МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация Базы данных

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных для банковского приложения»

Выполнил студент Глухова Д.В.

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

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

Консультант: асс. Н.И. Уласевич.

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

Нормоконтролер: асс. Н.И. Уласевич.

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

Курсовой проект защищен с оценкой

**Содержание**

[Введение 3](#_Toc166268802)

[1 Постановка задачи 4](#_Toc166268803)

[2 Проектирование базы данных 5](#_Toc166268804)

[2.1 Определение вариантов использования 5](#_Toc166268805)

[2.2 Диаграмма базы данных UML 6](#_Toc166268806)

[2.3 Вывод 7](#_Toc166268807)

[3 Разработка объектов базы данных 8](#_Toc166268808)

[3.1 Разработка таблиц базы данных 8](#_Toc166268809)

[3.2 Разработка индексов базы данных 10](#_Toc166268810)

[3.3 Разработка функций базы данных 12](#_Toc166268811)

[3.4 Разработка процедур базы данных 13](#_Toc166268812)

[3.5 Разработка пользователей базы данных 15](#_Toc166268813)

[3.6 Разработка представлений базы данных 16](#_Toc166268814)

[3.7 Разработка триггеров базы данных 17](#_Toc166268815)

[3.8 Вывод 18](#_Toc166268816)

[4 Описание импорта и экспорта 19](#_Toc166268817)

[4.1 Описание импорта данных 19](#_Toc166268818)

[4.2 Описание экспорта данных 19](#_Toc166268819)

[4.3 Вывод 20](#_Toc166268820)

[5 Тестирование производительности 21](#_Toc166268821)

[6 Описание технологии и ее применения в базе данных 24](#_Toc166268822)

[7 Руководство пользователя 25](#_Toc166268823)

[Заключение 28](#_Toc166268824)

[Список используемых источников 29](#_Toc166268825)

[Приложение А Листинг создания таблиц 30](#_Toc166268826)

[Приложение Б Листинг создания функций 32](#_Toc166268827)

[Приложение В Листинг создания процедур 39](#_Toc166268828)

[Приложение Г Листинг создания пользователей и ролей 52](#_Toc166268829)

[Приложение Д Листинг создания триггеров 54](#_Toc166268830)

# **Введение**

Мобильные устройства стали неотъемлемой составляющей в жизни большинства, банковские приложения становятся все более популярными и востребованными. Они предоставляют пользователям практичный и надежный способ для управления финансами.

Банковские приложения стали ассистентами, предоставляющими доступ к финансовым услугам в любое время. Они позволяют пользователям мониторить свои финансы, управлять карточками и счетами, осуществлять переводы и платежи.

Одной из ключевых особенностей банковских приложений является их безопасность. С помощью передовых методов шифрования и аутентификации, банки стремятся защитить конфиденциальность и безопасность финансовых данных своих клиентов. Это позволяет пользователям чувствовать себя уверенно и спокойно при работе с приложением, зная, что их финансы защищены.

В проекте будет рассмотрена разработка банковского приложения, которое сочетает в себе удобство использования, и обширный функционал. Будет рассмотрено проектирование и разработка базы данных, а также реализация следующего функционала:

* возможность регистрации и входа;
* переводы с карты на карту;
* подтверждение перевода;
* создание новых карт;
* оплата разных услуг;
* просмотр истории транзакций.

Целью данного проекта является создание интуитивно понятного и эффективного банковского приложения, базы данных для него, которое удовлетворит потребности клиентов и поможет им легко и управлять своими финансами.

**1 Постановка задачи**

Разработка банковского приложения представляет собой важный стратегический шаг, направленный на обеспечение высокого уровня обслуживания и удовлетворение потребностей современного клиента. Цель состоит в предоставлении широкого набора функций, которые позволят клиентам полностью контролировать свои финансы и получать высококачественное обслуживание.

Одной из основных задач проекта является разработка архитектуры банковского приложения с интуитивно понятным дизайном. Необходимо создать приложение, которое будет легким в использовании и приятным для пользователей. Это включает разработку удобного интерфейса, который позволит клиентам легко ориентироваться по функциям приложения и выполнять операции с минимальными усилиями.

Другой важной задачей является построение надежной базы данных, которая будет хранить информацию о клиентах, их счетах, транзакциях и других важных данных.

В рамках разработки планируются следующие роли: администратор и пользователь.

Необходимо также провести тестирование готового продукта, чтобы убедиться в его надежности, функциональности и соответствии требованиям. Тестирование позволит выявить и устранить возможные ошибки и проблемы, а также обеспечит высокое качество и надежность приложения перед его внедрением.

Итак, разработка банковского приложения имеет несколько ключевых задач: разработка архитектуры с интуитивно понятным дизайном, построение безопасной базы данных и проведение тестирования.

При этом должны быть выполнены следующие требования:

˗ база данных должна быть реализована в СУБД Microsoft SQL Server.

˗ доступ к данным должен осуществляться только через соответствующие процедуры;

˗ должен быть проведен импорт данных из Excel файлов, экспорт данных в формат JSON;

˗ протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости.

˗ проанализировать планы запросов к таблице;

˗ применить технологию разработки системы e-mail-уведомлений о событиях в своей базе данных.

**2 Проектирование базы данных**

**2.1 Определение вариантов использования**

Проектирование баз данных представляет собой процесс создания структуры базы данных и определения необходимых ограничений целостности.

К основным задачам проектирования БД относятся: обеспечение хранения в БД всей необходимой информации, обеспечение возможности получения данных по всем необходимым запросам, сокращение избыточности и дублирования данных, обеспечение целостности базы данных.

Проектирование базы данных проходит через два основных этапа: концептуальное и логическое проектирование. Концептуальное проектирование включает определение вариантов использования и функциональности разрабатываемой программной системы. На основе этого определяются таблицы, связи и ограничения целостности.

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. Диаграмма показана на рисунке 2.1.

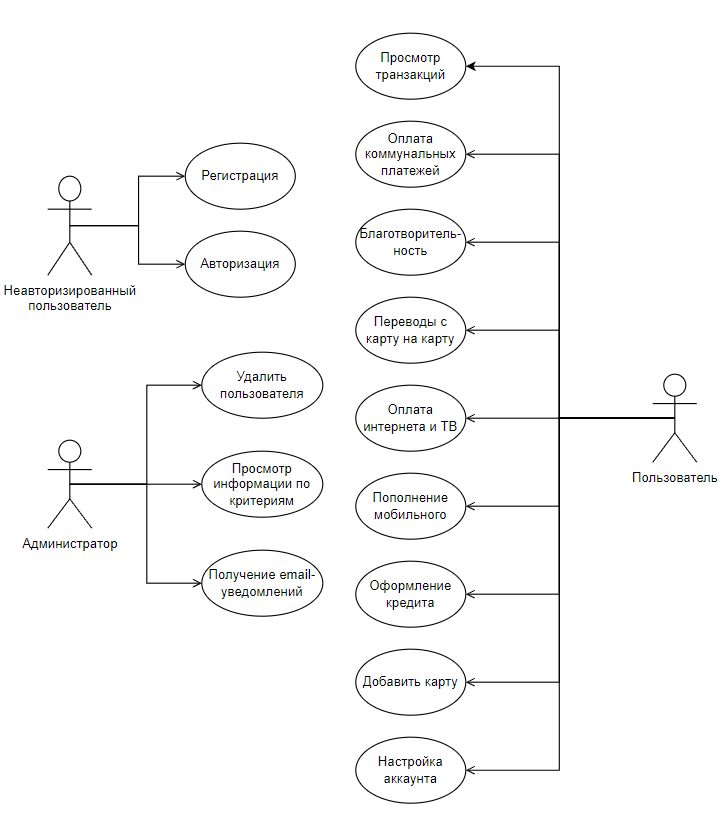


Рисунок 2.1 – UML диаграмма вариантов использования

При открытии приложения пользователь является гостем, он может войти под уже существующими данными или создать новый аккаунт. В обоих случаях после успешного входа пользователь видит главный экран приложения, где представлены доступные ему функции.

Пользователь может выбрать любую функцию из списка и выполнить необходимые действия. Авторизованному пользователю доступны следующие функции: переводы с карты на карту, оплата интернета и ТВ, коммунальных платежей, благотворительность, добавление новых карт, пополнение мобильного, кредит, настройка аккаунта. Все функции будут работать с базой данных.

**2.2 Диаграмма базы данных UML**

На следующем этапе процесса предстоит составить наглядную картину своей базы данных.

Любой сложный процесс разработки программного обеспечения должен начинаться с моделирования данных. Модели данных помогают программистам понять область и правильно спланировать свою работу. Диаграммы баз данных графически изображают схемы баз данных и связи между объектами баз данных. Хорошо построенная модель базы данных часто работает быстро, часто быстрее, чем предполагалось. Принципы, заложенные в модели данных, должны быть четкими и согласованными, чтобы работать наилучшим образом. После этого модель базы данных должна быть преобразована в проект базы данных с использованием правильных принципов. Диаграмма базы данных представлена на рисунке 2.2.

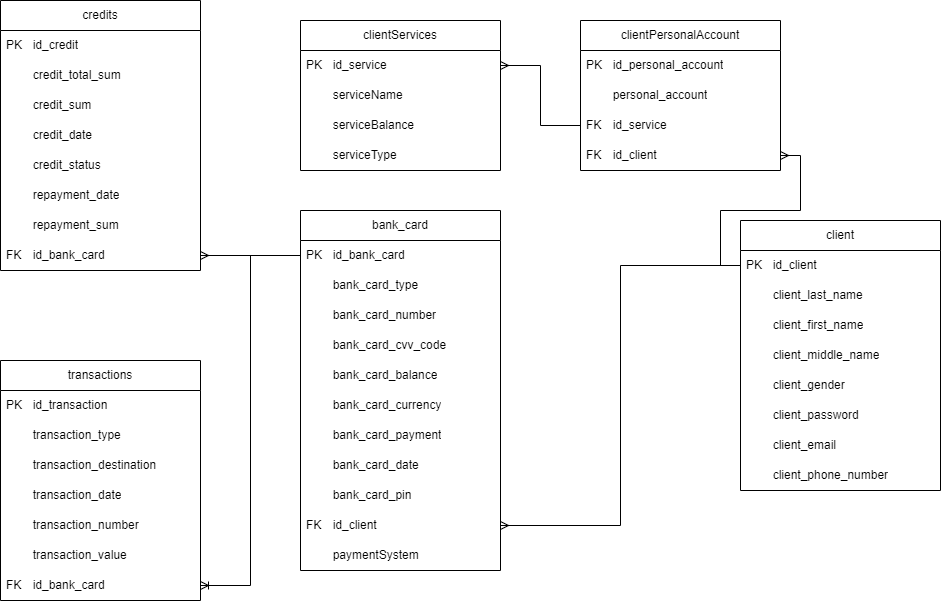


Рисунок 2.2 – Диаграмма базы данных

Эта диаграмма помогает программистам понять структуру базы данных и правильно спланировать свою работу. Она отображает схемы баз данных и связи между объектами баз данных. Хорошо построенная модель базы данных обеспечивает быструю и эффективную работу приложения.

Одним из важных аспектов проектирования базы данных является определение взаимосвязей между таблицами. Для этого используются связи, такие как один-ко-многим и многие-ко-многим. Эти связи позволяют связывать данные из разных таблиц и обеспечивают целостность и связность данных.

Чаще всего неправильное использование модели базы данных является ошибкой, чем неисправностью приложения базы данных (SQL Server). Когда этот вопрос решен, исполнение становится превосходным. База данных может быть понята с помощью моделирования, что позволяет оптимизировать ее для быстрой эффективности.

**2.3 Вывод**

Проектирование базы данных играет ключевую роль в обеспечении надежности, безопасности и эффективности работы приложения. Путем внимательного изучения требований к приложению, определены, какие данные будут храниться и как они будут использоваться. Правильное проектирование может повысить эффективность работы базы данных и уменьшить количество запросов к ней, что в свою очередь уменьшает время ответа, повышает производительность и общую надежность базы.

Данные разделены на логические таблицы, установлены связи и определены первичные и внешние ключи для обеспечения целостности данных. Была определена структура данных, применены принципы нормализации.

Таким образом, проектирование базы данных является ключевым этапом в разработке приложения, и при правильном подходе к проектированию можно обеспечить надежность, эффективность и безопасность работы базы данных.

**3 Разработка объектов базы данных**

**3.1 Разработка таблиц базы данных**

При разработке приложения для курсового проекта была использована база данных SQL Server Management Studio.

Таблица – это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Она состоит из столбцов и строк.

Столбцы таблицы называют полями; каждое поле характеризуется своим именем (названием соответствующего свойства) и типом данных, отражающих значения данного свойства. Каждое поле обладает определенным набором свойств (размер, формат и др.).

Поле базы данных – это столбец таблицы, включающий в себя значения определенного свойства.

В каждой таблице должно быть, по крайней мере, одно ключевое поле, содержимое которого уникально для любой записи в этой таблице.

Значения ключевого поля однозначно определяют каждую запись в таблице.

Для реализации базы данных было разработано 6 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: client, bank\_card, transactions, credits, clientServices, clientPersonalAccount. Ниже будет описание про каждую из них более подробно.

Таблица client содержит информацию о клиентах. Состоит из следующих столбцов (таблица 3.1):

Таблица 3.1 «client»

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип данных |
| id\_client | Уникальный идентификатор клиента, первичный ключ | INT |
| client\_last\_name | Фамилия клиента | NVARCHAR(50) |
| client\_first\_name | Имя клиента | NVARCHAR(50) |
| client\_middle\_name | Отчество клиента | NVARCHAR(50) |
| client\_gender | Пол клиента | NVARCHAR(50) |
| client\_password | Пароль клиента | NVARCHAR(50) |
| client\_email | Электронная почта клиента | NVARCHAR(50) |
| client\_phone\_number | Номер телефона клиента | NVARCHAR(50) |
| is\_admin | Администратор/пользователь | BIT |

Следующая таблица к разработке: Таблица bank\_card содержит информацию о банковских картах клиентов. Состоит из следующих столбцов (таблица 3.2):

Таблица 3.2 «bank\_card»

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип данных |
| id\_bank\_card | Уникальный идентификатор карты, первичный ключ | INT |
| bank\_card\_type | Тип карты | NVARCHAR(50) |
| bank\_card\_number | Номер карты | NVARCHAR(16) |
| bank\_card\_cvv\_code | CVV-код карты | NVARCHAR(3) |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
| bank\_card\_balance | Баланс карты | MONEY |
| bank\_card\_currency | Валюта карты | NVARCHAR(10) |
| bank\_card\_payment | Платежная система карты | NVARCHAR(50) |
| bank\_card\_date | Срок действия карты | DATE |
| bank\_card\_pin | Пин-код от карты | INT |
| id\_client | Уникальный идентификатор клиента, внешний ключ | INT |

Таблица transactions содержит информацию о транзакциях с карты. Состоит из следующих столбцов (таблица 3.3):

Таблица 3.3 «transactions»

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id\_transaction | Уникальный идентификатор транзакции, первичный ключ | INT |
| transaction\_type | Тип транзакции | VARCHAR(50) |
| transaction\_destination | Данные получателя | VARCHAR(200) |
| transaction\_date | Дата совершения транзакции | DATE |
| transaction\_number | Номер транзакции | NACHAR(50) |
| transaction\_value | Значение транзакции | MONEY |
| id\_bank\_card | Уникальный идентификатор карты, внешний ключ | INT |

Таблица credits содержит информацию о кредитах пользователя. Состоит из следующих столбцов (таблица 3.4):

Таблица 3.4 «credits»

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id\_credit | Уникальный идентификатор кредита, первичный ключ | INT |
| credit\_total\_sum | Общая сумма | MONEY |
| credit\_sum | Сумма по карте | MONEY |
| credit\_date | Дата оформления кредита | DATE |
| credit\_status | Статус кредита | BIT |
| repayment\_date | Дата платежа | DATE |
| repayment\_sum | Сумма платежа | MONEY |
| id\_bank\_card | Уникальный идентификатор карты, внешний ключ | INT |

Таблица clientServices является частью базы данных и содержит информацию о различных сервисах. Состоит из следующих столбцов (таблица 3.5):

Таблица 3.5 «clientServices»

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id\_service | Уникальный идентификатор сервиса, первичный ключ | INT |
| serviceName | Название сервиса | VARCHAR(100) |
| serviceBalance | Баланс на сервисе | MONEY |
| serviceType | Тип сервиса | VARCHAR(100) |

Таблица clientPersonalAccount содержит информацию о персональном аккаунте. Состоит из следующих столбцов (таблица 3.6):

Таблица 3.6 «clientPersonalAccount»

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id\_personal\_account | Уникальный идентификатор персонального аккаунта, первичный ключ | INT |
| personal\_account | Данные аккаунта(номер счета) | VARCHAR(10) |
| id\_service | Уникальный идентификатор сервиса, внешний ключ | INT |
| id\_client | Уникальный идентификатор клиента, внешний ключ | INT |

Скрипты создания таблиц и наложение ограничений целостности на них представлены в приложении А данной записки.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных.

Для более удобной работы с базой данных можно создать несколько функций и процедур.

**3.2 Разработка индексов базы данных**

Индекс – это объект БД, содержащий упорядоченные значения указанных столбцов таблицы и ссылки на физическое размещение записи с данными значениями. Индекс позволяет ускорить поиск данных в таблице и упорядочивание данных. Один из наиболее распространенных алгоритмов индексации – это B-дерево. B-дерево – это сбалансированное дерево поиска, которое обеспечивает эффективный поиск элементов, используя ключи для сортировки данных в индексе.

Индексы функционируют как предметные указатели в книге, позволяя быстро находить информацию без прочтения всего текста. Они представляют собой специальные таблицы, строки которых, в отличие от обычных таблиц данных, расположены в строго определённом порядке. Но вместо того, чтобы содержать все данные о некоторой записи, индекс содержит только столбец (или столбцы), используемый, чтобы найти строки в таблице данных, вместе с информацией, описывающей, где физически расположена эта строка. Таким образом, роль индексов состоит в том, чтобы облегчить поиск подмножества строк и столбцов таблицы без необходимости сканировать каждую строку в таблице.

В качестве примера рассмотрим индекс idx\_bank\_card\_number на таблице bank\_card. Этот индекс создан на столбце bank\_card\_number таблицы bank\_card, который содержит номер банковской карты. Цель данного индекса заключается в ускорении операций поиска и сортировки клиентов по номеру карты. Создание части индексов представлено в листинге 3.1.

|  |
| --- |
| CREATE INDEX idx\_LastName ON client(client\_last\_name);  CREATE INDEX idx\_FirstName ON client(client\_first\_name);  CREATE INDEX idx\_Gender ON client(client\_gender);  CREATE INDEX idx\_bank\_card\_number ON bank\_card(bank\_card\_number);  CREATE INDEX idx\_bank\_card\_type ON bank\_card (bank\_card\_type);  CREATE INDEX idx\_transaction\_type ON transactions (transaction\_type);  CREATE INDEX idx\_transaction\_number ON transactions (transaction\_number);  CREATE INDEX idx\_credit\_total\_sum ON credits (credit\_total\_sum);  CREATE INDEX idx\_credit\_date ON credits (credit\_date);  CREATE INDEX idx\_serviceName ON clientServices(serviceName);  CREATE INDEX idx\_serviceType ON clientServices (serviceType);  CREATE INDEX idx\_personal\_account ON clientPersonalAccount(personal\_account); |

Листинг 3.1 – Создание индексов

idx\_LastName – индекс для столбца client\_last\_name таблицы client.

idx\_FirstName – индекс для столбца client\_first\_name таблицы client.

idx\_Gender – индекс для столбца client\_gender таблицы client.

idx\_bank\_card\_number – индекс для столбца bank\_card\_number таблицы bank\_card.

idx\_bank\_card\_type – индекс для столбца bank\_card\_type таблицы bank\_card.

idx\_transaction\_type– индекс для столбца transaction\_type таблицы transactions.

idx\_transaction\_number– индекс для столбца transaction\_number таблицы transactions.

idx\_credit\_total\_sum– индекс для столбца credit\_total\_sum таблицы credits.

idx\_credit\_date – индекс для столбца credit\_date таблицы credits.

idx\_serviceName– индекс для столбца serviceName таблицы clientServices.

idx\_ serviceType – индекс для столбца serviceType таблицы clientServices.

idx\_personal\_account – индекс для столбца personal\_account таблицы clientPersonalAccount.

Каждый индекс представляет собой таблицу (пусть и особый тип таблицы, но все же это таблица). Следовательно, каждый раз, когда строка добавляется в таблицу или удаляется из неё, должны быть изменены все индексы в этой таблице. При обновлении строки любые индексы для столбца (или столбцов), которые были затронуты, также должны быть изменены. Следовательно, чем больше у вас индексов, тем больше должен работать СУБД, чтобы поддерживать все объекты схемы в актуальном состоянии — что приводит к замедлению работы.

Более того, индексы занимают дополнительное место на диске и требуют внимательного управления со стороны администраторов баз данных. Поэтому оптимальным решением является создание индексов только тогда, когда это действительно необходимо. Если индекс нужен временно, например, для выполнения месячного отчёта, его можно добавить перед началом процедуры и удалить после её завершения.

В итоге, идеальный подход заключается в нахождении баланса: необходимо иметь достаточно индексов для эффективной работы, но не столько, чтобы это сказывалось на производительности.

**3.3 Разработка функций базы данных**

SQL-функция выполняет произвольный набор команд на языке SQL, разделённых точкой с запятой, и возвращает результат последнего запроса в списке. Помимо запросов SELECT, эти команды могут включать запросы, изменяющие данные (INSERT, UPDATE и DELETE), а также другие SQL-команды.

С SQL Server вы можете создавать ваши собственные функции, добавляющие и расширяющие функции, предоставляемые системой. Функции могут получать 0 или более параметров и возвращать скалярное значение или таблицу.

Скалярные функции – похожи на встроенные функции.

Функция, возвращающая таблицу – возвращает результат единичного оператора SELECT. Он похож на объект просмотра, но имеет большую эластичность благодаря использованию параметров, и расширяет возможности индексированного объекта просмотра;

Многооператорная функция – возвращает таблицу, созданную одним или несколькими операторами Transact-SQL, чем напоминает хранимые процедуры. В отличие от процедур, на такие функции можно ссылаться в WHERE как на объект просмотра.

Рассмотрим функцию, которая используется для поиска по фамилии, имени или номеру банковской карты одновременно. Пример создания функции представлен на листинге 3.2.

|  |
| --- |
| CREATE FUNCTION SearchClients  (  @searchString NVARCHAR(100)  )  RETURNS TABLE  AS  RETURN  (  SELECT \*  FROM client  WHERE client\_first\_name LIKE '%' + @searchString + '%'  OR client\_last\_name LIKE '%' + @searchString + '%'  OR EXISTS (  SELECT 1  FROM bank\_card  WHERE bank\_card.id\_client = client.id\_client  AND (  bank\_card.bank\_card\_number LIKE '%' + @searchString + '%'  OR CAST(client.id\_client AS NVARCHAR(50)) = @searchString  )  )  ); |

Листинг 3.2 – Функция поиска по нескольким параметрам

Функция SearchClients принимает один параметр @searchString типа NVARCHAR(100) и возвращает таблицу результатов. Внутри функции используется оператор SELECT для выборки всех полей из таблицы client. Затем применяются условия фильтрации, которые выполняются в операторе WHERE. Условия фильтрации позволяют найти клиентов, удовлетворяющих следующим критериям: client\_first\_name, client\_last\_name, bank\_card\_number.

Функция SearchClients может быть использована для поиска клиентов по имени, фамилии или номеру банковской карты. При передаче значения параметра @searchString, функция вернет все записи из таблицы client, удовлетворяющие заданным условиям фильтрации.

GetTransById – получение транзакции по айди.

GetTransByNumber – получение транзакций по номеру карты.

SearchClientByLastName – поиск клиента по фамилии.

CalculateTotalBalance – подсчет общего баланса на всех картах клиента.

SearchByLastName – поиск клиентов по фамилии на полное совпадение.

GetClientsByGender – поиск клиентов по полу.

GetRecentTransactions – получение последних транзакций.

GetClientsWithCreditDebt – клиенты с активными кредитами.

GetLastTransactionForEachCard – получение последней транзакции для каждой карты.

HasActiveCredit – имеет ли клиент активный кредит.

SearchClients – поиск по фамилии, имени или номеру карты.

GetTransactionTotalAmount – получение суммы транзакций.

GetClientCount – получение количества клиентов.

GetBankCardCount – получение количества карт.

GetTransactionCount – получение количества транзакций.

GetCreditCount – получение количества кредитов.

GetActiveServiceCount – получение количества активных сервисов.

GetClientsWithoutActiveBankCards – получение клиентов с активными банковскими картами.

Код всех разработанных функции представлены в приложении Б.

**3.4 Разработка процедур базы данных**

В простейшем виде хранимая процедура - это не более чем набор операторов Transact-SQL (T-SQL), который хранится в базе данных и может выполняться при вызове хранимой процедуры по имени. Имя присваивается хранимой процедуре при ее создании.

Наиболее часто используются пользовательские хранимые процедуры. Такая процедура может создаваться в пользовательской базе данных или в любой системной базе данных за исключением базы данных Resource.

Временная хранимая процедура - это вид пользовательской хранимой процедуры, которая очень похожа на временные таблицы. Временные хранимые процедуры хранятся в базе данных "tempdb", видной на вкладке "System Databases". Эти временные хранимые процедуры могут использоваться как локальные или глобальные.

Довольно часто используются системные хранимые процедуры. Полезно познакомиться с ними, чтобы не изобретать колесо. Эти хранимые процедуры включены по умолчанию в установку SQL Server и всегда будут начинаться с префикса "sys.sp\_".

Хранимые процедуры имеют много преимуществ. Он уменьшают сетевой трафик, поскольку только имя передается по сети от приложения на сервер. Правда, мы можем передавать еще несколько параметров, но передача имени хранимой процедуры и нескольких параметров не так сильно загружает сеть, как передача всего кода всякий раз, когда нужно этот код выполнить.

Хранимые процедуры усиливают безопасность. Код в хранимой процедуре определяет, какие действия производятся над объектами базы данных. Это означает, что при всяком вызове хранимой процедуры один и тот же блок кода выполняется одним и тем же способом. То есть мы не должны беспокоиться об ошибках в коде, повторяя набор одного и того же кода всякий раз, когда вам нужно выполнить его.

В качестве примера рассмотрим процедуру для добавления клиента в таблицу client. Пример создания функции представлен на листинге 3.3.

Данная процедура называется CreateClient и служит для создания новой записи о клиенте в базе данных.

Каждое значение параметра соответствует соответствующему столбцу в таблице client Таким образом, новая запись будет содержать информацию о фамилии, имени, отчестве, поле, пароле, электронной почте, номере телефона и флаге администратора указанного клиента.

|  |
| --- |
| CREATE OR ALTER PROCEDURE dbo.CreateClient  @client\_last\_name nvarchar(50),  @client\_first\_name nvarchar(50),  @client\_middle\_name nvarchar(50),  @client\_gender nvarchar(50),  @client\_password nvarchar(50),  @client\_email nvarchar(50),  @client\_phone\_number nchar(13),  @is\_admin bit  AS  BEGIN  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number, is\_admin)  VALUES (@client\_last\_name, @client\_first\_name, @client\_middle\_name, @client\_gender, @client\_password, @client\_email, @client\_phone\_number, @is\_admin)  END |

Листинг 3.3 – Процедура добавления клиента

Таким образом, в результате выполнения процедуры мы получаем последнюю транзакцию для каждой карты, связанной с указанным клиентом.

CreateBankCard – процедура создания карты.

CreateClient – процедура создания клиента.

CreateClientPersonalAccount – процедура создания акканта клиента.

CreateClientService – процедура создания клиент сервиса.

CreateCredit – процедура создания кредита.

CreateTransaction – процедура создания транзакции.

DeleteBankCard – процедура удаления карты.

DeleteClient – удаления клиента.

DeleteClientPersonalAccount – удаление персонального аккаунта.

DeleteClientService – удаление клиент-сервиса.

DeleteCredit – удаление кредита.

DeleteTransaction – удаление транзакций.

PopulateClients – заполнение таблицы клиенты.

SaveBankCardDataToJson – сохранение таблицы с картами в JSON.

SaveClientDataToJson – сохранение таблицы с клиентами в JSON.

SaveClientPersonalAccountDataToJson – сохранение транзакций в JSON.

SaveTableDataToJson – сохранение таблиц по параметру в JSON.

UpdateBankCard – обновление карты.

UpdateClient – обновление клиента.

UpdateClientPersonalAccount – обновление персонального аккаунта.

UpdateClientService – обновление клиент-сервиса.

UpdateCredit – обновление кредита.

UpdateTransaction – обновление транзакции.

Код всех разработанных процедур представлен в приложении В.

**3.5 Разработка пользователей базы данных**

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

При проектировании базы данных было использовано 2 пользователя: администратор, пользователь. Каждый пользователь имеет разные привилегии в соответствии с его позицией.

Администратор управляет общей информацией и данными базами данных, ему доступны процедуры и функции. Доступ со стороны приложения предоставляется обоим пользователям. При вводе конкретного логина и пароля можно зайти с администратора на отдельную форму и просматривать данные, при вводе номера телефона и пароля – с пользователя.

Создание пользователей представлено в листинге 3.4.

|  |
| --- |
| CREATE USER admin\_bank FOR LOGIN admin\_bank\_login;  CREATE USER user\_bank FOR LOGIN user\_bank\_login;  CREATE ROLE administrator\_role;  CREATE ROLE user\_role;  ALTER ROLE administrator\_role ADD MEMBER admin\_bank;  ALTER ROLE user\_role ADD MEMBER user\_bank; |

Листинг 3.4 – Создание пользователей и выдача привилегий

Полный код создания пользователей и выдача привилегий будет представлен в приложении Г.

Далее для созданного пользователя базы данных были выданы необходимые привилегии необходимые для функционала.

**3.6 Разработка представлений базы данных**

В SQL представления (views) содержат строки и столбцы, аналогичные таблицам, однако без фактических данных. Представление можно рассматривать как виртуальную таблицу, созданную из одной или нескольких таблиц, чтобы упростить работу с данными.

Представления создаются в случаях, когда необходимо дать пользователю возможность просматривать не всю таблицу, а только ее часть, а также для объединения двух или более таблиц.

В листинге 3.5 представлены все разработанные представления.

|  |
| --- |
| --------------------КЛИЕНТЫ И КАРТЫ----------------------  CREATE VIEW ClientBankInfo AS  SELECT c.id\_client, c.client\_last\_name, c.client\_first\_name, c.client\_middle\_name, c.client\_gender, b.id\_bank\_card, b.bank\_card\_number, b.bank\_card\_balance  FROM client AS c  JOIN bank\_card AS b ON c.id\_client = b.id\_client;  -------------------------------------------------------------  SELECT \* FROM ClientBankInfo;  -------------------------------------------------------------  -----------------ИНФО О КРЕДИТАХ---------------------------  CREATE VIEW ClientCreditInfo AS  SELECT c.id\_client, c.client\_last\_name, c.client\_first\_name, c.client\_middle\_name, SUM(cr.credit\_sum) AS total\_credit\_sum  FROM client AS c  JOIN bank\_card AS b ON c.id\_client = b.id\_client  JOIN credits AS cr ON b.id\_bank\_card = cr.id\_bank\_card  GROUP BY c.id\_client, c.client\_last\_name, c.client\_first\_name, c.client\_middle\_name;  -------------------------------------------------------------  SELECT \* FROM ClientCreditInfo;  -------------------------------------------------------------  -----------------АКТИВНЫЕ УСЛУГИ---------------------------  CREATE VIEW ClientServiceInfo AS  SELECT c.id\_client, c.client\_last\_name, c.client\_first\_name, c.client\_middle\_name, COUNT(cs.id\_service) AS active\_service\_count  FROM client AS c  JOIN clientPersonalAccount AS cpa ON c.id\_client = cpa.id\_client  JOIN clientServices AS cs ON cpa.id\_service = cs.id\_service  WHERE cs.serviceBalance > 0  GROUP BY c.id\_client, c.client\_last\_name, c.client\_first\_name, c.client\_middle\_name; |

Листинг 3.5 – Создание представлений

Представление ClientBankInfo предоставляет информацию о клиентах и связанных с ними банковских картах. Оно объединяет таблицы client и bank\_card по идентификатору клиента и включает столбцы, такие как идентификатор клиента, фамилию, имя, отчество, пол клиента, идентификатор банковской карты, номер карты и баланс карты. Это представление позволяет получить все эти данные в одном запросе.

Представление ClientCreditInfo предоставляет информацию о клиентах и общей сумме их кредитов. Оно связывает таблицы client, bank\_card и credits, чтобы получить сумму кредитов для каждого клиента. Данные группируются по идентификатору клиента, фамилии, имени и отчеству, и включают общую сумму кредитов клиента.

Представление ClientServiceInfo предоставляет информацию о клиентах и количестве активных услуг на их счетах. Оно связывает таблицы client, clientPersonalAccount и clientServices, чтобы подсчитать количество активных услуг для каждого клиента. Услуги считаются активными, если баланс счета по услуге больше нуля. Данные группируются по идентификатору клиента, фамилии, имени и отчеству, и включают количество активных услуг на счетах клиента.

Используя эти представления, можно получить нужные данные о клиентах, их банковских картах, сумме кредитов и количестве активных услуг в удобном и эффективном формате, без необходимости писать сложные и длинные запросы с каждым разом.

**3.7 Разработка триггеров базы данных**

Триггер SQL Server – это часть процедурного кода, как и хранимая процедура, но который выполняется только при наступлении определенного события. Имеются различные типы событий, которые могут вызвать срабатывание триггера. Например, вставка строки в таблицу, изменение структуры таблицы или авторизация пользователя в экземпляре SQL Server. В листинге 3.5 представлен один из триггеров.

|  |
| --- |
| CREATE TRIGGER i\_tgLastName ON dbo.client  FOR INSERT  AS  DECLARE @Name varchar(50)  SELECT @Name=client\_last\_name  FROM inserted  IF @Name='admin'  BEGIN  PRINT 'ОШИБКА ПРИ ПОПЫТКЕ ВСТАВИТЬ ADMIN. ВЫ НЕ ИМЕЕТЕ ДОСТАТОЧНЫХ ПРАВ.'  ROLLBACK TRANSACTION  END; |

Листинг 3.4 – Создание триггера i\_tgLastName

Выше приведен триггер, который запрещает вставку записей с client\_last\_name равным "admin" в таблицу client. Если происходит попытка вставки таких данных, триггер откатывает транзакцию и выводит сообщение об ошибке. Ниже представлен перечень триггеров. Скрипты представлены в приложении Д.

i\_tgLastName – обработка ошибки при попытке вставить admin.

SetClientIsAdminFlag – запрет на is\_admin = 1.

CheckTransactionBalance – будет автоматически проверять баланс банковской карты перед вставкой новой транзакции и отклонять транзакцию, если баланс недостаточен.

CheckCreditSum – триггер для проверки суммы кредита перед вставкой.

CheckBankCardType – триггер для проверки типа банковской карты перед вставкой.

CheckClientFields – проверка на заполнение обязательных полей.

CheckBankCardNumber – проверка на уникальность карты.

CheckUniquePhoneNumber – проверка на уникальность номера.

**3.8 Вывод**

В результате были разработаны необходимые таблицы. Для эффективного использования базы данных используются индексы на столбцах, которые часто используются в запросах с поиском данных. Индексы позволяют ускорить выполнение запросов и повысить производительность системы. Разработанные функции используются для обработки данных в базе данных и предоставляют удобные средства для выполнения различных операций, таких как вычисления, преобразования данных, агрегирование и т. д. Хранимые процедуры в базе данных позволяют увеличить производительность приложения, повторно использовать код, упростить безопасность и управление базой данных, а также повысить производительность сети. Они представляют собой набор инструкций SQL, выполняемых на сервере базы данных, и обеспечивают модульность, эффективность и централизованное управление бизнес-логикой и операциями базы данных. Разделение ролей позволяет ограничить доступ пользователя к определенным функциям и данным. Администратор имеет расширенные привилегии и может управлять системой, в то время как обычные пользователи имеют доступ только к необходимым им функциям. Это помогает предотвратить несанкционированный доступ и защитить данные от нежелательного изменения или утечки. Все эти концепции содействуют разработке и управлению программными и базами данных, способствуя созданию надежных и гибких систем.

**4 Описание импорта и экспорта**

1. **Описание импорта данных**

Импортировать данные из файлов Excel в SQL Server или базу данных SQL Azure можно несколькими способами. Некоторые методы позволяют импортировать данные за один шаг непосредственно из файлов Excel. Для других методов необходимо экспортировать данные Excel в виде текста (CSV-файла), прежде чем их можно будет импортировать. Импорт осуществляется с помощью мастера импорта и экспорта SQL Server. Для этого происходит подключение к экземпляру, выбирается база данных для импорта, затем в пункте “Задачи” необходимо выбрать “Импортировать данные” и ввести нужные данные.

Для импорта данных из файла в формате CSV был разработан скрипт, который представлен в листинге 4.1.

В качестве разделителя в исходном CSV-файле используются запятые, а для разделителя строки – символ переноса на новую строку. Как правило, данные из CSV-файла нужно сохранять во временную таблицу, а затем из нее вставлять в основную, т.к. зачастую необходимо парсить даты, числа с плавающей запятой и другие форматы. Их можно спарсить в строку во временной таблице и привести к нужному формату в основной.

|  |
| --- |
| CREATE OR ALTER PROCEDURE SaveTableDataToJson  @tableName NVARCHAR(128)  AS  BEGIN  DECLARE @fileName NVARCHAR(128);  SET @fileName = REPLACE(@tableName, ' ', '') + '.json';  DECLARE @sql NVARCHAR(1000);  SET @sql = 'bcp "SELECT \* FROM ' + QUOTENAME(@tableName) + ' ' +  'FOR JSON PATH, INCLUDE\_NULL\_VALUES" ' +  'queryout "e:\' + @fileName + '" ' +  '-c -S DESKTOP-F644G2I -d MobileBank -T';  EXEC sys.XP\_CMDSHELL @sql;  END;  ---------------------------------------------  EXEC SaveTableDataToJson @tableName = 'client'; |

Листинг 4.1 – Импорт данных из CSV

Таким образом, пользователи базы данных могут легко импортировать данные в формате CSV.

1. **Описание экспорта данных**

Для таблицы клиенты в базе данных доступна процедура экспорта данных в формате JSON. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в JSON, была разработана функция, результатом которой стало создание JSON файла, с записанными строками из определенной таблицы в JSON формат.

Скрипт запроса для экспорта данных представлен в листинге 4.2.

|  |
| --- |
| CREATE OR ALTER PROCEDURE SaveTableDataToJson  @tableName NVARCHAR(128)  AS  BEGIN  DECLARE @fileName NVARCHAR(128);  SET @fileName = REPLACE(@tableName, ' ', '') + '.json';  DECLARE @sql NVARCHAR(1000);  SET @sql = 'bcp "SELECT \* FROM ' + QUOTENAME(@tableName) + ' ' +  'FOR JSON PATH, INCLUDE\_NULL\_VALUES" ' +  'queryout "e:\' + @fileName + '" ' +  '-c -S DESKTOP-F644G2I -d MobileBank -T';  EXEC sys.XP\_CMDSHELL @sql;  END;  ---------------------------------------------  EXEC SaveTableDataToJson @tableName = 'client'; |

Листинг 4.1 – Процедура экспорта EXPORT \_TO\_JSON\_FILE

Таким образом, пользователи базы данных могут легко экспортировать в формате JSON, что делает управление базой данных более удобным и эффективным.

**4.3 Вывод**

В данном разделе были представлены различные способы импорта данных из файлов Excel в SQL Server или базу данных SQL Azure. Кроме того, была представлена функция EXPORT\_TO\_JSON\_FILE, которая позволяет экспортировать данные из определенной таблицы в базе данных в формате JSON. В целом, возможность импорта данных из CVS и экспорта в JSON формат значительно облегчает управление базой данных и позволяет пользователям эффективно перемещать и сохранять данные.

**5 Тестирование производительности**

Тестирование отыгрывает важную роль при разработке любого программного продукта. Чем качественнее тестирование, тем лучше в итоге должен выйти конечный продукт. Выявление проблем с производительностью до того, как они повлияют на конечного пользователя, имеет жизненно важное значение для успеха любого приложения в разработке. Пользователи будут избегать приложений, которые не работают должным образом, и большинство из них предпочитают переходить к программному обеспечению, которое было должным образом протестировано.

Нагрузочное тестирование базы данных – важный шаг, который разработчики должны предпринять, чтобы обеспечить оптимизацию производительности перед развертыванием. Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса. Разработанная процедура позволяет добавить 100000 строк за одно выполнение (листинг 5.1).

|  |
| --- |
| CREATE OR ALTER PROCEDURE PopulateClients AS  BEGIN DECLARE @i INT = 1  WHILE @i <= 100000 BEGIN  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number, is\_admin)  VALUES ('Last Name ' + CAST(@i AS NVARCHAR(50)), 'First Name ' + CAST(@i AS NVARCHAR(50)), 'Middle Name ' + CAST(@i AS NVARCHAR(50)), 'Gender ' + CAST(@i AS NVARCHAR(50)), 'Password ' + CAST(@i AS NVARCHAR(50)), 'Email ' + CAST(@i AS NVARCHAR(50)), CAST(@i AS NCHAR(13)), CAST(@i % 2 AS BIT))  SET @i = @i + 1  END  END |

Листинг 5.1 – Заполнение таблицы

Для получения выборки данных использовался запрос, который представлен на листинге 5.2.

|  |
| --- |
| SET STATISTICS TIME ON;  SELECT client\_first\_name  FROM client  WHERE client\_first\_name = 'First Name 2024';  SET STATISTICS TIME OFF; |

Листинг 5.2 – Запрос к таблице

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. В запросе мы включаем вывод статистики времени выполнения, чтобы получить информацию о времени, затраченном на анализ и компиляцию запроса, а также статистика времени выполнения, включая время CPU и общее время выполнения запроса. Результаты запроса будут представлены на рисунке 5.1.

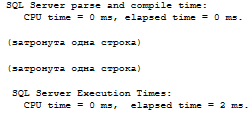


Рисунок 5.1 – Результат выполнения запроса

План выполнения – это большое дерево, на котором много объектов. Это дерево – инструкция, что нужно сделать, чтобы получить результат запроса. Операторы в плане вызываются по порядку слева направо (с левого верхнего угла), а данные в плане перемещаются справа налево. Результат будет представлен на рисунке 5.2.

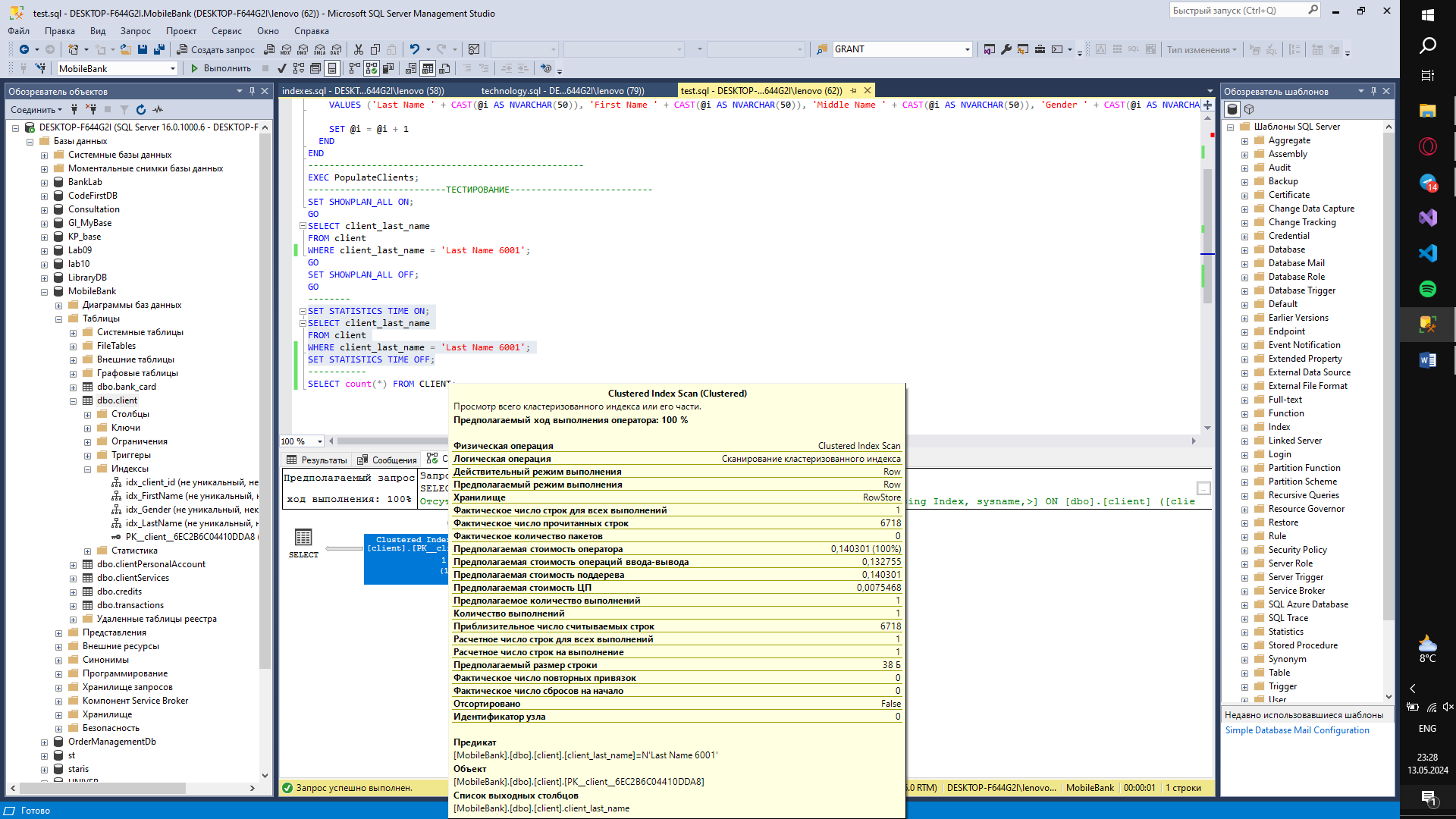


Рисунок 5.2 – Результат плана выполнения

Для сокращения времени выполнения запроса на выборку данных был добавлен индекс idx\_LastName для таблицы client.

В результате время поиска уменьшилось, это показано на рисунке 5.3.

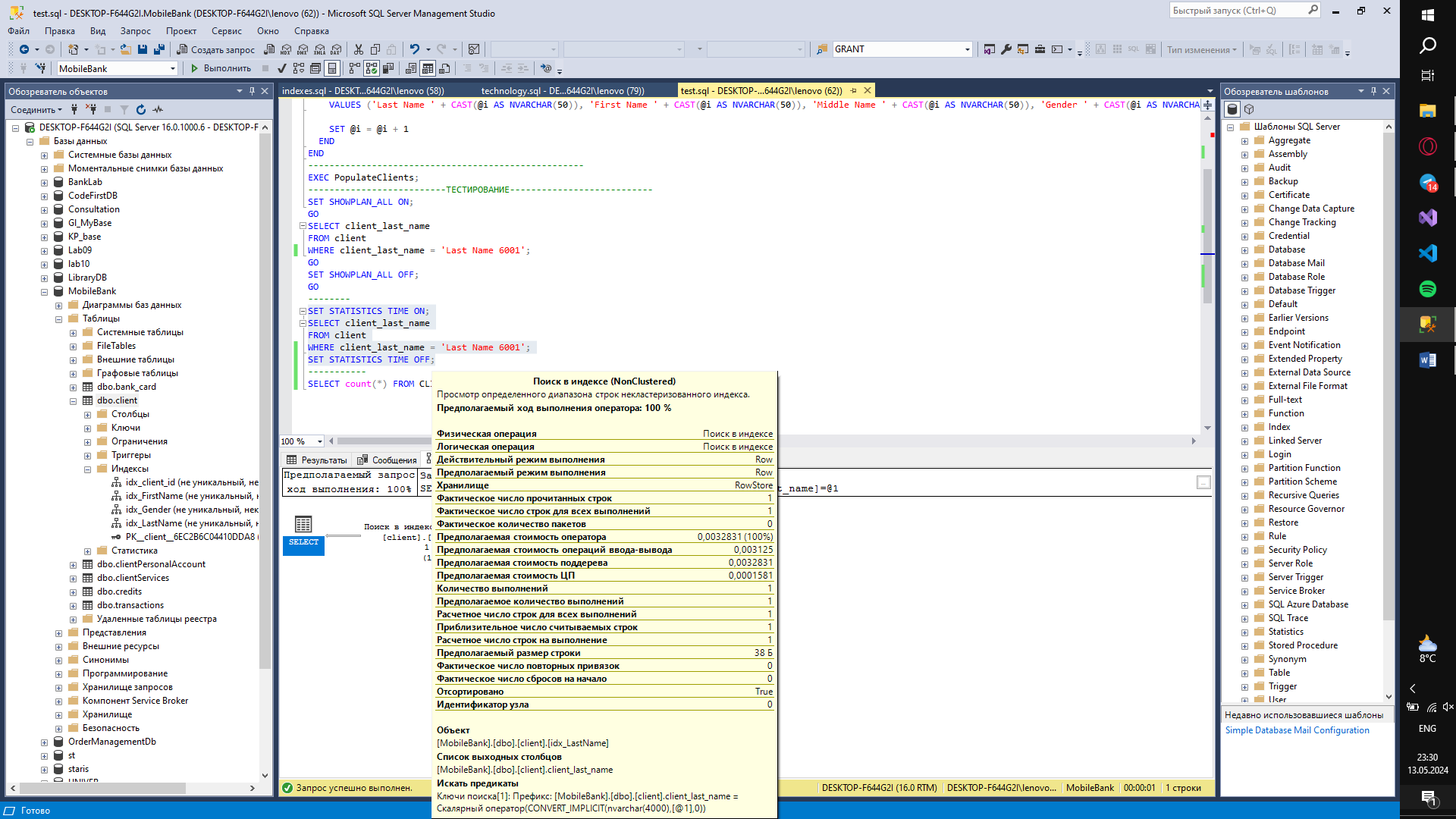


Рисунок 5.3 – План выполнения с индексом

Для других таблиц также были добавлены индексы, такие как idx\_FirstName и idx\_Gender для таблицы client, idx\_bank\_card\_number и idx\_bank\_card\_type для таблицы bank\_card, idx\_transaction\_type и idx\_transaction\_number для таблицы transactions, idx\_credit\_total\_sum и idx\_credit\_date для таблицы credits, idx\_serviceName и idx\_serviceType для таблицы clientService, idx\_personal\_account для таблицы personalAccount.

1. **Описание технологии и ее применения в базе данных**

Рассылка уведомлений - очень удобный инструмент отслеживания выполнения заданий в SQL Server. Настройка таких рассылок состоит из нескольких шагов. Для этого открываем Management Studio, заходим в «Управление» и нажимаем правой кнопкой мыши по пункту «Компонент Database Mail», затем в появившемся меню щелкаем по пункту «Настроить компонент Database Mail». Для создания учетной записи SMTP указываем e-mail, с которого будут приходить уведомления, и настройки сервера исходящей почты. После чего в списке учетных записей SMTP отобразится только что созданная учетная запись, жмем «Далее». Работу компоненты Database Mail можно проверить, отправив тестовое сообщение.

Database Mail появился в SQL Server 2005, заменив собой компонент SQLMail. Database Mail (как ранее SQLMail) используется для отправки сообщений электронной почты самим SQL Server'ом (точнее, его компонентом Database Engine). Коренным отличием Database Mail от SQLMail является то, что последний, для отправки сообщений, использовал стороний клиент (Microsoft Outlook), который должен был быть установлен на машине с SQL Server'ом, а Database Mail сам общается с почтовым сервером по протоколу SMTP.

Для настройки в командную строку PowerShell вводились следующие данные, которые представлены в листинге 6.1.

|  |
| --- |
| $EmailFrom = "ssjjitt@mail.ru"  $EmailPass = "f7T8Pw81sexdrP4yLHJ9"  $EmailTo = "ssjjitt@mail.ru"  $Port = 587  $Subject = "Test From PowerShell"  $Body = "Did this work?"  $SMTPServer = "smtp.mail.ru"  $SMTPClient = New-Object Net.Mail.SmtpClient($SmtpServer, $Port)  $SMTPClient.EnableSsl = $true  $SMTPClient.Credentials = New-Object System.Net.NetworkCredential($EmailFrom, $EmailPass);  $SMTPClient.Send($EmailFrom, $EmailTo, $Subject, $Body) |

Листинг 6.1 – код для строки PowerShell

Письма на отправляются на почту mail.ru через процедуру SaveTableDataToJsonAndSend, письмо содержит файл JSON с переданной таблицей. Листинг процедуры представлен в листинге 6.2.

|  |
| --- |
| CREATE OR ALTER PROCEDURE SaveTableDataToJsonAndSend  @tableName NVARCHAR(128),  @recipients NVARCHAR(100),  @subject NVARCHAR(255),  @body NVARCHAR(MAX)  AS  BEGIN  DECLARE @fileName NVARCHAR(128);  SET @fileName = REPLACE(@tableName, ' ', '') + '.json';  DECLARE @sql NVARCHAR(1000);  SET @sql = 'bcp "SELECT \* FROM ' + QUOTENAME(@tableName) + ' ' +  'FOR JSON PATH, INCLUDE\_NULL\_VALUES" ' +  'queryout "e:\' + @fileName + '" ' +  '-c -S DESKTOP-F644G2I -d MobileBank -T';  EXEC sys.XP\_CMDSHELL @sql;  DECLARE @attachment NVARCHAR(255);  SET @attachment = 'e:\' + @fileName;  -- Отправка письма с вложением  EXEC msdb.dbo.sp\_send\_dbmail  @profile\_name = 'database profile',  @recipients = @recipients,  @subject = @subject,  @body = @body,  @file\_attachments = @attachment;  END; |

Листинг 6.2 – код для отправки письма на почту

Процедура генерирует имя файла для сохранения данных в формате JSON. Имя файла формируется путем удаления пробелов из имени таблицы и добавления расширения json. Формирует динамический SQL запрос для получения данных из таблицы @tableName в формате JSON с использованием FOR JSON PATH, INCLUDE\_NULL\_VALUES. Результаты запроса сохраняются в файле. Вызывает команду XP\_CMDSHELL сформированным SQL запросом. Это выполняет команду bcp (Bulk Copy Program) для выполнения SQL-запроса и сохранения результатов в файле. Задает путь к сохраненному файлу как вложение для отправки письма. Использует системную процедуру sp\_send\_dbmail для отправки письма с вложением. Параметры письма включают имя профиля отправителя, список получателей, тему, текст письма и путь к вложенному файлу.

**7 Руководство пользователя**

При входе отображается форма логина, далее пользователь может или создать аккаутн, или войти с уже существующего, так же есть возможность войти с администратора, если ввести конкретный логин и пароль. Рисунок 7.1.

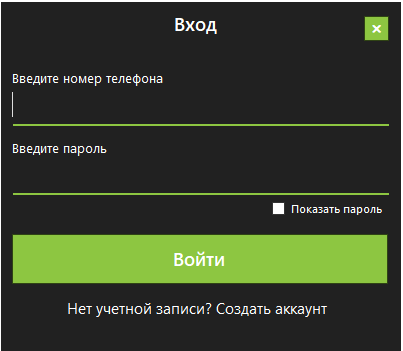


Рисунок 7.1. Вид формы входа

При нажатии “Зарегистрироваться” открывается форма, где пользователь заполнит информацию о себе. Форма представлена на рисунке 7.2.

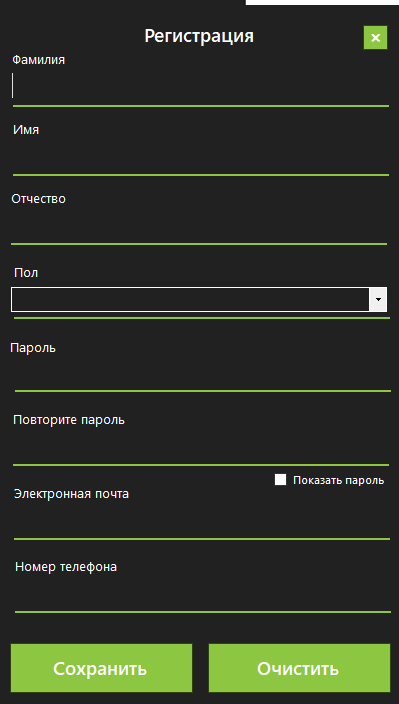


Рисунок 7.2. – Форма регистрации

После регистрации и последующей авторизации, пользователь уже имеет одноименный статус – пользователь. У него открывается новая форма с возможностями: перевод с карты на карту, коммунальные платежи, благотворительность, пополнение мобильного, интернет и ТВ, оформление кредита. У пользователя есть возможность добавить карту или использовать уже существующую. В верхнем левом углу экрана есть настройка личных данных, а также история транзакций. Вид главного окна представлен на рисунке 7.3.

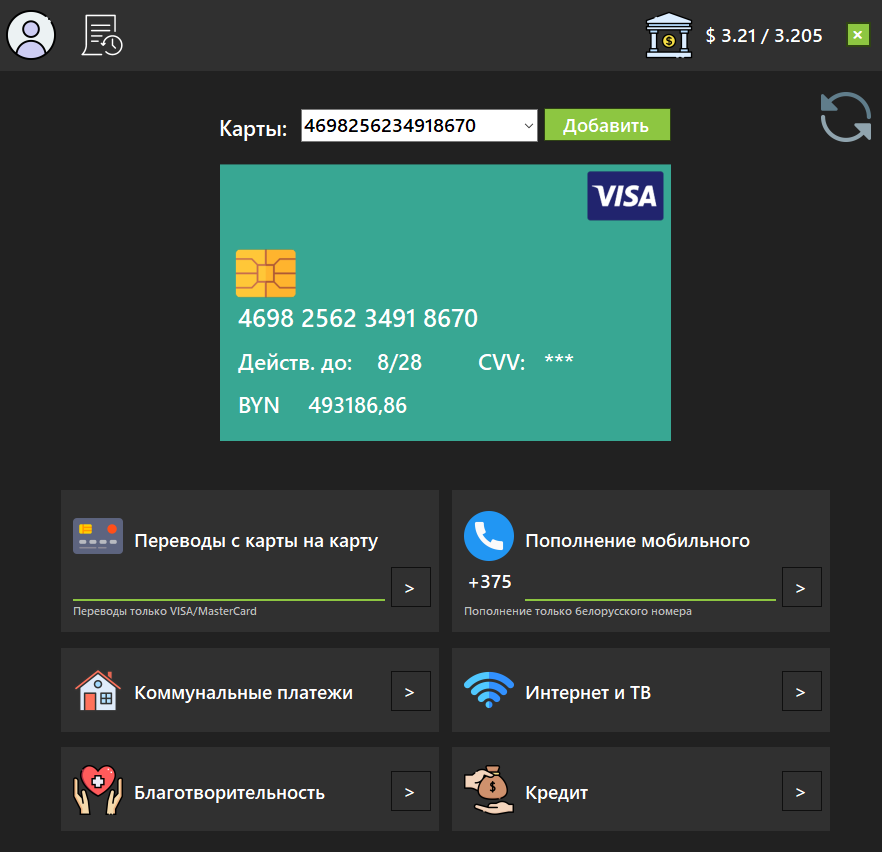


Рисунок 7.3. – Вид главного окна приложения

Для примера работы базовых банковских операций на рисунке 7.4. представлена форма перевода с карты на карту.

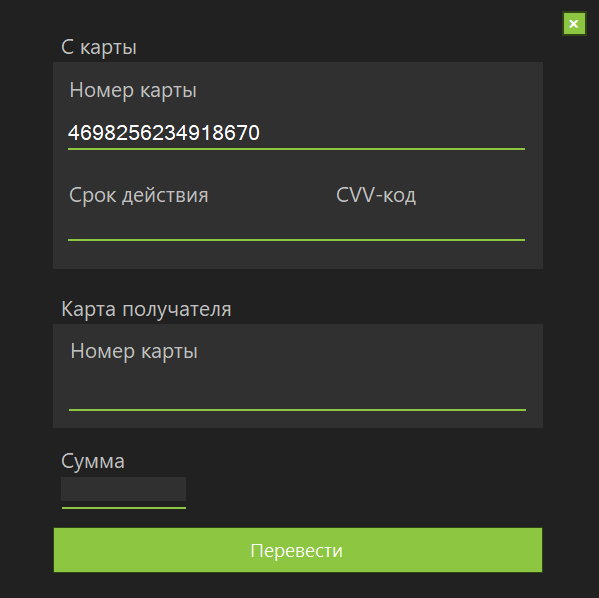


Рисунок 7.4. – Форма перевода с карты на карту

Для того, чтобы войти в приложение под администратором, необходимо ввести admin в оба поля формы входа. Вход с учетной записью администратора дает нам управление базой данных, выполняя процедуры, функции. Вид формы администратора представлен на рисунке 7.5.

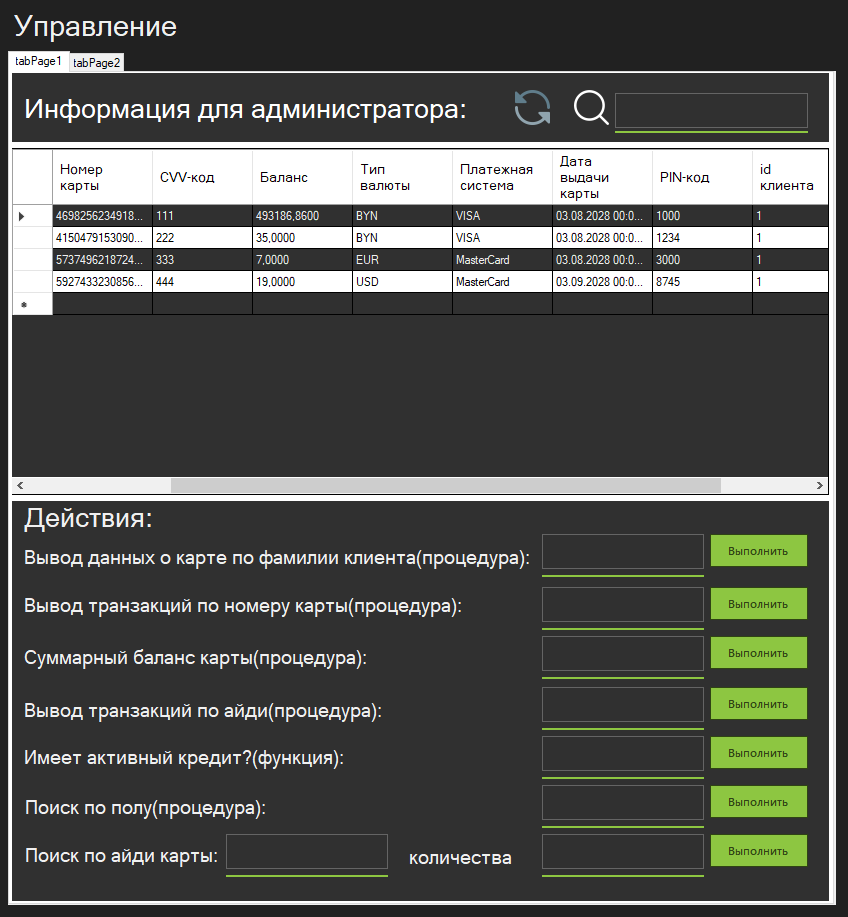


Рисунок 7.5. – Вид формы администратора

Реализация каждой из вышеупомянутой роли приведена в приложении Г.

Заключение

В результате выполнения данного курсового проекта были реализованы ключевые элементы базы данных, включая таблицы для хранения информации. Индексы были применены для оптимизации производительности запросов, а процедуры обеспечивают управление данными, соблюдая принципы безопасности.

Также были созданы пользователи базы данных с различными уровнями доступа – пользователь и администратор, что обеспечивает отделение прав доступа и повышает безопасность системы.

Разработанные процедуры обеспечивают базовые операции управления данными, а также функционал поиска, сортировки и фильтрации, что сделает взаимодействие с системой интуитивно понятным и удобным для пользователей.

Благодаря внедрению индексов, процедур, и оптимизированным структурам таблиц, система способна эффективно обрабатывать запросы, в том числе на больших объемах данных, что существенно повышает производительность системы.

В целом, разработанная база данных представляет собой мощный инструмент, сочетающий в себе гибкость, производительность и безопасность. Технология email-оповещений позволяет уведомлять о действиях в базе данных.

**Список используемых источников**

1. Таблицы в SQL Server [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/tables/tables?view=sql-server-ver16 – Дата доступа: 01.05.2024.
2. Transact-SQL [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver16 – Дата доступа: 01.05.2024.
3. Настройка DatabaseMail [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/database-mail/configure-database-mail?view=sql-server-ver16 – Дата доступа: 01.05.2024.
4. Данные JSON в SQL Server [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/json/json-data-sql-server?view=sql-server-ver16 – Дата доступа: 01.05.2024.
5. Импорт CSV в SQL Server [Электронный ресурс]. – Режим доступа: https://professorweb.ru/my/it/blog/net/csv\_sql\_server.php - Дата доступа: 01.05.2024.

**Приложение А Листинг создания таблиц**

|  |
| --- |
| create table client(  id\_client int identity(1,1) not null primary key,  client\_last\_name nvarchar(50) not null,  client\_first\_name nvarchar(50) not null,  client\_middle\_name nvarchar(50) not null,  client\_gender nvarchar(50) not null,  client\_password nvarchar(50) not null,  client\_email nvarchar(50) not null,  client\_phone\_number nchar(13) not null,  )  ------------------------------------------------------------------------------------------  create table bank\_card(  id\_bank\_card int identity(1,1) not null primary key,  bank\_card\_type nvarchar(50) not null,  bank\_card\_number nvarchar(16) not null,  bank\_card\_cvv\_code nvarchar(3) not null,  bank\_card\_balance money default 0,  bank\_card\_currency nvarchar(10) not null,  bank\_card\_payment nvarchar(50) not null,  bank\_card\_date date not null,  bank\_card\_pin int not null  )  ------------------------------------------------------------------------------------------  alter table client add is\_admin bit  alter table bank\_card add id\_client int  alter table bank\_card add paymentSystem nvarchar(50) not null  alter table bank\_card add foreign key(id\_client) references dbo.client(id\_client) on delete no action on update cascade  ------------------------------------------------------------------------------------------  create table transactions(  id\_transaction int identity(1,1) not null primary key,  transaction\_type varchar(50) not null,  transaction\_destination varchar(200) not null,  transaction\_date date not null,  transaction\_number nchar(50),  transaction\_value money  )  ------------------------------------------------------------------------------------------  alter table transactions add id\_bank\_card int  alter table transactions add foreign key (id\_bank\_card) references dbo.bank\_card(id\_bank\_card) on delete no action on update cascade  ------------------------------------------------------------------------------------------  create table credits(  id\_credit int identity(1,1) not null primary key,  credit\_total\_sum money not null,  credit\_sum money not null,  credit\_date date not null,  credit\_status bit not null default 0,  repayment\_date date,  repayment\_sum money  )  ------------------------------------------------------------------------------------------  alter table credits add id\_bank\_card int  alter table credits add foreign key (id\_bank\_card) references dbo.bank\_card(id\_bank\_card) on delete no action on update cascade  ------------------------------------------------------------------------------------------  create table clientServices(  id\_service int identity(1,1) not null primary key,  serviceName nvarchar(100) not null,  serviceBalance money default 0,  serviceType nvarchar(100) not null  )  ------------------------------------------------------------------------------------------  create table clientPersonalAccount(  id\_personal\_account int identity(1,1) not null primary key,  personal\_account nvarchar(10) not null  )  ------------------------------------------------------------------------------------------  alter table clientPersonalAccount add id\_service int  alter table clientPersonalAccount add foreign key (id\_service) references dbo.clientServices(id\_service) on delete no action on update cascade  ------------------------------------------------------------------------------------------  alter table clientPersonalAccount add id\_client int  alter table clientPersonalAccount add foreign key (id\_client) references dbo.client(id\_client) on delete no action on update cascade  ------------------------------------------------------------------------------------------ |

**Приложение Б Листинг создания функций**

|  |
| --- |
| -----------------------ПОИСК ТРАНЗАКЦИЙ ПО АЙДИ(ИСП)-------------------------  CREATE FUNCTION GetTransById(  @bankCardId nvarchar(16)  )  RETURNS TABLE  AS  RETURN  (  SELECT  t.id\_transaction,  t.transaction\_type,  t.transaction\_destination,  t.transaction\_date,  t.transaction\_number,  t.transaction\_value  FROM  transactions t  WHERE  t.id\_bank\_card = @bankCardId  )  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.GetTransById('11');  ---------------ПОИСК ТРАНЗАКЦИЙ ПО НОМЕРУ КАРТЫ(ИСП)---------------  CREATE FUNCTION GetTransByNumber  (  @bankCardNumber nvarchar(16)  )  RETURNS TABLE  AS  RETURN  (  SELECT  t.id\_transaction,  t.transaction\_type,  t.transaction\_destination,  t.transaction\_date,  t.transaction\_number,  t.transaction\_value  FROM  transactions t  WHERE  t.id\_bank\_card = (SELECT id\_bank\_card FROM bank\_card WHERE bank\_card\_number = @bankCardNumber)  );  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.GetTransByNumber('4698256234918670');  ----------------------ПОИСКА ПО ИМЕНИ(ИСП)-------------------------  CREATE FUNCTION dbo.SearchClientByLastName  (  @lastName NVARCHAR(50)  )  RETURNS TABLE  AS  RETURN  (  SELECT  c.client\_first\_name,  c.client\_last\_name,  c.client\_middle\_name,  bc.id\_bank\_card,  bc.bank\_card\_type,  bc.bank\_card\_number,  bc.bank\_card\_cvv\_code,  bc.bank\_card\_balance,  bc.bank\_card\_currency,  bc.bank\_card\_payment,  bc.bank\_card\_date,  bc.bank\_card\_pin,  bc.id\_client  FROM client AS c  INNER JOIN bank\_card AS bc ON c.id\_client = bc.id\_client  WHERE c.client\_last\_name = @lastName  );  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.SearchClientByLastName('Иванов');  -----------------СРЕДНИЙ БАЛАНС ПО ВСЕМ КАРТАМ(ИСП)+КУРСОР---------------  CREATE FUNCTION dbo.CalculateTotalBalance  (  @clientId INT  )  RETURNS DECIMAL(18, 2)  AS  BEGIN  DECLARE @totalBalance DECIMAL(18, 2)  DECLARE @balance DECIMAL(18, 2)  SET @totalBalance = 0  DECLARE @cur CURSOR  SET @cur = CURSOR FOR  SELECT bank\_card\_balance  FROM bank\_card  WHERE id\_client = @clientId  OPEN @cur  FETCH NEXT FROM @cur INTO @balance  WHILE @@FETCH\_STATUS = 0  BEGIN  SET @totalBalance = @totalBalance + @balance  FETCH NEXT FROM @cur INTO @balance  END  CLOSE @cur  DEALLOCATE @cur  RETURN @totalBalance  END  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.CalculateTotalBalance(1) as [Общий баланс на всех картах]  -----ДЛЯ ПОИСКА ПО ИМЕНИ И ЕГО ЧАСТЬ(НЕ ИСП) + ИНДЕКС------  CREATE FUNCTION dbo.SearchByLastName  (  @LastNameSearch VARCHAR(50)  )  RETURNS TABLE  AS  RETURN  (  SELECT \*  FROM client  WHERE client\_last\_name LIKE '%' + @LastNameSearch + '%'  )  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.SearchByLastName('Ива');  -------------------ПОИСК ПО ГЕНДЕРУ(И)----------------  CREATE FUNCTION dbo.GetClientsByGender  (  @Gender NVARCHAR(50)  )  RETURNS TABLE  AS  RETURN  (  SELECT \*  FROM client  WHERE client\_gender = @Gender  )  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.GetClientsByGender('мужской');  ----ПОЛУЧЕНИЕ ПОСЛЕДНИХ N ТРАНЗАКЦИЙ ДЛЯ КАРТЫ(ИСП):--------------  CREATE FUNCTION dbo.GetRecentTransactions  (  @BankCardID INT,  @NumTransactions INT  )  RETURNS TABLE  AS  RETURN  (  SELECT TOP (@NumTransactions) \*  FROM transactions  WHERE id\_bank\_card = @BankCardID  ORDER BY transaction\_date DESC  )  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.GetRecentTransactions(10, 5);  ---------КЛИЕНТЫ К КОТОРЫМ ПРИДУТ КОЛЛЕКТОРЫ(ИСП):----------------  CREATE FUNCTION dbo.GetClientsWithCreditDebt()  RETURNS TABLE  AS  RETURN  (  SELECT c.\*  FROM client c  INNER JOIN bank\_card b ON c.id\_client = b.id\_client  INNER JOIN credits cr ON b.id\_bank\_card = cr.id\_bank\_card  WHERE cr.credit\_status = 1 AND cr.repayment\_sum > 0  )  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.GetClientsWithCreditDebt();  ---------ПОСЛЕДНЯЯ ТРАНЗАКЦИЯ ДЛЯ КАЖДОЙ КАРТЫ(ИСП):----------------  CREATE FUNCTION dbo.GetLastTransactionForEachCard (@ClientID int)  RETURNS TABLE  AS  RETURN  (  SELECT \*  FROM  (  SELECT t.\*, ROW\_NUMBER() OVER (PARTITION BY t.id\_bank\_card ORDER BY t.transaction\_date DESC) AS RowNum  FROM transactions t  WHERE t.id\_bank\_card IN (  SELECT b.id\_bank\_card  FROM bank\_card b  WHERE b.id\_client = @ClientID  )  ) AS Subquery  WHERE RowNum = 1  )  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT \* FROM dbo.GetLastTransactionForEachCard(1);  -------------ФУНКЦИЯ ДЛЯ ПРОВЕРКИ АКТИВНОГО КРЕДИТА(ИСП)-----------  CREATE FUNCTION dbo.HasActiveCredit(@clientId INT)  RETURNS BIT  AS  BEGIN  DECLARE @hasActiveCredit BIT;  IF EXISTS (  SELECT 1  FROM credits  WHERE id\_bank\_card IN (  SELECT id\_bank\_card  FROM bank\_card  WHERE id\_client = @clientId  ) AND credit\_status = 1  )  BEGIN  SET @hasActiveCredit = 1;  END  ELSE  BEGIN  SET @hasActiveCredit = 0;  END  RETURN @hasActiveCredit;  END  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.HasActiveCredit(1) AS HasActiveCredit;  ---ФУНКЦИЯ ПОИСКА ПО НЕСКОЛЬКИМ ПАРАМЕТРАМ(ИМЯ,ФАМИЛИЯ,КАРТА)(ИСП)----  CREATE FUNCTION SearchClients  (  @searchString NVARCHAR(100)  )  RETURNS TABLE  AS  RETURN  (  SELECT \*  FROM client  WHERE client\_first\_name LIKE '%' + @searchString + '%'  OR client\_last\_name LIKE '%' + @searchString + '%'  OR EXISTS (  SELECT 1  FROM bank\_card  WHERE bank\_card.id\_client = client.id\_client  AND (  bank\_card.bank\_card\_number LIKE '%' + @searchString + '%'  OR CAST(client.id\_client AS NVARCHAR(50)) = @searchString  )  )  );  SELECT \* FROM dbo.SearchClients('Ива');  SELECT \* FROM dbo.SearchClients(46);  ---ФУНКЦИЯ ДЛЯ ПОЛУЧЕНИЯ ОБЩЕЙ СУММЫ ТРАНЗАКЦИЙ ДЛЯ ОПРЕДЕЛЕННОЙ БАНКОВСКОЙ КАРТЫ(ИСП)----  CREATE FUNCTION dbo.GetTransactionTotalAmount(@id\_bank\_card INT)  RETURNS MONEY  AS  BEGIN  DECLARE @TotalAmount MONEY;  SELECT @TotalAmount = SUM(transaction\_value) FROM transactions WHERE id\_bank\_card = @id\_bank\_card;  RETURN @TotalAmount;  END;  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.GetTransactionTotalAmount(11);  ---------------ФУНКЦИЯ ПОДСЧЕТА ОБЩЕГО КОЛИЧЕСТВА КЛИЕНТОВ-------------  CREATE FUNCTION dbo.GetClientCount()  RETURNS INT  AS  BEGIN  DECLARE @Count INT;  SELECT @Count = COUNT(\*) FROM client;  RETURN @Count;  END;  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.GetClientCount();  ---ФУНКЦИЯ ДЛЯ ПОДСЧЕТА ОБЩЕГО КОЛИЧЕСТВА БАНКОВСКИХ КАРТ----  CREATE FUNCTION dbo.GetBankCardCount()  RETURNS INT  AS  BEGIN  DECLARE @Count INT;  SELECT @Count = COUNT(\*) FROM bank\_card;  RETURN @Count;  END;  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.GetBankCardCount();  ---ФУНКЦИЯ ДЛЯ ПОДСЧЕТА ОБЩЕГО КОЛИЧЕСТВА ТРАНЗАКЦИЙ----  CREATE FUNCTION dbo.GetTransactionCount()  RETURNS INT  AS  BEGIN  DECLARE @Count INT;  SELECT @Count = COUNT(\*) FROM transactions;  RETURN @Count;  END;  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.GetTransactionCount();  ---ФУНКЦИЯ ДЛЯ ПОДСЧЕТА ОБЩЕГО КОЛИЧЕСТВА КРЕДИТОВ----  CREATE FUNCTION dbo.GetCreditCount()  RETURNS INT  AS  BEGIN  DECLARE @Count INT;  SELECT @Count = COUNT(\*) FROM credits;  RETURN @Count;  END;  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.GetCreditCount();  ---ФУНКЦИЯ ДЛЯ ПОДСЧЕТА ОБЩЕГО КОЛИЧЕСТВА АКТИВНЫХ КЛИЕНТСКИХ УСЛУГ----  CREATE FUNCTION dbo.GetActiveServiceCount()  RETURNS INT  AS  BEGIN  DECLARE @Count INT;  SELECT @Count = COUNT(\*) FROM clientServices WHERE serviceBalance > 0;  RETURN @Count;  END;  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  SELECT dbo.GetActiveServiceCount();  ---ФУНКЦИЯ ДЛЯ ПОЛУЧЕНИЯ СПИСКА КЛИЕНТОВ, У КОТОРЫХ НЕТ АКТИВНЫХ БАНКОВСКИХ КАРТ----  CREATE FUNCTION dbo.GetClientsWithoutActiveBankCards()  RETURNS TABLE  AS  RETURN  SELECT \* FROM client WHERE id\_client NOT IN (SELECT DISTINCT id\_client FROM bank\_card);  -------------------------------------------------------------  SELECT \* from GetClientsWithoutActiveBankCards(); |

**Приложение В Листинг создания процедур**

|  |
| --- |
| -------------------------------------------------------------  -------------ДОБАВЛЕНИЕ КЛИЕНТА--------------------------------  CREATE OR ALTER PROCEDURE dbo.CreateClient  @client\_last\_name nvarchar(50),  @client\_first\_name nvarchar(50),  @client\_middle\_name nvarchar(50),  @client\_gender nvarchar(50),  @client\_password nvarchar(50),  @client\_email nvarchar(50),  @client\_phone\_number nchar(13),  @is\_admin bit  AS  BEGIN  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number, is\_admin)  VALUES (@client\_last\_name, @client\_first\_name, @client\_middle\_name, @client\_gender, @client\_password, @client\_email, @client\_phone\_number, @is\_admin)  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.CreateClient  @client\_last\_name = 'Фамилия',  @client\_first\_name = 'Имя',  @client\_middle\_name = 'Отчество',  @client\_gender = 'Пол',  @client\_password = 'Пароль',  @client\_email = 'email@example.com',  @client\_phone\_number = '1234567890123',  @is\_admin = 0;  select \* from client  -------------------------------------------------------------  ------------ОБНОВЛЕНИЕ ИНФОРМАЦИИ О КЛИЕНТЕ--------------------  CREATE OR ALTER PROCEDURE dbo.UpdateClient  @id\_client int,  @client\_last\_name nvarchar(50),  @client\_first\_name nvarchar(50),  @client\_middle\_name nvarchar(50),  @client\_gender nvarchar(50),  @client\_password nvarchar(50),  @client\_email nvarchar(50),  @client\_phone\_number nchar(13),  @is\_admin bit  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM client WHERE id\_client = @id\_client)  BEGIN  RAISERROR ('Клиента с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  BEGIN TRANSACTION  UPDATE client  SET client\_last\_name = @client\_last\_name,  client\_first\_name = @client\_first\_name,  client\_middle\_name = @client\_middle\_name,  client\_gender = @client\_gender,  client\_password = @client\_password,  client\_email = @client\_email,  client\_phone\_number = @client\_phone\_number,  is\_admin = @is\_admin  WHERE id\_client = @id\_client  COMMIT TRANSACTION  END TRY  BEGIN CATCH  IF @@TRANCOUNT > 0  ROLLBACK TRANSACTION  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.UpdateClient  @id\_client = 2209,  @client\_last\_name = 'Фамилия',  @client\_first\_name = 'Имя',  @client\_middle\_name = 'Отчество',  @client\_gender = 'Пол',  @client\_password = 'Пароль',  @client\_email = 'email@example.com',  @client\_phone\_number = '1234567890123',  @is\_admin = 0;  select \* from client  -------------------------------------------------------------  ------------УДАЛЕНИЕ ИНФОРМАЦИИ О КЛИЕНТЕ--------------------  CREATE OR ALTER PROCEDURE dbo.DeleteClient  @id\_client int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM client WHERE id\_client = @id\_client)  BEGIN  RAISERROR ('Клиента с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  BEGIN TRANSACTION  DELETE FROM client  WHERE id\_client = @id\_client  COMMIT TRANSACTION  END TRY  BEGIN CATCH  IF @@TRANCOUNT > 0  ROLLBACK TRANSACTION  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.DeleteClient @id\_client = 2209;  -------------------------------------------------------------  --------------СОЗДАНИЕ БАНКОВСКОЙ КАРТЫ----------------------  select \* from bank\_card;  CREATE OR ALTER PROCEDURE dbo.CreateBankCard  @bank\_card\_type nvarchar(50),  @bank\_card\_number nvarchar(16),  @bank\_card\_cvv\_code nvarchar(3),  @bank\_card\_balance money,  @bank\_card\_currency nvarchar(10),  @bank\_card\_payment nvarchar(50),  @bank\_card\_date date,  @bank\_card\_pin int,  @id\_client int,  @paymentSystem nvarchar(50)  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM dbo.Client WHERE id\_client = @id\_client)  BEGIN  RAISERROR('Клиента с указанным идентификатором не существует.', 16, 1)  RETURN  END  BEGIN TRY  INSERT INTO bank\_card (bank\_card\_type, bank\_card\_number, bank\_card\_cvv\_code, bank\_card\_balance, bank\_card\_currency, bank\_card\_payment, bank\_card\_date, bank\_card\_pin, id\_client, paymentSystem)  VALUES (@bank\_card\_type, @bank\_card\_number, @bank\_card\_cvv\_code, @bank\_card\_balance, @bank\_card\_currency, @bank\_card\_payment, @bank\_card\_date, @bank\_card\_pin, @id\_client, @paymentSystem)  END TRY  BEGIN CATCH  IF ERROR\_NUMBER() = 547  BEGIN  RAISERROR('Нарушение ограничения внешнего ключа. Проверьте правильность идентификатора клиента.', 16, 1)  END  ELSE  BEGIN  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.CreateBankCard  @bank\_card\_type = 'дебетовая',  @bank\_card\_number = '1234567891234567',  @bank\_card\_cvv\_code = '132',  @bank\_card\_balance = 1000,  @bank\_card\_currency = 'BYN',  @bank\_card\_payment = 'VISA',  @bank\_card\_date = '2024-05-09',  @bank\_card\_pin = 1234,  @id\_client = 2220,  @paymentSystem = 'VISA'  -------------------------------------------------------------  --------------ОБНОВЛЕНИЕ БАНКОВСКОЙ КАРТЫ----------------------  CREATE OR ALTER PROCEDURE dbo.UpdateBankCard  @id\_bank\_card int,  @bank\_card\_type nvarchar(50),  @bank\_card\_number nvarchar(16),  @bank\_card\_cvv\_code nvarchar(3),  @bank\_card\_balance money,  @bank\_card\_currency nvarchar(10),  @bank\_card\_payment nvarchar(50),  @bank\_card\_date date,  @bank\_card\_pin int,  @id\_client int,  @paymentSystem nvarchar(50)  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM bank\_card WHERE id\_bank\_card = @id\_bank\_card)  BEGIN  RAISERROR('Карты с указанным идентификатором не существует.', 16, 1)  RETURN  END  BEGIN TRY  UPDATE bank\_card  SET bank\_card\_type = @bank\_card\_type,  bank\_card\_number = @bank\_card\_number,  bank\_card\_cvv\_code = @bank\_card\_cvv\_code,  bank\_card\_balance = @bank\_card\_balance,  bank\_card\_currency = @bank\_card\_currency,  bank\_card\_payment = @bank\_card\_payment,  bank\_card\_date = @bank\_card\_date,  bank\_card\_pin = @bank\_card\_pin,  id\_client = @id\_client,  paymentSystem = @paymentSystem  WHERE id\_bank\_card = @id\_bank\_card  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.UpdateBankCard  @id\_bank\_card = 22,  @bank\_card\_type = 'кредитная',  @bank\_card\_number = '8765432193547809',  @bank\_card\_cvv\_code = '123',  @bank\_card\_balance = 1000,  @bank\_card\_currency = 'BYN',  @bank\_card\_payment = 'MasterCard',  @bank\_card\_date = '2024-05-09',  @bank\_card\_pin = 1234,  @id\_client = 2220,  @paymentSystem = 'MasterCard'  -------------------------------------------------------------  --------------УДАЛЕНИЕ БАНКОВСКОЙ КАРТЫ----------------------  CREATE OR ALTER PROCEDURE dbo.DeleteBankCard  @id\_bank\_card int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM bank\_card WHERE id\_bank\_card = @id\_bank\_card)  BEGIN  RAISERROR('Карты с указанным идентификатором не существует.', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM bank\_card WHERE id\_bank\_card = @id\_bank\_card  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.DeleteBankCard @id\_bank\_card = 22;  -------------------------------------------------------------  ------------СОЗДАНИЕ ТРАНЗАКЦИИ------------  CREATE OR ALTER PROCEDURE dbo.CreateTransaction  @transaction\_type varchar(50),  @transaction\_destination varchar(200),  @transaction\_date date,  @transaction\_number nchar(50),  @transaction\_value money,  @id\_bank\_card int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM bank\_card WHERE id\_bank\_card = @id\_bank\_card)  BEGIN  RAISERROR('Карты с указанным идентификатором не существует.', 16, 1)  RETURN  END  BEGIN TRY  INSERT INTO transactions (transaction\_type, transaction\_destination, transaction\_date, transaction\_number, transaction\_value, id\_bank\_card)  VALUES (@transaction\_type, @transaction\_destination, @transaction\_date, @transaction\_number, @transaction\_value, @id\_bank\_card)  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.CreateTransaction  @transaction\_type = 'Payment',  @transaction\_destination = 'Online Store',  @transaction\_date = '2024-05-09',  @transaction\_number = '1234567890',  @transaction\_value = 100.00,  @id\_bank\_card = 23;  -------------------------------------------------------------  ------------ОБНОВЛЕНИЕ ИНФОРМАЦИИ О ТРАНЗАКЦИИ------------  CREATE OR ALTER PROCEDURE dbo.UpdateTransaction  @id\_transaction int,  @transaction\_type varchar(50),  @transaction\_destination varchar(200),  @transaction\_date date,  @transaction\_number nchar(50),  @transaction\_value money,  @id\_bank\_card int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM transactions WHERE id\_transaction = @id\_transaction)  BEGIN  RAISERROR ('Транзакция с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  UPDATE transactions  SET transaction\_type = @transaction\_type,  transaction\_destination = @transaction\_destination,  transaction\_date = @transaction\_date,  transaction\_number = @transaction\_number,  transaction\_value = @transaction\_value,  id\_bank\_card = @id\_bank\_card  WHERE id\_transaction = @id\_transaction  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.UpdateTransaction  @id\_transaction = 1,  @transaction\_type = 'New Payment',  @transaction\_destination = 'Updated Online Store',  @transaction\_date = '2024-05-09',  @transaction\_number = '0987654321',  @transaction\_value = 200.00,  @id\_bank\_card = 45;  -------------------------------------------------------------  ------------УДАЛЕНИЕ ТРАНЗАКЦИИ------------  CREATE OR ALTER PROCEDURE dbo.DeleteTransaction  @id\_transaction int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM transactions WHERE id\_transaction = @id\_transaction)  BEGIN  RAISERROR ('Транзакция с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM transactions WHERE id\_transaction = @id\_transaction  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  exec dbo.DeleteTransaction @id\_transaction = 1;  ------------СОЗДАНИЕ КРЕДИТА------------  CREATE OR ALTER PROCEDURE dbo.CreateCredit  @credit\_total\_sum money,  @credit\_sum money,  @credit\_date date,  @credit\_status bit = 0,  @repayment\_date date = NULL,  @repayment\_sum money = NULL,  @id\_bank\_card int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM dbo.bank\_card WHERE id\_bank\_card = @id\_bank\_card)  BEGIN  RAISERROR ('Банковской карты с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  INSERT INTO credits (credit\_total\_sum, credit\_sum, credit\_date, credit\_status, repayment\_date, repayment\_sum, id\_bank\_card)  VALUES (@credit\_total\_sum, @credit\_sum, @credit\_date, @credit\_status, @repayment\_date, @repayment\_sum, @id\_bank\_card)  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.CreateCredit  @credit\_total\_sum = 10000,  @credit\_sum = 5000,  @credit\_date = '2024-05-09',  @credit\_status = 1,  @repayment\_date = '2024-06-09',  @repayment\_sum = 2500,  @id\_bank\_card = 123456789;  ------------ОБНОВЛЕНИЕ КРЕДИТА------------  CREATE OR ALTER PROCEDURE dbo.UpdateCredit  @id\_credit int,  @credit\_total\_sum money,  @credit\_sum money,  @credit\_date date,  @credit\_status bit,  @repayment\_date date,  @repayment\_sum money,  @id\_bank\_card int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM credits WHERE id\_credit = @id\_credit)  BEGIN  RAISERROR ('Кредит с таким ID не существует.', 16, 1)  RETURN  END  IF NOT EXISTS (SELECT 1 FROM dbo.bank\_card WHERE id\_bank\_card = @id\_bank\_card)  BEGIN  RAISERROR ('Банковской карты с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  UPDATE credits  SET credit\_total\_sum = @credit\_total\_sum,  credit\_sum = @credit\_sum,  credit\_date = @credit\_date,  credit\_status = @credit\_status,  repayment\_date = @repayment\_date,  repayment\_sum = @repayment\_sum,  id\_bank\_card = @id\_bank\_card  WHERE id\_credit = @id\_credit  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.UpdateCredit  @id\_credit = 1,  @credit\_total\_sum = 15000,  @credit\_sum = 8000,  @credit\_date = '2024-05-09',  @credit\_status = 1,  @repayment\_date = '2024-07-09',  @repayment\_sum = 4000,  @id\_bank\_card = 987654321;  ------------УДАЛЕНИЕ КРЕДИТА------------  CREATE OR ALTER PROCEDURE dbo.DeleteCredit  @id\_credit int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM credits WHERE id\_credit = @id\_credit)  BEGIN  RAISERROR ('Кредита с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM credits  WHERE id\_credit = @id\_credit  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.DeleteCredit @id\_credit = 1;  ------------СОЗДАНИЕ КЛИЕНТ-СЕРВИСА------------  CREATE OR ALTER PROCEDURE dbo.CreateClientService  @serviceName nvarchar(100),  @serviceBalance money = 0,  @serviceType nvarchar(100)  AS  BEGIN  BEGIN TRY  INSERT INTO clientServices (serviceName, serviceBalance, serviceType)  VALUES (@serviceName, @serviceBalance, @serviceType)  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.CreateClientService @serviceName = 'Интернет', @serviceBalance = 0, @serviceType = 'Услуга связи'  ------------ОБНОВЛЕНИЕ КЛИЕНТ-СЕРВИСА------------  CREATE OR ALTER PROCEDURE dbo.UpdateClientService  @id\_service int,  @serviceName nvarchar(100),  @serviceBalance money,  @serviceType nvarchar(100)  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM clientServices WHERE id\_service = @id\_service)  BEGIN  RAISERROR ('Клиент-сервис с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  UPDATE clientServices  SET serviceName = @serviceName,  serviceBalance = @serviceBalance,  serviceType = @serviceType  WHERE id\_service = @id\_service  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.UpdateClientService @id\_service = 58, @serviceName = 'Телефония', @serviceBalance = 100, @serviceType = 'Услуга связи'  ------------УДАЛЕНИЕ КЛИЕНТ-СЕРВИСА------------  CREATE OR ALTER PROCEDURE dbo.DeleteClientService  @id\_service int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM clientServices WHERE id\_service = @id\_service)  BEGIN  RAISERROR ('Клиент-сервис с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM clientServices  WHERE id\_service = @id\_service  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.DeleteClientService @id\_service = 58;  ------------СОЗДАНИЕ ПЕРСОНАЛЬНОГО АККАУНТА------------  CREATE OR ALTER PROCEDURE dbo.CreateClientPersonalAccount  @personal\_account nvarchar(10),  @id\_service int,  @id\_client int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM dbo.clientServices WHERE id\_service = @id\_service)  BEGIN  RAISERROR ('Клиент-сервис с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  INSERT INTO clientPersonalAccount (personal\_account, id\_service, id\_client)  VALUES (@personal\_account, @id\_service, @id\_client)  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.CreateClientPersonalAccount @personal\_account = '1234567890', @id\_service = 1, @id\_client = 2209;  ------------ОБНОВЛЕНИЕ ПЕРСОНАЛЬНОГО АККАУНТА------------  CREATE OR ALTER PROCEDURE dbo.UpdateClientPersonalAccount  @id\_personal\_account int,  @personal\_account nvarchar(10),  @id\_service int,  @id\_client int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM clientPersonalAccount WHERE id\_personal\_account = @id\_personal\_account)  BEGIN  RAISERROR ('Аккаунта с таким ID не существует.', 16, 1)  RETURN  END  IF NOT EXISTS (SELECT 1 FROM dbo.clientServices WHERE id\_service = @id\_service)  BEGIN  RAISERROR ('Клиент-сервис с таким ID не существует.', 16, 1)  RETURN  END  IF NOT EXISTS (SELECT 1 FROM dbo.client WHERE id\_client = @id\_client)  BEGIN  RAISERROR ('Клиент с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  UPDATE clientPersonalAccount  SET personal\_account = @personal\_account,  id\_service = @id\_service,  id\_client = @id\_client  WHERE id\_personal\_account = @id\_personal\_account  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.UpdateClientPersonalAccount @id\_personal\_account = 1, @personal\_account = '9876543210', @id\_service = 2, @id\_client = 2;  ------------УДАЛЕНИЕ ПЕРСОНАЛЬНОГО АККАУНТА------------  CREATE OR ALTER PROCEDURE dbo.DeleteClientPersonalAccount  @id\_personal\_account int  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM clientPersonalAccount WHERE id\_personal\_account = @id\_personal\_account)  BEGIN  RAISERROR ('Аккаунта с таким ID не существует.', 16, 1)  RETURN  END  BEGIN TRY  DELETE FROM clientPersonalAccount  WHERE id\_personal\_account = @id\_personal\_account  END TRY  BEGIN CATCH  DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE()  DECLARE @ErrorSeverity INT = ERROR\_SEVERITY()  DECLARE @ErrorState INT = ERROR\_STATE()  RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState)  END CATCH  END  ------------РАБОТОСПОСОБНОСТЬ--------------  EXEC dbo.DeleteClientPersonalAccount @id\_personal\_account = 1 |

**Приложение Г Листинг создания пользователей и ролей**

|  |
| --- |
| CREATE LOGIN admin\_bank\_login WITH PASSWORD = 'admin\_bank';  CREATE LOGIN user\_bank\_login WITH PASSWORD = 'user\_bank';  CREATE USER admin\_bank FOR LOGIN admin\_bank\_login;  CREATE USER user\_bank FOR LOGIN user\_bank\_login;  CREATE ROLE administrator\_role;  CREATE ROLE user\_role;  ALTER ROLE administrator\_role ADD MEMBER admin\_bank;  ALTER ROLE user\_role ADD MEMBER user\_bank;  ------------------------РАБОТОСПОСОБНОСТЬ----------------------  EXECUTE AS USER = 'admin\_bank';  SELECT \* FROM dbo.GetTransById('11');  REVERT;  EXECUTE AS USER = 'user\_bank';  SELECT \* FROM dbo.GetTransById('11');  select \* from client;  REVERT;  ------------------------ГРАНТЫ----------------------  GRANT EXECUTE ON dbo.SaveTableDataToJson TO administrator\_role;  GRANT EXECUTE ON dbo.CreateCredit TO administrator\_role;  GRANT EXECUTE ON dbo.CreateTransaction TO administrator\_role;  GRANT EXECUTE ON dbo.DeleteBankCard TO administrator\_role;  GRANT EXECUTE ON dbo.DeleteClient TO administrator\_role;  GRANT EXECUTE ON dbo.DeleteClientPersonalAccount TO administrator\_role;  GRANT EXECUTE ON dbo.DeleteClientService TO administrator\_role;  GRANT EXECUTE ON dbo.DeleteCredit TO administrator\_role;  GRANT EXECUTE ON dbo.DeleteTransaction TO administrator\_role;  GRANT EXECUTE ON dbo.PopulateClients TO administrator\_role;  GRANT EXECUTE ON dbo.SaveBankCardDataToJson TO administrator\_role;  GRANT EXECUTE ON dbo.SaveClientDataToJson TO administrator\_role;  GRANT EXECUTE ON dbo.SaveClientPersonalAccountDataToJson TO administrator\_role;  GRANT EXECUTE ON dbo.CreateBankCard TO administrator\_role;  GRANT EXECUTE ON dbo.CreateClient TO administrator\_role;  GRANT EXECUTE ON dbo.CreateClientPersonalAccount TO administrator\_role;  GRANT EXECUTE ON dbo.CreateClientService TO administrator\_role;  GRANT EXECUTE ON dbo.UpdateBankCard TO administrator\_role;  GRANT EXECUTE ON dbo.UpdateClient TO administrator\_role;  GRANT EXECUTE ON dbo.UpdateClientPersonalAccount TO administrator\_role;  GRANT EXECUTE ON dbo.UpdateClientService TO administrator\_role;  GRANT EXECUTE ON dbo.UpdateCredit TO administrator\_role;  GRANT EXECUTE ON dbo.UpdateTransaction TO administrator\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.client TO administrator\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.bank\_card TO administrator\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.credits TO administrator\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.transactions TO administrator\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.clientServices TO administrator\_role;  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.clientPersonalAccount TO administrator\_role;  GRANT SELECT ON dbo.GetTransById TO administrator\_role;  GRANT SELECT ON dbo.GetTransByNumber TO administrator\_role;  GRANT SELECT ON dbo.SearchClientByLastName TO administrator\_role;  GRANT EXECUTE ON dbo.CalculateTotalBalance TO administrator\_role;  GRANT SELECT ON dbo.SearchByLastName TO administrator\_role;  GRANT SELECT ON dbo.GetClientsByGender TO administrator\_role;  GRANT SELECT ON dbo.GetRecentTransactions TO administrator\_role;  GRANT SELECT ON dbo.GetClientsWithCreditDebt TO administrator\_role;  GRANT SELECT ON dbo.GetLastTransactionForEachCard TO administrator\_role;  GRANT EXECUTE ON dbo.HasActiveCredit TO administrator\_role;  GRANT SELECT ON dbo.SearchClients TO administrator\_role;  GRANT EXECUTE ON dbo.GetTransactionTotalAmount TO administrator\_role;  GRANT EXECUTE ON dbo.GetClientCount TO administrator\_role;  GRANT EXECUTE ON dbo.GetBankCardCount TO administrator\_role;  GRANT EXECUTE ON dbo.GetTransactionCount TO administrator\_role;  GRANT EXECUTE ON dbo.GetCreditCount TO administrator\_role;  GRANT EXECUTE ON dbo.GetActiveServiceCount TO administrator\_role;  GRANT SELECT ON dbo.GetClientsWithoutActiveBankCards TO administrator\_role;  GRANT SELECT ON dbo.GetTransById TO administrator\_role;  GRANT SELECT ON dbo.client TO user\_role;  GRANT SELECT ON dbo.bank\_card TO user\_role;  GRANT SELECT ON dbo.credits TO user\_role;  GRANT SELECT ON dbo.transactions TO user\_role;  GRANT SELECT ON dbo.clientServices TO user\_role;  GRANT SELECT ON dbo.clientPersonalAccount TO user\_role; |

**Приложение Д Листинг создания триггеров**

|  |
| --- |
| ---------------ЗАПРЕТ НА ВСТАВКУ С ADMIN---------------------  CREATE TRIGGER i\_tgLastName ON dbo.client  FOR INSERT  AS  DECLARE @Name varchar(50)  SELECT @Name=client\_last\_name  FROM inserted  IF @Name='admin'  BEGIN  PRINT 'ОШИБКА ПРИ ПОПЫТКЕ ВСТАВИТЬ ADMIN. ВЫ НЕ ИМЕЕТЕ ДОСТАТОЧНЫХ ПРАВ.'  ROLLBACK TRANSACTION  END;  -------------------------------------------------------------  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number, is\_admin)  VALUES('admin', 'A', 'A', 'A', 'A', 'A', 'A', 0);  ---------------ЗАПРЕТ НА IS\_ADMIN = 1---------------------  CREATE TRIGGER SetClientIsAdminFlag  ON client  AFTER INSERT  AS  BEGIN  UPDATE client  SET is\_admin = 0  FROM inserted  WHERE client.id\_client = inserted.id\_client;  END;  -------------------------------------------------------------  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number, is\_admin)  VALUES('B', 'A', 'A', 'A', 'A', 'A', 'A', 1);  ----БУДЕТ АВТОМАТИЧЕСКИ ПРОВЕРЯТЬ БАЛАНС БАНКОВСКОЙ КАРТЫ ПЕРЕД ВСТАВКОЙ НОВОЙ ТРАНЗАКЦИИ И ОТКЛОНЯТЬ ТРАНЗАКЦИЮ, ЕСЛИ БАЛАНС НЕДОСТАТОЧЕН-----  CREATE OR ALTER TRIGGER CheckTransactionBalance  ON transactions  INSTEAD OF INSERT  AS  BEGIN  DECLARE @ins TABLE (  id\_transaction int,  transaction\_type varchar(50),  transaction\_destination varchar(200),  transaction\_date date,  transaction\_number nchar(50),  transaction\_value money,  id\_bank\_card int  );  INSERT INTO @ins  SELECT id\_transaction, transaction\_type, transaction\_destination, transaction\_date, transaction\_number, transaction\_value, id\_bank\_card  FROM inserted;  IF EXISTS (  SELECT 1  FROM @ins i  INNER JOIN bank\_card bc ON i.id\_bank\_card = bc.id\_bank\_card  WHERE i.transaction\_value > bc.bank\_card\_balance  )  BEGIN  RAISERROR('Недостаточно средств для выполнения операции.', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END;  INSERT INTO transactions (id\_transaction, transaction\_type, transaction\_destination, transaction\_date, transaction\_number, transaction\_value, id\_bank\_card)  SELECT id\_transaction, transaction\_type, transaction\_destination, transaction\_date, transaction\_number, transaction\_value, id\_bank\_card  FROM @ins;  END;  -------------------------------------------------------------  INSERT INTO transactions (transaction\_type, transaction\_destination, transaction\_date, transaction\_number, transaction\_value, id\_bank\_card)  VALUES ('Payment', 'Recipient', GETDATE(), '1234567890', 100000, 12);  select \* from transactions  select \* from bank\_card;  ----------ТРИГГЕР ДЛЯ ПРОВЕРКИ СУММЫ КРЕДИТА ПЕРЕД ВСТАВКОЙ----------  CREATE OR ALTER TRIGGER CheckCreditSum  ON credits  FOR INSERT  AS  BEGIN  DECLARE @credit\_sum money;  SELECT @credit\_sum = credit\_sum FROM inserted;  IF @credit\_sum <= 0  BEGIN  RAISERROR('Недостаточно средств для выполнения операции.', 16, 1);  ROLLBACK TRANSACTION;  END;  END;  -------------------------------------------------  ----------ТРИГГЕР ДЛЯ ПРОВЕРКИ ТИПА БАНКОВСКОЙ КАРТЫ ПЕРЕД ВСТАВКОЙ----------  CREATE OR ALTER TRIGGER CheckBankCardType  ON bank\_card  FOR INSERT  AS  BEGIN  DECLARE @bank\_card\_type nvarchar(50);  SELECT @bank\_card\_type = bank\_card\_type FROM inserted;    IF @bank\_card\_type NOT IN ('VISA', 'MasterCard')  BEGIN  RAISERROR('Тип карты не соответствует нормам.', 16, 1);  ROLLBACK TRANSACTION;  END;  END;  ------------------------------------------------------------  INSERT INTO bank\_card (bank\_card\_type, bank\_card\_number, bank\_card\_cvv\_code, bank\_card\_currency, bank\_card\_payment, bank\_card\_date, bank\_card\_pin, paymentSystem)  VALUES ('неVisa', '1111222233334444', '123', 'USD', 'Payment System', '2024-05-09', 1234, 'Visa');  --------------------ОБЯЗАТЕЛЬНЫЕ ПОЛЯ--------------------  CREATE OR ALTER TRIGGER CheckClientFields  ON client  FOR INSERT  AS  BEGIN  IF EXISTS (  SELECT 1  FROM inserted  WHERE client\_last\_name IS NULL  OR client\_first\_name IS NULL  OR client\_middle\_name IS NULL  OR client\_gender IS NULL  OR client\_password IS NULL  OR client\_email IS NULL  OR client\_phone\_number IS NULL  )  BEGIN  RAISERROR('Заполните обязательные поля.', 16, 1);  ROLLBACK TRANSACTION;  END;  END;  ---------------------------------------------------------------------------  select \* from client;  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number)  VALUES ('ало', 'ало', 'ало', 'ало', 'ало', 'ало@example.com', '155532131234');  ---------------УНИКАЛЬНОСТЬ КАРТЫ---------------  CREATE OR ALTER TRIGGER CheckBankCardNumber  ON bank\_card  FOR INSERT  AS  BEGIN  IF EXISTS (  SELECT 1  FROM inserted  WHERE bank\_card\_number IN (SELECT bank\_card\_number FROM bank\_card)  )  BEGIN  RAISERROR('Такая карта уже есть.', 16, 1);  ROLLBACK TRANSACTION;  END;  END;  ---------------------------------------------------------------------------  INSERT INTO bank\_card (bank\_card\_type, bank\_card\_number, bank\_card\_cvv\_code, bank\_card\_currency, bank\_card\_payment, bank\_card\_date, bank\_card\_pin, paymentSystem)  VALUES ('Visa', '1111222233334444', '123', 'USD', 'Payment System', '2024-05-09', 1234, 'Visa');  INSERT INTO bank\_card (bank\_card\_type, bank\_card\_number, bank\_card\_cvv\_code, bank\_card\_currency, bank\_card\_payment, bank\_card\_date, bank\_card\_pin, paymentSystem)  VALUES ('Visa', '1111222233334444', '123', 'USD', 'Payment System', '2024-05-09', 1234, 'Visa');  ---------------УНИКАЛЬНОСТЬ ТЕЛЕФОНА---------------  CREATE OR ALTER TRIGGER CheckUniquePhoneNumber  ON client  FOR INSERT  AS  BEGIN  IF EXISTS (  SELECT 1  FROM client  WHERE client\_phone\_number IN (SELECT client\_phone\_number FROM client)  GROUP BY client\_phone\_number  HAVING COUNT(\*) > 1  )  BEGIN  RAISERROR('Phone number must be unique.', 16, 1);  ROLLBACK TRANSACTION;  END;  END;  ------------------------------------------------------------  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number)  VALUES ('курсачевский', 'курсач', 'курсанович', 'кп', 'mypassword123', 'кп@example.com', '1234567890');  INSERT INTO client (client\_last\_name, client\_first\_name, client\_middle\_name, client\_gender, client\_password, client\_email, client\_phone\_number)  VALUES ('курсачевский2', 'курсач2', 'курсанович2', 'кп2', 'mypassword1232', 'кп2@example.com', '1234567890'); |