МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных банка с применением шифрования и маскирования данных»

Выполнил студент Ворович Кирилл Сергеевич

(Ф.И.О.)

Руководитель работы асс. Савельева Маргарита Геннадьевна

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

И.о. зав. кафедрой ст. преп. Блинова Е.А.

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

Курсовая работа защищена с оценкой

Минск 2023

Содержание

[Введение 5](#_Toc185409177)

[1Постановка задачи 6](#_Toc185409178)

[1.1 Аналитический обзор аналогов 6](#_Toc185409179)

[1.1.1 Аналог «M-Belarusbank» 6](#_Toc185409180)

[1.1.2 Аналог «BSB-bank» 7](#_Toc185409181)

[1.1.3 Аналог «Alpha-bank» 8](#_Toc185409182)

[1.2 Вывод по разделу 9](#_Toc185409183)

[2Изучение требований, определение вариантов использования 10](#_Toc185409184)

[2.1 Определение вариантов использования 10](#_Toc185409185)

[2.2 Диаграммы UML, взаимодействие всех компонентов 10](#_Toc185409186)

[2.3 Вывод по разделу 11](#_Toc185409187)

[3 Анализ и проектирование модели базы данных. Описание информационных объектов и ограничений целостности 12](#_Toc185409188)

[3.1 Разработка таблиц базы данных 12](#_Toc185409189)

[3.2 Разработка ролей 15](#_Toc185409190)

[3.3 Разработка функций и процедур 16](#_Toc185409191)

[3.4 Вывод по разделу 18](#_Toc185409192)

[4 Описание процедур экспорта и импорта данных 19](#_Toc185409193)

[4.1 Процедура экспорта данных из JSON-файла 19](#_Toc185409194)

[4.2 Процедура импорта данных из JSON-файла 20](#_Toc185409195)

[4.3 Вывод по разделу 21](#_Toc185409196)

[5 Тестирование производительности 22](#_Toc185409197)

[5.1 Заполнение таблицы 22](#_Toc185409198)

[5.2 Вывод по разделу 23](#_Toc185409199)

[6 Описание используемой технологии 24](#_Toc185409200)

[6.1 Реализация функций шифрования 24](#_Toc185409201)

[6.2 Реализация функции маскировки 26](#_Toc185409202)

[6.3 Вывод по разделу 27](#_Toc185409203)

[7 Руководство пользователя 28](#_Toc185409204)

[7.1 Вывод по разделу 29](#_Toc185409205)

[Заключение 30](#_Toc185409206)

[Список использованных источников 31](#_Toc185409207)

[Приложение A 32](#_Toc185409208)

[Приложение Б 33](#_Toc185409209)

[Приложение В 35](#_Toc185409210)

Введение

В современном мире стабильные финансовые системы необходимы для многих сфер деятельности. Одной из важнейших задач банковских систем является обеспечение безопасности данных клиентов, а также удобный доступ к финансовым операциям. Разрабатываемая система базы данных предназначена для управления финансовыми операциями клиентов и обеспечения надежного контроля со стороны руководства.

Создание базы данных с применением технологий шифрования и маскирования позволит защитить конфиденциальную информацию и предотвратить ее утечку. Клиенты смогут безопасно выполнять переводы средств, управлять своими счетами, а руководство получит инструмент для анализа и отслеживания финансовых операций.

Актуальность темы обусловлена растущими угрозами в области информационной безопасности и необходимостью соответствия современным требованиям защиты данных. Реализация шифрования и маскирования в базе данных поможет минимизировать риски, связанные с несанкционированным доступом, обеспечивая при этом высокую производительность и удобство использования.

Целью работы является разработка базы данных банка с учетом требований безопасности и эффективности. В рамках работы будет реализована архитектура базы данных, обеспечен доступ через безопасные процедуры и протестирована производительность на больших объемах данных. Кроме того, технологии шифрования и маскирования станут ключевыми компонентами для защиты информации о клиентах и финансовых операциях.

Шифрование позволяет скрывать информацию от неавторизованных лиц, сохраняя доступ для авторизованных пользователей. Маскировка, в свою очередь, обеспечивает защиту данных путем изменения исходных символов, создавая их безопасные версии для различных сценариев использования. Эти технологии играют ключевую роль в обеспечении конфиденциальности и безопасности данных в банковских системах.

1. Постановка задачи
   1. Аналитический обзор аналогов

В современном мире, когда скорость и точность играют важную роль, банковские компании становятся все более популярными. Одной из главных задач таких компаний является надежные финансовые операции.

1.1.1 Аналог «M-Belarusbank»

Мобильное приложение «M-Belarusbank» – приложение, предназначенное для управление банковского счета belarusbank, а также имеются финансовые операции, не связанные со счетом банка [1].

Интерфейса приведён на рисунке 1.1.

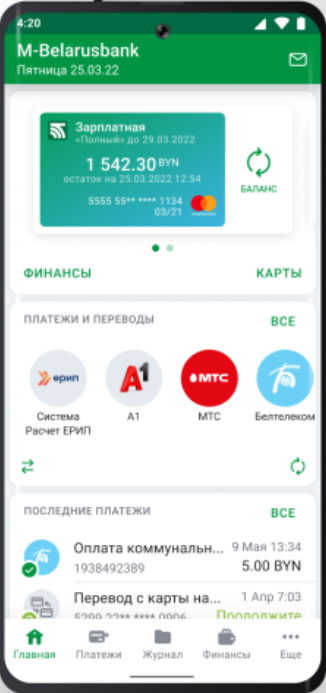


Рисунок 1.1 – Интерфейс приложения «M-Belarusbank»

Это приложение позволяет пользователям управлять своими банковскими картами и финансами прямо со смартфона.

На главной странице отображается информация о текущем балансе, включая название карты, её тип, остаток средств, а также валюту счёта. Пользователь может видеть подробности последних операций, таких как переводы, оплата услуг или пополнение счёта.

Приложение предоставляет удобный доступ к оформлению вкладов и кредитов, позволяя выбрать наиболее подходящие условия прямо из интерфейса. Быстрый доступ к популярным услугам, включая коммунальные платежи, связь и интернет, обеспечивается благодаря интеграции с системами, такими как ЕРИП.

В нижней части экрана расположены навигационные вкладки, которые позволяют просматривать журнал операций, управлять финансами, совершать платежи и использовать дополнительные функции. Удобный и понятный интерфейс приложения акцентирует внимание на простоте взаимодействия, быстроте выполнения операций и доступности всех необходимых функций.

1.1.2 Аналог «BSB-bank»

Мобильное приложение «BSB-bank» – приложение, предназначенное для управление банковского счета bsb-bank, а также для совершения платежей и переводов[2]. Интерфейса приведён на рисунке 1.2.



Рисунок 1.2 – Интерфейс приложения «BSB-Bank»

Приложение позволяет управлять счётом, просматривать баланс, выписки и детали операций, включая валюту счёта. Пользователь также может отслеживать актуальный курс валют, что особенно удобно для тех, кто совершает операции в разных валютах.

Доступны функции оплаты мобильной связи, интернета, коммунальных услуг и других услуг. Важные операции можно сохранить в избранное для быстрого доступа. Все платежи и операции структурированы, что позволяет легко находить нужную информацию.

Интерфейс интуитивный и поддерживает быструю навигацию между основными функциями. Дизайн приложения продуман для комфортного использования: яркие элементы выделяют ключевые функции, а детализированные разделы помогают эффективно управлять финансами и планировать расходы.

1.1.3 Аналог «Alpha-bank»

Мобильное приложение «Alpha-bank» – приложение, предназначенное для управление банковского счета alpha-bank, а также для совершения платежей и переводов [3]. Интерфейса приведён на рисунке 1.3

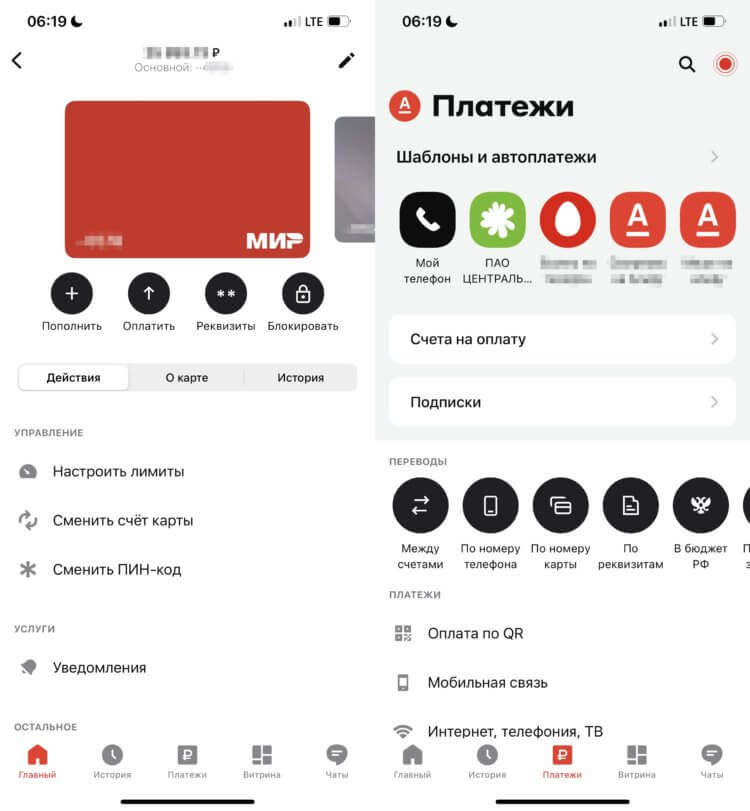


Рисунок 1.3 – Интерфейс приложения «Alpha-Bank»

Приложение предоставляет возможность полного управления банковскими картами, включая пополнение, оплату, просмотр реквизитов и блокировку карты. Пользователь может настроить лимиты, изменить ПИН-код, включить или отключить уведомления для контроля финансовых операций. В приложении также доступен просмотр валюты счёта с актуальными курсами, что удобно для пользователей, совершающих операции в разных валютах.

В разделе платежей доступен широкий выбор услуг: переводы между счетами, по номеру телефона или карты, оплата по реквизитам, а также платежи в бюджет РФ. Для удобства доступны шаблоны и автоплатежи, которые значительно упрощают выполнение регулярных операций. Приложение поддерживает оплату по QR-коду, а также предоставляет функции подключения и оплаты услуг мобильной связи, интернета и телевидения.

Пользователи могут обратиться в службу поддержки через встроенный чат с администрацией, где оперативно решаются вопросы, связанные с обслуживанием счёта или проведением операций. Интерфейс приложения выполнен в минималистичном стиле с акцентом на интуитивность, что делает процесс управления финансами максимально удобным, быстрым и понятным.

1.2 Вывод по разделу

Проанализировав аналогичные банковские приложения, были определены ключевые моменты, которые необходимо учесть для создания конкурентоспособного продукта:

– возможность управления банковскими картами (пополнение, оплата, смена имени карты);

– просмотр истории операций с детализацией;

– отображение валюты счёта и актуальных курсов валют;

– поддержка переводов между счетами и по номеру карты;

– быстрый доступ к часто используемым функциям, таким как избранные платежи и выписки;

– высокая скорость работы и удобная навигация по всем разделам.

Для обеспечения конкурентоспособности приложение также должно предоставлять современные функции безопасности, возможность блокировки карты. Это позволит пользователям чувствовать себя уверенно при работе с финансами и минимизирует риски несанкционированного доступа.

1. Изучение требований, определение вариантов использования
   1. Определение вариантов использования

Для формирования функциональных требований к базе данных необходимо определить способы обработки данных и обеспечить нужную функциональность пользовательскому интерфейсу. Важно описать, как хранятся данные, как выполняется их поиск и выборка, как осуществляется обновление информации и какие меры принимаются для обеспечения безопасности данных.

Также ключевым этапом проектирования является определение ролей пользователей и их взаимодействия с системой. Варианты использования системы помогают определить доступный функционал для каждой роли, описывая, как именно пользователи могут работать с базой данных.

Обычно такие варианты использования отображаются в виде диаграмм Use–Case, которые наглядно показывают взаимодействие пользователей с системой и помогают визуализировать распределение функций. UML диаграмма вариантов использования банковского приложения представлена в приложении А.

Роли пользователя представляют собой совокупность прав, которые могут быть предоставлены пользователю в системе. Обычно роли назначаются в зависимости от позиции, роли или ответственности человека в организации.

User (авторизированный пользователь) – пользователю доступна возможность просматривать информацию о своих счетах, создать счет, изменить свой профиль, совершить перевод между счетами.

Admin (администратор банка) – сотрудник, закреплённый за банком. В функции адинстратора входит: пополнение счета пользователя, подтверждение операций из очереди, имение операций в очереди, а также функции пользователя.

AdminPDB (модератор базы данных) – сотрудник который обладает всеми привилегиями, такими как: создание ролей, изменение ролей, удаление ролей, создание таблиц и другое.

Эти роли и варианты использования помогают ясно понять, какие функции и данные должны быть доступны для каждого типа пользователя, и как должна быть организована навигация в системе.

* 1. Диаграммы UML, взаимодействие всех компонентов

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.2.

|  |
| --- |
|  |

Рисунок 2.2 – Диаграмма базы данных

Как видно из таблицы, между всеми первичными и внешними ключами установлена связь один ко многим.

* 1. Вывод по разделу

Используя архитектуру, разработанную в рамках текущего раздела, получится создать базу данных банковской системы. Однако для обеспечения её возможности поддержки коммерческого продукта необходимо продумать остальные её аспекты. Эти аспекты будут описаны в следующих разделах.

3 Анализ и проектирование модели базы данных. Описание информационных объектов и ограничений целостности

Для разработки базы данных, способной эффективно и безопасно обрабатывать большие объемы данных, необходимо:

– создать грамотно спроектированные таблицы;

– индексы для часто используемых полей, по которым будет осуществляться выборка данных;

– представления, которые будут предоставлять только те данные, которые требуются в конкретной ситуации;

– функции и процедуры для выполнения всех операций с базой данных;

– триггеры, обеспечивающие безопасные вставки, удаления и изменения данных;

– пользователей, у которых будет доступ только к тем функциям, которые соответствуют их правам.

После выполнения всех этих шагов база данных будет готова для использования в коммерческой среде. В процессе курсового проектирования была выбрана система управления базами данных Oracle, поскольку она обеспечивает высокую производительность, гарантирует безопасность и поддерживает множество различных типов данных.

Для управления базой данных была использована SQL Developer, так как это официальный инструмент, предназначенный для работы с Oracle.

## **3.1 Разработка таблиц базы данных**

Для реализации базы данных банка было разработано 5 таблиц, подробнее про каждую таблицу написано ниже.

Таблица «Users» содержит информацию о пользователях, её описание приведено в таблице 3.1.

Таблица 3.1 – Описание таблицы «Users»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| User\_id | Уникальный идентификатор пользователя, первичный ключ | NUMBER |
| User\_role | ID роли пользователя  Не может быть пустым ссылается на таблицу Role | NUMBER |
| login | Логин пользователя должен быть уникальным и не может быть пустым | VARCHAR2(100) |
| pass | Хеш пароля, не может быть пустым | VARCHAR(255) |
| user\_name | Имя пользователя, не может быть пустым | VARCHAR(100) |
| last\_name | Фамилия пользователя, не может быть пустым | VARCHAR(100) |

Окончание таблицы 3.1 – Описание таблицы «Users»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| phone\_number | Номер телефона пользователя не может быть пустым, должен быть уникальным | VARCHAR(100) |
| pasport | Паспорт пользователя, должен быть уникальным, не может быть пустым | VARCHAR(100) |

Как видно из таблицы, для обеспечения безопасности пользователей, пароль записывается в виде хэша.

В таблице «Role» расположена информация о ролях пользователей, её структура приведена в таблице 3.2.

Таблице 3.2 – Описание таблицы «Role»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| Role\_id | Уникальный идентификатор роли, первичный ключ | NUMBER |
| name | Имя роли, не может быть пустой | VARCHAR (100) |
| role\_level | Уровень роли, не может быть пустой | NUMBER |

В таблице «Cash\_accounts» расположена информация о источниках медиа-данных для новостей, её структура приведена в таблице 3.3.

Таблица 3.3 – Описание таблицы «Cash\_accounts»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| Cash\_id | Уникальный идентификатор счета, первичный ключ | VARCHAR (100) |
| Cash\_owner | Идентификатор владельца счета  Ссылается на таблицу users | NUMBER |
| balance | Баланс счета, не может быть пустым | VARCHAR(100) |
| Cash\_name | Имя счета, не может быть пустым | VARCHAR(100) |
| IsBlocked | Указывает заблокированный счет или нет, по умолчанию 0 | NUMBER(1) |
| Creation\_date | Дата создания счета, по умолчанию устанавливается текущая дата | DATE |
| Currency\_id | Идентификатор валюты.  Ссылается на таблицу Currency | VARCHAR2(4) |

В таблице «Currencies» расположена информация валюте. Содержит название валюты, а также стоимость валюты для преобразования одной валюты в другую. Для удобства цена USD 1. Ее структура представлена в таблице 3.4

Таблица 3.4 – Описание таблицы «Currencies»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| Currency\_id | Уникальный идентификатор счета, первичный ключ | VARCHAR (100) |
| Currency\_name | Идентификатор владельца счета  Ссылается на таблицу users | NUMBER |
| Exchange\_rate | Баланс счета, не может быть пустым | VARCHAR(100) |

таблице «History» расположена информация о совершенных операциях, её структура приведена в таблице 3.4.

Таблица 3.5 – Описание таблицы «History»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| Operation\_id | Уникальный идентификатор операции в таблице, первичный ключ | NUMBER |
| Transactor | Идентификатор пользователя, совершившего операцию. Ссылается на таблицу users | NUMBER |
| cash\_from | Идентификатор счета, с которого совершена операция, ссылается на таблицу Cash\_accounts | VARCHAR (100) |
| cash\_to | Идентификатор счета, на который совершена операция, ссылается на таблицу Cash\_accounts | VARCHAR2(100) |
| operation | Название операции | VARCHAR (100) |
| amount | Сумма операции | VARCHAR2(100) |
| operation\_description | Описание операции | VARCHAR2(100) |

В таблице «Queue» расположена информация о операциях добавленых в очередь, её структура приведена в таблице 3.4.

Таблица 3.6 – Описание таблицы «Queue»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| Operation\_id | Уникальный идентификатор операции в таблице, первичный ключ | NUMBER |
| Transactor | Идентификатор пользователя, совершившего операцию  Ссылается на таблицу users | NUMBER |
| cash\_from | Идентификатор счета, с которого совершена операция, ссылается на таблицу Cash\_accounts | VARCHAR (100) |
| cash\_to | Идентификатор счета, на который совершена операция, ссылается на таблицу Cash\_accounts | VARCHAR2(100) |

Окончание таблицы 3.6 – Описание таблицы «Queue»

|  |  |  |
| --- | --- | --- |
| Атрибут | Описание | Тип данных |
| amount | Сумма операции | VARCHAR2(100) |
| operation\_description | Описание операции | VARCHAR2(100) |
| operation | Название операции | VARCHAR (100) |

Листинг SQL-кода для создания таблиц, а также инициализации структур представлен в приложении Б.

Описание шифрования представлена в разделе 6

## **3.2 Разработка ролей**

Для доступа к базе данных предусмотрено 3 роли: пользователь, администратор и администратор PDB. Для администратора предусмотрены отдельные процедуры и функции, которые недоступны пользователю. Для администратора PDB предоставлены все привилегии над PDB. Код для инициализации ролей представлен в Листинге 3.1.

CREATE USER pdbAdmin IDENTIFIED BY admin;

GRANT EXECUTE ON DBMS\_CRYPTO TO pdbAdmin;

CREATE USER Client\_user IDENTIFIED BY password;

CREATE USER Admin\_user IDENTIFIED BY password;

GRANT ALL PRIVILEGES TO pdbAdmin;

GRANT user\_pdb\_role TO Client\_user;

GRANT user\_pdb\_role TO Admin\_user;

GRANT admin\_pdb\_role TO Admin\_user;

CREATE ROLE user\_pdb\_role;

GRANT CREATE SESSION TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.Register\_User TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.Authorize\_User TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.UpdateUser TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.DeleteUser TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.CreateCashAccount TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.ProcessTransfer TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.Get\_User\_Info TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.Get\_Cash\_Accounts TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.GetUncryptHistory TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.GetUserRoleLevel TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.DeleteUser TO user\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.DeleteCashAccount TO user\_pdb\_role;

CREATE ROLE admin\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.UPDATEACCOUNTBALANCE TO admin\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.UpdateQueueRow TO admin\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.ExecuteQueueOperation TO admin\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.GetUncryptQueue TO admin\_pdb\_role;

GRANT EXECUTE ON pdbAdmin.DeleteQueueRow TO admin\_pdb\_role;

Листинг 3.1 – SQL-код для инициализации ролей

Для управления базой данных был создан пользователь pdbAdmin, который обладает полными привилегиями, позволяющими ему выполнять такие действия, как создание таблиц, управление ролями и разработка процедур и функций. Этот пользователь так же имеет привилегию на выполнения DBMS\_CRYPTO что позволило создать функции шифрования [4].

Пользователь Admin\_user наделён правами управления профилями других пользователей, изменением счетов, просмотром истории всех операций и управлением очередью операций. Он имеет роли admin\_pdb\_role и user\_pdb\_role, которые расширяют его функционал в системе.

Пользователь Client\_user имеет минимально необходимые привилегии для использования приложения, позволяя ему создавать счета, управлять своим профилем, отправлять переводы и выполнять другие операции.

## **3.3 Разработка функций и процедур**

Для обеспечения безопасности весь доступ к данным осуществляется через функции и процедуры. Этот подход позволяет не только защитить информацию, но и организовать структурированный доступ к необходимым операциям. Каждая функция и процедура реализует определённый бизнес-логический процесс, гарантируя целостность данных и соответствие установленным правилам. В дополнение к этому, в функциях реализована проверка прав пользователя, что позволяет контролировать доступ к критически важной информации и предотвращает несанкционированные операции. Такой механизм обеспечивает гибкость и безопасность работы с базой данных, минимизируя риски и повышая уровень доверия к системе. Пример функции создания счета представлена в приложение В.

Функция CreateCashAccount предназначена для создания нового денежного счета в системе. Она принимает три параметра: p\_executor\_id, идентификатор исполнителя, p\_cash\_owner, идентификатор владельца счета, и p\_cash\_name, имя создаваемого счета. Функция возвращает строку, указывающую на результат выполнения операции.

При начале работы функция выполняет несколько проверок для обеспечения корректности данных. Сначала она проверяет, что параметры не равны NULL, и если это так, возвращает соответствующее сообщение об ошибке. Затем происходит извлечение уровня роли исполнителя из представления пользователей и проверки прав доступа. Функция позволяет создавать счет только в том случае, если исполнитель является владельцем счета или имеет достаточные права.

Если все проверки пройдены, функция генерирует новый уникальный идентификатор счета с использованием последовательности Cash\_id\_seq и вставляет новую запись в таблицу Cash\_accounts. Также добавляется запись в историю операций, фиксирующая создание счета.

В случае успешного выполнения функция фиксирует изменения с помощью команды COMMIT и возвращает сообщение об успешном создании счета. Если возникает ошибка, функция обрабатывает исключения и возвращает соответствующее сообщение об ошибке, информируя о возможных проблемах, таких как отсутствие данных или другие непредвиденные ошибки.

Для добавления строк в таблицу history была создана функция. Пользователи не имеют доступа к данной функции. Данная функция используется только в других функциях и процедурах. Функция представлена в листинге 3.2

|  |
| --- |
| CREATE OR REPLACE PROCEDURE AddToHistory (  p\_transactor IN NUMBER, -- ID транзактора (пользователя, который выполняет операцию)  p\_cash\_from IN NUMBER DEFAULT NULL, -- ID счета, с которого производится операция  p\_cash\_to IN NUMBER DEFAULT NULL, -- ID счета, на который производится операция  p\_operation IN VARCHAR2, -- Тип операции  p\_amount IN NUMBER DEFAULT NULL, -- Сумма операции  p\_operation\_description IN VARCHAR2 DEFAULT NULL -- Описание операции  ) AS  BEGIN    INSERT INTO History (Transactor, cash\_from, cash\_to, operation, amount, operation\_description)  VALUES (p\_transactor, p\_cash\_from, p\_cash\_to, p\_operation, p\_amount, p\_operation\_description);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20004, 'Произошла ошибка: ' || SQLERRM);  END;  / |

Листинг 3.1 – SQL-код для создания функции AddToHistory

Процедура AddToHistory предназначена для добавления записей об операциях в историю транзакций. Она принимает несколько параметров: p\_transactor (идентификатор пользователя, выполняющего операцию), p\_cash\_from (идентификатор счета, с которого производится операция), p\_cash\_to (идентификатор счета, на который производится операция), p\_operation (тип операции), p\_amount (сумма операции) и p\_operation\_description (описание операции).

При выполнении процедуры происходит вставка данных в таблицу History. Если операция успешна, изменения фиксируются с помощью команды COMMIT. В случае возникновения ошибки процедура перехватывает исключение и возвращает сообщение об ошибке с помощью RAISE\_APPLICATION\_ERROR, что позволяет разработчикам и пользователям получить информацию о проблеме [5].

Для обновления счета пользователя была добавлена функция UpdateAccountBalance. Листинг функции представлен в приложение В. Функция предназначена для обновления баланса счета пользователя. Она принимает три параметра: p\_user\_id (идентификатор пользователя), p\_cash\_id (идентификатор счета) и p\_amount (сумма, на которую нужно изменить баланс).

Сначала функция проверяет уровень роли пользователя, чтобы убедиться, что у него достаточно прав для выполнения операции. Затем осуществляется проверка существования счета и получение текущего баланса. После этого вычисляется новый баланс, и проверяется, чтобы он не оказался отрицательным. Если баланс корректен, вызывается процедура SetBalance для обновления баланса счета, а также записывается операция в историю с помощью процедуры AddToHistory.

Функция возвращает сообщение о результате операции: успешном обновлении баланса или ошибке в случае недостатка прав, отсутствия счета или пользователя.

3.4 Вывод по разделу

В этом разделе были разработаны ключевые объекты базы данных, которые обеспечивают взаимодействие конечного пользователя с системой. При создании таблиц были учтены потенциальные трудности в управлении данными, что позволило внедрить ограничения для отдельных значений и установить связи между таблицами.

Процедуры и функции, разработанные для работы с таблицами, инкапсулируют логику управления данными, что значительно упрощает взаимодействие с ними и способствует разделению задач по ролям.

Кроме того, для различных ролей пользователей был создан набор привилегий, оформленный в виде ролей. Это позволяет эффективно управлять доступом к функциям и данным, наделяя пользователей определёнными правами в зависимости от их ролей в системе.

# 4 Описание процедур экспорта и импорта данных

В данном разделе представлены процедуры для экспорта и импорта данных из таблиц в формате JSON в Oracle. Эти процедуры позволяют эффективно управлять данными, обеспечивая их сохранность и удобство переноса.

4.1 Процедура экспорта данных из JSON-файла

Экспорт в контексте баз данных — это процесс сохранения данных из базы данных во внешний файл или другое хранилище. Он может использоваться для резервного копирования, обмена данными между системами или анализа данных с помощью сторонних инструментов. В результате экспорта данные становятся доступными в структурированном формате, что упрощает их передачу и анализ вне самой базы данных.

Для каждой таблице была разработана функция экспорта В процессе работы создаются переменные для хранения сгенерированного JSON и для работы с файлами. С помощью SQL-запроса генерируется JSON, где каждая запись преобразуется в JSON-объект с определенными полями. Полученный JSON сохраняется в переменной.

Затем процедура открывает файл для записи, передавая имя директории и имя файла. Данные в формате JSON записываются в открытый файл, после чего файл закрывается. В листинге 4.1 приведён пример процедуры экспорта ролей

CREATE OR REPLACE PROCEDURE Export\_Role\_To\_JSON (

p\_file\_name IN VARCHAR2

) AS

l\_json\_clob CLOB;

l\_file UTL\_FILE.FILE\_TYPE;

BEGIN

-- Генерация JSON из данных таблицы Role

SELECT JSON\_ARRAYAGG(

JSON\_OBJECT(

'Role\_id' VALUE Role\_id,

'name' VALUE name,

'role\_level' VALUE role\_level

) RETURNING CLOB

)

INTO l\_json\_clob

FROM Role;

-- Запись JSON в файл

l\_file := UTL\_FILE.FOPEN('MY\_DIRECTORY', p\_file\_name, 'w', 32767);

UTL\_FILE.PUT\_LINE(l\_file, l\_json\_clob);

UTL\_FILE.FCLOSE(l\_file);

END;

Листинг 4.1 – SQL-код процедуры «Export\_Role\_To\_JSON»

4.2 Процедура импорта данных из JSON-файла

Импорт — это процедура перемещения данных из внешних источников, таких как файлы или другие базы данных, в текущую систему или приложение. В контексте баз данных импорт служит для загрузки данных из различных форматов файлов или из других баз в уже существующую базу данных. Этот процесс дает возможность обновлять, дополнять или восстанавливать данные с учетом изменений, происходящих во внешних источниках.

Процедура предназначена для импорта данных из файла в формате JSON в базу данных. Она принимает одно входное значение — имя файла (p\_file\_name), из которого будут извлекаться данные.

В начале процедуры создается переменная для хранения данных JSON (l\_json\_clob) и переменная для работы с файлом (l\_file). Процедура открывает файл для чтения с помощью функции UTL\_FILE.FOPEN, передавая имя директории (MY\_DIRECTORY) и имя файла. Затем она считывает содержимое файла в переменную l\_json\_clob с помощью функции UTL\_FILE.GET\_LINE и закрывает файл.

Далее выполняется разбор содержимого JSON с помощью функции JSON\_TABLE, которая извлекает данные в структуре JSON [6]. Каждый элемент JSON обрабатывается в цикле, и для каждого элемента проверяется наличие поля Role\_id. Если значение Role\_id присутствует, выполняется вставка записи с указанием этого идентификатора в таблицу. Если Role\_id отсутствует, запись добавляется без него.

Таким образом, процедура позволяет загружать данные из JSON-файла в таблицу базы данных, учитывая возможность наличия или отсутствия идентификатора для каждой записи. В листинге 4.2 приведён пример процедуры импорта ролей

CREATE OR REPLACE PROCEDURE Import\_Role\_From\_JSON (

p\_file\_name IN VARCHAR2

) AS

l\_json\_clob CLOB;

l\_file UTL\_FILE.FILE\_TYPE;

BEGIN

-- Чтение JSON из файла

l\_file := UTL\_FILE.FOPEN('MY\_DIRECTORY', p\_file\_name, 'r', 32767);

UTL\_FILE.GET\_LINE(l\_file, l\_json\_clob);

UTL\_FILE.FCLOSE(l\_file);

-- Разбор JSON и вставка данных в таблицу Role

FOR rec IN (

SELECT \*

FROM JSON\_TABLE(

l\_json\_clob,

'$[\*]'

COLUMNS (

Role\_id NUMBER PATH '$.Role\_id', -- Считываем Role\_id, если он есть

name VARCHAR2(100) PATH '$.name',

role\_level NUMBER PATH '$.role\_level'

)

)

) LOOP

-- Проверяем, если Role\_id указан

IF rec.Role\_id IS NOT NULL THEN

INSERT INTO Role (Role\_id, name, role\_level)

VALUES (rec.Role\_id, rec.name, rec.role\_level);

ELSE

-- Если Role\_id не указан, вставляем только name и role\_level

INSERT INTO Role (name, role\_level)

VALUES (rec.name, rec.role\_level);

END IF;

END LOOP;

END;

/

Листинг 4.2 – SQL-код процедуры «Import\_Role\_To\_JSON»

Процедура Import\_Role\_From\_JSON позволяет загружать данные о ролях из JSON-файла в таблицу Role. С помощью пакета UTL\_FILE процедура читает содержимое файла, разбирает JSON-структуру с использованием функции JSON\_TABLE, а затем вставляет данные в таблицу. Если в JSON указан Role\_id, он включается в запись; иначе данные сохраняются без него.

4.3 Вывод по разделу

В этом разделе были рассмотрены процедуры экспорта и импорта данных в базу данных с использованием формата JSON. Процедура экспорта позволяет извлекать данные из базы данных и сохранять их в файл JSON, что упрощает процесс передачи и анализа информации. Процедура импорта, в свою очередь, обеспечивает загрузку данных из JSON-файла обратно в базу данных, с возможностью обработки записей с учетом идентификаторов.

Использование JSON в качестве формата обмена данными обеспечивает гибкость и структурированность, позволяя легко интегрировать данные между различными системами. Описанные процедуры служат основой для реализации процессов резервного копирования, миграции данных и обмена информацией между различными приложениями.

Таким образом, понимание и внедрение процедур экспорта и импорта данных в формате JSON является важным аспектом работы с базами данных, что способствует эффективному управлению данными и улучшению взаимодействия между различными системами.

5 Тестирование производительности

Тестирование производительности базы данных является критически важным этапом в процессе разработки и оптимизации систем. Оно позволяет измерить, насколько эффективно база данных может обрабатывать запросы и выдавать результаты в заданные сроки. Оценка производительности помогает выявить узкие места, которые могут негативно сказаться на быстродействии приложения, а также определить, насколько база данных справляется с нагрузкой в реальных условиях эксплуатации. Понимание производительности системы позволяет разработчикам и администраторам баз данных принимать обоснованные решения по оптимизации запросов, улучшению структуры данных и масштабированию системы для обеспечения надежной и быстрой работы приложения.

5.1 Заполнение таблицы

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для этой цели была создана функция, которая заполняет таблицу USERS на 100000 строк, представленная на листинге 5.1

|  |
| --- |
| CREATE OR REPLACE PROCEDURE Generate\_Users AS  v\_login VARCHAR2(100);  v\_pass VARCHAR2(100) := 'pass';  v\_user\_name VARCHAR2(100) := 'name';  v\_last\_name VARCHAR2(100) := 'last\_name';  v\_phone\_number VARCHAR2(100);  v\_passport VARCHAR2(100);  v\_user\_role NUMBER := 3; -- Роль пользователя  BEGIN  FOR i IN 2000..100000 LOOP  -- Генерация уникального номера телефона и паспорта  v\_phone\_number := '375' || LPAD(i, 9, '0'); -- 375 и 9 последовательных цифр  v\_passport := LPAD(i, 7, '0'); -- 7 последовательных цифр  -- Генерация уникального логина  v\_login := v\_phone\_number || v\_passport; -- Логин формируется из номера телефона и паспорта  -- Вызов процедуры для регистрации пользователя  BEGIN  -- Вставка данных в таблицу Users  INSERT INTO Users (login, pass, user\_name, last\_name, phone\_number, pasport, user\_role)  VALUES (v\_login, v\_pass, v\_user\_name, v\_last\_name, v\_phone\_number, v\_passport, v\_user\_role);  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Duplicate value found for login: ' || v\_login);  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error inserting user: ' || SQLERRM);  END;  END LOOP;  COMMIT; -- Сохранение изменений  END; |

Листинг 5.1 – Заполнение таблицы на 100000

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

|  |
| --- |
| SELECT \* FROM Users WHERE Login = EncryptData('admin'); |

Листинг 5.2 – Запрос к таблице USERS

Результаты выполнения запроса к таблице указывают на незначительные затраты времени и ресурсов. Время выполнения запроса составило 0,011 мс. Результаты запроса будут представлены на рисунке 5.1

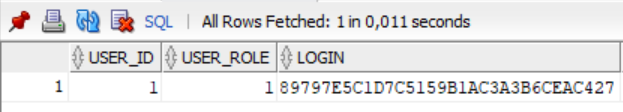


Рисунок 5.1 – Результат выполнения запроса

Поскольку индекс для таблиц был создан автоматически, создание иных индексов не требуется. Поскольку таблица Users хранит данные в зашифрованном виде для поиска требуется использовать EncryptData для шифрования данных.

5.2 Вывод по разделу

Тестирование производительности проводилось на таблице, содержащей 100,000 строк, что обеспечило репрезентативную выборку для анализа времени выполнения запросов. Все тесты были успешно завершены, продемонстрировав эффективное выполнение запросов и отсутствие значительных узких мест.

Полученные результаты подтверждают, что система способна обрабатывать большие объемы данных без ухудшения производительности. Это говорит о правильной структуре базы данных и оптимизации запросов, что является важным фактором для обеспечения стабильной работы приложения при высоких нагрузках.

# 6 Описание используемой технологии

Технологией, используемой в данной базе данных, является шифрование и маскирование. Ее суть заключается в защите конфиденциальных данных от несанкционированного доступа и обеспечения безопасности информации на всех уровнях.

Шифрование преобразует данные в нечитабельный формат, который может быть расшифрован только авторизованными пользователями с соответствующими ключами. Это гарантирует, что даже в случае утечки данных злоумышленники не смогут получить доступ к их содержимому.

Маскирование данных, в свою очередь, позволяет скрыть часть информации, заменяя ее подделанными значениями или замещающими символами. Это особенно полезно при работе с тестовыми данными или при предоставлении доступа третьим лицам, когда полная информация не должна быть раскрыта.

Таким образом, шифрование и маскирование не только повышают уровень безопасности, но и помогают соответствовать требованиям законодательства и стандартов защиты данных.

6.1 Реализация функций шифрования

Для реализации технологии было разработана функция шифрования EncryptData и функция расшифровки DecryptData. Функция EncryptData принимает строку в качестве входного параметра и возвращает зашифрованные данные в формате RAW. Внутри функции сначала извлекается ключ шифрования из таблицы EncryptionKeys, используя идентификатор ключа. Затем данные шифруются с использованием алгоритма AES с 128-битным ключом в режиме цепочки (CBC) и с PKCS для дополнения [7]. Представлена на листинге 6.1.

CREATE OR REPLACE FUNCTION EncryptData (

p\_data IN VARCHAR2

) RETURN RAW IS

v\_key RAW(32);

v\_encrypted\_data RAW(2000);

BEGIN

SELECT Key\_value INTO v\_key FROM EncryptionKeys WHERE Key\_id = 1;

v\_encrypted\_data := DBMS\_CRYPTO.ENCRYPT(src => UTL\_RAW.CAST\_TO\_RAW(p\_data), typ => DBMS\_CRYPTO.ENCRYPT\_AES128 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5, => v\_key

);

RETURN v\_encrypted\_data;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Ошибка при шифровании: ' || SQLERRM);

END;

/

Листинг 6.1 – SQL-код функции «EncryptData»

Функция DecryptData принимает зашифрованные данные в формате RAW и возвращает расшифрованную строку. Аналогично функции шифрования, сначала извлекается ключ из той же таблицы. Затем данные дешифруются с использованием тех же параметров, что и при шифровании. Представлена на листинге 6.2.

CREATE OR REPLACE FUNCTION DecryptData (

p\_encrypted\_data IN RAW

) RETURN VARCHAR2 IS

v\_key RAW(32);

v\_decrypted\_data RAW(2000);

BEGIN

SELECT Key\_value INTO v\_key FROM EncryptionKeys WHERE Key\_id = 1;

-- Дешифрование данных

v\_decrypted\_data := DBMS\_CRYPTO.DECRYPT(

src => p\_encrypted\_data,

typ => DBMS\_CRYPTO.ENCRYPT\_AES128 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,

key => v\_key

);

RETURN UTL\_RAW.CAST\_TO\_VARCHAR2(v\_decrypted\_data);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Ошибка при дешифровании: ' || SQLERRM);

END;

/

Листинг 6.2 – SQL-код функции «DecryptData»

Обе функции обеспечивают защиту данных, обеспечивая их шифрование и последующее расшифрование с использованием надежного алгоритма и ключа, что позволяет сохранить конфиденциальность информации в базе данных.

Для того что бы данные шифровались были созданы триггеры для таблиц Users, History, Queue, Cash\_accounts. Данные триггеры используют функцию шифрования при вставке и обновления строк таблиц. Триггер для Cash\_accounts представлен на листинг 6.3

CREATE OR REPLACE TRIGGER trg\_encrypt\_cash\_accounts

BEFORE INSERT OR UPDATE ON Cash\_accountsFOR EACH ROWBEGIN

IF :OLD.Cash\_id IS NULL OR :OLD.Cash\_id != :NEW.Cash\_id THEN

:NEW.Cash\_id := EncryptCashId(:NEW.Cash\_id); END IF;

IF :OLD.balance IS NULL OR :OLD.balance != :NEW.balance THEN

:NEW.balance := EncryptData(To\_char(:NEW.balance)); balance END IF;

IF :OLD.Cash\_name IS NULL OR :OLD.Cash\_name != :NEW.Cash\_name THEN :NEW.Cash\_name := EncryptData(:NEW.Cash\_name); Cash\_name END IF;END;/

Листинг 6.3 – SQL-код триггера «trg\_encrypt\_cash\_accounts»

Для расшифровки были созданные представления для таблиц с использованием DecryptData. Созданные представления UsersView, uncrypt\_cash\_accounts, uncrypt\_Queue, uncrypt\_History. Представление uncrypt\_History представлено на листинге 6.4.

CREATE OR REPLACE VIEW uncrypt\_History AS

SELECT

Operation\_id,

Transactor,

DecryptData(cash\_from) AS cash\_from, -- Дешифровка cash\_from

DecryptData(cash\_to) AS cash\_to, -- Дешифровка cash\_to

DecryptData(operation) AS operation, -- Дешифровка operation

TO\_NUMBER(DecryptData(amount)) AS amount, -- Дешифровка amount, преобразуем обратно в число

DecryptData(operation\_description) AS operation\_description -- Дешифровка operation\_description

FROM History;

/

Листинг 6.4 – SQL-код представления «uncrypt\_History»

В представлении выбираются поля Operation\_id, Transactor, cash\_from, cash\_to, operation, amount и operation\_description, при этом для каждого из них применяется функция DecryptData, которая выполняет дешифровку. Поле amount дополнительно преобразуется в числовой формат с помощью функции TO\_NUMBER. Это представление позволяет пользователям получать доступ к расшифрованной информации о транзакциях, сохраняя при этом безопасность хранения данных.

6.2 Реализация функции маскировки

Для реализации маскировки была разработана функция Get\_User\_Info, которая возвращает курсор с информацией о пользователе, в котором маскируются поля phone\_number и pasport. У phone\_number отображаются первые и последние 3 символа, у pasport отображается первые 2 и последние 2 символа. Курсор представлен на листинге 6.5

OPEN v\_cursor FOR

SELECT User\_id, login, user\_name, last\_name,

SUBSTR(phone\_number, 1, 3) || '\*\*\*\*\*' || SUBSTR(phone\_number, -3) AS phone\_number,

SUBSTR(pasport, 1, 2) || '\*\*\*\*\*' || SUBSTR(pasport, -2) as pasport

FROM UsersView

Листинг 6.5 – SQL-код курсора функции Get\_User\_Info

Данный код открывает курсор v\_cursor для выполнения SQL-запроса, который выбирает информацию о пользователях из представления UsersView. Запрашиваются поля User\_id, login, user\_name, last\_name, а также зашифрованные версии phone\_number и pasport. Номера телефонов и паспорта скрываются с помощью функции SUBSTR, оставляя видимыми только первые три и последние три символа, при этом заменяя центральные цифры символами \*\*\*\*\*. Это позволяет обеспечить конфиденциальность данных пользователей.

6.3 Вывод по разделу

В разделе шифрования и маскировки было продемонстрировано, как защищать данные в базе данных с помощью функций EncryptData и DecryptData. Эти функции обеспечивают шифрование и дешифрование информации, используя надежный алгоритм AES128 с цепочкой CBC и дополнением PKCS [8]. Также были созданы представления, которые позволяют пользователям получать расшифрованные данные, не нарушая принципов безопасности. Внедрение шифрования и маскировки данных существенно повышает уровень защиты конфиденциальной информации и позволяет соответствовать современным требованиям безопасности. Кроме того, обеспечивается контроль доступа к данным, что минимизирует риски утечек и несанкционированного доступа. Маскировка данных, реализованная через функции отображения, защищает личные данные пользователей, позволяя видеть только необходимую информацию, что является важным шагом в соблюдении законодательства о защите персональных данных. Все эти меры способствуют созданию надежной системы управления данными, обеспечивая безопасность и доверие со стороны пользователей.

Использование современных технологий шифрования позволяет оперативно адаптироваться к новым вызовам в области кибербезопасности и сохранять целостность данных на протяжении всего их жизненного цикла. В результате пользователи могут быть уверены в том, что их личная информация защищена на всех уровнях, что является ключевым фактором для успешного функционирования современного бизнеса.

# 7 Руководство пользователя

При входе у пользователя будет форма авторизации, так же будет кнопка регистрации. Для регистрации необходимо заполнить нужные поля, с условием что логин должен быть уникальным, поле номер телефона должно начинаться на 375 и содержать 10 цифр, поле паспорт должно быть уникальным и содержатся 7 символов. После регистрации пользователь попадет на страницу авторизации. После спешной авторизации пользователь перейдет на главную страницу. Главная страница представлена на рисунке 7.1

|  |
| --- |
|  |

Рисунок 7.1 – Главная страница банка

Находясь на главное странице, пользователь имеет возможность просмотреть свои счета, для этого необходимо нажать на кнопку «обновить». Так же может перейти на странице «профиль», «создать счет», «перевод», «история».

На странице профиль пользователь может увидеть информацию о своём профиле. Поля «телефон», и «паспорт» замаскированы. Для получения информации о профиле нужно нажать на кнопку «обновить информацию». Так же пользователь может обновить информацию о профиле заполнив поля, которые он хочет изменить. Профиль представлен на рисунке 7.2

|  |
| --- |
|  |

Рисунок 7.2 – Страница профиля пользователя

Для создания нового счета пользователю необходимо перейти с главной страницы на раздел «Создать счет». После этого нужно заполнить все обязательные поля формы, такие как название счета, сумма начального баланса и другие параметры. После нажатия кнопки «Создать», новый счет будет успешно создан и отображен в списке доступных счетов.

На странице «перевод» пользователь может отправить перевод на другой счет. Для этого ему надо указать с какого своего счета он хочет перевести и на какой, а также сумму перевода. Если пользователь укажет не свой счет ему выдаст ошибку.

Чтобы просмотреть историю операций, пользователю необходимо на главной странице нажать на кнопку «Получить историю». Затем, перейдя на страницу истории, нужно нажать на кнопку «Получить». После этого в таблице отобразятся все операции, связанные с пользователем, позволяя удобно просмотреть детали каждой транзакции и операции. Страница истории представлена на рисунке 7.3.

|  |
| --- |
|  |

Рисунок 7.3 –Страница истории операций пользователя

На данной странице пользователь может посмотреть историю своих операций. Отображаются не только переводы, но и другие операции, совершенные пользователем

7.1 Вывод по разделу

В разделе «Руководство пользователя» представлены ключевые шаги и инструкции, необходимые для эффективного взаимодействия с системой. Пользователи могут узнать, как создать новый счет, просмотреть историю операций и выполнять другие важные действия. Четкие указания и пошаговые инструкции помогают обеспечить простоту использования, минимизируя риск ошибок и улучшая общее взаимодействие с приложением. Таким образом, данное руководство служит важным инструментом для пользователей, позволяя им максимально эффективно использовать функциональность системы.

# Заключение

В ходе выполнения курсового проектирования была создана база данных для банка, которая включает в себя 6 таблиц, 3 роли, набор функций и процедур. Каждая роль имеет ограниченный доступ к функциям, что обеспечивает безопасное взаимодействие с базой данных, позволяя пользователям выполнять только необходимые действия.

Для поддержки импорта и экспорта данных в удобочитаемом формате разработаны 5 функций для импорта и 5 для экспорта. Это позволяет легко передавать данные между системами и обеспечивает гибкость в управлении информацией. Внедренные технологии резервного шифрования и маскировки данных обеспечивают безопасность пользователей, защищая их личную информацию.

Все процедуры и функции разработаны с учетом обработки ошибок, что позволяет пользователю сразу видеть все причины, по которым запросы не были выполнены. Это значительно упрощает процесс отладки и повышает удобство работы с базой данных. Структура базы данных также спроектирована таким образом, чтобы обеспечивать возможность добавления нового функционала без необходимости изменения существующего.

Кроме того, для повышения безопасности реализованы три роли, и добавлены проверки в функции. Например, перед выполнением перевода система проверяет, является ли пользователь владельцем счета. Это предотвращает несанкционированный доступ к функциям, доступным только администраторам.

В системе реализованы технологии шифрования и маскирования данных, которые обеспечивают надежную защиту конфиденциальной информации пользователей. Для шифрования используется алгоритм AES128, что гарантирует высокий уровень безопасности. Маскирование применяется для отображения данных в обезличенном виде, скрывая часть информации, например, номера телефонов или паспортных данных. Это позволяет минимизировать риски утечек и несанкционированного доступа, обеспечивая соответствие современным стандартам защиты данных.

Таким образом, можно заключить, что все поставленные цели были достигнуты, и база данных успешно выполняет возложенные на нее задачи.

# Список использованных источников

1. belarusbank [Электронный ресурс] – Режим доступа: https://belarusbank.by – Дата доступа 24.09.2024.
2. bsb [Электронный ресурс] – Режим доступа: https://www.bsb.by – Дата доступа 24.09.2024.
3. alfabank [Электронный ресурс] – https://www.alfabank.by – Дата доступа 25.09.2024.
4. Oracle DBMS\_CRYPTO [Электронный ресурс] – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle- database/21/arpls/DBMS\_CRYPTO.html– Дата доступа 10.11.2024.
5. Error Handling [Электронный ресурс] – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/21/lnpls/plsql-error-handling.html. – Дата доступа 24.10.2024.
6. JSON\_ TABLE (Transact-SQL) [Электронный ресурс]  – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/JSON\_TABLE.html
7. Как устроен AES \ habr.com [Электронный ресурс] – https://habr.com/ru/articles/112733/ – Дата доступа 12.11.2024.
8. PKCS [Электронный ресурс] – Режим доступа: – https://ru.wikipedia.org/wiki/PKCS– Дата доступа 13.11.2024. PL/SQL

# Приложение A

|  |
| --- |
|  |

Рисунок А.1 – Диаграмма вариантов использования

# Приложение Б

CREATE TABLE Role (

Role\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

name VARCHAR2(100) NOT NULL,

role\_level NUMBER NOT NULL

);

-- Создание таблицы Users

CREATE TABLE Users (

User\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

User\_role NUMBER REFERENCES Role(Role\_id),

login VARCHAR2(100) UNIQUE NOT NULL,

pass VARCHAR2(255) NOT NULL,

user\_name VARCHAR2(100) NOT NULL,

last\_name VARCHAR2(100) NOT NULL,

phone\_number VARCHAR2(100) UNIQUE NOT NULL,

pasport VARCHAR2(100) UNIQUE NOT NULL

);

--создание таблицы Currencies

CREATE TABLE Currencies (

Currency\_id VARCHAR2(4) PRIMARY KEY, -- ID валюты, максимум 4 символа

Currency\_name VARCHAR2(100) NOT NULL, -- Название валюты

Exchange\_rate NUMBER(10, 4) NOT NULL -- Цена валюты, может быть дробной

);

-- Создание таблицы Cash\_accounts

CREATE TABLE Cash\_accounts (

Cash\_id VARCHAR2(100) PRIMARY KEY,

Cash\_owner NUMBER REFERENCES Users(User\_id) NOT NULL,

balance VARCHAR2(100) NOT NULL,

Cash\_name VARCHAR2(100) NOT NULL,

IsBlocked NUMBER(1) DEFAULT 0,

Creation\_date DATE DEFAULT TRUNC(SYSDATE),

Currency\_id VARCHAR2(4) REFERENCES Currencies(Currency\_id)

);

-- Создание таблицы History

CREATE TABLE History (

Operation\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

Transactor NUMBER REFERENCES Users(User\_id),

cash\_from VARCHAR2(100) REFERENCES Cash\_accounts(Cash\_id),

cash\_to VARCHAR2(100) REFERENCES Cash\_accounts(Cash\_id),

operation VARCHAR2(100),

amount VARCHAR2(100),

operation\_description VARCHAR2(1000)

);

-- Создание таблицы Queue

CREATE TABLE Queue (

Operation\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

Transactor NUMBER REFERENCES Users(User\_id),

cash\_from VARCHAR2(100) REFERENCES Cash\_accounts(Cash\_id),

cash\_to VARCHAR2(100) REFERENCES Cash\_accounts(Cash\_id),

operation VARCHAR2(100),

amount VARCHAR2(100),

operation\_description VARCHAR2(1000)

);

Листинг Б.1 – SQL код создания таблиц

# Приложение B

CREATE OR REPLACE FUNCTION CreateCashAccount (

p\_executor\_id IN NUMBER,

p\_cash\_owner IN NUMBER,

p\_cash\_name IN VARCHAR2

) RETURN VARCHAR2 AS

v\_executor\_role\_level NUMBER;

v\_new\_cash\_id NUMBER;

v\_count NUMBER;

BEGIN

-- Проверка на NULL значения

IF p\_executor\_id IS NULL THEN

RETURN 'Ошибка: p\_executor\_id не может быть NULL.';

END IF;

IF p\_cash\_owner IS NULL THEN

RETURN 'Ошибка: p\_cash\_owner не может быть NULL.';

END IF;

IF p\_cash\_name IS NULL OR TRIM(p\_cash\_name) IS NULL THEN

RETURN 'Ошибка: p\_cash\_name не может быть NULL или пустым.';

END IF;

-- Получение уровня роли исполнителя

SELECT r.role\_level

INTO v\_executor\_role\_level

FROM UsersView u

JOIN Role r ON u.User\_role = r.Role\_id

WHERE u.User\_id = p\_executor\_id;

-- Проверка прав доступа

IF NOT (p\_executor\_id = p\_cash\_owner OR v\_executor\_role\_level < 1) THEN

RETURN 'Недостаточно прав.';

END IF;

-- Проверка, что владелец счета существует

SELECT COUNT(\*)

INTO v\_count

FROM UsersView

WHERE User\_id = p\_cash\_owner;

IF v\_count = 0 THEN

RETURN 'Пользователь с ID ' || p\_cash\_owner || ' не найден.';

END IF;

-- Генерация нового уникального Cash\_id

SELECT Cash\_id\_seq.NEXTVAL INTO v\_new\_cash\_id FROM DUAL;

-- Вставка новой записи в таблицу Cash\_accounts

INSERT INTO Cash\_accounts (Cash\_id, Cash\_owner, balance, Cash\_name)

VALUES (v\_new\_cash\_id, p\_cash\_owner, 0, p\_cash\_name);

-- Добавление записи в историю

AddToHistory(

p\_transactor => p\_cash\_owner,

p\_cash\_from => NULL,

p\_cash\_to => NULL,

p\_operation => 'CreateCashAccount',

p\_amount => NULL,

p\_operation\_description => 'Счет создан id: ' || v\_new\_cash\_id || ' name: ' || p\_cash\_name

);

COMMIT;

RETURN 'Счет успешно создан.';

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'Ошибка: Указанный пользователь не найден.';

WHEN OTHERS THEN

RETURN 'Произошла ошибка при добавлении счета: ' || SQLERRM;

END;

Листинг Г.1 – SQL код создания функции добавления нового счета

CREATE OR REPLACE FUNCTION UpdateAccountBalance(

p\_user\_id IN NUMBER,

p\_cash\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN VARCHAR2 AS

v\_role\_level NUMBER;

v\_current\_balance NUMBER;

v\_new\_balance NUMBER;

BEGIN

SELECT r.role\_level INTO v\_role\_level

FROM UsersView u JOIN Role r ON u.User\_role = r.Role\_id

WHERE u.User\_id = p\_user\_id;

IF v\_role\_level > 1 THEN

RETURN 'Ошибка: У вас недостаточно прав для выполнения этой операции.';

END IF;

SELECT balance

INTO v\_current\_balance

FROM uncrypt\_cash\_accounts

WHERE Cash\_id = p\_cash\_id;

v\_new\_balance := v\_current\_balance + p\_amount;

IF v\_new\_balance < 0 THEN

RETURN 'Ошибка: Баланс счета не может быть отрицательным. Текущий баланс: ' || v\_current\_balance;

END IF;

-- Обновление баланса счета

SetBalance(p\_cash\_id, v\_new\_balance);

AddToHistory(

p\_transactor => p\_user\_id,

p\_cash\_from => p\_cash\_id,

p\_cash\_to => NULL,

p\_operation => 'UpdateAccountBalance',

p\_amount => p\_amount,

p\_operation\_description => 'Изменение баланса счета ID ' || p\_cash\_id || ' на сумму ' || p\_amount || ' успешно обновлен. Новый баланс: ' || v\_new\_balance

);

RETURN 'Баланс счета ID ' || p\_cash\_id || ' успешно обновлен. Новый баланс: ' || v\_new\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'Ошибка: Счет или пользователь не найдены.';

WHEN OTHERS THEN

RETURN GetCustomErrorMessage(SQLERRM);END

Листинг Г.2 – SQL код создания функции обновления баланса счета