL, 790.00, N'Острая с колбасой пепперони и сырным бортиком', 950, 600, 1),

(2, N'Ролл "Филадельфия"', NULL, 480.00, N'С лососем, сливочным сыром и огурцом', 320, 240, 1),

(2, N'Сет "Самурай"', NULL, 1200.00, N'12 роллов с тунцом, угрем и креветкой', 650, 450, 1),

(3, N'Круассан с миндалем', NULL, 150.00, N'Слоеная выпечка с миндальной начинкой', 280, 120, 1),

(3, N'Багет "Деревенский"', NULL, 90.00, N'Свежий французский багет', 210, 300, 0),

(4, N'Бургер с тофу', NULL, 340.00, N'Веганский бургер с овощами и соусом', 400, 350, 1),

(4, N'Смузи "Зеленый"', NULL, 220.00, N'Шпинат, киви и яблоко', 180, 400, 1),

(5, N'Лапша с говядиной', NULL, 450.00, N'Жареная лапша с овощами и соевым соусом', 520, 400, 1),

(5, N'Wok с креветками', NULL, 680.00, N'С тигровыми креветками и кунжутом', 480, 380, 1),

(6, N'Капучино "Премиум"', NULL, 180.00, N'Двойная порция эспрессо с молочной пенкой', 120, 300, 1),

(6, N'Тирамису', NULL, 240.00, N'Классический итальянский десерт', 380, 150, 1),

(7, N'Пельмени "Сибирские"', NULL, 290.00, N'С говядиной и свининой, 15 шт', 550, 300, 1),

(7, N'Бульон куриный', NULL, 120.00, N'С домашней лапшой и зеленью', 150, 350, 0),

(8, N'Стейк Рибай', NULL, 890.00, N'Мраморная говядина на гриле', 720, 400, 1),

(8, N'Картофель "Гратен"', NULL, 210.00, N'Запеченный с сыром и сливками', 450, 200, 1),

(9, N'Салат "Цезарь"', NULL, 320.00, N'С курицей и пармезаном', 280, 250, 1),

(9, N'Салат "Греческий"', NULL, 270.00, N'С фетаксой и оливками', 210, 220, 1),

(10, N'Пончик "Карамельный"', NULL, 90.00, N'С карамельной глазурью', 320, 80, 1),

(10, N'Пончик "Шоколадный"', NULL, 95.00, N'С темным шоколадом и посыпкой', 350, 85, 1);

# **ПРИЛОЖЕНИЕ Ж: сценарий тестирования хранимых процедур БД**

/\* Белый ящик \*/

EXEC RegisterClient 'User', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', NULL

EXEC RegisterClient 'UserLogin', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', NULL

EXEC RegisterClient 'AnotherLogin', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Na', '+77777777771', NULL

EXEC RegisterClient 'AnotherLogin', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', NULL

EXEC RegisterClient 'AnotherLogin', x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+777777777', NULL

EXEC RegisterClient 'AnotherLogin', x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777772', 'MyEmail@mailru'

EXEC RegisterClient 'AnotherLogin', x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777779', NULL

EXEC DeleteClient 'RandomLogin', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92

EXEC DeleteClient 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92

EXEC ChangeClientLogin 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Evdok'

EXEC ChangeClientLogin 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'UserLogin'

EXEC ChangeClientLogin 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'User'

EXEC ChangeClientLogin 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'TestUser'

EXEC ChangeClientName 'Evdokimov', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Evdok'

EXEC ChangeClientName 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Al'

EXEC ChangeClientName 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Alex'

EXEC ChangeClientPhone 'Evdokimov', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, '+79605899969'

EXEC ChangeClientPhone 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, '+77777777777'

EXEC ChangeClientPhone 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, '+777777775'

EXEC ChangeClientPhone 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, '+77777777789'

EXEC ChangeClientEmail 'Evdokimov', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'test@mail.ru'

EXEC ChangeClientEmail 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'test@mailru'

EXEC ChangeClientEmail 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'test@mail.ru'

DECLARE @newPass binary(32);

SET @newPass = CAST('new pass' AS BINARY(32))

EXEC ChangeClientPassword 'Evdokimov', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, @newPass

EXEC ChangeClientPassword 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, @newPass

EXEC AddClientAddress 'Evdokimov', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Рязань', 'Советский', 'Советской Армии', '10A', '10'

EXEC AddClientAddress 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Рязань', 'Советский', 'Советской Армии', '10A', '10'

EXEC AddClientAddress 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Рязань', 'Советский', 'Советской Армии', '10A', '10'

EXEC EditClientAddress 13, 'Evdokimov', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Рязань', 'Советский', 'Советской Армии', '10A', '10'

EXEC EditClientAddress 5, 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Рязань', 'Советский', 'Советской Армии', '10A', '10'

EXEC EditClientAddress 16, 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Рязань', 'Советский', 'Советской Армии', '10Б', '10'

EXEC DeleteClientAddress 16, 'Evdokimov', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92

EXEC DeleteClientAddress 25, 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92

EXEC DeleteClientAddress 20, 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92

EXEC InitOrder 'Evdokimov', 20

EXEC InitOrder 'Evdokim', 21

EXEC AddToOrder 80, 5, 1

EXEC AddToOrder 21, 5, 1

EXEC AddToOrder 21, 5, 5

EXEC DeleteFromOrder 80, 5

EXEC AddToOrder 21, 6, 1

EXEC DeleteFromOrder 21, 6

EXEC DeleteFromOrder 21, 5

EXEC ApplyOrder 80

EXEC ApplyOrder 21

EXEC AddClientAddress 'Evdokim', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Рязань', 'Советский', 'Советская', '222', '80'

EXEC InitOrder 'Evdokim', 21

EXEC ChangeOrderAddress 22, 220

EXEC ChangeOrderAddress 220, 22

EXEC ChangeOrderAddress 21, 22

EXEC ChangeOrderAddress 22, 22

/\* Чёрный ящик \*/

/\* Эквивалентности \*/

EXEC RegisterClient 'Ur', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestUserLoginWith50SymbolsShouldFailProcedureCase1', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestUserBasicLogin12', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Na', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestUserBasicLogin12', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'TooLongNameTestItsShouldBeErro', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestUserBasicLogin12', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+7777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestUserBasicLogin12', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mailru'

EXEC RegisterClient 'TestUserBasicLogin12', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

/\* Граничные условия \*/

EXEC RegisterClient 'Test', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test5', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test56', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestWith29SymbolsTestStringTe', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestWith29SymbolsTestStringTes', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'TestWith29SymbolsTestStringTest', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test56', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Na', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test56', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Nam', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test56', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'Name', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test56', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'TestNameWith19Symbo', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test56', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'TestNameWith20Symbol', '+77777777777', 'mymail@mail.com'

EXEC RegisterClient 'Test56', 0x8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92, 'TestNameWith21Symbols', '+77777777777', 'mymail@mail.com'

# **ПРИЛОЖЕНИЕ З: сценарий заполнения таблиц БД изображениями**

using System;

using System.Data;

using System.Data.SqlClient;

using System.IO;

public static class Fill

{

private static string path = "C:\\images\\";

private static string ext = ".jpg";

private static void Main()

{

for (int i = 0; i < 20; i++)

{

SaveImageToDatabase(i);

}

}

public static void SaveImageToDatabase(int number)

{

byte[] imageData = File.ReadAllBytes(path + number.ToString() + ext);

string fileName = Path.GetFileName(path + number.ToString() + ext);

using (SqlConnection connection = new SqlConnection("" +

"Data Source = LAPTOP\\SQLEXPRESS; Initial Catalog = Delivery; Integrated Security = true;"))

{

string sql = "UPDATE Dish SET Image = @data WHERE ID = @id";

SqlCommand command = new SqlCommand(sql, connection);

command.Parameters.Add("@data", SqlDbType.VarBinary).Value = imageData;

command.Parameters.Add("@id", SqlDbType.Int).Value = number;

connection.Open();

command.ExecuteNonQuery();

}

}

}

# 

# **ПРИЛОЖЕНИЕ К: исходный текст клиентского приложения – класс Database**

using System;

using System.Data;

using System.Data.SqlClient;

using System.Data.SqlTypes;

using System.Collections.Generic;

using DeliveryApp.EF;

using System.Linq;

namespace DeliveryApp

{

static class Database

{

public static bool IsValidUser(string Login, byte[] Password)

{

using (var context = new DeliveryAppContext())

{

return context.Client.Where(p => p.Login == Login && p.Password == Password && p.Active\_Account == 1).Count() != 0;

}

}

public static void InsertUser(string Login, byte[] Password, string Name, string Phone, string Email)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

var name = new SqlParameter("@name", Name);

var phone = new SqlParameter("@phone", Phone);

SqlParameter email = Email.Length == 0

? new SqlParameter("@email", SqlInt32.Null)

: new SqlParameter("@email", Email);

context.Database.ExecuteSqlCommand("exec RegisterClient @login, @password, @name, @phone, @email",

login, password, name, phone, email);

}

}

public static List<Order\_Status\_Table> GetClosedUserOrders(string Login)

{

using (var context = new DeliveryAppContext())

{

return context.Order\_Status\_Table.Where(p => p.Client\_Login == Login && p.Status\_ID != 0).ToList();

}

}

public static Client GetUserInfo(string Login, byte[] Password)

{

using (var context = new DeliveryAppContext())

{

return context.Client.Where(p => p.Login == Login && p.Password == Password).First();

}

}

public static void ChangePassword(string Login, byte[] OldPassword, byte[] NewPassword)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var oldPassword = new SqlParameter("@oldPassword", OldPassword);

var newPassword = new SqlParameter("@newPassword", NewPassword);

context.Database.ExecuteSqlCommand("exec ChangeClientPassword @login, @oldPassword, @newPassword",

login, oldPassword, newPassword);

}

}

public static void ChangeLogin(string Login, byte[] Password, string NewLogin)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

var newLogin = new SqlParameter("@newLogin", NewLogin);

context.Database.ExecuteSqlCommand("exec ChangeClientLogin @login, @password, @newLogin",

login, password, newLogin);

}

}

public static void ChangeName(string Login, byte[] Password, string NewName)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

var newName = new SqlParameter("@newName", NewName);

context.Database.ExecuteSqlCommand("exec ChangeClientName @login, @password, @newName",

login, password, newName);

}

}

public static void ChangePhone(string Login, byte[] Password, string NewPhone)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

var newPhone = new SqlParameter("@newPhone", NewPhone);

context.Database.ExecuteSqlCommand("exec ChangeClientPhone @login, @password, @newPhone",

login, password, newPhone);

}

}

public static void ChangeEmail(string Login, byte[] Password, string NewEmail)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

var newEmail = new SqlParameter("@newEmail", NewEmail);

context.Database.ExecuteSqlCommand("exec ChangeClientEmail @login, @password, @newEmail",

login, password, newEmail);

}

}

public static void DeleteAccount(string Login, byte[] Password)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

context.Database.ExecuteSqlCommand("exec DeleteClient @login, @password", login, password);

}

}

public static List<Address\_By\_Login> GetAddresses(string Login, byte[] Password)

{

using (var context = new DeliveryAppContext())

{

return context.Address\_By\_Login.Where(p => p.Client\_Login == Login && p.Password == Password && p.Active\_Address == 1).ToList();

}

}

public static List<Order\_Set> GetOrderSet(int order)

{

using (var context = new DeliveryAppContext())

{

return context.Order\_Set.Where(p => p.Order\_ID == order).ToList();

}

}

public static List<All\_Dishes> GetAllDishes()

{

using (var context = new DeliveryAppContext())

{

return context.All\_Dishes.ToList();

}

}

public static bool HasOpenedOrder(string Login)

{

using (var context = new DeliveryAppContext())

{

return context.Order\_Status\_Table.Where(p => p.Client\_Login == Login && p.Status\_ID == 0).Count() == 1;

}

}

public static int GetOpenedOrderID(string Login)

{

using (var context = new DeliveryAppContext())

{

return context.Order\_Status\_Table.Where(p => p.Client\_Login == Login && p.Status\_ID == 0).First().ID;

}

}

public static void InitOrder(string Login, int Address)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var address = new SqlParameter("@address", Address);

context.Database.ExecuteSqlCommand("exec InitOrder @login, @address",login, address);

}

}

public static void AddToOrder(int OrderID, int DishID, int Count)

{

using (var context = new DeliveryAppContext())

{

var order = new SqlParameter("@orderID", OrderID);

var dish = new SqlParameter("@dishID", DishID);

var count = new SqlParameter("@count", Count);

context.Database.ExecuteSqlCommand("exec AddToOrder @orderID, @dishID, @count", order, dish, count);

}

}

public static void DeleteFromOrder(int OrderID, int DishID)

{

using (var context = new DeliveryAppContext())

{

var order = new SqlParameter("@orderID", OrderID);

var dish = new SqlParameter("@dishID", DishID);

context.Database.ExecuteSqlCommand("exec DeleteFromOrder @orderID, @dishID", order, dish);

}

}

public static void ApplyOrder(string Login)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@orderID", Login);

context.Database.ExecuteSqlCommand("exec ApplyOrder @login", login);

}

}

public static void AddAdress(string Login, byte[] Password, string City, string District, string Street, string Building, string Room)

{

using (var context = new DeliveryAppContext())

{

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

var city = new SqlParameter("@city", City);

var district = new SqlParameter("@district", District);

var street = new SqlParameter("@street", Street);

var building = new SqlParameter("@building", Building);

var room = new SqlParameter("@room", Room);

context.Database.ExecuteSqlCommand("exec AddClientAddress @login, @password, @city, @district, @street, @building, @room",

login, password, city, district, street, building, room);

}

}

public static void DeleteAddress(int Address, string Login, byte[] Password)

{

using (var context = new DeliveryAppContext())

{

var address = new SqlParameter("@address", Address);

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

context.Database.ExecuteSqlCommand("exec DeleteClientAddress @address, @login, @password",

address, login, password);

}

}

public static void EditAdress(int Address, string Login, byte[] Password, string City, string District, string Street, string Building, string Room)

{

using (var context = new DeliveryAppContext())

{

var address = new SqlParameter("@address", Address);

var login = new SqlParameter("@login", Login);

var password = new SqlParameter("@password", Password);

var city = new SqlParameter("@city", City);

var district = new SqlParameter("@district", District);

var street = new SqlParameter("@street", Street);

var building = new SqlParameter("@building", Building);

var room = new SqlParameter("@room", Room);

context.Database.ExecuteSqlCommand("exec EditClientAddress @address, @login, @password, @city, @district, @street, @building, @room",

address, login, password, city, district, street, building, room);

}

}

public static void ApplyOrder(int Order)

{

using (var context = new DeliveryAppContext())

{

var order = new SqlParameter("@order", Order);

context.Database.ExecuteSqlCommand("exec ApplyOrder @order", order);

}

}

public static int GetAddressIdByOrder(int Order)

{

using (var context = new DeliveryAppContext())

{

return context.Order\_Status\_Table.Where(p => p.ID == Order).First().Client\_Address\_ID;

}

}

public static void ChangeAddressInOrder(int Order, int NewAddress)

{

using (var context = new DeliveryAppContext())

{

var order = new SqlParameter("@order", Order);

var address = new SqlParameter("@address", NewAddress);

context.Database.ExecuteSqlCommand("exec ChangeOrderAddress @order, @address", order, address);

}

}

public static List<Order\_On\_Producer> GetProducerByOrder(int Order)

{

using (var context = new DeliveryAppContext())

{

return context.Order\_On\_Producer.Where(p => p.Order\_ID == Order).ToList();

}

}

public static List<Dish\_All\_Info> GetDishInfo(int dishID)

{

using (var context = new DeliveryAppContext())

{

return context.Dish\_All\_Info.Where(p => p.ID == dishID).ToList();

}

}

public static Order\_Status\_Table GetAddressOfOrder(int Order)

{

using (var context = new DeliveryAppContext())

{

return context.Order\_Status\_Table.Where(p => p.ID == Order).First();

}

}

}

}

# **ПРИЛОЖЕНИЕ Л: исходный текст клиентского приложения – класс ClientActions**

using System;

using System.Data.SqlClient;

using System.Security.Cryptography;

using System.Text;

using System.Collections.Generic;

using DeliveryApp.EF;

namespace DeliveryApp

{

class ClientActions

{

private static byte[] HashString(string Input)

{

using (SHA256 sha256 = SHA256.Create())

{

byte[] InputBytes = Encoding.UTF8.GetBytes(Input);

return sha256.ComputeHash(InputBytes);

}

}

static public void Login(string Login, string Password)

{

if (Login.Length == 0 || Password.Length == 0) throw new Exception("Есть незаполненные поля");

try

{

byte[] password = HashString(Password);

if (!Database.IsValidUser(Login, password))

throw new Exception("Неправильный логин или пароль");

User.userInfo = new User(Login, password);

}

catch (SqlException ex)

{

throw new Exception($"Database error {ex.Number}\n{ex.Message}");

}

}

public static string FormatPhone(string phone)

{

string[] charsToRemoveFromPhone = new string[] { "(", ")", "-", " " };

foreach (string c in charsToRemoveFromPhone)

phone = phone.Replace(c, string.Empty);

if (phone[0] == '8')

{

phone = "+7" + phone;

}

return phone;

}

public static void Register(string Login, string Password, string Name, string Phone, string Email)

{

if (Password.Length < 5 || Password.Length > 16)

throw new Exception("Пароль должен быть от 5 до 16 символов");

byte[] password = HashString(Password);

Phone = FormatPhone(Phone);

try

{

Database.InsertUser(Login, password, Name, Phone, Email);

User.userInfo = new User(Login, password);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50000:

throw new Exception("Логин должен быть от 5 до 30 символов");

case 50001:

throw new Exception("Пользователь с таким логином уже существует");

case 50002:

throw new Exception("Имя должно быть от 3 до 20 символов");

case 50003:

throw new Exception("Телефон уже используется");

case 50004:

throw new Exception("Телефон заполнен неверно");

case 50005:

throw new Exception("Почта заполнена неверно");

default:

throw new Exception($"Database error {ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Processing error. Try again\n{ex.Message}");

}

}

public static List<Order\_Status\_Table> GetClosedUserOrders(string Login)

{

try

{

return Database.GetClosedUserOrders(Login);

}

catch(Exception ex)

{

throw new Exception($"Error retriving orders\n{ex.Message}");

}

}

public static Client getClientInfo()

{

try

{

return Database.GetUserInfo(User.userInfo.Login, User.userInfo.Password);

}

catch(Exception ex)

{

throw new Exception($"Error retriving user data\n{ex.Message}");

}

}

public static void ChangePassword(string NewPassword, string RetypedPassword)

{

if (NewPassword.Length < 5 || NewPassword.Length > 16) throw new

Exception("Пароль должен быть от 5 до 16 символов");

if (NewPassword != RetypedPassword) throw new Exception("Пароли не совпадают");

byte[] newPassword = HashString(NewPassword);

if (User.userInfo.Password == newPassword)

throw new Exception("Новый и старый пароли одинаковы");

try

{

Database.ChangePassword(User.userInfo.Login, User.userInfo.Password, newPassword);

User.userInfo.Password = newPassword;

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50006:

throw new Exception("Пользователь с логином и паролем не существует");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during changing password\n{ex.Message}");

}

}

public static void ChangeLogin(string NewLogin)

{

try

{

Database.ChangeLogin(User.userInfo.Login, User.userInfo.Password, NewLogin);

User.userInfo.Login = NewLogin;

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50000:

throw new Exception("Логин должен быть от 5 до 30 символов");

case 50001:

throw new Exception("Логин уже существует");

case 50006:

throw new Exception("Пользователь не существует");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during changing password\n{ex.Message}");

}

}

public static void ChangeName(string NewName)

{

try

{

Database.ChangeName(User.userInfo.Login, User.userInfo.Password, NewName);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50002:

throw new Exception("Имя должно быть от 3 до 20 символов");

case 50006:

throw new Exception("Неправильный логин или пароль");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during changing name\n{ex.Message}");

}

}

public static void ChangePhone(string NewPhone)

{

try

{

NewPhone = FormatPhone(NewPhone);

Database.ChangePhone(User.userInfo.Login, User.userInfo.Password, NewPhone);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50003:

throw new Exception("Телефон уже используется");

case 50004:

throw new Exception("Телефон заполнен неверно");

case 50006:

throw new Exception("Пользователь не существует");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during changing phone\n{ex.Message}");

}

}

public static void ChangeEmail(string NewEmail)

{

try

{

Database.ChangeEmail(User.userInfo.Login, User.userInfo.Password, NewEmail);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50005:

throw new Exception("Почта заполнена неверно");

case 50006:

throw new Exception("Пользователь не существует");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during changing email\n{ex.Message}");

}

}

public static void DeleteAccount()

{

try

{

Database.DeleteAccount(User.userInfo.Login, User.userInfo.Password);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50006:

throw new Exception("Пользователь не существет");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during deleting account\n{ex.Message}");

}

}

public static List<Address\_By\_Login> GetAddresses()

{

try

{

return Database.GetAddresses(User.userInfo.Login, User.userInfo.Password);

}

catch(Exception ex)

{

throw new Exception($"Error during reciving addresses\n{ex.Message}");

}

}

public static List<All\_Dishes> GetAllDishes()

{

try

{

return Database.GetAllDishes();

}

catch(Exception ex)

{

throw new Exception($"Error during reciving dishes\n{ex.Message}");

}

}

public static bool HasOpenedOrder()

{

try

{

return Database.HasOpenedOrder(User.userInfo.Login);

}

catch(Exception ex)

{

throw new Exception($"Error during reciving order\n{ex.Message}");

}

}

public static int GetOpenOrderID()

{

try

{

return Database.GetOpenedOrderID(User.userInfo.Login);

}

catch(InvalidOperationException)

{

throw new Exception($"Адрес не выбран");

}

catch(Exception ex)

{

throw new Exception($"Error during reciving order\n{ex.Message}");

}

}

public static void InitOrder(int Address)

{

try {

Database.InitOrder(User.userInfo.Login, Address);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50006:

throw new Exception("Пользователь с данным логином не существует");

case 50011:

throw new Exception("У Вас более одного заказа");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during initing order\n{ex.Message}");

}

}

public static void AddToOrder(int OrderID, int DishID)

{

try

{

Database.AddToOrder(OrderID, DishID, 1);

}

catch(SqlException ex)

{

switch (ex.Number)

{

case 50009:

throw new Exception("Заказ не создан");

case 50010:

throw new Exception("Разные блюда в одном заказе не могут находиться");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during adding dish to order\n{ex.Message}");

}

}

public static void DeleteFromOrder(int OrderID, int DishID)

{

try

{

Database.DeleteFromOrder(OrderID, DishID);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50012:

throw new Exception("Блюдо не найдено в заказе");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during deleting dish from order\n{ex.Message}");

}

}

public static void AddAdress(string City, string District, string Street, string Building, string Room)

{

try

{

Database.AddAdress(User.userInfo.Login, User.userInfo.Password, City, District, Street, Building, Room);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50006:

throw new Exception("Пользователь не найден");

case 50014:

throw new Exception("Адрес уже существует");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during adding address\n{ex.Message}");

}

}

public static void DeleteAddress(int Address)

{

try

{

Database.DeleteAddress(Address, User.userInfo.Login, User.userInfo.Password);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50006:

throw new Exception("Пользователь не существует");

case 50007:

throw new Exception("Адрес не существует");

case 50013:

throw new Exception("Адрес находится в заказе");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during deleting address\n{ex.Message}");

}

}

public static void EditAddress(int Address, string City, string District, string Street, string Building, string Room)

{

try

{

Database.EditAdress(Address, User.userInfo.Login, User.userInfo.Password, City, District, Street, Building, Room);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50006:

throw new Exception("Пользователь не существует");

case 50007:

throw new Exception("Адрес не существует");

case 50014:

throw new Exception("Адрес уже существует");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during deleting address\n{ex.Message}");

}

}

public static List<Order\_Set> GetOrderSet(int order)

{

try

{

return Database.GetOrderSet(order);

}

catch(Exception ex)

{

throw new Exception($"Error during reciving order data\n{ex.Message}");

}

}

public static void ApplyOrder(int Order)

{

try

{

Database.ApplyOrder(Order);

}

catch (SqlException ex)

{

switch (ex.Number)

{

case 50008:

throw new Exception("Заказ не может быть собран");

default:

throw new Exception($"Database error{ex.Number}\n{ex.Message}");

}

}

catch(Exception ex)

{

throw new Exception($"Error during applying order\n{ex.Message}");

}

}

public static int GetAddressIdByOrder(int Order)

{

try

{

return Database.GetAddressIdByOrder(Order);

}

catch(Exception ex)

{

throw new Exception($"Error reciving address by order\n{ex.Message}");

}

}

public static void ChangeAddressInOrder(int Order, int NewAddress)

{

try

{

Database.ChangeAddressInOrder(Order, NewAddress);

}

catch(Exception ex)

{

throw new Exception($"Error during changing address in order\n{ex.Message}");

}

}

public static List<Order\_On\_Producer> GetProducerByOrder(int Order)

{

try

{

return Database.GetProducerByOrder(Order);

}

catch(Exception ex)

{

throw new Exception($"Error during reciving producers by order\n{ex.Message}");

}

}

public static List<Dish\_All\_Info> GetDishInfo(int DishID)

{

try

{

return Database.GetDishInfo(DishID);

}

catch(Exception ex)

{

throw new Exception($"Error during reciving dish data\n{ex.Message}");

}

}

public static Order\_Status\_Table GetAddressOfOrder(int Order)

{

try

{

return Database.GetAddressOfOrder(Order);

}

catch(Exception ex)

{

throw new Exception($"Error during reciving address of order\n{ex.Message}");

}

}

}

}

# **ПРИЛОЖЕНИЕ М: исходный текст клиентского приложения – класс Program**

using System;

using System.Windows.Forms;

namespace DeliveryApp

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

(new LoginForm()).Show();

Application.Run();

}

}

}

# **ПРИЛОЖЕНИЕ Н: исходный текст клиентского приложения – класс User**

namespace DeliveryApp

{

class User

{

public string Login { get; set; }

public byte[] Password { get; set; }

public User(string login, byte[] password)

{

Login = login;

Password = password;

}

public static User userInfo = null;

}

}

# **ПРИЛОЖЕНИЕ О: исходный текст клиентского приложения – класс DeliveryAppContext**

namespace DeliveryApp.EF

{

using System.Data.Entity;

public partial class DeliveryAppContext : DbContext

{

public DeliveryAppContext() : base()

{

Database.SetInitializer<DeliveryAppContext>(null);

}

public virtual DbSet<Client> Client { get; set; }

public virtual DbSet<Client\_Address> Client\_Address { get; set; }

public virtual DbSet<Courier> Courier { get; set; }

public virtual DbSet<Dish> Dish { get; set; }

public virtual DbSet<Dishes\_Order> Dishes\_Order { get; set; }

public virtual DbSet<Ingredient> Ingredient { get; set; }

public virtual DbSet<Notifications> Notifications { get; set; }

public virtual DbSet<Order> Order { get; set; }

public virtual DbSet<Producer> Producer { get; set; }

public virtual DbSet<Status> Status { get; set; }

public virtual DbSet<Address\_By\_Login> Address\_By\_Login { get; set; }

public virtual DbSet<All\_Dishes> All\_Dishes { get; set; }

public virtual DbSet<Dish\_All\_Info> Dish\_All\_Info { get; set; }

public virtual DbSet<Order\_On\_Producer> Order\_On\_Producer { get; set; }

public virtual DbSet<Order\_Set> Order\_Set { get; set; }

public virtual DbSet<Order\_Status\_Table> Order\_Status\_Table { get; set; }

}

}