МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка базы данных для банка с использованием технологий шифрования и маскирования в БД»

Выполнил студент Метла Полина Георгиевна

(Ф.И.О.)

Руководитель проекта пр.-стаж. Нистюк О.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты пр.-стаж. Нистюк О.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер пр.-стаж. Нистюк О.А. ы

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2019

**Реферат**

Пояснительная записка курсового проекта содержит 47 страниц пояснительной записки, 46 иллюстраций, 9 источников литературы, 5 приложения.

C#, WPF, microsoft SQL SERVER 2017, БАЗА ДАННЫХ, ADO.NET

Целью выполнения курсового проекта является создание базы данных для банка, ознакомление и применение технологии шифрования и маскирования в базе данных, а также разработка приложения, которое должно показать возможности работы с базой данных

В первой главе проводится аналитический обзор литературы по тематике курсовой работы и содержит описание технологий, использованных во время выполнения проекта.

Вторая глава посвящена процессу проектирования системы.

В третьей главе описывается процесс разработки, принципы функционирования и назначение созданных компонент проекта.

В четвёртой главе описано тестирование используемых технологий и программного средства.

В пятой главе описано руководство пользователя, позволяющее подробно понять интерфейс программного средства.

В заключении приведены результаты проделанной работы.

**Содержание**

[Введение 5](#_Toc27214130)

[1 Аналитический обзор литературы и постановка задачи 6](#_Toc27214131)

[1.1 Обзор прототипов 6](#_Toc27214132)

[1.2 Анализ прототипов 7](#_Toc27214133)

[1.3 Описание используемых технологий 8](#_Toc27214134)

[2 Проектирование модели базы данных 10](#_Toc27214135)

[3 Реализация базы данных 12](#_Toc27214136)

[3.1 Таблицы 12](#_Toc27214137)

[3.2 Процедуры 17](#_Toc27214138)

[3.2.1 Процедуры для решения поставленных задач 17](#_Toc27214139)

[3.2.2 Процедура экспорта данных в XML-файл 17](#_Toc27214140)

[3.2.3 Процедура импорта данных из XML-файла 18](#_Toc27214141)

[3.3 Функции 18](#_Toc27214142)

[3.4 Триггеры 19](#_Toc27214143)

[3.5 Пользователи 19](#_Toc27214144)

[3.6 Шифрование и маскирование в базе данных 21](#_Toc27214145)

[4 Тестирование 23](#_Toc27214146)

[4.1 Тестирование используемой технологии 23](#_Toc27214147)

[4.2 Тестирование работоспособности приложения 24](#_Toc27214148)

[5 Руководство пользователя 28](#_Toc27214149)

[Заключение 37](#_Toc27214150)

[Список литературных источников 38](#_Toc27214151)

[Приложение А 39](#_Toc27214152)

[Приложение Б 41](#_Toc27214153)

[Приложение В 45](#_Toc27214154)

[Приложение Г 46](#_Toc27214155)

[Приложение Д 47](#_Toc27214156)

# **Введение**

В современном мире сталкиваться с такими учреждениями, как банки, приходится уже с юного возраста. У каждого второго подростка имеется своя банковская карта, даже и не одна.

На протяжении всей жизни человек пользуется услугами банка. Открытие счета, оформление банковских карт, обмен валют, перевод средств с одного счета на другой, выдача кредитов – все эти функции предоставляет нам учреждение банка.

Как бы далеко ни продвинулся научно-технический прогресс в 21 веке, мы все еще имеем проблему с очередями в отделениях банка. Электронные очереди и бронирование места в очереди через интернет полностью не могут решить эту проблему.

Современные приложения для предоставления банковских услуг могут помочь только с некоторыми тривиальными операциями. Также банковские приложения устроены так, что происходит только однонаправленное взаимодействие клиента с приложением. Это приводит к сложности использования клиентом приложения, ответственности за свои действия, боязни клиента потерять свои сбережения вследствие неверных действий, беспокойстве о безопасности проводимых транзакций. Опираясь на перечисленные проблемы, была сформулирована цель и задачи курсового проекта.

Целью моего курсового проекта является создание базы данных для банка, ознакомление и применение технологии шифрования и маскирования в базе данных, а также разработка приложения, которое должно показать возможности работы с базой данных.

Основными задачами курсовой работы являются:

* провести аналитический обзор литературы;
* спроектировать базу данных;
* реализовать функциональность базы данных;
* провести тестирование используемой технологии в базе данных;
* разработать приложение для работы с базой данных;
* написать руководство пользователя.

Для проектирования базы данных используется СУБД «Microsoft SQL Server 2017». В базе данных применяются технологии шифрования и маскирования. Приложение для работы с базой данных реализуется с помощью технологии WPF и языка программирования C#. Для того чтобы осуществлять связь между базой данных и приложением, используется технология ADO.NET.

1. **Аналитический обзор литературы и постановка задачи**

В соответствии с заданием курсового проекта следует не только создать базу данных, но и разработать программное средство, которое должно в полной мере показать возможности базы данных. Для этого необходимо проанализировать прототипы программных средств выбранной темы.

## **1.1 Обзор прототипов**

На сегодняшний день каждый банк имеет свое программное средство для предоставления услуг клиентам. Рассмотрим примеры web-приложений, сайтов известных банков.

Так как я являюсь владельцем карточки Беларусбанка и у меня подключен интернет-банкинг, мы можем проанализировать сайт интернет-банкинга Беларусбанка [1]. Основной список предоставляемых услуг находится во вкладке «Платежи и переводы» (рис. 1.1).

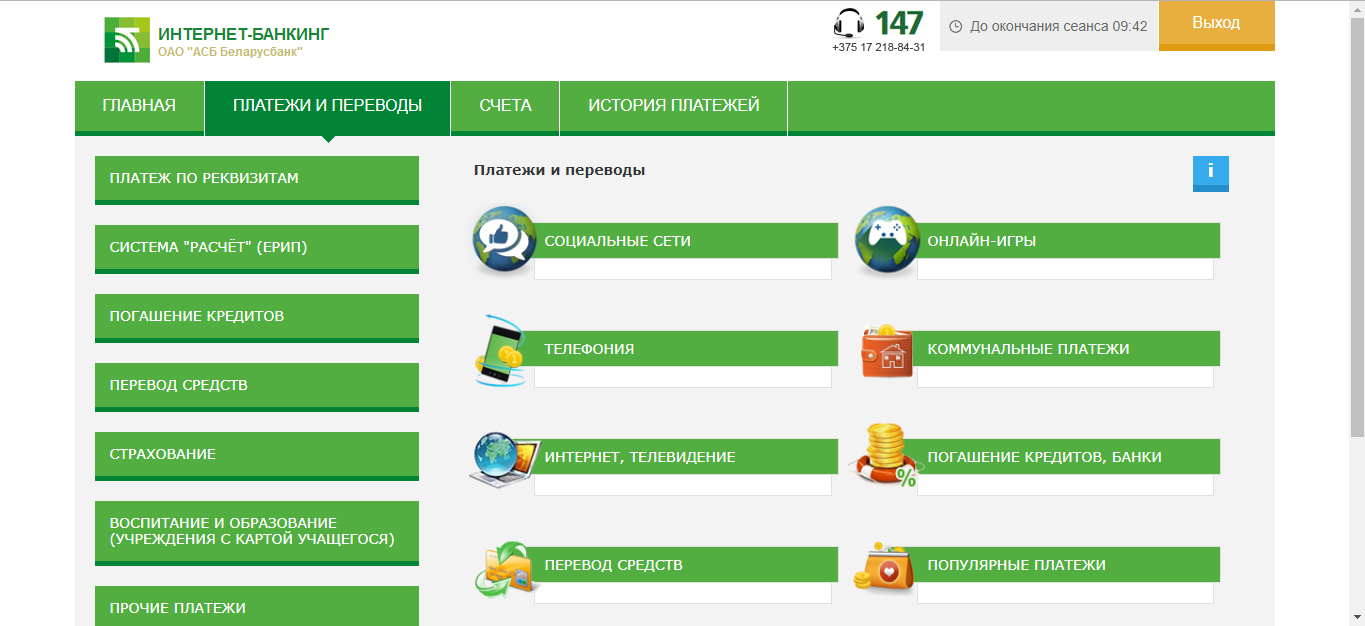


Рисунок 1.1 – Интерфейс интернет-банкинга Беларусбанка

Следующим примером рассмотрим сайт Альфа-банка [2]. Перейдем на вкладку «Банк онлайн», так как нас интересуют услуги, предоставляемые банком (рис. 1.2).

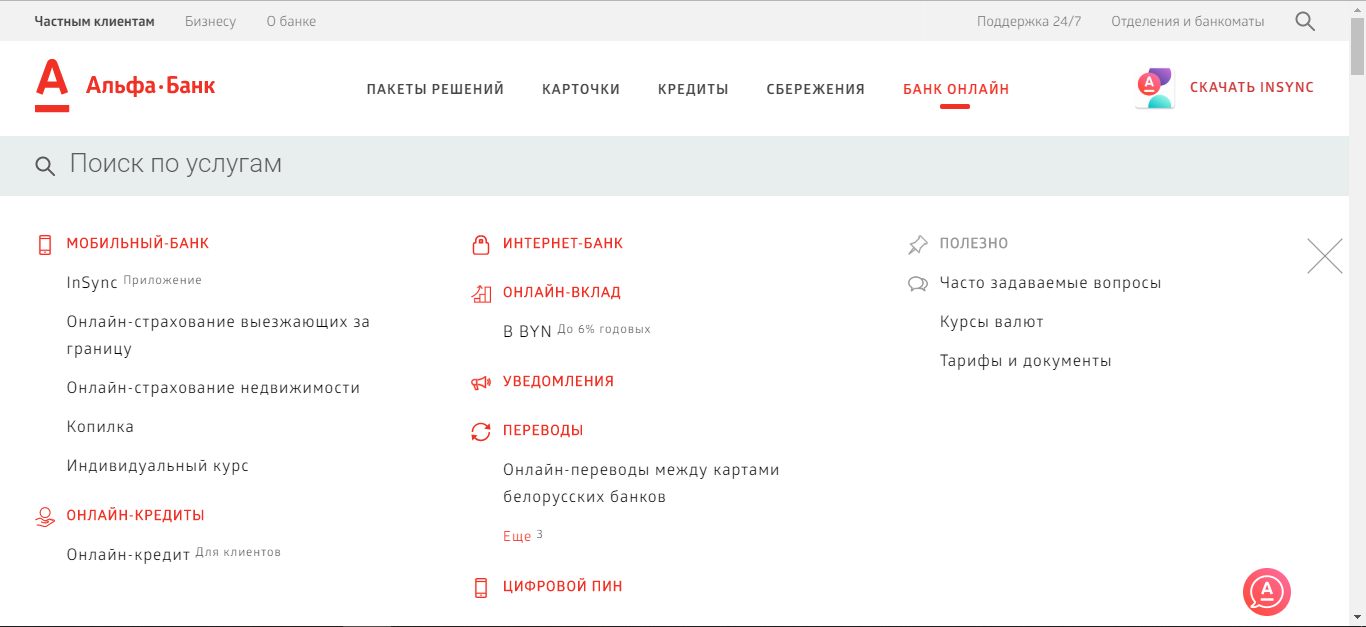


Рисунок 1.2 – Интерфейс сайта Альфа-банка

Далее перейдем на сайт Приорбанка [3]. Просмотрим вкладку «Наши продукты» (рис. 1.3).

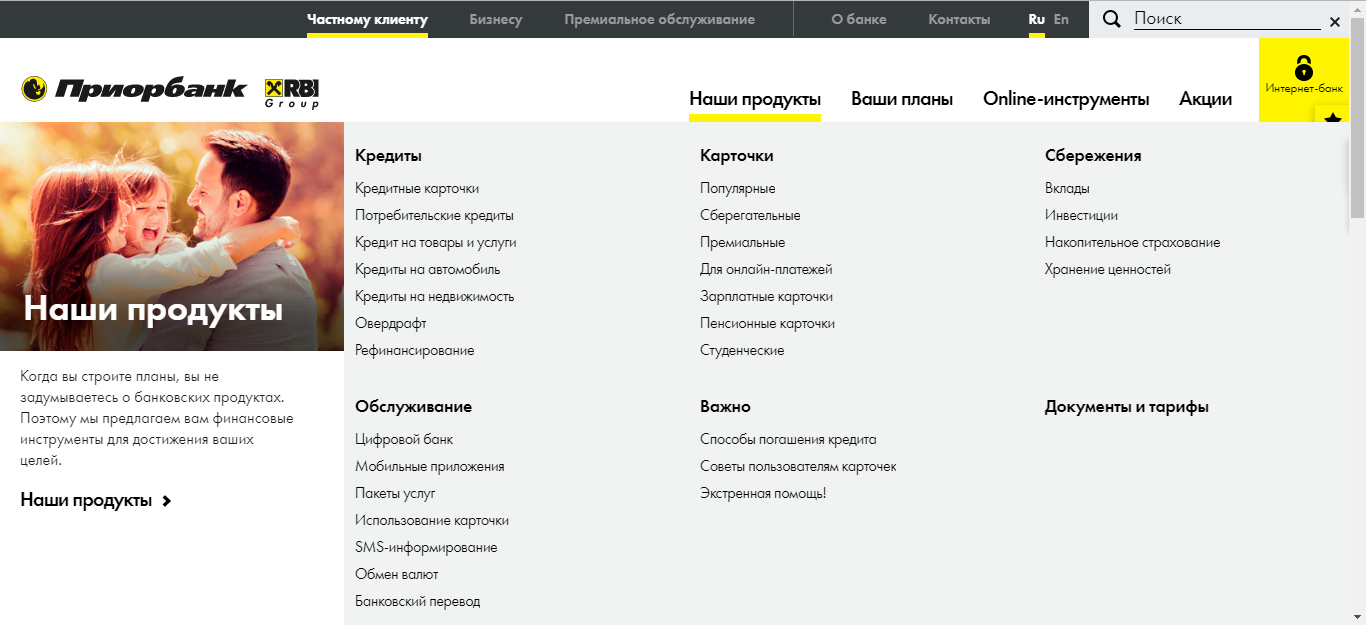


Рисунок 1.3 – Интерфейс сайта Приорбанка

## **1.2 Анализ прототипов**

Проанализировав прототипы сайтов различных банков, я заметила ожидаемую схожесть в списке предоставляемых услуг клиентам и выделила такие услуги, как оформление кредитных карт, кредитов, перевод средств, погашение кредитов и осуществление различных платежей.

Исходя из полученных результатов, я сформулировала основные функциональные требования курсовой работы:

* регистрация и авторизация клиентов;
* авторизация администратора;
* возможность оформления клиентом личного счета и банковской карты;
* возможность класть деньги на счет и снимать их;
* возможность перевода денег со счета на счет одного лица;
* возможность перевода денег со счета на счет другого лица;
* возможность перевода денег со счета на счет с обменом валют;
* возможность взятия кредита;
* возможность частичного и полного погашения кредита.

Данные требования будут реализованы в базе данных в виде хранимых процедур, которые описаны в главе 3 данной пояснительной записки.

## **1.3 Описание используемых технологий**

Для организации работы с базой данных было решено использовать одну из наиболее популярных СУБД «Microsoft SQL Server 2017» [4], так как она предоставляет необходимые возможности оперирования объектами базы данных.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс — Windows Presentation Foundation (WPF) [5], предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом. В основе графической технологии WPF лежит мощная инфраструктура, основанная на DirectX. Это является одним из основных отличий WPF от более ранней технологии создания пользовательских интерфейсов — Windows Forms.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом — С#, разработанный для создания приложений на платформе Microsoft .NET Framework [6].

Чтобы осуществлять связь между базой данных и приложением на C# необходим посредник. И именно таким посредником является технология ADO.NET [7]. ADO.NET представляет собой технологию работы с данными, которая основана на платформе .NET Framework. Эта технология представляет набор классов, через которые мы можем отправлять запросы к базам данных, установить подключения, получать ответ от базы данных и производить ряд других операций.

База данных для банка оперирует личной информацией клиентов, которую необходимо защитить. Для обеспечения защиты данных применяются технологии шифрования и маскирования.

Динамическое маскирование данных (DDM) [8] ограничивает возможность раскрытия конфиденциальных данных за счет маскирования этих данных для непривилегированных пользователей. Оно позволяет значительно упростить проектирование и написание кода для системы безопасности в приложении.

DDM можно настроить для отдельных полей базы данных, чтобы скрыть конфиденциальные данные в результирующих наборах запросов. При использовании DDM данные в базе данных не изменяются. Динамическое маскирование данных легко использовать с существующими приложениями, поскольку правила маскирования применяются к результатам запроса. Многие приложения могут маскировать конфиденциальные данные без изменения существующих запросов.

DDM включает функции полного и частичного маскирования, а также возможность использования случайной маски для числовых данных. Назначение и использование масок осуществляется простыми командами Transact-SQL.

Динамическое маскирование данных дополняет другие функции безопасности SQL Server (аудит, шифрование, безопасность на уровне строк и др.). Поэтому для лучшей защиты конфиденциальных данных в базе данных реализована технология шифрования [9].

SQL Server, как и другие распространенные коммерческие системы управления базами данных, располагает множеством вариантов шифрования, в том числе на уровне ячеек, базы данных и файлов через Windows, а также на транспортном уровне. Эти варианты шифрования обеспечивают безопасность информации на уровне базы данных и операционной системы. Кроме того, они снижают вероятность несанкционированного раскрытия конфиденциальных сведений, даже если поражены инфраструктура или база данных SQL Server.

Шифрование представляет собой способ скрытия данных с помощью ключа или пароля. Это делает данные бесполезными без соответствующего ключа или пароля для дешифрования. Шифрование не решает проблемы управления доступом. Однако оно повышает защиту за счет ограничения потери данных даже при обходе системы управления доступом. Например, если компьютер, на котором установлена база данных, был настроен неправильно и злоумышленник смог получить конфиденциальные данные, то украденная информация будет бесполезна, если она была предварительно зашифрована.

Реализация технологий шифрования и маскирования в базе данных для банка описана в главе 3, а тестирование – в главе 4.

1. **Проектирование модели базы данных**

Для базы данных банка были разработаны 12 таблиц, которые связаны друг с другом внешними ключами. На рисунке 2.1 представлена диаграмма структуры базы данных, разработанной в СУБД «Microsoft SQL Server 2017». База данных была подвергнута процедуре нормализации данных.



Рисунок 2.1 – Диаграмма базы данных

Таблица *Branch* хранит данные о филиалах банка.

Таблица *Department* хранит данные департаментов банка.

Таблица *Employee* содержит информацию о работниках банка.

Таблица *Customer* предназначена для информации о клиентах банка.

Таблица *Individual* хранит данные о физических лицах.

Таблица *Business* хранит данные о юридических лицах.

Таблица *Product* создана для данных о продуктах услуг банка.

Таблица *Currency* содержит данные денежных сбережений банковского счета в валюте.

Таблица *Account* создана для данных банковского счета.

Таблица *Credit\_Сard* предназначена для хранения данных о банковских картах клиентов.

Таблица *Acc\_Transaction* хранит информацию транзакции с банком определенного счета.

Таблица *Temp\_Transaction* хранит информацию о действующих длительных операциях/услугах, предоставляемых банком.

Создание перечисленных таблиц с установлением внешних связей описано в главе 3.

1. **Реализация базы данных**

## **3.1 Таблицы**

Ниже представлена внутренняя структура таблиц базы данных с описанием столбцов.

Таблица *Branch* (рис. 3.1) имеет следующую структуру:

* Id – первичный ключ, идентификатор филиала;
* Name – название филиала;
* Region – область, в которой находится филиал и которая содержит значение из списка (Брестская, Витебская, Гомельская, Гродненская, Минская, Могилевская);
* City – город, в котором находится филиал;
* Address – адрес, где находится филиал;
* Index – индекс филиала, который должен содержать шестизначное число.

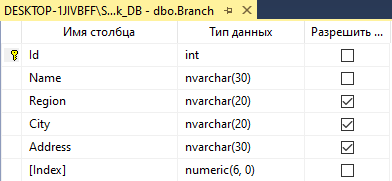


Рисунок 3.1 – Описание таблицы Branch

Таблица *Department* (рис. 3.2) имеет следующую структуру:

* Id – первичный ключ, идентификатор департамента;
* Name – название департамента.

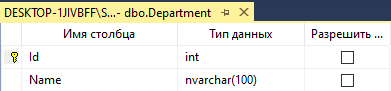


Рисунок 3.2 – Описание таблицы Department

Таблица *Employee* (рис. 3.3) имеет следующую структуру:

* Id – первичный ключ, идентификатор сотрудника;
* Login – уникальный логин сотрудника;
* Password – пароль сотрудника;
* First\_Name – имя сотрудника;
* Last\_Name – фамилия сотрудника;
* Position – должность сотрудника;
* Start\_Date – дата приема на работу сотрудника;
* Assigned\_Branch\_Id­ – идентификатор филиала, внешний ключ к таблице Branch;
* Depart\_Id – идентификатор департамента, внешний ключ к таблице Department;
* Superior\_Emp\_Id – идентификатор начальника, внешний ключ к таблице Employee;
* End\_Date – дата увольнения с работы сотрудника.

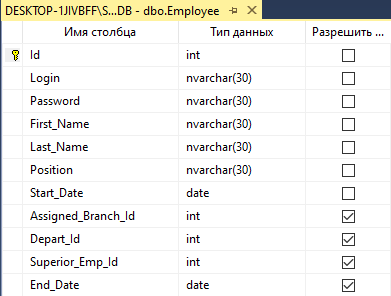


Рисунок 3.3 – Описание таблицы Employee

Таблица *Customer* (рис. 3.4) имеет следующую структуру:

* Id – первичный ключ, идентификатор клиента;
* Login – уникальный логин клиента;
* Password – пароль клиента;
* Region – область, в которой проживает клиент;
* City – город проживания клиента;
* Address – адрес проживания клиента.

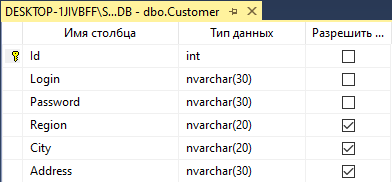


Рисунок 3.4 – Описание таблицы Customer

Таблица *Individual* (рис. 3.5) имеет следующую структуру:

* Cust\_Id – первичный ключ, идентификатор клиента, внешний ключ к таблице Customer;
* First\_Name – имя физического клиента;
* Last\_Name – фамилия физического клиента;
* Birth\_Date – дата рождения физического клиента.

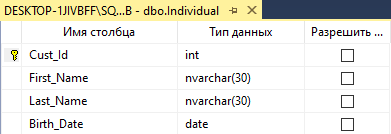


Рисунок 3.5 – Описание таблицы Individual

Таблица *Business* (рис. 3.6) имеет следующую структуру:

* Cust\_Id – первичный ключ, идентификатор юридического клиента, внешний ключ к таблице Customer;
* Name – название юридического клиента (компании);
* Incorp\_Date – дата основания компании.

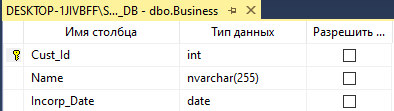


Рисунок 3.6 – Описание таблицы Business

Таблица *Product* (рис. 3.7) имеет следующую структуру:

* Id – первичный ключ, идентификатор услуги, предоставляемой банком;
* Name – название услуги банка;

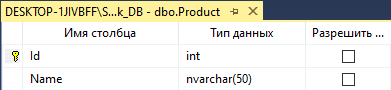


Рисунок 3.7 – Описание таблицы Product

Таблица *Currency* (рис. 3.8) имеет следующую структуру:

* Id – первичный ключ, идентификатор валюты;
* Name – название валюты, содержит значения из списка (RUB, BYN, USD).

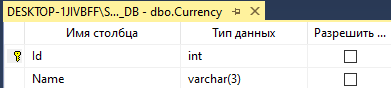


Рисунок 3.8 – Описание таблицы Currency

Таблица *Account* (рис. 3.9) имеет следующую структуру:

* Id – первичный ключ, идентификатор счета клиента;
* Cust\_Id – идентификатор клиента, которому принадлежит счет, внешний ключ к таблицу Customer;
* Open\_Date – дата открытия счета;
* Open\_Branch\_Id – идентификатор филиала, в котором работает сотрудник банка, внешний ключ к таблицу Branch;
* Open\_Emp\_Id – идентификатор сотрудника, который открыл счет, внешний ключ к таблицу Employee;
* Balance – остаток на счете;
* Curr\_Id – идентификатор валюты счета, внешний ключ к таблицу Currency;
* Last\_Activity\_Date­ – дата последней активности счета;
* Close\_Date – дата закрытия счета.

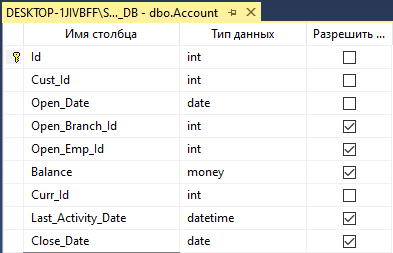


Рисунок 3.9 – Описание таблицы Account

Таблица *Credit\_Card* (рис. 3.10) имеет следующую структуру:

* Id – первичный ключ, идентификатор кредитной карты;
* Account\_Id – идентификатор счета, к которому привязана кредитная карта, внешний ключ к таблицу Account;
* Open\_Date – дата оформления кредитной карты;
* Validity – действительность кредитной карты.

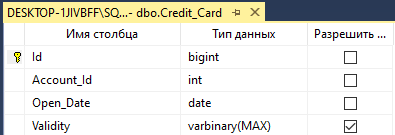


Рисунок 3.10 – Описание таблицы Credit\_Card

Таблица *Acc\_Transaction* (рис. 3.11) имеет следующую структуру:

* Id – первичный ключ, идентификатор транзакции клиента;
* Account\_Id – идентификатор счета, внешний ключ к таблицу Account;
* Amount – сумма выплаты в процессе транзакции;
* Rest – остаток выплаты в процессе транзакции;
* DateTime – дата и время выполнения транзакции;
* Type\_Prod\_Id – идентификатор услуги, внешний ключ к таблице Product;
* Execution\_Branch\_Id\_Id – идентификатор филиала, внешний ключ к таблицу Branch;
* Teller\_Emp\_Id­ – идентификатор сотрудника, внешний ключ к таблице Employee.

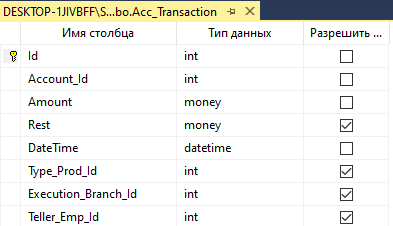


Рисунок 3.11 – Описание таблицы Acc\_Transaction

Таблица *Temp\_Transaction* (рис. 3.12) имеет следующую структуру:

* Id – первичный ключ, идентификатор текущей транзакции;
* Acc\_Tran\_Id – идентификатор транзакции, внешний ключ к таблицу Acc\_Transaction;
* Date\_Retired – дата, до которой клиент должен закрыть текущую транзакцию;
* Completed – завершенность текущей транзакции.

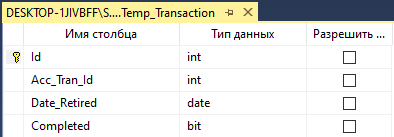


Рисунок 3.12 – Описание таблицы Temp\_Transaction

Скрипты для создания всех таблиц базы данных, а также зависимостей между ними представлены в Приложении А.

## **3.2 Процедуры**

### **3.2.1 Процедуры для решения поставленных задач**

Использование хранимых процедур позволяет ограничить либо вообще исключить непосредственный доступ пользователей к таблицам базы данных.

Всего было разработано 46 процедур для работы с данными. Описание функциональности процедур описано далее в пунктах 3.2.2, 3.2.3 и 3.2.5.

Пример создания хранимой процедуры, которая добавляет новый департамент в таблицу Department, представлен в листинге 3.1.

Данная процедура принимает входной параметр *name* (имя добавляемого департамента) и формирует выходной параметр *mess*, который содержит в случае ошибки уточняющее сообщение о ней. Тело процедуры содержит проверку на уникальность названия департамента, пройдя которую новый департамент добавляется в таблицу Department.

CREATE PROCEDURE AddDepartment

@name NVARCHAR(50),

@mess NVARCHAR(200) OUTPUT

AS

BEGIN

SET @mess='';

BEGIN TRY

DECLARE @checkDepartment int;

SELECT @checkDepartment = count(\*) FROM Department WHERE Name = @name;

IF(@checkDepartment = 0)

INSERT INTO Department(Name) values(@name);

ELSE SET @mess = 'Департамент с таким названием уже существует!';

END TRY

BEGIN CATCH

SET @mess = 'Ошибка системы!';

END CATCH;

END

Листинг 3.1 – Процедура добавления департамента AddDepartment

Листинги некоторых хранимых процедур представлены в Приложении Б. Все скрипты хранимых процедур приложены в отдельном файле в корне директории прилагаемого диска.

### **3.2.2 Процедура экспорта данных в XML-файл**

Код создания процедуры экспорта данных таблицы Employee в xml формат представлен в листинге 3.2.

Для вызова процедуры экспорта была разработана процедура, код которой представлен в листинге 3.3.

CREATE PROCEDURE ExportToFile

AS

SELECT Id, Login, Password, First\_Name, Last\_Name, Position, Start\_Date, Assigned\_Branch\_Id, Depart\_Id, Superior\_Emp\_Id, End\_Date

FROM Employee FOR XML PATH('Employee'), ROOT('Root');

Листинг 3.2 – Процедура экспорта

CREATE PROCEDURE ExecExport

AS

DECLARE @cmd nvarchar(200);

SET @cmd = 'bcp "EXEC ExportToFile" queryout "d:\Bank\data.xml" -S .\SQLEXPRESS -E -dBank\_DB -w -C1251 -r -T';

EXEC master..xp\_cmdshell @cmd;

Листинг 3.3 – Вызов процедуры экспорта

Приложение вызывает процедуру ExecExport, которая в свою очередь с помощью консоли и служебной программы bcp вызывает процедуру ExportToFile, в результате чего данные из таблицы Employee сохраняются в файл в формате xml.

### **3.2.3 Процедура импорта данных из XML-файла**

Код создания процедуры импорта представлен в Приложении В.

Процедура принимает со стороны клиентского приложения xml код и дополняет таблицу всеми записями, у которых идентификатор сотрудников не совпадает с уже имеющимися записями.

## **3.3 Функции**

В базе данных разработана функция, которая принимает исходную валюту и валюту, в которую необходимо конвертировать исходную, и возвращает коэффициент для преобразования валют. Данная функция используется в процедуре для перевода средств с обменом валют. Код функции представлен в листинге 3.4.

CREATE FUNCTION CurrExchange(@curr1 VARCHAR(3), @curr2 VARCHAR(3)) RETURNS FLOAT

BEGIN

DECLARE @rc FLOAT;

SET @rc = CASE

WHEN @curr1='BYN' AND @curr2='RUB' THEN 31.18

WHEN @curr1='BYN' AND @curr2='USD' THEN 0.49

WHEN @curr1='USD' AND @curr2='BYN' THEN 2.05

WHEN @curr1='USD' AND @curr2='RUB' THEN 63.75

WHEN @curr1='RUB' AND @curr2='BYN' THEN 0.032

WHEN @curr1='RUB' AND @curr2='USD' THEN 0.016 ELSE 1

END;

RETURN @rc;

END;

Листинг 3.4 – Функция для преобразования валют

## **3.4 Триггеры**

Изменение или добавление данных в некоторых таблицах требует автоматического изменения данных в других таблицах. Для выполнения этой задачи были разработаны триггеры, которые заносят данные о транзакции при оформлении счета и банковской карты клиентом. Также был разработан триггер для автоматического обновления начальника у сотрудников (листинг 3.5), который выполняется при изменении начальника в определенном филиале и департаменте.

CREATE TRIGGER TRANS\_AFTER\_UPD\_EMP

ON Employee AFTER UPDATE

AS

DECLARE @id INT, @pos1 NVARCHAR(30), @pos2 NVARCHAR(30), @branchId INT, @departId INT;

SELECT @id = Id, @pos1 = Position, @branchId = Assigned\_Branch\_Id, @departId = Depart\_Id FROM deleted;

SELECT @pos2 = Position FROM inserted;

IF(@pos1 != @pos2 AND @pos2 = 'Руководитель отдела')

UPDATE Employee SET Superior\_Emp\_Id = @id WHERE Assigned\_Branch\_Id = @branchId AND Depart\_Id = @departId AND Id != @id;

RETURN;

Листинг 3.5 – Триггер для обновления начальника у сотрудников

Листинги остальных транзакций представлены в Приложении Г.

## **3.5 Пользователи**

Пользователь базы данных – это лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. При проектировании базы данных понадобилось 2 пользователя: Client осуществляет операции над своим счетом в банке и Admin, который может просматривать, изменять и удалять информацию в таблицах базы данных.

Пользователь Admin является администратором, который может получать доступ к базе данных и использовать следующие процедуры:

* AddAccount – добавление счета клиента;
* AddBranch – добавление филиала;
* AddBusinessClient – добавление юридического клиента;
* AddDepartment – добавление департамента;
* AddEmployee – добавление сотрудника;
* AddIndividualClient – добавление физического клиента;
* AddProduct – добавление услуги банка;
* DelAccount – закрытие счета клиента;
* DelBranch – удаление филиала;
* DelDepartment – удаление департамента;
* DelEmployee – увольнение сотрудника;
* DelTempTransaction – закрытие текущей транзакции;
* UpdCustomer – изменение информации о клиенте;
* UpdEmployee – изменение информации о сотруднике;
* UpdSuperior – сменить руководителя отдела;
* GetAllAccounts – получение информации о счетах клиентов;
* GetAllBranches – получение информации о филиалах банка;
* GetAllCurrency – получение информации о валюте банка;
* GetAllDepartments – получение информации о департаментах банка;
* GetAllEmployees – получение информации о сотрудниках банка;
* GetAllProducts – получение информации об услугах банка;
* GetAllTransactions – получение информации о транзакциях клиента;
* GetBusinessClients – получение информации о юридических клиентах;
* GetIndividualClients – получение информации о физических клиентах;
* GetOneClient – получение информации о клиенте по логину;
* GetOneEmployee – получение информации о сотруднике по логину;
* ExportToFile – экспорт данных таблицы Employee в файл xml;
* ExecExport – вызов процедуры ExportToFile через консоль;
* ImportFromFile – импорт данных таблицы Employee из файла xml;

Пользователь Client наделён правами клиента, может обращаться к базе данных через следующие процедуры:

* AddAccount – добавление счета;
* CloseCredit – полное погашение кредита;
* CreateCard – оформление платежной карты;
* DelAccount – закрытие счета;
* DelCard – удаление платежной карты;
* GetAllBranches – получение информации о филиалах банка;
* GetAllCurrency – получение информации о валюте банка;
* GetClientAccounts – получение информации о счетах клиента;
* GetClientCard – получение информации о счетах клиента;
* GetTempTransactions – получение информации о текущих транзакциях по логину клиента;
* GetAccTransactions – получение информации о транзакциях по логину клиена;
* PayCredit – пополнение кредита;
* PutMoney – пополнить счет;
* TakeCredit – оформить кредит;
* TransAnother – перевод денежных средств со счета на счет другого клиента;
* TransExchange – перевод денежных средств с обменом валюты;
* TransYourself – перевод денежных средств со счета на счет одного клиента;
* WithdrawMoney – снятие денежных средств со счета;
* dbo.CurrExchange – функция-конвертер валют;

Скрипты выдачи привилегий пользователям приведены в Приложении Д.

## **3.6 Шифрование и маскирование в базе данных**

Чтобы замаскировать данные столбца таблицы, необходимо указать правило, с помощью которого данные будут замаскированы. В своей базе данных я применила метод маскирования пользовательской строкой (*partial(prefix, padding, suffix)*). Этот метод может раскрывать несколько первых (*prefix*) и последних (*suffix*) букв и добавляет пользовательскую строку заполнения (*padding*) в середине.

Пример реализации технологии маскирования данных в базе данных представлен в листинге 3.6. Данные столбца *Password* таблицы *Customer* полностью маскируются строкой «XXXXXX».

ALTER TABLE Customer

ALTER COLUMN [Password]

ADD MASKED WITH (FUNCTION = 'partial(0, "XXXXXX", 0)');

Листинг 3.6 – Маскирование столбца Password таблицы Customer

Шифрование данных в моей базе данных реализовано с помощью симметричного ключа на уровне ячеек. Метод шифрования с помощью симметричного ключа достаточно надежен, удовлетворяет большинству требований к безопасности данных и обеспечивает достаточное быстродействие. Для шифрования и расшифровки данных используется один ключ.

В первую очередь необходимо создать главный ключ DMK (*Database master key*) базы данных, шифруемый с помощью парольной фразы $tr0nGPa$$w0rd (листинг 3.7).

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '$tr0nGPa$$w0rd';

Листинг 3.7 – Создание главного ключа DMK

Как было сказано ранее, данные шифруются с использованием симметричного ключа, который будет зашифрован с помощью асимметричного ключа. Для этого необходимо создать асимметричный ключ, зашифровать его парольной фразой $tr0nGPa$$w0rd, создать симметричный ключ и зашифровать симметричный ключ с помощью только что созданного асимметричного ключа. Программный код представлен в листинге 3.8.

CREATE ASYMMETRIC KEY MyAsymmetricKey

WITH ALGORITHM = RSA\_2048

ENCRYPTION BY PASSWORD = 'StrongPa$$w0rd!';

GO

CREATE SYMMETRIC KEY MySymmetricKey

WITH ALGORITHM = AES\_256

ENCRYPTION BY ASYMMETRIC KEY MyAsymmetricKey;

Листинг 3.8 – Создание ассиметричного и симметричного ключа

Теперь можно приступить к шифрованию данных. Для этого необходимо сначала открыть симметричный ключ, только что созданный с помощью команды OPEN SYMMETRIC KEY, за которой следует имя симметричного ключа. Затем указать, что нужно расшифровать его с использованием заданного асимметричного ключа. Программный код представлен в листинге 3.9.

OPEN SYMMETRIC KEY MySymmetricKey

DECRYPTION BY ASYMMETRIC KEY MyAsymmetricKey

WITH PASSWORD = 'StrongPa$$w0rd!';

 Листинг 3.9 – Открытие симметричного ключа

SQL Server располагает встроенными функциями для шифрования и расшифровки на уровне ячеек. Функция шифрования, которая использует симметричный ключ: ENCRYPTBYKEY.

Шифрование данных столбца *Validity* таблицы *Credit\_Card* симметричным ключом показано в листинге 3.10.

DECLARE @SymmetricKeyGUID AS [uniqueidentifier];

SET @SymmetricKeyGUID = KEY\_GUID('MySymmetricKey');

IF (@SymmetricKeyGUID IS NOT NULL)

BEGIN

DECLARE @accId INT, @open DATE;

SELECT @accId = Id, @open = Open\_Date FROM Account WHERE Id = @id+@i;

INSERT INTO Credit\_Card(Account\_Id, Open\_Date, Validity)

VALUES( @accId, @open, ENCRYPTBYKEY(@SymmetricKeyGUID,N'09/21'));

END;

Листинг 3.10 – Пример шифрования данных столбца Validity таблицы Credit\_Card с симметричным ключом

С помощью функции DECRYPTBYKEY можно просмотреть зашифрованные данные (листинг 3.11).

SELECT Id [ID], Account\_Id [ID счета], Open\_Date [Дата открытия],

CONVERT(NVARCHAR(32), DECRYPTBYKEY(Validity)) [Действительно]

FROM Credit\_Card WHERE Account\_Id = @accId;

Листинг 3.11 – Дешифрование данных столбца Validity таблицы Credit\_Card

# **Тестирование**

## **4.1 Тестирование используемой технологии**

Технология маскирования данных в курсовой работе была применена к данным столбцов Password таблиц Customer и Employee. Протестируем работу маскирования с помощью вызова хранимой процедуры, которая возвращает информацию об индивидуальных клиентах (join таблиц Individual и Customer). Результат вызова процедуры *GetIndividualClients* представлен на рисунке 4.1.

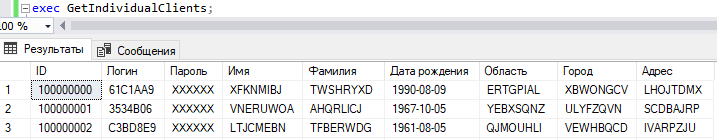


Рисунок 4.1 – Результат вызова процедуры GetIndividualClients

Шифрование данных в базе данных осуществлялось с помощью симметричного ключа и применялось к данным столбца Validity таблицы Credit\_Card. На рисунке 4.2 показан результат SELECT-запроса к таблице Credit\_Card после шифрования данных.

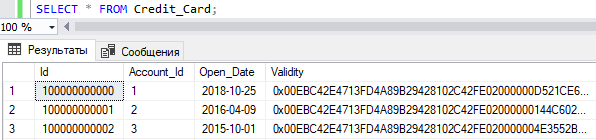


Рисунок 4.2 – Результат SELECT-запроса к таблице Credit\_Card

Для пользователя Client было осуществлено дешифрование данных для предоставления собственных данных в первоначальном виде. На рисунке 4.3 показан результат хранимой процедуры, которая дешифрует столбец Validity (Действительно) и выводит данные о кредитной карте определенного счета.

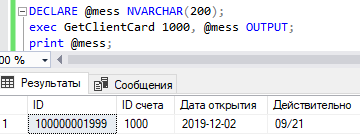


Рисунок 4.3 – Результат вызова процедуры GetClientCard

## **4.2 Тестирование работоспособности приложения**

В момент авторизации возможна ситуация, когда пользователь вводит некорректные данные. Обработка таких ошибок происходит с помощью добавления окна с описанием ошибки. Пример обработки неверных данных показан на рисунке 4.4.

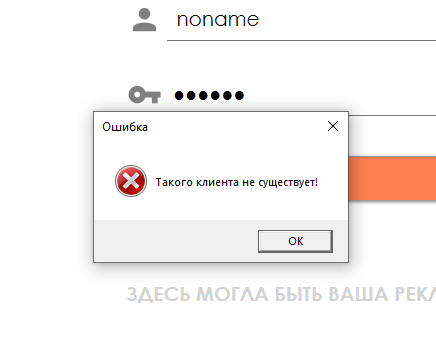


Рисунок 4.4 – Обработка некорректного логина или пароля

При регистрации пользователь может ввести данные не во все поля, пример обработки таких ошибок представлен на рисунке 4.5.

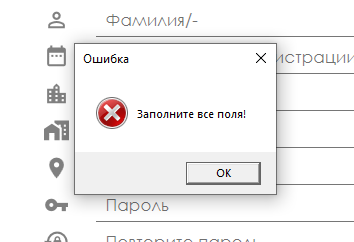


Рисунок 4.5 – Обработка ввода пустых данных

В приложении существуют определенные ограничения на некоторые действия. Пример обработки ограничения на количество добавляемых клиентом счетов показано на рисунке 4.6. Пример обработки ограничения на оформление платежной картой представлен на рисунке 4.7.

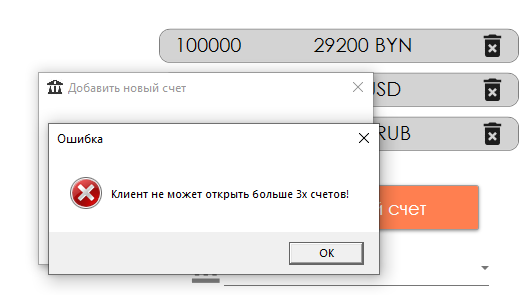


Рисунок 4.6 – Обработка ограничения на добавление счетов клиента

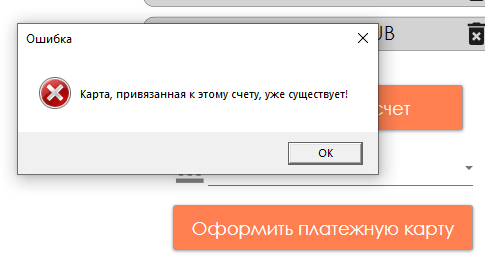


Рисунок 4.7 – Обработка ошибки добавления кредитной карты

В процессе использования клиентом приложения, он может не замечать некоторых ошибок при вводе данных. Поэтому чтобы избежать ошибок согласованности данных в таблицах, приложение указывает на неправильность действий клиентов. Пример ввода некорректных данных представлены на рисунках 4.8 - 4.10.

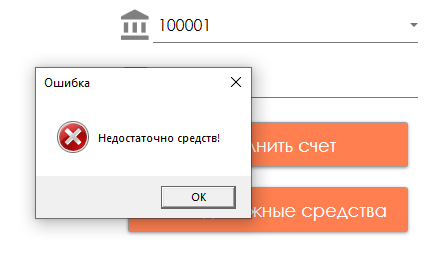


Рисунок 4.8 – Тестирование вкладки «Операции» аккаунта клиента

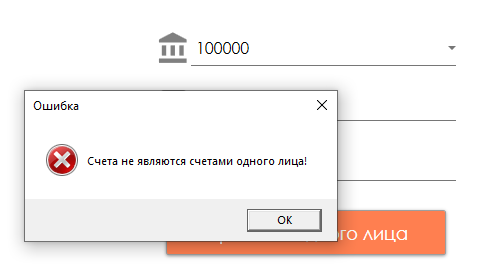


Рисунок 4.9 – Тестирование перевода средств в рамках одного лица

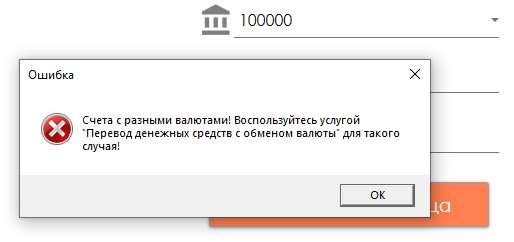


Рисунок 4.10 – Обработка ошибки при несовпадении валюты

Приложение обрабатывает ошибки не только в аккаунте клиента, но и контролирует действия администратора. На рисунке 4.11 представлена обработка ошибки при добавлении клиента под неуникальным логином

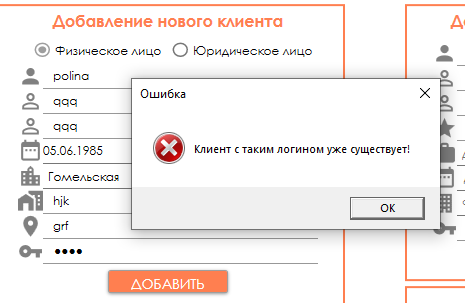


Рисунок 4.11 – Тестирование добавления нового клиента

При добавлении нового клиента также существует ограничение на возраст добавляемого, он должен быть не младше 18 лет. Обработка такого ограничения представлена на рисунке 4.12.

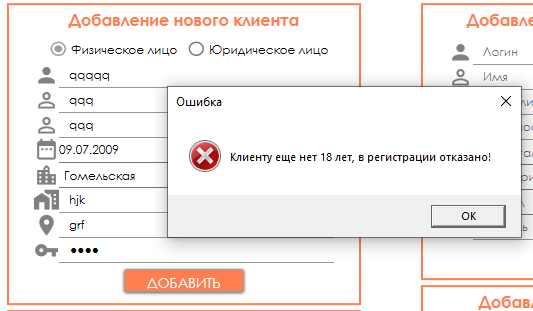


Рисунок 4.12 – Обработка ограничения на возраст при добавлении клиента

# **Руководство пользователя**

Начальной страницей приложения является вход в приложение. Здесь клиент может ввести имя и пароль и войти в приложение, либо перейти к регистрации. Данная страница представлена на рисунке 5.1, а вкладка регистрации на рисунке 5.2.



Рисунок 5.1 – Начальная страница авторизации

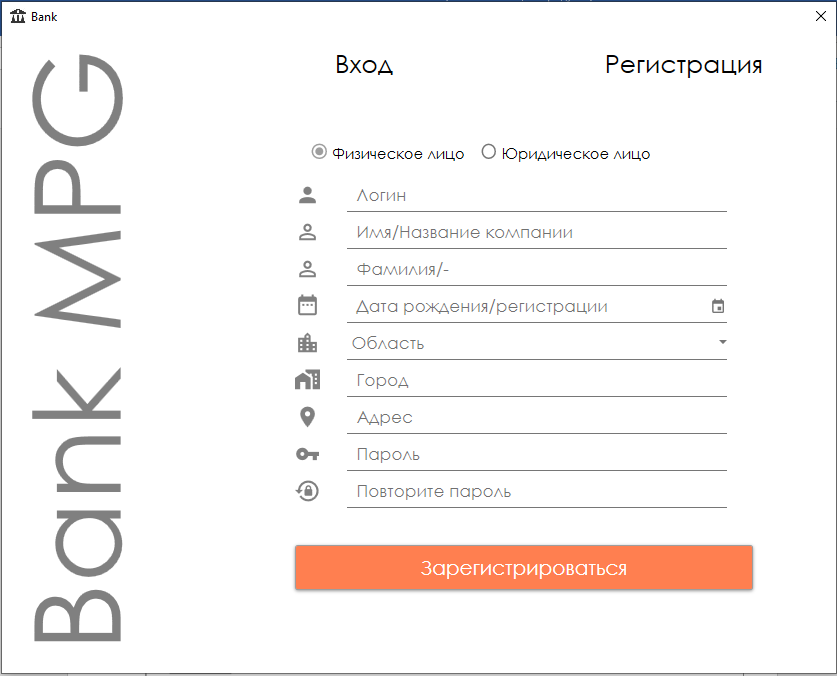


Рисунок 5.2 – Вкладка регистрации клиентов

После успешной авторизации или регистрации клиент попадает на главную страницу, которой является страница личных счетов (рис. 5.3). На главной странице можно увидеть меню для перехода по страницам, таким как «Счета», «Операции», «Кредиты», «Перевод средств» и «Транзакции».

На странице «Счета» клиент может создать новый личный счет и оформить платежную карту для имеющегося счета.

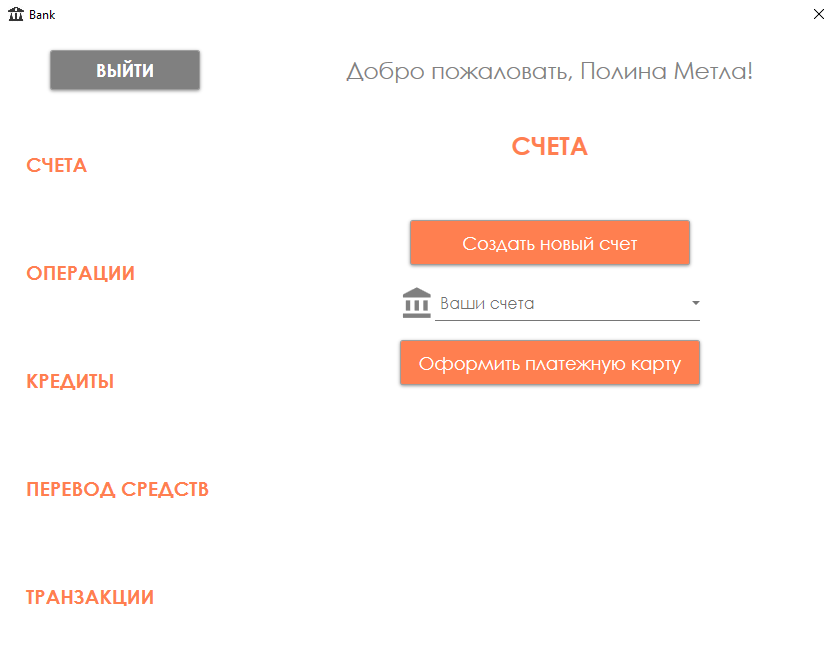


Рисунок 5.3 – Страница банковских счетов клиента

При нажатии на кнопку «Создать новый счет», открывается окно для оформления счета, которое представлено на рисунке 5.4.

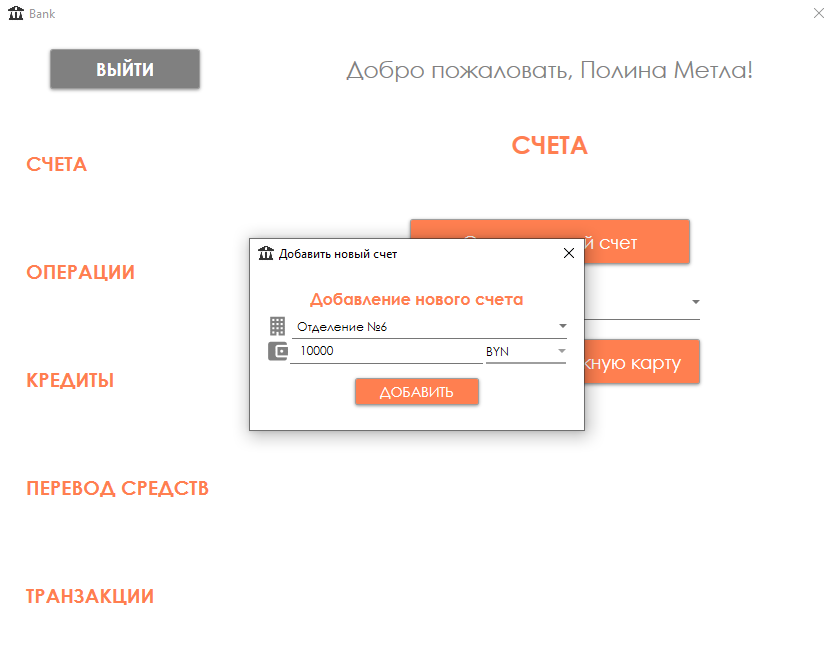


Рисунок 5.4 – Окно добавления нового счета клиента

При добавлении нового счета он автоматически появляется на странице счетов (рис. 5.5) и доступен в выпадающем списке для создания платежной карты. Выбрав счет из этого списка и нажав кнопку «Оформить платежную карту», необходимо перезагрузить страницу для появления возле счета подсказки с номером созданной платежной карты. Результат оформления платежной карты представлен на рисунке 5.6.

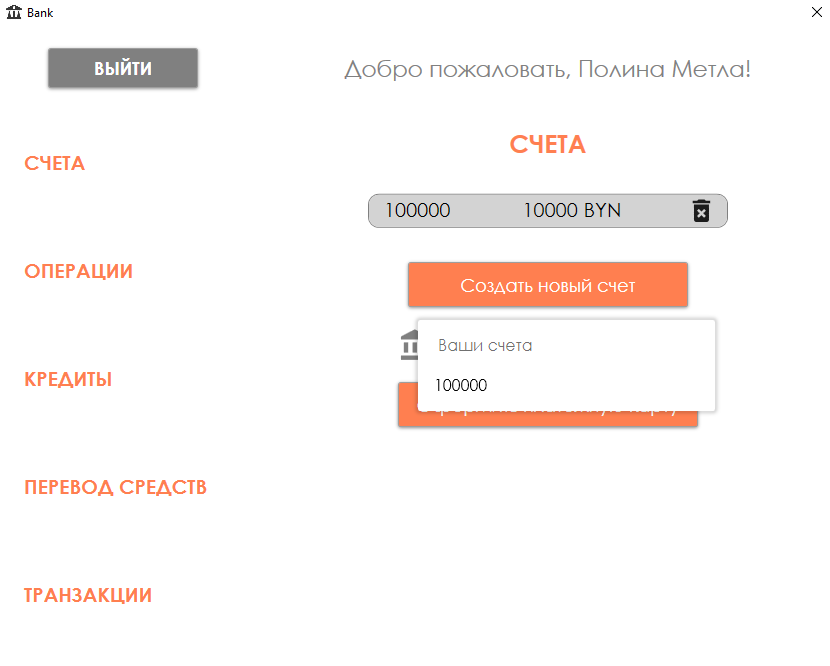


Рисунок 5.5 – Страница счетов клиента после добавления нового счета

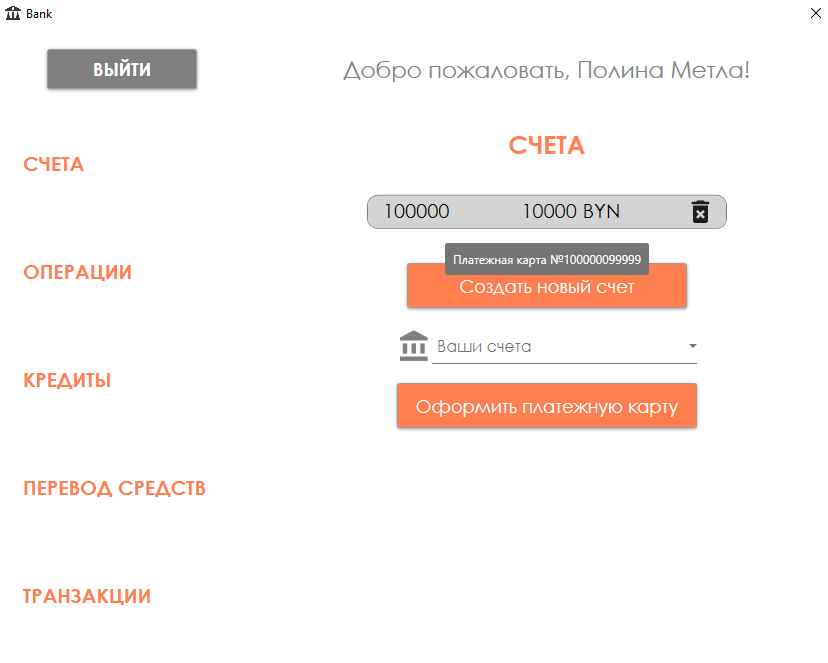


Рисунок 5.6 – Страница счетов клиента после добавления платежной карты

Перейдем на страницу «Операции» (рис. 5.7). Клиент может как класть денежные средства на выбранный счет, так и снимать их.

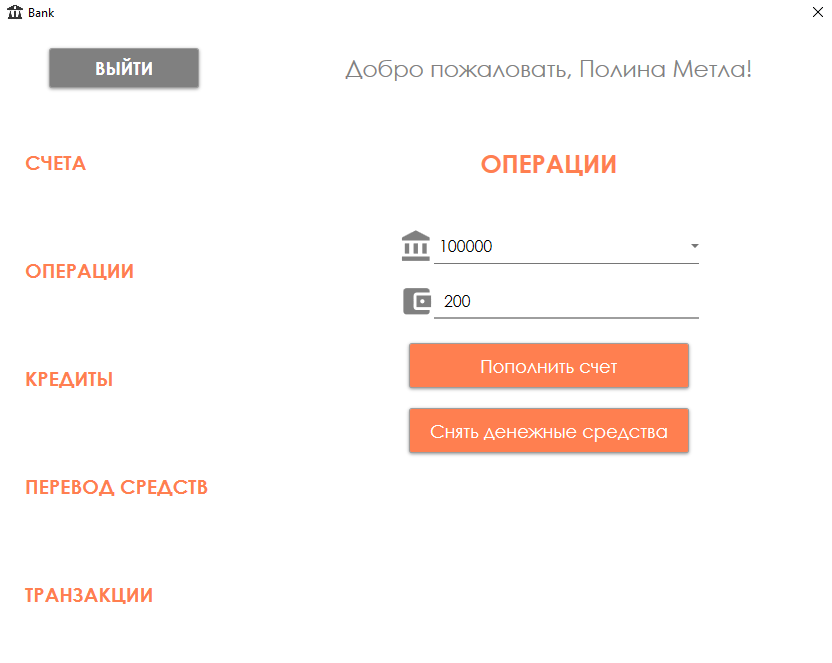


Рисунок 5.7 – Страница выполнения операций клиентом

На странице «Кредиты» клиент может оформлять и выплачивать свои кредиты (рис. 5.8). Чтобы оформить кредит клиент должен выбрать счет, сумму и до какой даты он должен выплатить кредит и нажать кнопку «Заявка на кредит».

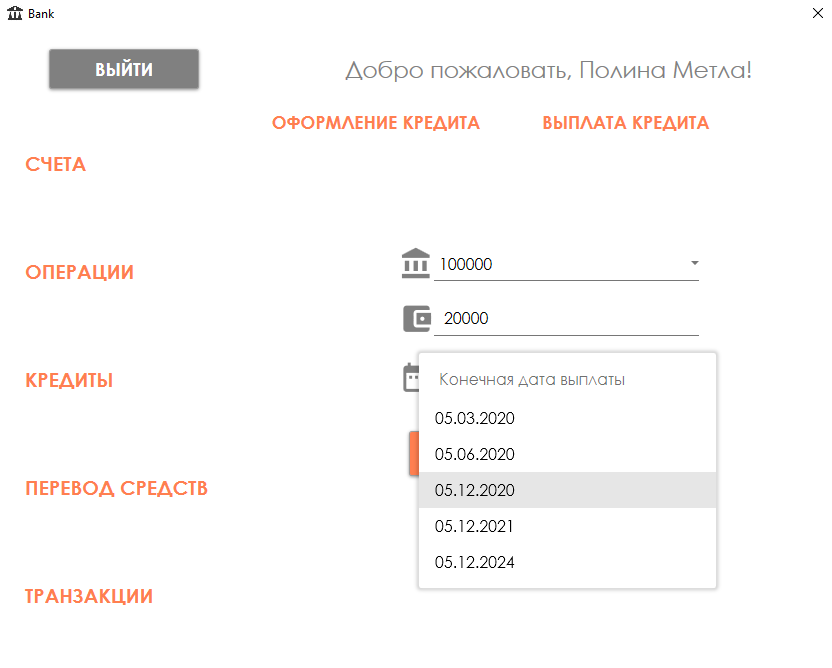


Рисунок 5.8 – Страница оформления кредита клиентом

Далее перейдем на страницу с транзакциями клиента. На рисунке 5.9 мы можем увидеть таблицу, в которой зафиксированы все действия клиента, произведенные ранее (оформление счета, оформление платежной карты, пополнение счета и оформление кредита).

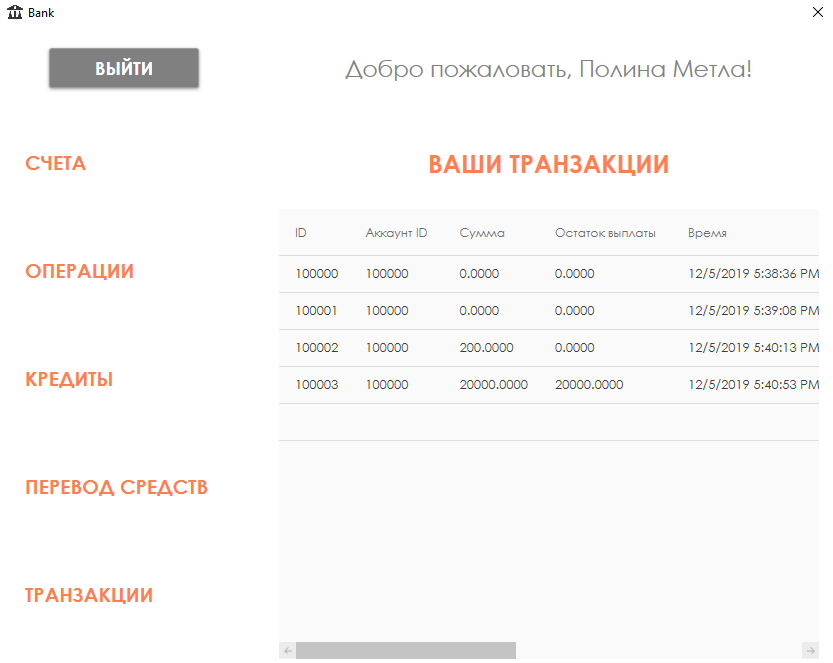


Рисунок 5.9 – Страница транзакций клиента

Вернемся на страницу «Кредиты» и перейдем на вкладку «Выплата кредитов». На данной странице клиент может выплатить кредит частично либо полностью. При досрочной выплате кредита поле «Сумма» можно не заполнять. Данная страница представлена на рисунке 5.10.

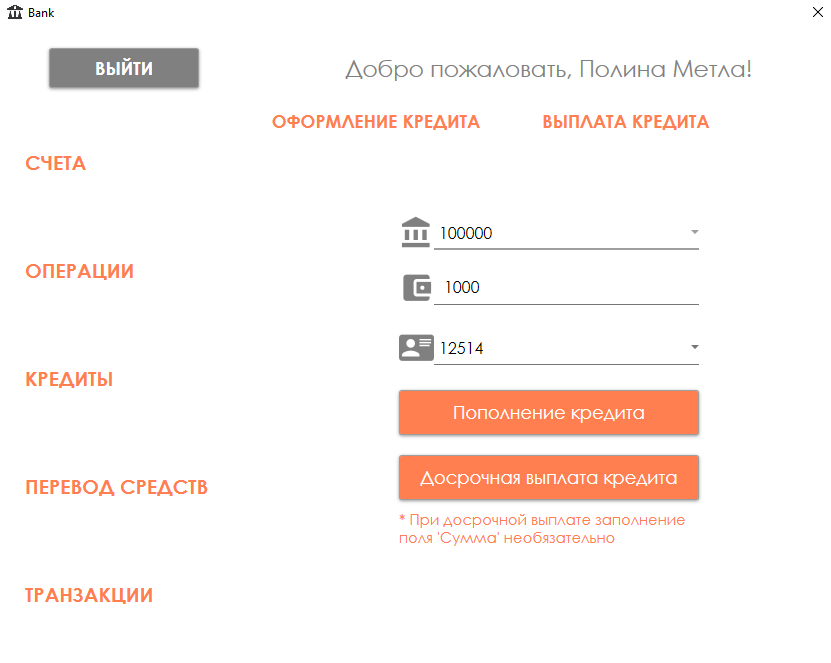


Рисунок 5.10 – Страница выплаты кредита клиентом

На странице «Перевод средств» клиент может переводить денежные средства со счета клиента на собственный счет, другому лицу, а также переводить средства с обменом валют (рис. 5.11).

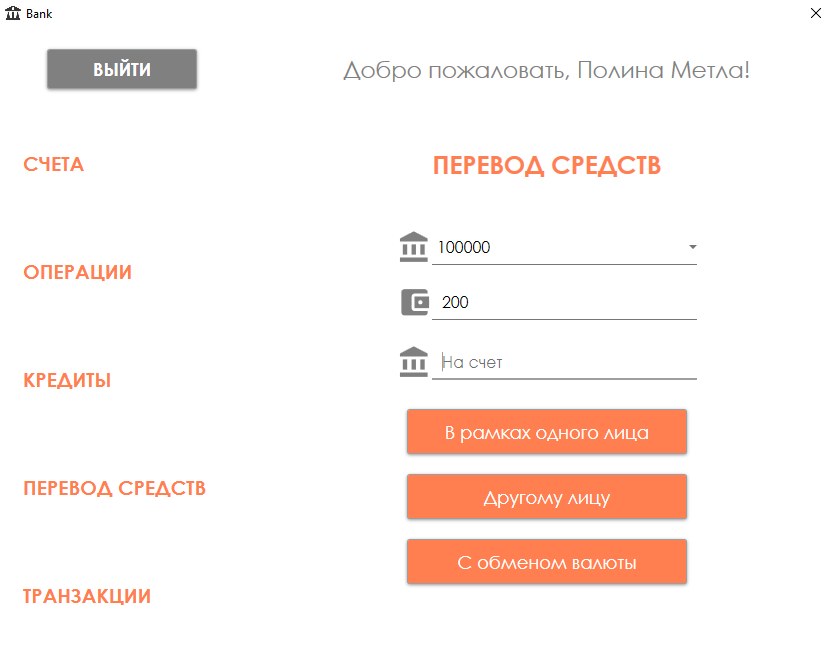


Рисунок 5.11 – Страница перевода средств клиентом

Для выхода с аккаунта клиента необходимо нажать на кнопку «Выйти». На рисунке 5.12 представлено окно для подтверждения выхода.

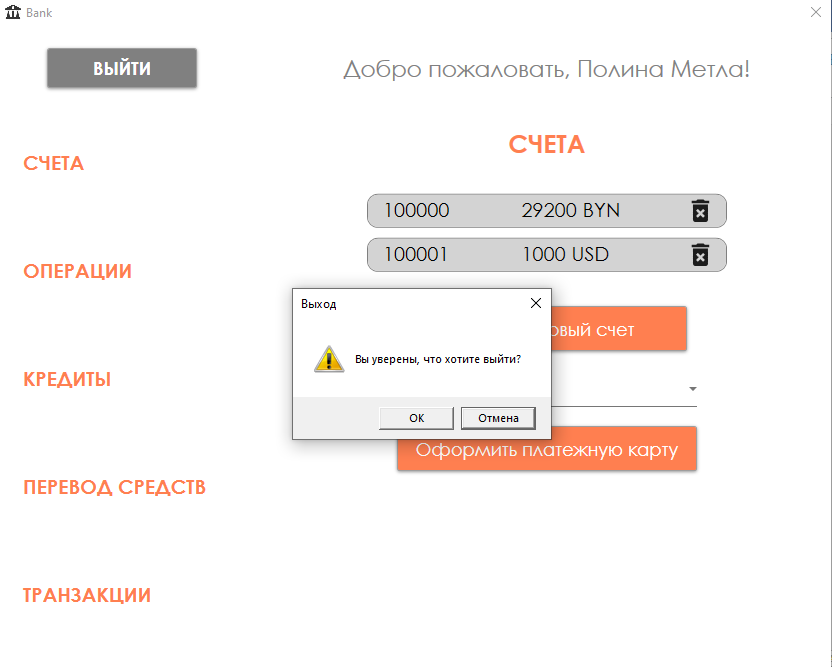


Рисунок 5.12 – Окно для выхода клиента из аккаунта

Вернемся на начальную страницу авторизации и нажмем кнопку «Войти как сотрудник» для входа в аккаунт администратора (рис. 5.13).

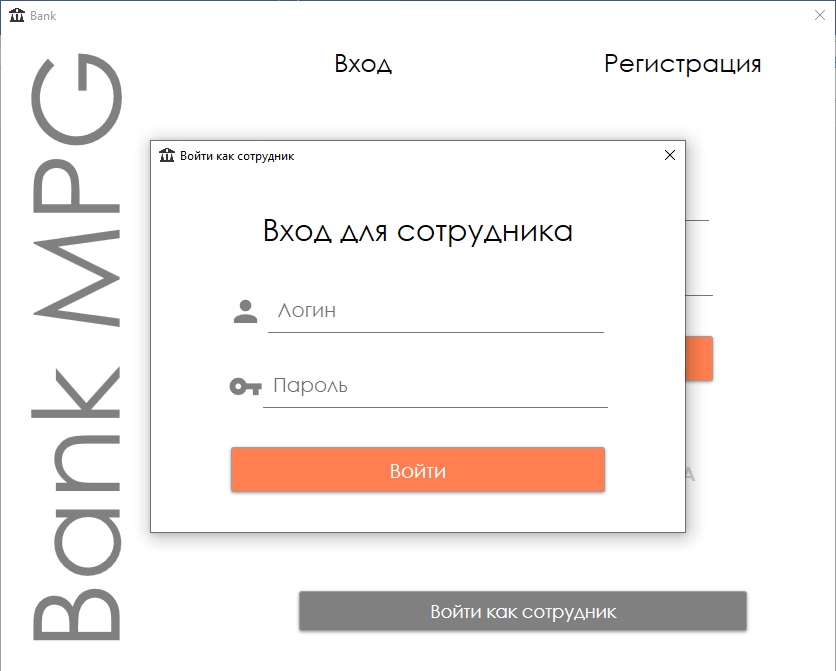


Рисунок 5.13 – Окно для авторизации администратора

После входа в аккаунт администратора мы попадаем на страницу просмотра данных из таблиц базы данных. На странице находятся кнопки, нажав на которые данные из соответствующих таблиц базы данных заносятся в таблицу, как показано на рисунке 5.14.

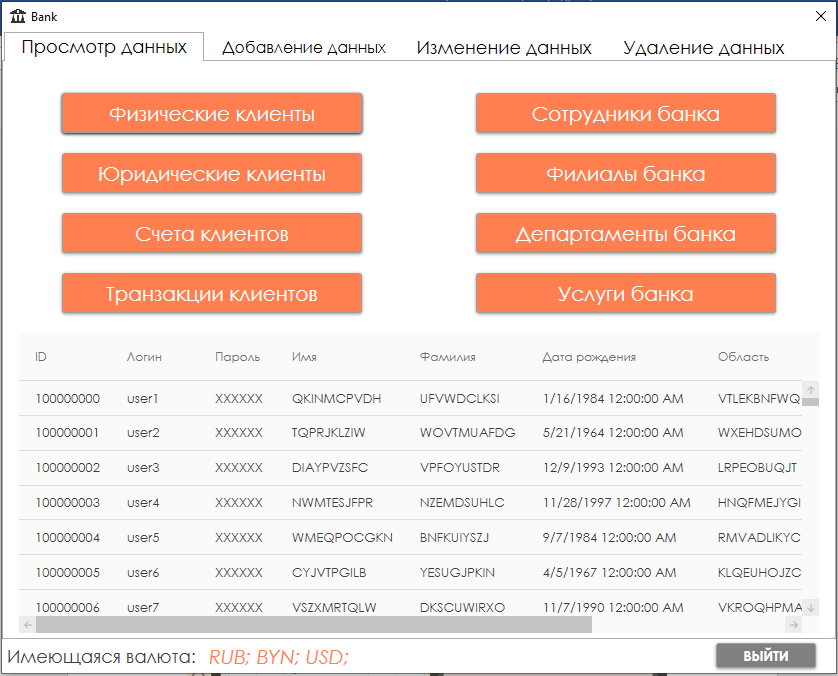


Рисунок 5.14 – Страница с получением данных таблиц БД

На вкладке «Добавление данных» находятся формы для добавления данных в таблицы базы данных (рис. 5.15).

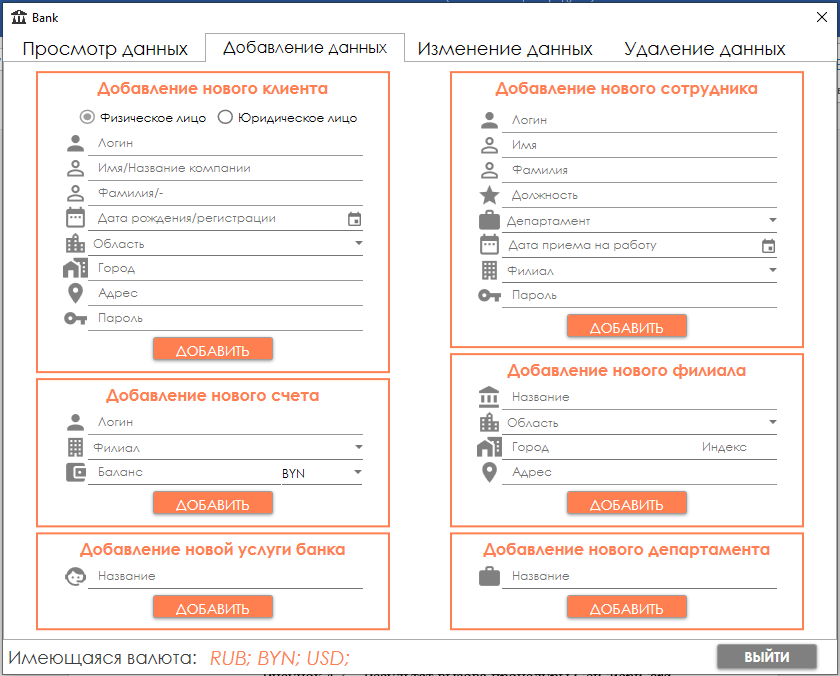


Рисунок 5.15 – Страница с добавлением данных в таблицы БД

Вкладка «Изменение данных» содержит формы для изменения данных в таблицах базы данных (рис. 5.16). Чтобы изменить данные о сотруднике или клиенте банка, необходимо ввести логин в поле «Логин» и нажать кнопку «Найти». После этого данные о найденном человеке появятся в полях, находящихся под кнопкой «Найти». Изменив эти данные и нажав кнопку «Изменить», новая информация будет зафиксирована в базе данных.

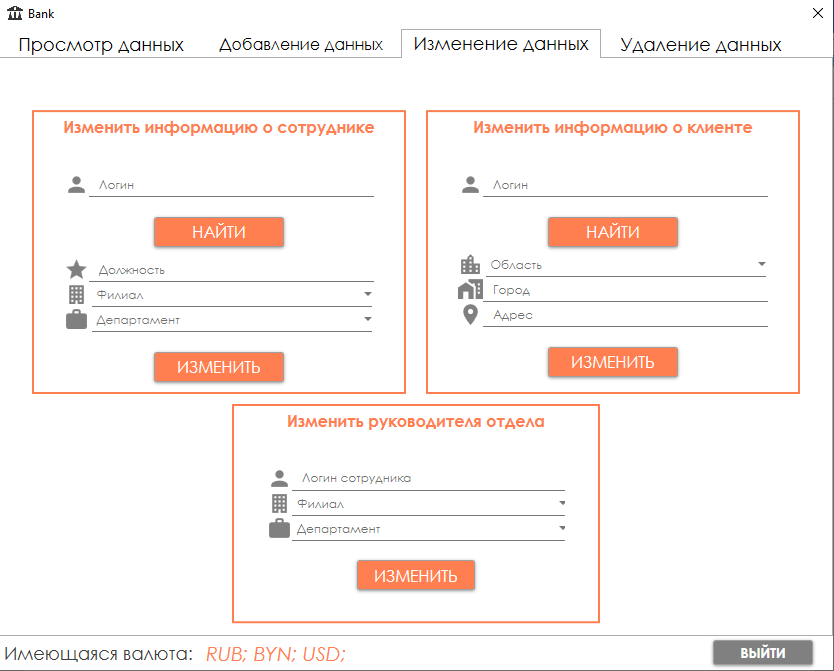


Рисунок 5.16 – Страница с изменением данных в некоторых таблицах БД

Перейдя на вкладку «Удаление данных», мы увидим формы для удаления данных из таблиц базы данных (рис. 5.17). Также на этой вкладке осуществляется экспорт и импорт данных таблицы Employee в xml файл с помощью соответствующих кнопок.

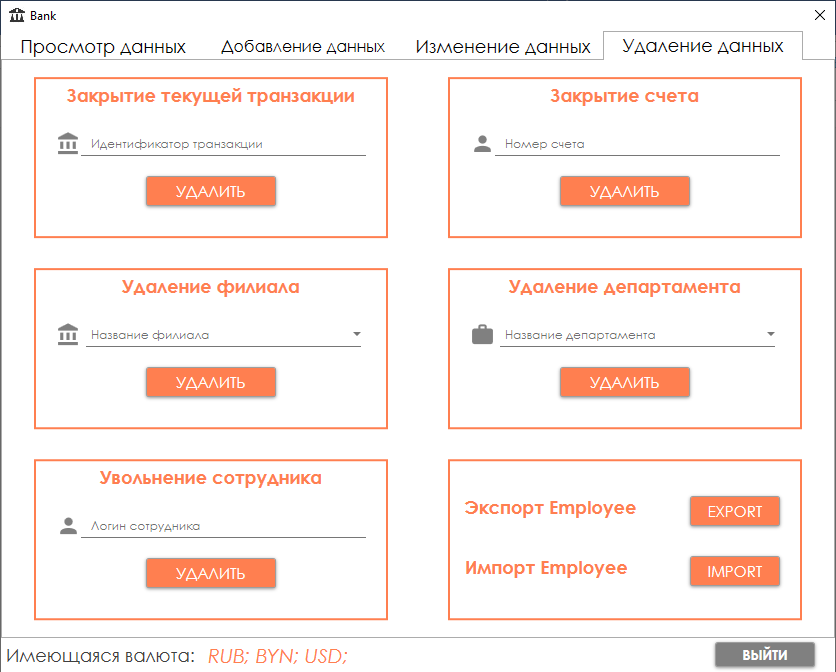


Рисунок 5.17 – Страница с удалением данных из БД

Для выхода с аккаунта администратора необходимо нажать на кнопку «Выйти». На рисунке 5.18 представлено окно для подтверждения выхода.

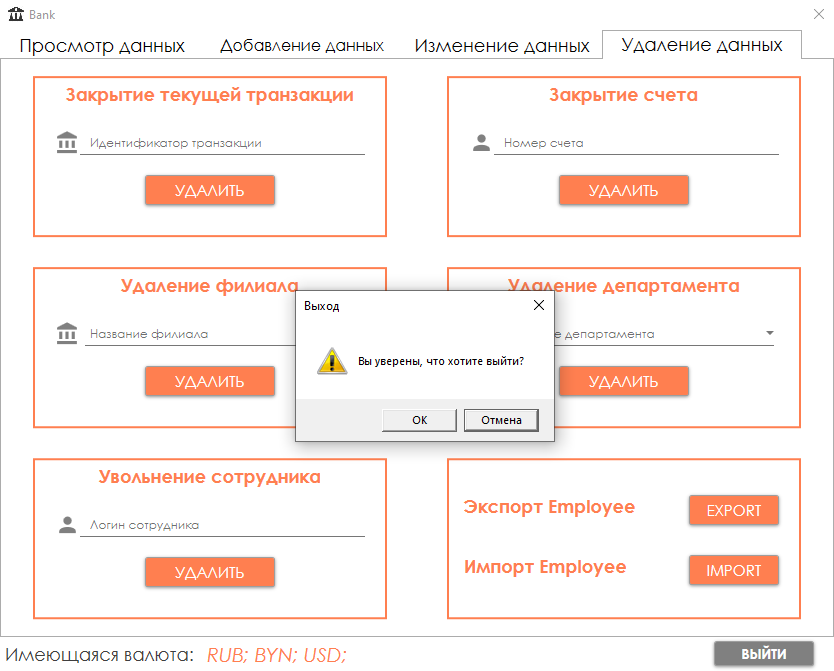


Рисунок 5.18 – Окно для выхода администратора с аккаунта

# **Заключение**

Перед началом разработки базы данных был произведен аналитический обзор прототипов приложений подобной тематики и определение функциональных возможностей моей базы данных.

В процессе выполнения курсовой работы была спроектирована база данных для банка. База данных была разработана с помощью системы управления базами данных «Microsoft SQL Server 2017». Помимо этого, для обеспечения безопасности данных я ознакомилась с технологией маскирования и шифрования данных и применила их к своей БД.

При разработке базы данных было созданы следующие объекты:

* 12 таблиц с внешними связями;
* 46 хранимых процедур;
* 1 функция;
* 3 триггера;
* 2 пользователя базы данных.

Также было разработано приложение, демонстрирующее работу с базой данных. Приложение написано с помощью технологии WPF и языка программирования C#. Связь между базой данных и приложением осуществляется с помощью технологии ADO.NET, которая предоставляет набор классов, через которые мы можем установить подключение к базе данных и производить ряд других операций. Также было написано руководство пользователя для созданного приложения.

В соответствии с полученным результатом, можно сказать, что разработанная база данных функционирует верно, требования технического задания реализованы в полном объеме, в созданном приложении предоставлены возможности базы данных, поэтому цель курсового проекта можно считать достигнутой.

# **Список литературных источников**

1. Система «Интернет-банкинг» ОАО «АСБ Беларусбанк» [Электронный ресурс] – Режим доступа: <https://ibank.asb.by/wps/portal/ibank/Home/login/>
2. Официальный сайт «Альфа-банк в Беларуси» [Электронный ресурс] – Режим доступа: <https://www.alfabank.by/>
3. Официальный сайт «Приорбанк» ОАО [Электронный ресурс] – Режим доступа: <https://www.priorbank.by/>
4. СУБД «Microsoft SQL Server 2017» [Электронный ресурс] – Режим доступа: <https://metanit.com/sql/sqlserver/>
5. Мэтью Макдональд — WPF: Windows Presentation Foundation в .NET 4.0 с примерами на C#
6. Язык C# и .NET Framework [Электронный ресурс] — Режим доступа: <http://professorweb.ru/my/csharp/charp_theory/level1/>
7. Руководство по ADO.NET и работе с базами данных [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/adonet/>
8. Динамическое маскирование данных – SQL Server [Электронный ресурс] – Режим доступа: [https://docs.microsoft.com/ru-ru/sql/relational-databases/security/ dynamic-data-masking?view=sql-server-2017](https://docs.microsoft.com/ru-ru/sql/relational-databases/security/%20dynamic-data-masking?view=sql-server-2017)
9. Шифрование в SQL Server [Электронный ресурс] – Режим доступа: [h<https://docs.microsoft.com/ru-ru/sql/relational-databases/security/encryption/sql-server-encryption?view=sql-server-ver15>](https://metanit.com/sharp/adonet/)

# **Приложение А**

USE Bank\_DB;

CREATE TABLE Branch(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

[Name] NVARCHAR(30) NOT NULL,

Region NVARCHAR(20) CHECK(Region IN ('Брестская', 'Витебская', 'Гомельская', 'Гродненская', 'Минская', 'Могилевская')),

City NVARCHAR(20),

[Address] NVARCHAR(30),

[Index] NUMERIC(6) NOT NULL

);

CREATE TABLE Department(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

[Name] NVARCHAR(100) NOT NULL

);

CREATE TABLE Employee(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

[Login] NVARCHAR(30) UNIQUE NOT NULL,

[Password] NVARCHAR(30) NOT NULL,

First\_Name NVARCHAR(30) NOT NULL,

Last\_Name NVARCHAR(30) NOT NULL,

Position NVARCHAR(30) NOT NULL,

[Start\_Date] DATE NOT NULL,

Assigned\_Branch\_Id INT,

Depart\_Id INT,

Superior\_Emp\_Id INT,

End\_Date DATE,

CONSTRAINT FK\_Employee\_Branch FOREIGN KEY(Assigned\_Branch\_Id) REFERENCES Branch,

CONSTRAINT FK\_Employee\_Department FOREIGN KEY(Depart\_Id) REFERENCES Department,

CONSTRAINT FK\_Employee\_Employee FOREIGN KEY(Superior\_Emp\_Id) REFERENCES Employee

);

ALTER TABLE Employee

ALTER COLUMN [Password] ADD MASKED WITH (FUNCTION = 'partial(0, "XXXXXX", 0)');

CREATE TABLE Customer(

Id INT IDENTITY(100000000,1) PRIMARY KEY NOT NULL,

[Login] NVARCHAR(30) UNIQUE NOT NULL,

[Password] NVARCHAR(30) NOT NULL,

Region NVARCHAR(20),

City NVARCHAR(20),

[Address] NVARCHAR(30)

);

ALTER TABLE Customer

ALTER COLUMN [Password] ADD MASKED WITH (FUNCTION = 'partial(0, "XXXXXX", 0)');

CREATE TABLE Individual(

Cust\_Id INT PRIMARY KEY NOT NULL,

First\_Name NVARCHAR(30) NOT NULL,

Last\_Name NVARCHAR(30) NOT NULL,

Birth\_Date DATE NOT NULL,

CONSTRAINT FK\_Individual\_Customer FOREIGN KEY(Cust\_Id) REFERENCES Customer

);

CREATE TABLE Business(

Cust\_Id INT PRIMARY KEY NOT NULL,

[Name] NVARCHAR(255) NOT NULL,

Incorp\_Date DATE NOT NULL,

CONSTRAINT FK\_Business\_Customer FOREIGN KEY(Cust\_Id) REFERENCES Customer

);

CREATE TABLE Product(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

[Name] NVARCHAR(50) NOT NULL

);

CREATE TABLE Currency(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

[Name] VARCHAR(3) NOT NULL CHECK([Name] IN ('RUB', 'BYN', 'USD'))

);

CREATE TABLE Account(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

Cust\_Id INT NOT NULL,

Open\_Date DATE NOT NULL,

Open\_Branch\_Id INT,

Open\_Emp\_Id INT,

Balance MONEY DEFAULT 0.0,

Curr\_Id INT NOT NULL,

Last\_Activity\_Date DATETIME,

Close\_Date DATE,

CONSTRAINT FK\_Account\_Customer FOREIGN KEY(Cust\_Id) REFERENCES Customer,

CONSTRAINT FK\_Account\_Branch FOREIGN KEY(Open\_Branch\_Id) REFERENCES Branch,

CONSTRAINT FK\_Account\_Employee FOREIGN KEY(Open\_Emp\_Id) REFERENCES Employee,

CONSTRAINT FK\_Account\_Currency FOREIGN KEY(Curr\_Id) REFERENCES Currency

);

CREATE TABLE Credit\_Card(

Id BIGINT IDENTITY(100000000000,1) PRIMARY KEY NOT NULL,

Account\_Id INT NOT NULL,

Open\_Date DATE NOT NULL,

Validity VARBINARY(MAX),

CONSTRAINT FK\_Card\_Account FOREIGN KEY(Account\_Id) REFERENCES Account

);

CREATE TABLE Acc\_Transaction(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

Account\_Id INT NOT NULL,

Amount MONEY NOT NULL DEFAULT 0.0,

Rest MONEY DEFAULT 0.0,

[DateTime] DATETIME NOT NULL,

Type\_Prod\_Id INT,

Execution\_Branch\_Id INT,

Teller\_Emp\_Id INT,

CONSTRAINT FK\_Acc\_Transaction\_Product FOREIGN KEY(Type\_Prod\_Id) REFERENCES Product,

CONSTRAINT FK\_Acc\_Transaction\_Account FOREIGN KEY(Account\_Id) REFERENCES Account,

CONSTRAINT FK\_Acc\_Transaction\_Branch FOREIGN KEY(Execution\_Branch\_Id) REFERENCES Branch,

CONSTRAINT FK\_Acc\_Transaction\_Employee FOREIGN KEY(Teller\_Emp\_Id) REFERENCES Employee

);

CREATE TABLE Temp\_Transaction(

Id INT IDENTITY(1,1) PRIMARY KEY NOT NULL,

Acc\_Tran\_Id INT NOT NULL,

Date\_Retired DATE NOT NULL,

Completed BIT NOT NULL,

CONSTRAINT FK\_Temp\_Acc\_Transaction FOREIGN KEY(Acc\_Tran\_Id) REFERENCES Acc\_Transaction

);

# **Приложение Б**

--информация о сотруднике по логину

CREATE PROCEDURE GetOneEmployee

@login NVARCHAR(30)

AS

SELECT E.Id [ID], E.Login [Логин], E.Password [Пароль], E.Position [Должность], E2.Last\_Name [Начальник], D.Name [Департамент], B.Name [Филиал]

FROM Employee E LEFT OUTER JOIN Branch B ON E.Assigned\_Branch\_Id=B.Id

LEFT OUTER JOIN Department D ON E.Depart\_Id=D.Id

LEFT OUTER JOIN Employee E2 ON E.Superior\_Emp\_Id=E2.Id

WHERE E.Login = @login;

--добавить физического клиента

CREATE PROCEDURE AddIndividualClient

@login NVARCHAR(30),

@password NVARCHAR(30),

@firstName NVARCHAR(30),

@lastName NVARCHAR (30),

@birth DATE,

@region NVARCHAR(20),

@city NVARCHAR(20),

@address NVARCHAR(30),

@mess NVARCHAR(200) OUTPUT

AS

BEGIN

SET @mess = '';

BEGIN TRY

DECLARE @checkLogin int;

SELECT @checkLogin = count(\*) FROM Customer WHERE Login = @login;

IF(@checkLogin = 0)

BEGIN

IF(DATEDIFF(year, @birth, GETDATE()) >= 18)

BEGIN

INSERT INTO Customer(Login, Password, Region, City, Address) values(@login, @password, @region, @city, @address);

DECLARE @userId int;

SELECT TOP(1) @userId = Id FROM Customer WHERE Login = @login;

INSERT INTO Individual(Cust\_Id, First\_Name, Last\_Name, Birth\_Date) values(@userId, @firstName, @lastName, @birth);

END;

ELSE SET @mess = 'Клиенту еще нет 18 лет, в регистрации отказано!';

END;

ELSE SET @mess = 'Клиент с таким логином уже существует!';

END TRY

BEGIN CATCH

SET @mess = 'Ошибка системы!';

END CATCH;

END

--удалить филиал банка

CREATE PROCEDURE DelBranch

@name NVARCHAR(30),

@mess NVARCHAR(200) OUTPUT

AS

BEGIN

SET @mess='';

BEGIN TRY

DECLARE @checkBranch int, @branchId int, @checkEmployee int, @checkAccount int, @checkTrans int;

SELECT @checkBranch = count(\*) FROM Branch WHERE Name = @name;

IF(@checkBranch != 0)

BEGIN

SELECT @branchId = Id FROM Branch WHERE Name = @name;

SELECT @checkEmployee = count(\*) FROM Employee WHERE Assigned\_Branch\_Id = @branchId;

SELECT @checkAccount = count(\*) FROM Account WHERE Open\_Branch\_Id = @branchId;

SELECT @checkTrans = count(\*) FROM Acc\_Transaction T join Temp\_Transaction TT ON T.Id=TT.Acc\_Tran\_Id WHERE T.Execution\_Branch\_Id = @branchId AND TT.Completed=0;

IF(@checkEmployee != 0)

UPDATE Employee SET End\_Date=GETDATE(), Assigned\_Branch\_Id = NULL WHERE Assigned\_Branch\_Id = @branchId;

IF(@checkAccount != 0)

UPDATE Account SET Open\_Branch\_Id=NULL WHERE Open\_Branch\_Id = @branchId;

IF(@checkTrans != 0)

UPDATE Acc\_Transaction SET Execution\_Branch\_Id=NULL WHERE Execution\_Branch\_Id = @branchId;

DELETE FROM Branch WHERE Name LIKE '%'+@name+'%';

END;

ELSE SET @mess = 'Филиала с таким названием не существует!';

END TRY

BEGIN CATCH

SET @mess = 'Ошибка системы!';

END CATCH;

END

--сменить руководителя отдела

CREATE PROCEDURE UpdSuperior

@login NVARCHAR(30),

@branch NVARCHAR(30),

@depart NVARCHAR(100),

@mess NVARCHAR(200) OUTPUT

AS

BEGIN

SET @mess='';

BEGIN TRY

DECLARE @checkEmployee int, @branchId int, @departId int, @superiorId int;

SELECT @checkEmployee = Id FROM Employee WHERE Login = @login;

IF(@checkEmployee > 0)

BEGIN

SELECT @branchId = Id FROM Branch WHERE Name = @branch;

SELECT @departId = Id FROM Department WHERE Name = @depart;

SELECT @superiorId = Id FROM Employee WHERE Assigned\_Branch\_Id = @branchId AND Depart\_Id = @departId AND Position LIKE '%Руководитель%';

IF(@superiorId > 0)

UPDATE TOP(1) Employee SET Position = 'Менеджер', Superior\_Emp\_Id = @checkEmployee WHERE Id = @superiorId;

UPDATE TOP(1) Employee SET Position = 'Руководитель отдела', Assigned\_Branch\_Id = @branchId, Depart\_Id = @departId, Superior\_Emp\_Id = NULL WHERE Login = @login;

END;

ELSE SET @mess = 'Сотрудника с таким логином не существует!';

END TRY

BEGIN CATCH

SET @mess = 'Ошибка системы!';

END CATCH;

END

--пополнить счет

CREATE PROCEDURE PutMoney

@accountId INT,

@amount MONEY,

@mess NVARCHAR(200) OUTPUT

AS

BEGIN

SET @mess='';

BEGIN TRY

DECLARE @checkAccount INT;

SELECT @checkAccount = count(\*) FROM Account WHERE Id = @accountId AND Close\_Date is NULL;

IF(@checkAccount != 0)

BEGIN

DECLARE @prodId INT, @branchId INT, @empId INT;

UPDATE Account SET Balance += @amount WHERE Id = @accountId;

SELECT @branchId = Open\_Branch\_Id, @empId = Open\_Emp\_Id FROM Account WHERE Id = @accountId;

SELECT @prodId = Id FROM Product WHERE Name = 'Пополнение счета';

INSERT INTO Acc\_Transaction(Account\_Id, Amount, Rest, DateTime, Type\_Prod\_Id, Execution\_Branch\_Id, Teller\_Emp\_Id) values(@accountId, @amount, default, GETDATE(), @prodId, @branchId, @empId);

END;

ELSE SET @mess = 'Такого счета не существует или он уже закрыт!';

END TRY

BEGIN CATCH

SET @mess = 'Ошибка системы!';

END CATCH;

END

--перевод денежных средств с обменом валюты

CREATE PROCEDURE TransExchange

@fromId INT,

@amount MONEY,

@toId INT,

@mess NVARCHAR(200) OUTPUT

AS

BEGIN

SET @mess='';

BEGIN TRY

DECLARE @checkFrom INT, @checkTo INT, @balance MONEY, @currIdFrom INT, @currIdTo INT, @curr1 VARCHAR(3), @curr2 VARCHAR(3);

SELECT @checkFrom = count(\*) FROM Account WHERE Id = @fromId AND Close\_Date is NULL;

SELECT @checkTo = count(\*) FROM Account WHERE Id = @toId AND Close\_Date is NULL;

IF(@checkFrom != 0 AND @checkTo != 0)

BEGIN

SELECT @balance = Balance, @currIdFrom = Curr\_Id FROM Account WHERE Id = @fromId;

SELECT @currIdTo = Curr\_Id FROM Account WHERE Id = @toId;

SELECT @curr1 = Name FROM Currency WHERE Id = @currIdFrom;

SELECT @curr2 = Name FROM Currency WHERE Id = @currIdTo;

IF(@balance>=@amount)

BEGIN

DECLARE @prodId INT, @branchId INT, @empId INT;

UPDATE Account SET Balance -= @amount WHERE Id = @fromId;

UPDATE Account SET Balance += @amount\*dbo.CurrExchange(@curr1,@curr2) WHERE Id = @toId;

SELECT @branchId = Open\_Branch\_Id, @empId = Open\_Emp\_Id FROM Account WHERE Id = @fromId;

SELECT @prodId = Id FROM Product WHERE Name = 'Перевод средств с обменом валюты';

INSERT INTO Acc\_Transaction(Account\_Id, Amount, Rest, DateTime, Type\_Prod\_Id, Execution\_Branch\_Id, Teller\_Emp\_Id) values(@fromId, @amount \* dbo.CurrExchange(@curr1, @curr2), default, GETDATE(), @prodId, @branchId, @empId);

END;

ELSE SET @mess = 'Недостаточно средств для перевода!';

END;

ELSE SET @mess = 'Такого(их) счета(ов) не существует или он(и) уже закрыт(ы)!';

END TRY

BEGIN CATCH

SET @mess = 'Ошибка системы!';

END CATCH;

END

--оформить кредит

CREATE PROCEDURE TakeCredit

@accountId INT,

@amount MONEY,

@end DATE,

@mess NVARCHAR(200) OUTPUT

AS

BEGIN

SET @mess='';

BEGIN TRY

DECLARE @checkAccount INT, @checkCredit INT, @prodId INT, @branchId INT, @empId INT, @tId INT;

SELECT @checkAccount = count(\*) FROM Account WHERE Id = @accountId AND Close\_Date is NULL;

IF(@checkAccount != 0)

BEGIN

SELECT @checkCredit = count(\*) FROM Temp\_Transaction WHERE Acc\_Tran\_Id = @accountId AND Completed = 0;

IF(@checkCredit < 2)

BEGIN

SELECT @prodId = Id FROM Product WHERE Name = 'Оформление кредита';

SELECT @branchId = Open\_Branch\_Id, @empId = Open\_Emp\_Id FROM Account WHERE Id = @accountId;

INSERT INTO Acc\_Transaction(Account\_Id, Amount, Rest, DateTime, Type\_Prod\_Id, Execution\_Branch\_Id, Teller\_Emp\_Id) VALUES( @accountId, @amount, @amount, GETDATE(), @prodId, @branchId, @empId);

SELECT TOP(1) @tId = Id FROM Acc\_Transaction ORDER BY Id DESC;

INSERT INTO Temp\_Transaction(Acc\_Tran\_Id, Date\_Retired, Completed) VALUES(@tid, @end, 0);

UPDATE Account SET Balance += @amount WHERE Id = @accountId;

END;

ELSE SET @mess = 'В кредите отказано! Одному клиенту допустимо иметь не более двух открытых кредитов!';

END;

ELSE SET @mess = 'Такого счета не существует или он уже закрыт!';

END TRY

BEGIN CATCH

SET @mess = 'Ошибка системы!';

END CATCH;

END

# **Приложение В**

--импорт из файла

DROP PROCEDURE ImportFromFile;

GO

CREATE PROCEDURE ImportFromFile AS

DECLARE @xml XML;

SELECT @xml = CONVERT(XML,bulkcolumn,2) FROM OPENROWSET(BULK 'D:\Bank\data.xml', SINGLE\_BLOB) AS X

SET ARITHABORT ON

DECLARE temp\_cursor CURSOR LOCAL READ\_ONLY FOR

SELECT Tbl.Col.value('Id[1]', 'int') AS Id,

Tbl.Col.value('Login[1]', 'nvarchar(30)') AS [Login],

Tbl.Col.value('Password[1]', 'nvarchar(30)') AS [Password],

Tbl.Col.value('First\_Name[1]', 'nvarchar(30)') AS First\_Name,

Tbl.Col.value('Last\_Name[1]', 'nvarchar(30)') AS Last\_Name,

Tbl.Col.value('Position[1]', 'nvarchar(30)') AS Position,

Tbl.Col.value('Start\_Date[1]', 'date') AS Start\_Date,

Tbl.Col.value('Assigned\_Branch\_Id[1]', 'int') AS Assigned\_Branch\_Id,

Tbl.Col.value('Depart\_Id[1]', 'int') AS Depart\_Id,

Tbl.Col.value('Superior\_Emp\_Id[1]', 'int') AS Superior\_Emp\_Id,

Tbl.Col.value('End\_Date[1]', 'date') AS End\_Date FROM @xml.nodes('/Root/Employee') Tbl(Col);

DECLARE @Id INT, @Login NVARCHAR(30), @Password NVARCHAR(30), @First\_Name NVARCHAR(30), @Last\_Name NVARCHAR(30), @Position NVARCHAR(30), @Start\_Date DATE, @Assigned\_Branch\_Id INT, @Depart\_Id INT, @Superior\_Emp\_Id INT, @End\_Date DATE;

SET IDENTITY\_INSERT Employee ON;

OPEN temp\_cursor;

FETCH NEXT FROM temp\_cursor INTO @Id, @Login, @Password, @First\_Name, @Last\_Name, @Position, @Start\_Date, @Assigned\_Branch\_Id, @Depart\_Id, @Superior\_Emp\_Id, @End\_Date;

WHILE @@FETCH\_STATUS = 0

BEGIN

INSERT INTO Employee(Id, Login, Password, First\_Name, Last\_Name, Position, Start\_Date, Assigned\_Branch\_Id, Depart\_Id, Superior\_Emp\_Id, End\_Date) SELECT @Id, @Login, @Password, @First\_Name, @Last\_Name, @Position, @Start\_Date, @Assigned\_Branch\_Id, @Depart\_Id, @Superior\_Emp\_Id, @End\_Date WHERE NOT EXISTS(SELECT \* FROM Employee WHERE Id = @Id) AND EXISTS(SELECT \* FROM Branch WHERE Id = @Assigned\_Branch\_Id) AND EXISTS(SELECT \* FROM Department WHERE Id = @Depart\_Id);

FETCH NEXT FROM temp\_cursor INTO @Id, @Login, @Password, @First\_Name, @Last\_Name, @Position, @Start\_Date, @Assigned\_Branch\_Id, @Depart\_Id, @Superior\_Emp\_Id, @End\_Date;

END;

SET IDENTITY\_INSERT Employee OFF;

GO

# **Приложение Г**

--триггер добавляет транзакцию "Оформление счета" при insert в таблицу Account

DROP TRIGGER TRANS\_AFTER\_ADD\_ACC;

GO

CREATE TRIGGER TRANS\_AFTER\_ADD\_ACC

ON Account AFTER INSERT

AS

DECLARE @id INT, @prodId INT, @branchId INT, @empId INT;

SELECT @id = Id, @branchId = Open\_Branch\_Id, @empId = Open\_Emp\_Id FROM inserted;

SELECT @prodId = Id FROM Product WHERE Name = 'Оформление счета';

INSERT INTO Acc\_Transaction(Account\_Id, Amount, Rest, DateTime, Type\_Prod\_Id, Execution\_Branch\_Id, Teller\_Emp\_Id) values(@id, default, default, GETDATE(), @prodId, @branchId, @empId);

RETURN;

GO

--триггер добавляет транзакцию "Оформление банковской карты" при insert в таблицу Credit\_Card

DROP TRIGGER TRANS\_AFTER\_ADD\_ACC;

GO

CREATE TRIGGER TRANS\_AFTER\_ADD\_CARD

ON Credit\_Card AFTER INSERT

AS

DECLARE @id INT, @prodId INT, @branchId INT, @empId INT;

SELECT @id = Account\_Id FROM inserted;

SELECT @branchId = Open\_Branch\_Id, @empId = Open\_Emp\_Id FROM Account WHERE Id = @id;

SELECT @prodId = Id FROM Product WHERE Name = 'Оформление банковской карты';

INSERT INTO Acc\_Transaction(Account\_Id, Amount, Rest, DateTime, Type\_Prod\_Id, Execution\_Branch\_Id, Teller\_Emp\_Id) values(@id, default, default, GETDATE(), @prodId, @branchId, @empId);

RETURN;

GO

--триггер обновляет начальника у сотрудников при обновлении начальника отдела

DROP TRIGGER TRANS\_AFTER\_UPD\_EMP;

GO

CREATE TRIGGER TRANS\_AFTER\_UPD\_EMP

ON Employee AFTER UPDATE

AS

DECLARE @id INT, @pos1 NVARCHAR(30), @pos2 NVARCHAR(30), @branchId INT, @departId INT;

SELECT @id = Id, @pos1 = Position, @branchId = Assigned\_Branch\_Id, @departId = Depart\_Id FROM deleted;

SELECT @pos2 = Position FROM inserted;

IF(@pos1 != @pos2 AND @pos2 = 'Руководитель отдела')

UPDATE Employee SET Superior\_Emp\_Id = @id WHERE Assigned\_Branch\_Id = @branchId AND Depart\_Id = @departId AND Id != @id;

RETURN;

GO

# **Приложение Д**

USE Bank\_DB;

CREATE LOGIN Admin WITH PASSWORD = 'Admin';

CREATE USER Admin FOR LOGIN Admin;

GRANT EXECUTE ON AddAccount TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON AddBranch TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON AddBusinessClient TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON AddDepartment TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON AddEmployee TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON AddIndividualClient TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON AddProduct TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON DelAccount TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON DelBranch TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON DelDepartment TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON DelEmployee TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON DelTempTransaction TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON UpdCustomer TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON UpdEmployee TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON UpdSuperior TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetAllAccounts TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetAllBranches TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetAllCurrency TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetAllDepartments TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetAllEmployees TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetAllProducts TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetAllTransactions TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetBusinessClients TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetIndividualClients TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetOneClient TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON GetOneEmployee TO Admin WITH GRANT OPTION;

ALTER SERVER ROLE [bulkadmin] ADD MEMBER Admin;

GRANT ALTER ON Employee TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON ExportToFile TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON ExecExport TO Admin WITH GRANT OPTION;

GRANT EXECUTE ON ImportFromFile TO Admin WITH GRANT OPTION;

CREATE LOGIN Client WITH PASSWORD = 'Client';

CREATE USER Client FOR LOGIN Client;

GRANT EXECUTE ON AddAccount TO Client;

GRANT EXECUTE ON CloseCredit TO Client;

GRANT EXECUTE ON CreateCard TO Client;

GRANT EXECUTE ON DelAccount TO Client;

GRANT EXECUTE ON DelCard TO Client;

GRANT EXECUTE ON GetAllBranches TO Client;

GRANT EXECUTE ON GetAllCurrency TO Client;

GRANT EXECUTE ON GetClientAccounts TO Client;

GRANT EXECUTE ON GetClientCard TO Client;

GRANT EXECUTE ON GetTempTransactions TO Client;

GRANT EXECUTE ON GetAccTransactions TO Client;

GRANT EXECUTE ON PayCredit TO Client;

GRANT EXECUTE ON PutMoney TO Client;

GRANT EXECUTE ON TakeCredit TO Client;

GRANT EXECUTE ON TransAnother TO Client;

GRANT EXECUTE ON TransExchange TO Client;

GRANT EXECUTE ON TransYourself TO Client;

GRANT EXECUTE ON WithdrawMoney TO Client;

GRANT EXECUTE ON dbo.CurrExchange TO Client;