МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

Утверждаю

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Блинова

подпись инициалы и фамилия

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

**ЗАДАНИЕ**

**к курсовой работе**

**по дисциплине** «Базы данных»

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Курс: 3 Группа: 4

Студент: Точило Олег Вячеславович

**Тема: «**Реализация базы данных приложения учёта доходов, расходов и долгов с применением технологии In-Memory в БД»

**1. Срок сдачи студентом законченной работы**: «11» декабря 2023 г.

**2. Исходные данные к курсовой работе:**

**2.1**. Функционально должны быть выполнены следующие задачи:

* управление пользователями и паролями (добавление, удаление, изменение);
* управление ролями (администратор, обычный пользователь, гость);
* управление должниками (добавление, удаление, изменение);
* управление транзакциями (добавление, удаление, изменение);
* анализ доходов и расходов (объём долга каждого пользователя перед каждым другим пользователем, общий долг каждого пользователя, долг за указанный период);

**2.2. Требования.**

* База данных должна быть реализована в СУБД Oracle.
* Доступ к данным должен осуществляться только через соответствующие процедуры.
* Количество объектов БД (таблиц, представлений, индексов, пользователей и пр.) регламентируется задачей.
* Должен быть проведен импорт данных из JSON-файлов, экспорт данных в формат JSON.
* Необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице.
* Применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.
* Листинги проекта должны содержать комментарии.

**3. Содержание расчетно-пояснительной записки**

* Введение
* Постановка задачи
* Проектирование базы данных.
* Разработка объектов базы данных
* Описание процедур импорта и экспорта
* Тестирование производительности
* Описание технологии и ее применения в базе данных
* Краткое описание приложения для демонстрации
* Руководство пользователя
* Заключение
* Список используемых источников
* Приложения

**4. Форма представления, выполненного курсовой работы:**

* Пояснительная записка оформляется в MS Word.
* Оформление пояснительной записки должно соответствовать требованиям к оформлению пояснительной записки для курсовых работ.
* Листинги всех скриптов представляются в приложении.
* К записке необходимо приложить DVD-диск, который должен содержать пояснительную записку, листинги и файлы базы данных.

**Календарный план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсовой работы | Срок выполнения этапов работы | Примечание |
| 1 | Введение | 19.09.2023 |  |
| 2 | Аналитический обзор литературы по теме работы | 26.09.2023 |  |
| 3 | Изучение требований, определение вариантов использования | 03.10.2023 |  |
| 4 | Анализ и проектирование модели базы данных. Описание информационных объектов и ограничений целостности. | 10.10.2023 |  |
| 5 | Создание необходимых объектов | 24.10.2023 |  |
| 6 | Импорт и экспорт данных | 07.11.2023 |  |
| 7 | Описание используемой технологии | 14.11.2023 |  |
| 8 | Тестирование производительности | 21.11.2023 |  |
| 9 | Оформление пояснительной записки | 05.12.2023 |  |
| 10 | Сдача курсовой работы | 11.12.2023 |  |

**5. Дата выдачи задания** «07» сентября 2023 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.А. Нистюк

(подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата и подпись студента)

**Содержание**

[Введение 3](#_Toc153522426)

[1 Постановка задачи 4](#_Toc153522427)

[1.1 Обзор аналогичных решений 4](#_Toc153522428)

[1.2 Спецификация требований 4](#_Toc153522429)

[2 Проектирование базы данных 6](#_Toc153522430)

[3 Разработка объектов базы данных 9](#_Toc153522431)

[3.1 Разработка таблиц 9](#_Toc153522432)

[3.2 Разработка хранимых процедур 9](#_Toc153522433)

[3.3 Создание индексов 11](#_Toc153522434)

[3.4 Создание пользователей 11](#_Toc153522435)

[4 Описание процедур импорта и экспорта 13](#_Toc153522436)

[5 Тестирование производительности 14](#_Toc153522437)

[6 Описание технологии и её применения в базе данных 15](#_Toc153522438)

[7 Краткое описание приложения для демонстрации 17](#_Toc153522439)

[8 Руководство пользователя 18](#_Toc153522440)

[Заключение 28](#_Toc153522441)

[Список используемых источников 29](#_Toc153522442)

[Приложение А 30](#_Toc153522443)

[Приложение Б 32](#_Toc153522444)

[Приложение В 42](#_Toc153522445)

[Приложение Г 43](#_Toc153522446)

Введение

В современном мире поиск и предложение работы становятся все более актуальными, поскольку технологические инновации меняют традиционные способы трудоустройства. Сегодня множество людей стремятся найти и предложить работу через специализированные приложения, которые облегчают этот процесс и создают связь между работодателями и соискателями.

Разработанное приложение предоставляет удобную и эффективную платформу для взаимодействия между работодателями и соискателями. Оно предлагает. Приложение позволяет соискателям создавать профили и свои резюме и представлять свои навыки, образование и опыт работы. Они могут просматривать доступные вакансии и отправлять свои заявки на эти вакансии. Работодатели могут размещать вакансии и просматривать профили и резюме откликнувшихся кандидатов. Помимо базовых функций поиска и предложения работы, приложение предоставляет возможность оценивать и оставлять отзывы о работодателях и соискателях.

В приложении разграничены возможности гостей, соискателей, работодателей и администраторов. Так, гости могут только просматривать публичные вакансии, страницы компаний и отзывы на них; соискателям доступно изменение своих персональных данных, изменение списка резюме и возможность отклика на доступные вакансии; работодатели могут изменять список своих вакансий, просматривать список откликов, принимать и отклонять их; работодателям и соискателям доступна возможность оставлять отзывы друг на друга, но только при условии, что соискатель откликнулся на вакансию работодателя, и последний принял отклик; администраторам доступны возможности по удалению учётных записей, подтверждению статусов компаний и удалению комментариев.

В ходе выполнения курсового проектирования будут решены следующие задачи:

– анализ литературы по теме работы;

– изучение требований и определение вариантов использования;

– анализ и проектирование модели базы данных, описание информационных объектов и ограничений целостности;

– создание необходимых объектов;

– реализация импорта и экспорта данных;

– описание используемой технологии;

– тестирование производительности;

– создание руководства пользователя.

Целями базы данных являются хранение, организация и обеспечение безопасности данных, а задачами являются структуризация данных в таблицы и использование индексов для ускоренного доступа к данным.

1 Постановка задачи

1.1 Обзор аналогичных решений

В качестве первого аналогичного решения была рассмотрена платформа Fiverr, на которой предлагаются услуги фрилансеров со всего мира. Она специализируется на небольших задачах, а не на поиске постоянного места работы. Пользователи могут найти исполнителей по различным областям, таким как дизайн, письменное творчество, программирование и многое другое. Основные особенности Fiverr включают простой процесс заказа, возможность просмотра портфолио и отзывов о соискателях и ориентация на заказчиков, то есть соискатели должны размещать свои резюме, а заказчики должны выбирать из имеющихся на данный момент соискателей.

В качестве второго аналогичного решения была рассмотрена платформа Upwork, предлагающая более широкий спектр услуг и проектов для фрилансеров. Платформа предоставляет доступ к множеству заказов в различных областях, включая разработку ПО, маркетинг и так далее. Upwork обеспечивает более сложный процесс поиска и найма, включая возможность просмотра профилей, проведения собеседований и оценки работы. Также на платформе присутствует система обратной связи и защиты платежей. К преимуществам платформы относится возможность размещения как вакансий, так и публичных резюме, а также возможность предлагать работу как на основе проектной занятости, так и на полный рабочий день.

В качестве последнего аналогичного решения была рассмотрена платформа HeadHunter – платформа с мощных функционалом по поиску работы и работников в странах СНГ. Она предлагает широкий функционал по размещению вакансий и резюме, а также обширные возможности фильтрации оп множеству критериев. Для связи соискателя и работодателя предусмотрен чат.

Недостатком всех рассмотренных аналогов является невозможность узнать, была ли изменена вакансия, без перезагрузки страницы. В худшем случае соискатель может откликнуться не на ту вакансию, на которую собирался, так как работодатель изменил её, пока соискатель находился на её странице.

На основе рассмотренных аналогичных решений были сформированы следующие требования к программному продукту:

– возможность размещать вакансии;

– возможность оценивать соискателей и работодателей;

– автоматическое обновление данных о просматриваемой вакансии в случае её изменения.

1.2 Спецификация требований

База данных должна быть реализована в СУБД PostgreSQL.

Всем пользователям, включая гостей, должна быть доступна возможность просматривать публичные вакансии, страницы компаний и отзывы о них и жаловаться на отзывы.

Соискателю должна быть доступна возможность добавлять резюме, изменять и удалять их, откликаться и удалять свои отклики на публичные вакансии и оставлять отзывы на работодателей, которые приняли его отклик.

Работодателю должна быть доступна возможность добавлять вакансии, изменять и удалять их, получать список откликов на свои вакансии, принимать и отклонять их, просматривать данные о соискателях, откликнувшихся на его вакансии, просматривать отзывы о них и оставлять отзывы о тех соискателях, чьи отклики он принял и запрашивать подтверждение компании у администратора.

Работодателям и соискателям должна быть доступна возможность редактировать данные о себе, менять пароль, удалять свои отзывы и запрашивать удаление своей учётной записи.

Администратору должна быть доступна возможность блокировать возможность пользователя оставлять отзывы о другом пользователе, удалять комментарии и учётные записи и подтверждать компании. Диаграмма вариантов использования представлена на рисунке 1.1.

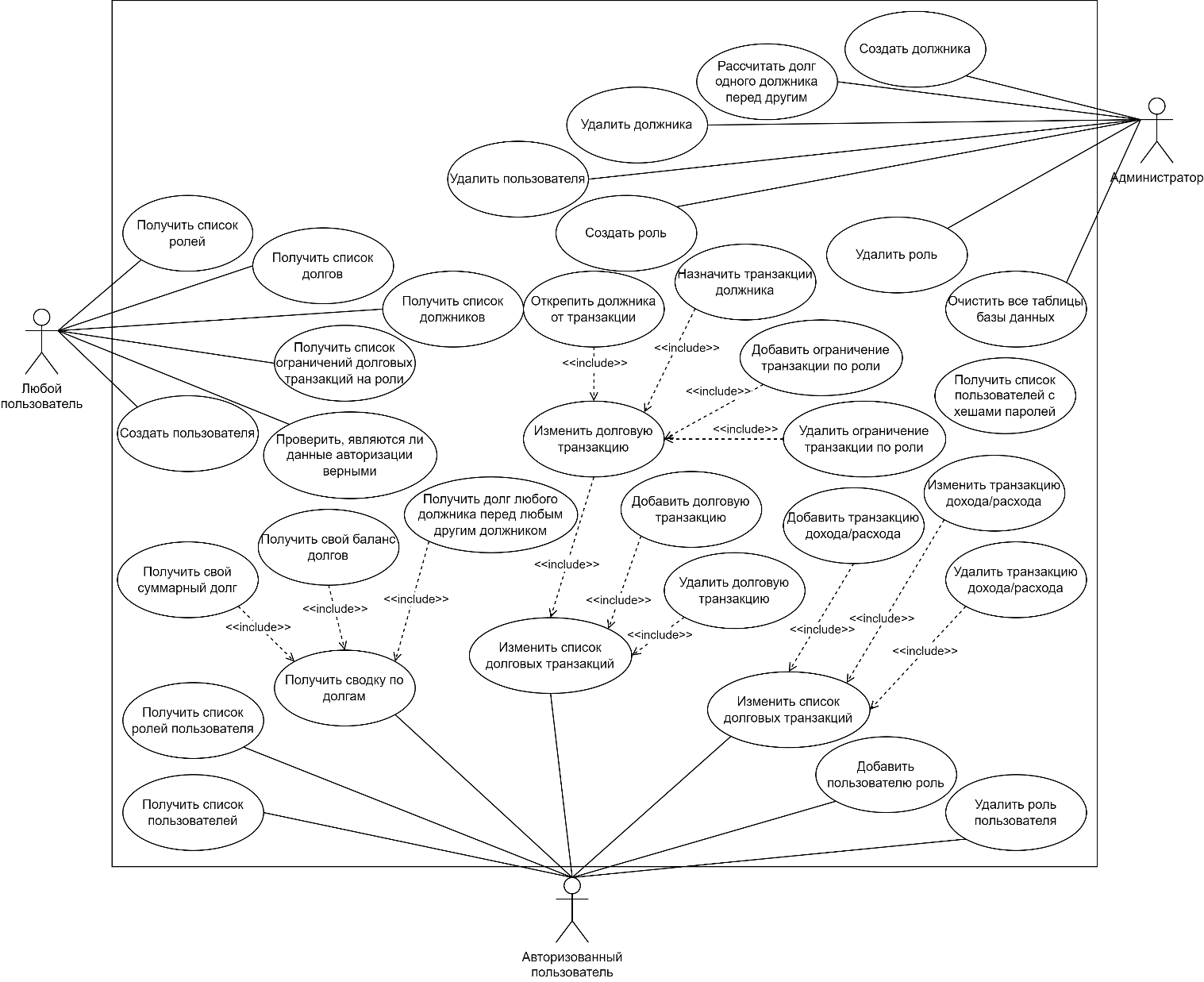


Рисунок 1.1 – Диаграмма вариантов использования базы данных

В категорию «Любые пользователи» входят гости, обычные пользователи и администраторы. В категорию «Авторизованные пользователи» входят обычные пользователи и администраторы.

2 Проектирование базы данных

Логическая схема базы данных представлена на рисунке 2.1.

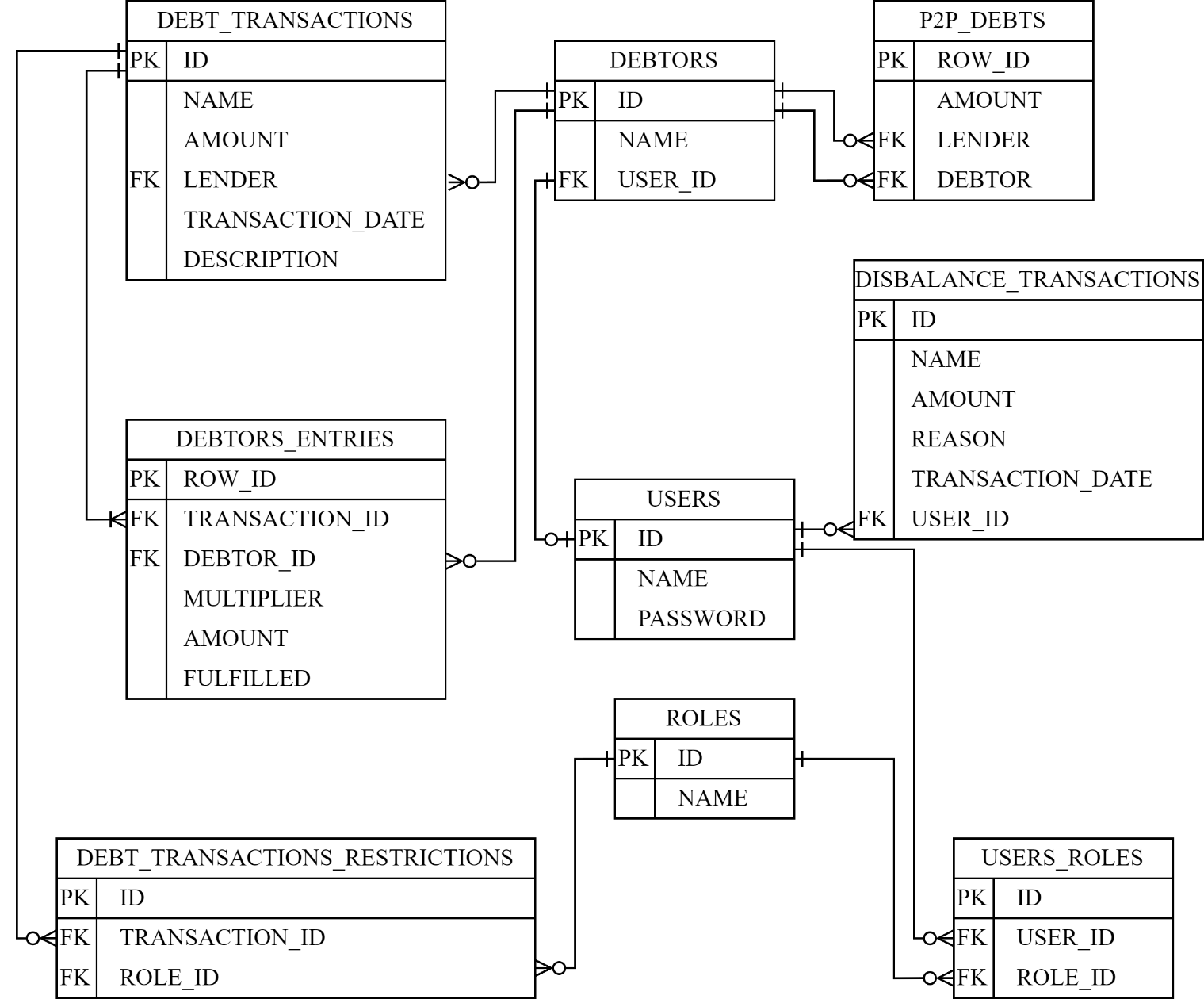


Рисунок 2.1 – Логическая схема базы данных

База данных содержит девять таблиц, хранящих информацию о ролях, пользователях, должниках, транзакциях и совокупных долгах. Листинг создания таблиц представлен в приложении А.

Таблица ROLES хранит информацию о ролях, доступных пользователям. Описание её столбцов представлено в таблице 2.1.

Таблица 2.1 – Описание таблицы ROLES

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | идентификатор роли, первичный ключ |
| name | nvarchar2(50) | название роли |

Таблица USERS хранит информацию о зарегистрированных пользователях. Описание её столбцов представлено в таблице 2.2.

Таблица 2.2 – Описание таблицы USERS

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | идентификатор пользователя, первичный ключ |
| name | nvarchar2(50) | имя пользователя |
| password\_hash | nvarchar2(256) | хэш пароля пользователя |

Таблица DEBTORS хранит информацию о должниках. Описание её столбцов представлено в таблице 2.3.

Таблица 2.3 – Описание таблицы DEBTORS

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | идентификатор должника, первичный ключ |
| name | nvarchar2(50) | имя должника |
| user\_id | int | идентификатор пользователя, ассоциированного с этим должником, внешний ключ |

Таблица DEBT\_TRANSACTIONS хранит информацию о долговых транзакциях. Описание её столбцов представлено в таблице 2.4.

Таблица 2.4 – Описание таблицы DEBT\_TRANSACTIONS

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | идентификатор транзакции, первичный ключ |
| name | nvarchar2(50) | название транзакции |
| amount | int | количество потраченных денег |
| lender | int | какой пользователь совершил транзакцию, внешний ключ |
| transaction\_date | date | дата транзакции |
| description | nvarchar2(256) | описание транзакции |

Таблица DEBTORS\_ENTRIES хранит информацию о долях должников в долговых транзакциях. Описание её столбцов представлено в таблице 2.5.

Таблица 2.5 – Описание таблицы DEBTORS\_ENTRIES

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| row\_id | int | идентификатор строки, первичный ключ |
| transaction\_id | int | идентификатор транзакции |
| debtor\_id | int | идентификатор должника |
| multiplier | float | фактор доли должника в транзакции |
| fulfilled | char(1) | показатель того, вернул ли должник свою долю в данной транзакции |

Таблица DISBALANCE\_TRANSACTIONS хранит информацию о доходах и расходах пользователей. Описание её столбцов представлено в таблице 2.6.

Таблица 2.6 – Описание таблицы DISBALANCE\_TRANSACTIONS

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | идентификатор транзакции, первичный ключ |
| name | nvarchar2(50) | идентификатор транзакции |
| amount | int | сумма транзакции |
| reason | nvarchar2(50) | причина совершения транзакции |
| transaction\_date | date | дата транзакции |
| user\_id | int | идентификатор пользователя, которому принадлежит транзакция, внешний ключ |

Таблица P2P\_DEBTS хранит информацию о долгах каждого должника перед каждым другим должником. Описание её столбцов представлено в таблице 2.7.

Таблица 2.7 – Описание таблицы P2P\_DEBTS

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| row\_id | int | идентификатор строки, первичный ключ |
| lender | int | идентификатор кредитора |
| debtor | int | идентификатор заёмщика |
| amount | int | сумма долга |

Таблица DEBT\_TRANSACTIONS\_RESTRICTIONS хранит информацию о суммарных долгах каждого должника. Описание её столбцов представлено в таблице 2.8.

Таблица 2.8 – Описание таблицы DEBT\_TRANSACTIONS \_RESTRICTIONS

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | идентификатор строки, первичный ключ |
| transaction\_id | int | идентификатор транзакции |
| role\_id | int | идентификатор роли, которой разрешено просматривать транзакцию |

Таблица USERS\_ROLES хранит информацию о ролях, назначенных себе пользователями. Описание её столбцов представлено в таблице 2.9.

Таблица 2.9 – Описание таблицы USERS\_ROLES

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | int | Идентификатор строки |
| user\_id | int | Идентификатор пользователя, внешний ключ |
| role\_id | int | Идентификатор роли, внешний ключ |

3 Разработка объектов базы данных

3.1 Разработка таблиц

Все таблицы создаются в табличном пространстве администратора подключаемой базы данных Goblin\_perm\_tbs.

Каждая таблица имеет столбец идентификатора строки типа int. При добавлении строки в таблицу её автоматически присваивается идентификатор при помощи отдельной последовательности для каждой таблицы. Для хранения строковых значений используется тип данных nvarchar2. Для хранения дат – date. Для хранения денежных значений – int. Для хранения булевых переменных используется тип char с допустимыми значениями «Y» и «N». Описание столбцов таблиц приведено в разделе 2.

3.2 Разработка хранимых процедур

Описание используемых хранимых процедур представлено в таблице 3.1.

Таблица 3.1 – Описание используемых процедур

|  |  |
| --- | --- |
| Название процедуры | Описание процедуры |
| Add\_debt\_transaction | Добавляет новую строку в Debt\_transactions |
| Add\_debtor | Добавляет новую строку в Debtors и новые строки в P2P\_debts для каждого уже существующего должника |
| Add\_debtor\_entry | Добавляет новую строку в Debtors\_entries и вызывает Calculate\_p2p\_debts для должника и кредитора данной транзакции |
| Add\_disbalance\_transaction | Добавляет новую строку в Disbalance\_transactions |
| Add\_role | Добавляет новую строку в Roles |
| Add\_transaction\_restriction | Добавляет новую строку в Debt\_transactions\_restrictions |
| Add\_user | Добавляет новую строку в Users и Debtors. Если должник с таким именем уже существует, задаёт его user\_id равным id созданного пользователя |
| Add\_user\_role | Добавляет новую строку в Users\_roles |
| Calculate\_p2p\_debts | Обновляет значения в строках таблицы P2P\_debts, в которых lender равен параметру p\_lender, а debtor равен параметру p\_debtor, и наоборот, на основе существующих транзакций в Debt\_transactions и соответствующих этим транзакциям строкам в таблице Debtors\_entries |
| Delete\_debt\_transaction | Удаляет строку из Debt\_transactions по идентификатору |
| Delete\_debtor | Удаляет строку из Debtors по идентификатору |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Название процедуры | Описание процедуры |
| Delete\_debtor\_entry | Удаляет строку из Debtors\_entries по идентификатору и вызывает Calculate\_p2p\_debts для данного должника и кредитора транзакции |
| Delete\_disbalance\_transaction | Удаляет строку из Disbalance\_transactions по идентификатору |
| Delete\_role | Удаляет роль по её идентификатору |
| Delete\_transactions\_restriction | Удаляет строку из Debt\_transactions\_restrictions |
| Delete\_user | Удаляет пользователя по идентификатору |
| Delete\_user\_role | Удаляет строку из Users\_roles по идентификаторам пользователя и роли |
| Get\_debt\_transactions | Открывает курсор, переданный в качестве параметра, запросом к Debt\_transactions с выбором всех столбцов |
| Get\_debtors | Открывает курсор, переданный в качестве параметра, запросом к Debtors с выбором всех столбцов |
| Get\_debtors\_entries | Открывает курсор, переданный в качестве параметра, запросом к Debtors\_entries с выбором всех столбцов по идентификатору транзакции |
| Get\_disbalance\_transactions | Открывает курсор, переданный в качестве параметра, запросом к Disbalance\_transactions с выбором всех столбцов по переданным в качестве параметров фильтрам |
| Get\_p2p\_debts\_by\_person\_id | Открывает курсор, переданный в качестве параметра, запросом к P2P\_debts, если даты, переданные в качестве параметров, не пустые. Иначе открывает курсор запросом к объединению таблиц Debt\_transactions и Debtors\_entries с фильтром по дате и должнику, если параметр is\_debtor равен ‘Y’, иначе по дате и кредитору |
| Get\_roles | Открывает курсор, переданный в качестве параметра, запросом к Roles с выбором всех столбцов |
| Get\_total\_balance | Возвращает с выходным параметром разницу суммы одолжений и суммы долгов должника за определённый период |
| Get\_total\_debt | Возвращает с выходным параметром сумму долгов должника за определённый период |

Окончание таблицы 3.1

|  |  |
| --- | --- |
| Название процедуры | Описание процедуры |
| Get\_transaction\_restrictions | Открывает курсор, переданный в качестве параметра, запросом к Debt\_transactions\_restrictions с выбором role\_id по идентификатору транзакции |
| Get\_user\_roles | Открывает курсор, переданный в качестве параметра, запросом к Users\_roles с выбором role\_id по идентификатору пользователя |
| Get\_users | Открывает курсор, переданный в качестве параметра, запросом к Users с выбором идентификатора и имени |
| Get\_users\_for\_export | Открывает курсор, переданный в качестве параметра, запросом к Users с выбором всех столбцов, включая хэш пароля |
| Is\_login\_correct | В out параметре возвращает 1, если пользователь с переданными в качестве параметров именем и хэшем пароля существует, иначе возвращает -1 |
| Update\_debt\_transaction | Устанавливает новые данные, переданные в качестве параметров, в строку таблицы Debt\_transactions по её идентификатору |
| Update\_disbalance\_transaction | Устанавливает новые данные, переданные в качестве параметров, в строку таблицы Disbalance\_transactions по её идентификатору |
| Wipe\_database | Удаляет все строки из всех таблиц для подготовки к импорту данных |

Листинг создания процедур представлен в приложении Б.

3.3 Создание индексов

Для ускоренного доступа к данным были созданы индексы для таблиц Users\_roles по столбцу user\_id, Debt\_transactions по столбцу lender, Disbalance\_transactions по столбцам user\_id, transaction\_date, amount и reason, Debtors\_entries по столбцу debtor\_id и P2P-debts по столбцам debtor и lender.

3.4 Создание пользователей

Для взаимодействия приложения с базой данных созданы три пользователя: Guest, Common\_user и Admin. Им предоставлены права на вызов различных процедур.

Процедуры, которые разрешено вызывать гостю:

– Get\_roles;

– Get\_debt\_transactions;

– Get\_debtors;

– Get\_debtors\_entries;

– Get\_transaction\_restrictions;

– Is\_login\_correct;

– Add\_user.

Процедуры, которые разрешено вызывать обычному пользователю:

– Get\_total\_balance;

– Get\_disbalance\_transactions;

– Get\_total\_debt;

– Get\_p2p\_debts\_by\_person\_id;

– Add\_debt\_transaction;

– Add\_debtor\_entry;

– Add\_disbalance\_transaction;

– Add\_transaction\_restriction;

– Add\_user\_role;

– Delete\_debt\_transaction;

– Delete\_debtor\_entry;

– Delete\_disbalance\_transaction;

– Delete\_transaction\_restriction;

– Delete\_user\_role;

– Get\_users;

– Get\_user\_roles;

– Update\_disbalance\_transaction;

– Update\_debt\_transaction.

Процедуры, которые разрешено вызывать администратору:

– Add\_debtor;

– Add\_role;

– Delete\_debtor;

– Delete\_role;

– Delete\_user;

– Calculate\_p2p\_debts;

– Wipe\_database;

– Get\_users\_for\_export.

Процедуры, которые разрешено вызывать гостю, разрешено вызывать обычному пользователю. Процедуры, которые разрешено вызывать обычному пользователю, разрешено вызывать администратору. Листинг создания пользователей и выдачи им необходимых привилегий представлен в приложении В.

4 Описание процедур импорта и экспорта

Для импорта и экспорта данных были созданы две дополнительные процедуры: Wipe\_database, удаляющая содержимое всех таблиц, и Get\_users\_for\_export, возвращающая список всех пользователей, включая их идентификаторы, имена и хэши паролей.

Первая процедура позволяет избавиться от необходимости проверки на наличие идентичных данных в таблице и вставки импортируемых строк в таблицы с новыми идентификаторами, так как в противном случае было бы необходимо учитывать новые идентификаторы для вставки других строк, ссылающихся на строки с изменившимися идентификаторами.

Вторая процедура отличается от Get\_users тем, что возвращает также пароли пользователей, которые не должны считываться из базы данных при обычном использовании приложения.

Так как данные процедуры сильно изменяют данные в базе данных или предоставляют доступ к конфиденциальной информации, разрешение на их вызов предоставлено исключительно администратору.

На Django-сервере для импорта и экспорта применяются функции, представленные в листинге в приложении Г.

5 Тестирование производительности

Для тестирования производительности было создано 1101 новых пользователей. При этом было создано столько же должников, что в свою очередь добавило 1215504 записи в таблицу P2P\_debts. Тестирование производилось при помощи процедуры Get\_p2p\_debts\_by\_person\_id. Запрос, используемый для тестирования производительности, представлен в листинге 5.1.

|  |
| --- |
| declare  p\_cursor sys\_refcursor;  cursor v\_cursor is select lender, debtor from P2P\_DEBTS;  v\_row v\_cursor%rowtype; begin  GET\_P2P\_DEBTS\_BY\_PERSON\_ID(p\_cursor, 74, 'Y', '', '');  fetch p\_cursor into v\_row;  while p\_cursor%found loop  DBMS\_OUTPUT.PUT\_LINE(v\_row.lender||' '||v\_row.debtor);  fetch p\_cursor into v\_row;  end loop;  close p\_cursor; end; |

Листинг 5.1 – Запрос для тестирования производительности базы данных

До перестройки индекса P2P\_debts\_index её вызов совершался за 303 миллисекунды. После перестройки вызов занял 16 миллисекунд.

6 Описание технологии и её применения в базе данных

Для ускорения доступа к данным была использована технология In-memory. Данная технология позволяет пользоваться некоторыми преимуществами колоночных баз данных путём хранения значений выбранных столбцов таблиц в оперативной памяти, а точнее, в области In-memory системной глобальной области памяти СУБД, что значительно ускоряет доступ к их значениям. Особенно это полезно в случаях, когда столбцы используются при группировке, сортировке и фильтрации.

Колоночные базы данных хранят каждый атрибут транзакции или записи в отдельной структуре столбцов. Колоночный формат идеально подходит для аналитики, поскольку позволяет быстрее получать данные, когда выбрано всего несколько столбцов, но запрос обращается к большой части набора данных. но запрос обращается к большой части набора данных.

Формат строк чрезвычайно эффективен для обработки DML-операций, поскольку он манипулирует всей записью за одну операцию (т. е. вставляет, обновляет или удаляет строку). Колоночный формат не так эффективен для обработки DML, поскольку для вставки или удаления одной записи в колоночном формате должны быть изменены все колоночные структуры таблицы. Это может потребовать одной или нескольких операций ввода-вывода для каждого столбца. СУБД, поддерживающие только один из форматов, в качестве компромисса выбирают либо неоптимальную обработку транзакций в реальном времени, либо неоптимальную производительность аналитических запросов.

Oracle Database In-Memory использует лучшие решения из двух подходов: двухформатную архитектуру, позволяющую данным существовать одновременно в формате строк в памяти (буферном кэше) и в новом формате столбцов в памяти. Демонстрация хранения данных в двух форматах одновременно представлена на рисунке 6.1.

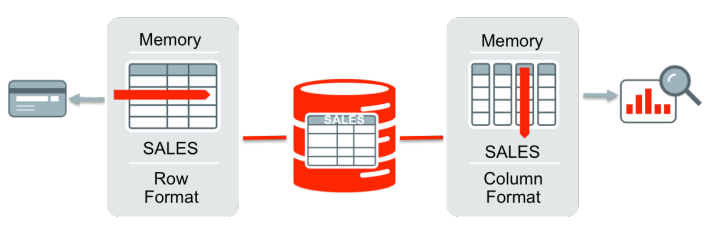


Рисунок 6.1 – Демонстрация хранения данных в двух форматах

Использование данной технологии оправданно при объединении таблиц по столбцу, для которого включено хранение в оперативной памяти. Использование технологии In-memory увеличивает потребление оперативной памяти на 20 процентов, но позволяет значительно ускорить аналитические запросы. Пример использования технологии In-memory при создании таблицы DEBTORS\_ENTRIES представлен в листинге 6.1.

|  |
| --- |
| create table DEBTORS\_ENTRIES (  row\_id int default DEBTORS\_ENTRIES\_ID\_SEQ.nextval primary key,  transaction\_id int not null constraint GA\_DEBTORS\_ENTRIES\_TRANSACTIONS\_ID\_FK  references DEBT\_TRANSACTIONS(id) on delete cascade,  debtor\_id int not null constraint GA\_DEBTORS\_ID\_FK references DEBTORS(id) on delete cascade,  multiplier float not null,  amount int not null,  fulfilled char(1) check ( fulfilled in ('Y', 'N') ) not null  );  alter table DEBTORS\_ENTRIES inmemory no inmemory(row\_id, multiplier, amount, fulfilled); |

Листинг 6.1 – Использование технологии In-memory при создании таблицы DEBTORS\_ENTRIES

Технология In-memory была использована для таблиц Users\_roles и Debtors полностью, так как запросы к этим таблицам выполняются часто, и к столбцам transaction\_id и debtor\_id таблицы Debtors\_entries, так как эти столбцы используются для фильтрации, а значения прочих столбцов считываются значительно реже.

7 Краткое описание приложения для демонстрации

Для демонстрации работоспособности приложения можно использовать следующий сценарий: пользователь в веб-браузере переходит по адресу http://127.0.0.1:8000/GoblinApp/debt\_transactions. На странице находятся публичные долговые транзакции. Пользователь нажимает на кнопку «Войти», вводит данные учётной записи и авторизуется в системе. Теперь на странице отображаются все транзакции, доступные пользователю в соответствии с назначенными ему ролями. Пользователь нажимает на кнопку «Добавить транзакцию», вводит в появившееся окно необходимые данные, устанавливает должников, их коэффициенты и ограничения видимости транзакции по ролям и сохраняет транзакцию. Страница обновляется, и новая транзакция появляется на ней. Пользователь переходит на страницу доходов и расходов по ссылке в боковом меню и просматривает свои доходы и расходы. Он добавляет новую транзакцию путём ввода данных в окно, открываемое по нажатию на кнопку «Добавить транзакцию», а затем фильтрует по нужной ему категории. Страница обновляется, и на ней остаются только отфильтрованные транзакции. Пользователь переходит на страницу сводки по ссылке в боковом меню, просматривает свои долги и одолжения, а также данные о своих доходах и расходах за нужный период. После этого пользователь выходит из учётной записи.

8 Руководство пользователя

При первом открытии сайта пользователь видит страницу долговых транзакций, изображённую на рисунке 4.1. При попытке перейти на любую другую страницу по прямой ссылке пользователь будет перенаправлен обратно на станицу долговых транзакций.

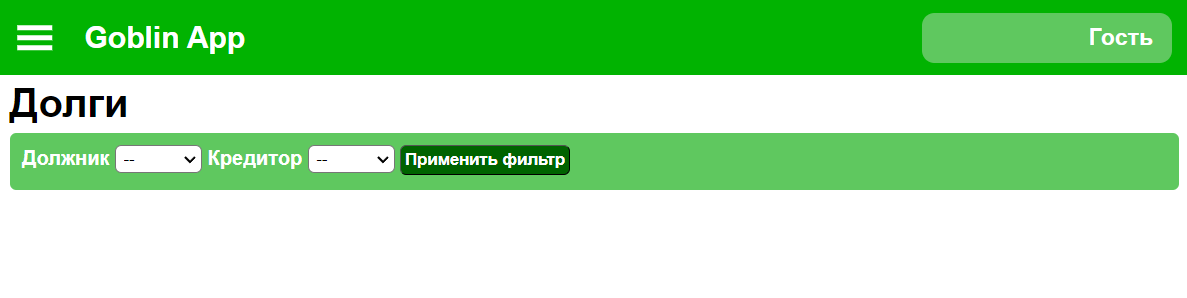


Рисунок 8.1 – Страница долговых транзакций до авторизации

Гостю доступна фильтрация долговых транзакций, на которые не было наложено ограничение видимости для определённых ролей, по должнику и кредитору. Для того, чтобы отфильтровать транзакции, пользователь должен выбрать нужные значения в выпадающих списках возле надписей «Должник» и «Заголовок» и нажать на кнопку «Применить фильтр». Страница обновится, и на ней будут отображены отфильтрованные транзакции. Форма фильтрации представлена на рисунке 8.1.

Для того, чтобы авторизоваться или зарегистрироваться, пользователь должен навести курсор мыши на элемент с надписью «Гость» и нажать на появившуюся кнопку «Войти» или «Зарегистрироваться» соответственно. Внешний вид кнопок при наведении курсора мыши представлен на рисунке 8.2.

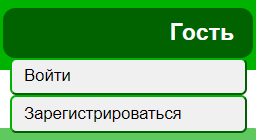


Рисунок 8.2 – Кнопки авторизации и регистрации

При нажатии на одну из вышеупомянутых кнопок появится форма авторизации или регистрации, в которую пользователь должен ввести своё имя пользователя и пароль. В случае, если пользователь пытается авторизоваться с неверными данными, появится окно об ошибке, представленное на рисунке 8.3.

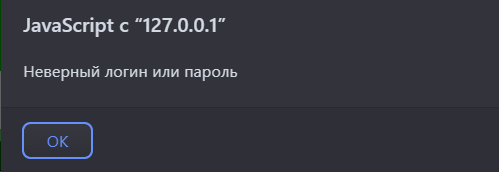


Рисунок 8.3 – Окно с сообщением о неверно введённых данных для авторизации

В случае, если пользователь пытается зарегистрироваться под существующим именем, появится окно, сообщающее об этом, изображённое на рисунке 8.4.

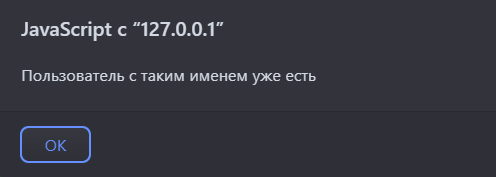


Рисунок 8.4 – Окно с сообщением об уже существующем пользователе с введённым именем

После появления формы авторизации или регистрации прочие действия на сайте блокируются. Чтобы скрыть форму и получить доступ к другим действиям, доступным гостю, пользователь должен нажать на кнопку «Отмена».

При попытке зарегистрироваться под именем уже существующего должника появится окно, представленное на рисунке 8.5, уведомляющее о том, что при подтверждении созданная учётная запись будет связана с уже существующим должником.

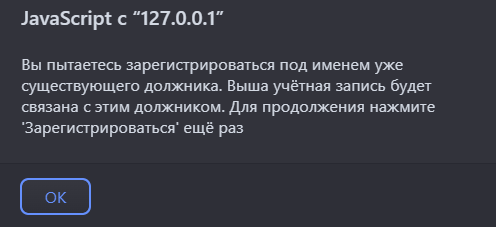


Рисунок 8.5 – Окно предупреждения о существовании должника с указанным именем

При успешной регистрации пользователь автоматически авторизуется. При успешной авторизации появляется окно, представленное на рисунке 8.6, сообщающее об этом, и страница обновляется. При этом кнопки «Войти» и «Зарегистрироваться» заменяются кнопками «Выйти» и «Управление ролями».

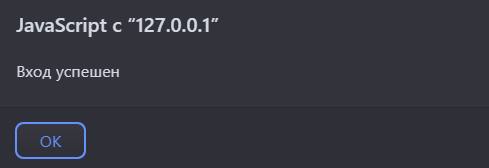


Рисунок 8.6 – Окно, информирующее об успешной авторизации

Обычному пользователю доступен выход из учётной записи и управление своими ролями. Назначения себе новых ролей и снятия имеющихся пользователь должен нажать на кнопку «Управление ролями». Появится соответствующее окно, и все прочие действия на сайте будут заблокированы. Окно управления ролями представлено на рисунке 8.7.

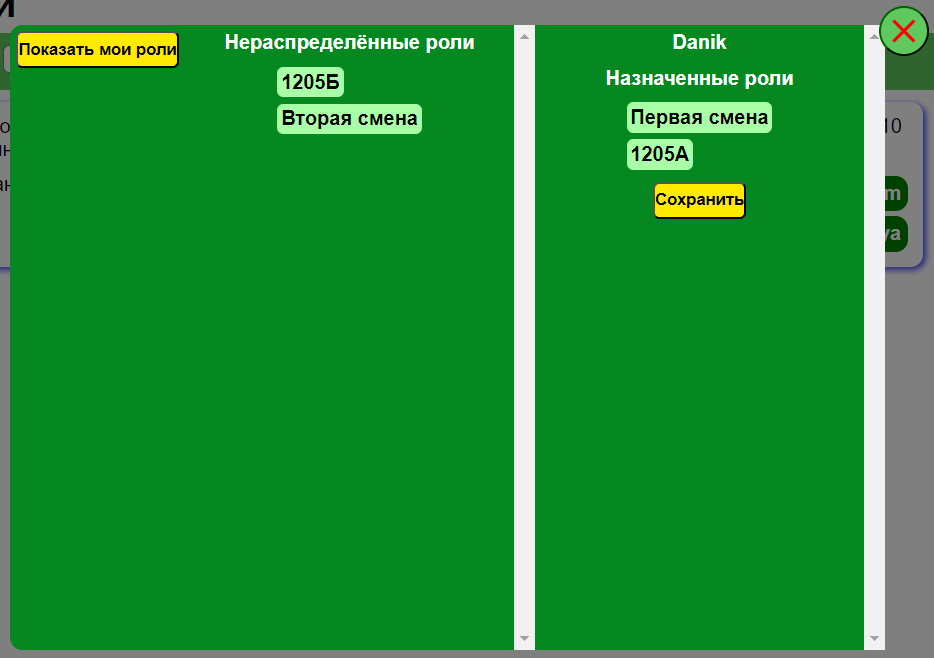


Рисунок 8.7 – Окно управления ролями обычного пользователя

Для того, чтобы назначить себе новую роль или снять с себя существующую, пользователь должен нажать на элемент роли. Назначенная роль переместится в блок нераспределённых ролей, а роль, которой у пользователя не было, переместится в блок назначенных ролей. Для сохранения изменений пользователь должен нажать на кнопку «Сохранить». Для закрытия окна пользователь должен нажать на кнопку в правом верхнем углу окна. При этом, если пользователь не изменял свои роли, то страница не перезагрузится, в противном случае произойдёт перезагрузка для обновления списка транзакций, доступных пользователю.

На странице долговых транзакций обычному пользователю доступно создание новой долговой транзакции. Для этого пользователь должен нажать на кнопку «Добавить транзакцию». Появится окно, представленное на рисунке 8.8, в которое пользователь должен ввести нужные данные и нажать на кнопку «Создать» для сохранения транзакции или на кнопку «Отмена» для сокрытия окна создания транзакции.

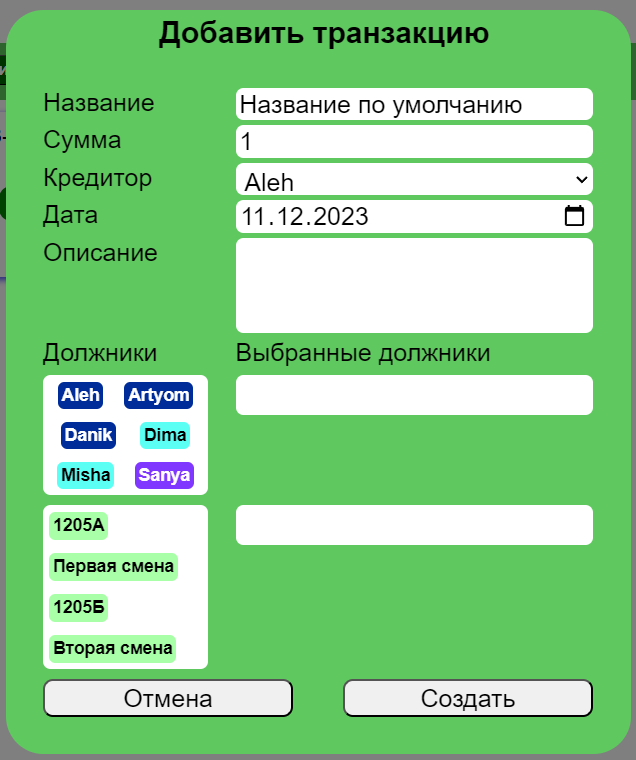


Рисунок 8.8 – Окно создания долговой транзакции

Чтобы прикрепить должника к транзакции, пользователь должен нажать на его элемент в блоке под надписью «Должники». Элемент неприкреплённого должника исчезнет из блока и появится в блоке выбранных должников. Изображение блока выбранных должников с выбранными должниками представлено на рисунке 8.9.

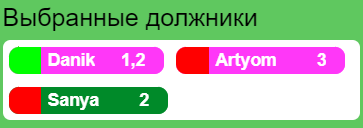


Рисунок 8.9 – Блок выбранных должников транзакции с прикреплёнными должниками

Пользователь может изменить вклад каждого должника в выбранную транзакцию путём изменения числового значения справа от имени должника. Пользователь может сделать это при помощи стрелок или клавиатуры. Также пользователь может указать, вернул ли выбранный должник свою часть транзакции путём нажатия на кнопку слева от имени должника. Её зелёный цвет означает, что должник вернул свой долг, а красный означает, что долг ещё не был возвращён. Для открепления должника от транзакции пользователь должен дважды нажать на его имя. Элемент должника исчезнет из блока выбранных должников и появится в блоке невыбранных должников.

Пользователь также может изменять и удалять транзакции, в которых он является кредитором. Для изменения транзакции он должен нажать на карточку транзакции. Появится окно создания транзакции с заголовком «Изменить транзакцию» и введёнными данными изменяемой транзакции. Для сохранения транзакции пользователь должен нажать на кнопку «Сохранить». При попытке сохранения транзакции с недопустимыми данными, например, пустым списком должников или названием, содержащим недопустимые символы, появится окно, сообщающее об этом, и сохранение транзакции прервётся. Окно сообщения об ошибке в названии представлено на рисунке 8.10.

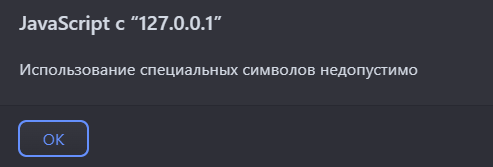


Рисунок 8.10 – Окно сообщения об использовании недопустимых символов в названии транзакции

Для удаления транзакции пользователь должен навести курсор на карточку транзакции и нажать на появившуюся в правом верхнем углу карточки кнопку удаления. Транзакция удалится, и страница обновится.

В боковом меню, в котором для гостя присутствует всего одна ссылка: на страницу долговых транзакций – авторизованным пользователям доступны дополнительные страницы. Внешний вид бокового меню авторизованного пользователя представлен на рисунке 8.11.

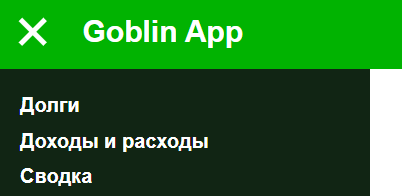


Рисунок 8.11 – Боковое меню авторизованного пользователя

Обычному пользователю также доступна страница доходов и расходов, представленная на рисунке 8.12.

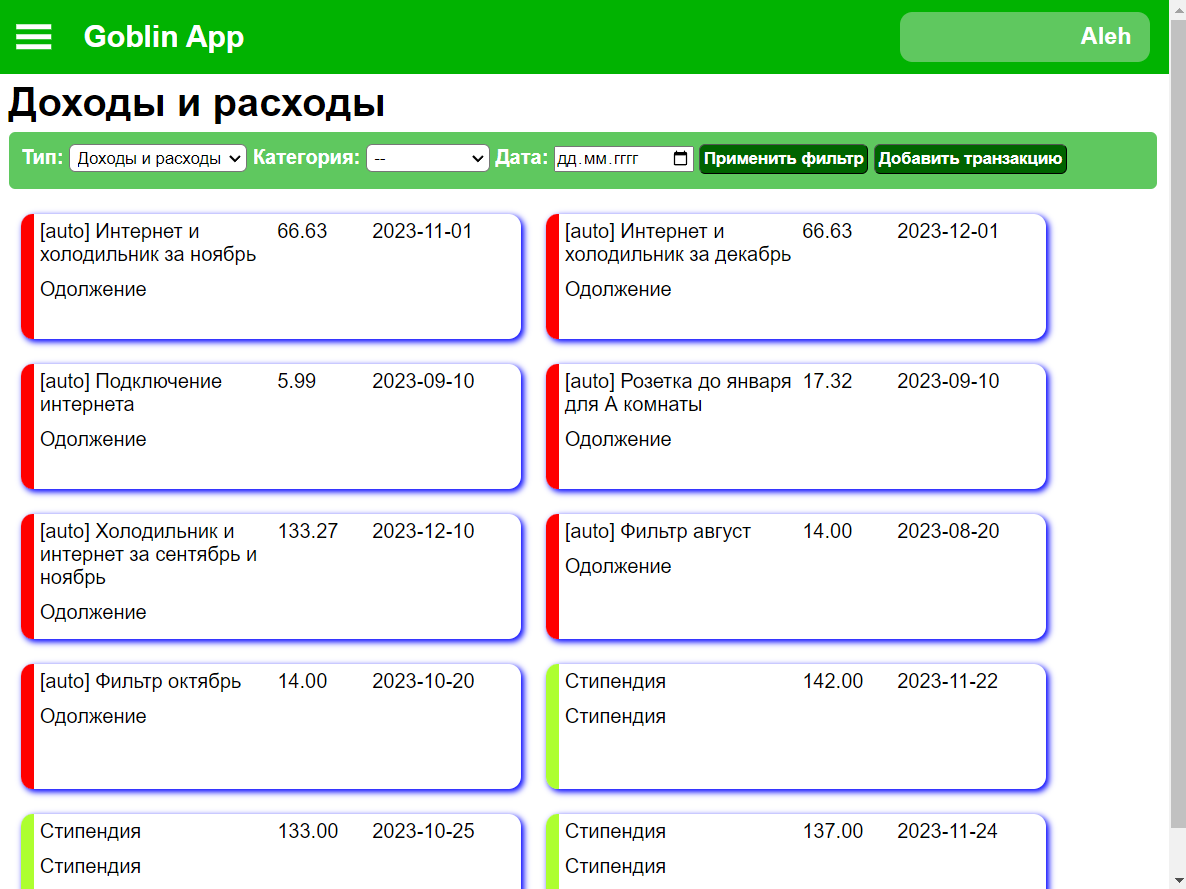


Рисунок 8.12 – Страница доходов и расходов

На странице доходов и расходов зелёным цветом помечены доходы, а красным – расходы. Карточка транзакции содержит её название, сумму, дату и её причину. Для добавления новой транзакции пользователь должен нажать на кнопку «Добавить транзакцию». Появится окно добавления транзакции, представленное на рисунке 8.13, в которое пользователь должен ввести данные и нажать на кнопку «Добавить», чтобы сохранить транзакцию, или нажать на кнопку «Отмена», чтобы закрыть окно добавления транзакции.

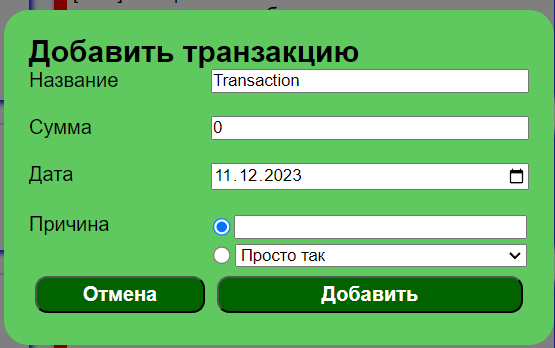


Рисунок 8.13 – Окно добавления транзакции

Отрицательное значение в поле «Сумма» означает расход, а положительное – доход. Пользователь может ввести новое значение причины или выбрать существую. Пользователь должен выбрать нужный режим ввода причины путём нажатия на одну из двух радиокнопок справа от надписи «Причина». Верхняя выбирает причину, введённую пользователем вручную, а нижняя выбирает причину, выбранную пользователем из выпадающего списка.

Пользователь может изменять и удалять свои транзакции. Для изменения он должен нажать на карточку транзакции. Появится окно добавления транзакции с заголовком «Изменить транзакцию» и данными изменяемой транзакции. Для сохранения новых данных пользователь должен нажать на кнопку «Сохранить». Для удаления транзакции пользователь должен навести курсор на карточку транзакции и нажать на появившуюся в верхнем правом углу кнопку удаления.

Пользователь может фильтровать транзакции по категории (доходы и расходы, только доходы и только расходы), причине и дате. Для этого он должен выбрать нужные значения в форме фильтрации, представленной на рисунке 8.12, и нажать на кнопку «Применить фильтр». При этом страница обновится, и на ней будут отображены отфильтрованные данные. Для сброса фильтра по причине пользователь должен выбрать пункт «--» в выпадающем списке, а для сброса фильтра по дате ещё раз нажать на кнопку «Применить фильтр», не вводя никакую дату.

Обычному пользователю также доступна страница сводки, представленная на рисунке 8.14.

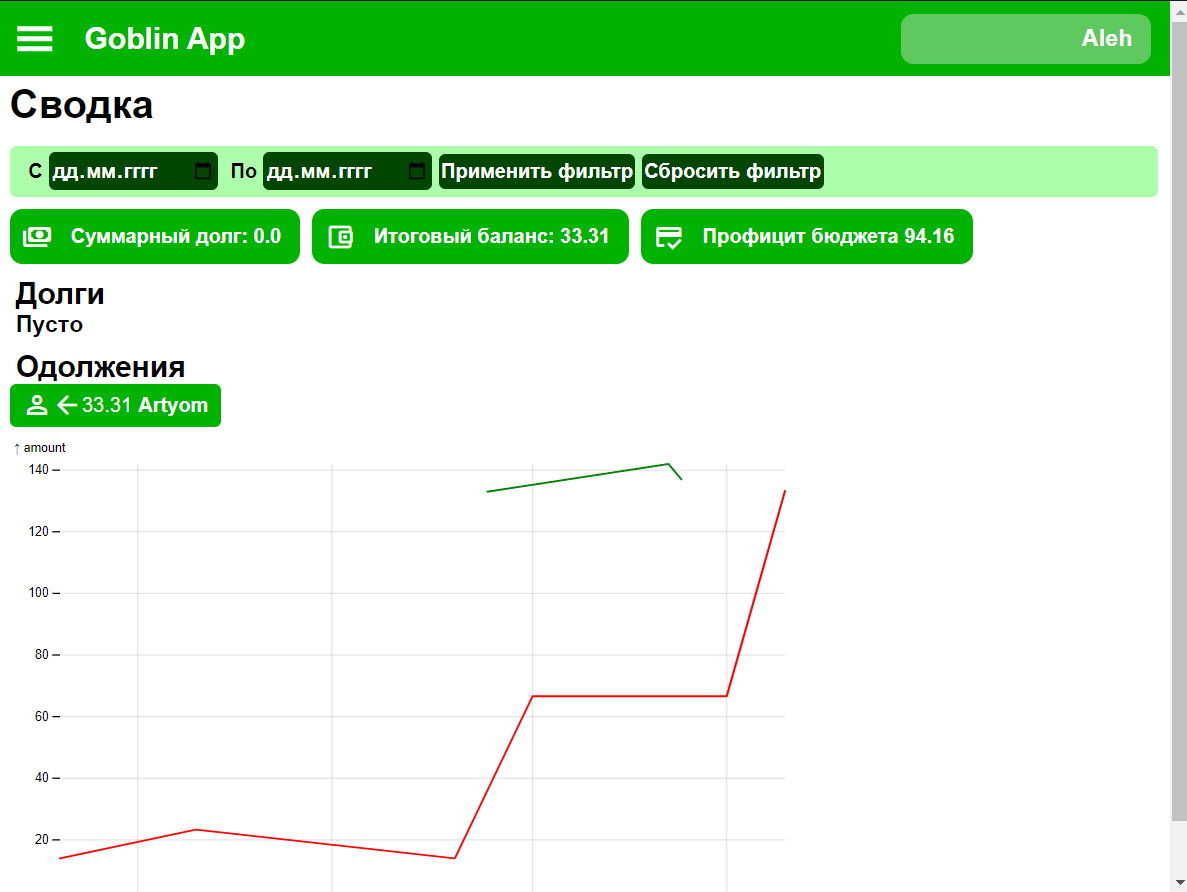


Рисунок 8.14 – Сводка финансов пользователя

На ней пользователь может ознакомиться с количеством денег, которое он должен всем прочим должникам в сумме, в блоке с надписью «Суммарный долг», разностью количества денег, которое остальные должники должны пользователю, и своим суммарным долгом, в блоке с надписью «Итоговый баланс» и разностью между своими доходами и расходами за указанный период в блоке с надписью «Дефицит бюджета» или «Профицит бюджета» в зависимости от того, больше ли полученная разность нуля или меньше.

Также пользователь может получать сводку за определённый период при помощи двух полей ввода даты в форме фильтрации. Чтобы применить фильтр, пользователь должен нажать на кнопку «Применить фильтр», а чтобы сбросить его – «Сбросить фильтр». При применении фильтра страница обновляется, и в полях ввода даты появляются выбранные пользователем временные рамки. Внешний вид формы фильтрации при выбранном периоде представлен на рисунке 8.15.



Рисунок 8.15 – Внешний вид формы фильтрации после выбора периода

Пользователь также может оценить свои доходы и расходы по автоматически создаваемому на основе его доходов и расходов графику. По оси абсцисс определяются даты, в которые баланс пользователя изменялся, а по оси ординат – значения расходов и доходов в указанный день. Зелёная ломаная линия обозначает доходы, а красная – расходы.

Администратору доступны две дополнительные кнопки в меню пользователя, кроме «Выход» и «Управление ролями»: «Импортировать» и «Экспортировать». Первая кнопка отправляет запрос на импорт данных из архива backup.zip в папке приложения с файлами в формате JSON, а вторая отправляет запрос на экспорт информации из базы данных в данный архив. В случае возникновения ошибки выводится окно с кодом ответа сервера.

Также администратор может редактировать и удалять любые долговые транзакции. Его окно управления ролями содержит три дополнительные вкладки: добавления и удаления роли по названию, а также изменение ролей любого пользователя. Внешний вид окна управления ролями администратора представлен на рисунке 8.16.

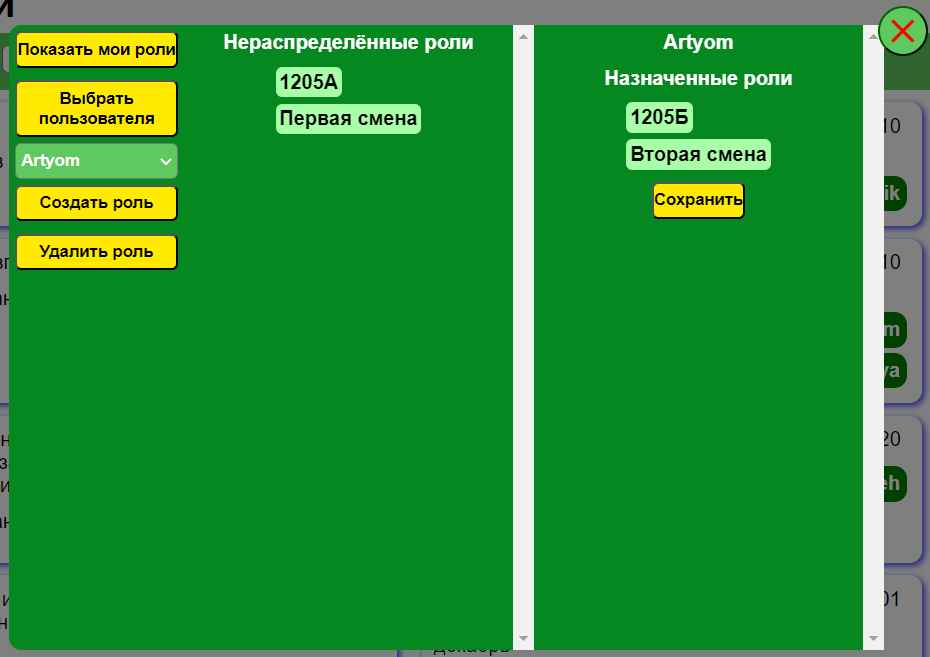


Рисунок 8.16 – Внешний вид окна управления ролями администратора

Для создания новой роли пользователь должен нажать на кнопку «Создать роль». Текущий блок исчезнет, и появится блок создания роли с заголовком «Создать роль», текстовым полем ввода и кнопкой «Сохранить». Пользователь должен ввести уникальное название роли, состоящее из кириллических или латинских букв, цифр и пробелов, и нажать на кнопку «Сохранить». Если роль с таким названием уже существует или название роли не отвечает вышеописанным требованиям, появится окно с сообщением об ошибке.

Для удаления существующей роли пользователь должен нажать на кнопку «Удалить роль». Текущий блок исчезнет, и появится блок создания роли с заголовком «Удалить роль», выпадающим списком и кнопкой «Удалить». Пользователь должен выбрать нужную роль и нажать на кнопку «Удалить». Если роль была удалена успешно, появится окно с надписью «Успешно», список ролей обновится, и при закрытии окна управления ролями страница перезагрузится. В противном случае появится окно с кодом ответа сервера.

Администратору доступна страница управления должниками и пользователями, внешний вид которой представлен на рисунке 8.16.

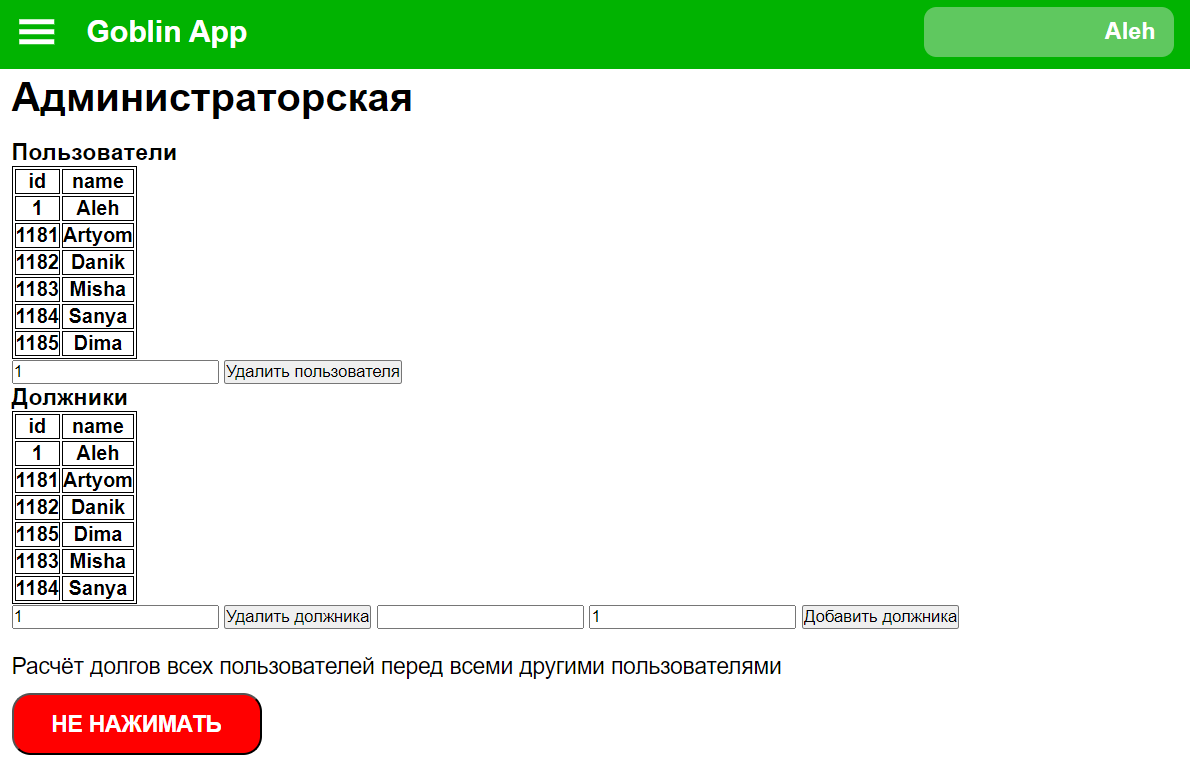


Рисунок 8.16 – Страница управления должниками и пользователями

На данной странице администратор может удалить пользователя по его идентификатору, который можно узнать из таблицы «Пользователи», таким же образом удалить должника и создать должника с указанным именем и идентификатором пользователя. При попытке удалить самого себя или создать должника с уже занятым именем или уже занятым пользователем появится окно с сообщением о произошедшей ошибке.

Также администратор может запустить процесс перерасчёта всех пользователей перед всеми прочими пользователями путём нажатия на кнопку с надписью «Не нажимать». Данный текст предупреждает о потенциально чрезвычайно больших временных затратах на эту операцию. При успешном завершении перерасчёта появится окно с текстом «Расчёт выполнен».

Заключение

При выполнении курсового проекта была создана база данных для приложения учёта доходов, расходов и долгов, а также приложение для работы с ней. База данных была реализована в СУБД Oracle. Были реализованы все функциональные требования, а именно:

– управление пользователями и паролями (добавление, удаление, изменение);

– управление ролями (администратор, обычный пользователь, гость);

– управление должниками (добавление, удаление, изменение);

– управление транзакциями (добавление, удаление, изменение);

– анализ доходов и расходов (объём долга каждого пользователя перед каждым другим пользователем, общий долг каждого пользователя, долг за указанный период).

Был реализован импорт и экспорт данных из базы данных в формат JSON. Была протестирована производительность базы данных при помощи таблицы, содержащей более 100000 строк, и созданы индексы для улучшения производительности базы данных. Исключения, возникающие при выполнении процедур, обрабатываются процедурами и приложением.

Список используемых источников

1. Oracle Database Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/index.html. Дата доступа: 20.09.2023.
2. Oracle Database In-Memory with Oracle Database 19c [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.oracle.com/a/tech/docs/twp-oracle-database-in-memory-19c.pdf. Дата доступа: 10.12.2023.
3. Django documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.djangoproject.com/en/5.0/ . Дата доступа: 23.10.2023.
4. Python 3.12.1 documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: 30.11.2023.

Приложение А

Листинг создания таблиц в базе данных

|  |
| --- |
| create sequence ROLES\_ID\_SEQ cycle;  create table ROLES (  id int default ROLES\_ID\_SEQ.nextval primary key,  name nvarchar2(50) not null unique  );  create sequence USERS\_ID\_SEQ cycle;  create table USERS (  id int default USERS\_ID\_SEQ.nextval primary key,  name nvarchar2(50) not null unique,  password nvarchar2(256) not null  );  create sequence USERS\_ROLES\_ID\_SEQ cycle;  create table USERS\_ROLES (  id int default USERS\_ROLES\_ID\_SEQ.nextval primary key,  user\_id int not null constraint USERS\_ROLES\_USER\_ID\_FK references USERS(id) on delete cascade,  role\_id int not null constraint USERS\_ROLES\_ROLE\_ID\_FK references ROLES(id) on delete cascade  );  create sequence DEBTORS\_ID\_SEQ cycle;  create table DEBTORS (  id int default DEBTORS\_ID\_SEQ.nextval primary key,  name nvarchar2(50) unique not null,  user\_id int constraint ga\_debtors\_user\_id\_fk references USERS(id) on delete set null  );  create sequence DEBT\_TRANSACTIONS\_ID\_SEQ cycle;  create table DEBT\_TRANSACTIONS (  id int default DEBT\_TRANSACTIONS\_ID\_SEQ.nextval primary key,  name nvarchar2(50) not null,  amount int not null,  lender int constraint ga\_debt\_transactions\_lender\_fk references DEBTORS(id) on delete cascade not null,  transaction\_date date,  description nvarchar2(256)  );  create sequence DEBTORS\_ENTRIES\_ID\_SEQ cycle;  create table DEBTORS\_ENTRIES (  row\_id int default DEBTORS\_ENTRIES\_ID\_SEQ.nextval primary key,  transaction\_id int not null constraint GA\_DEBTORS\_ENTRIES\_TRANSACTIONS\_ID\_FK  references DEBT\_TRANSACTIONS(id) on delete cascade,  debtor\_id int not null constraint GA\_DEBTORS\_ID\_FK references DEBTORS(id) on delete cascade,  multiplier float not null,  amount int not null,  fulfilled char(1) check ( fulfilled in ('Y', 'N') ) not null  );  create sequence DEBT\_TRANSACTIONS\_RESTRICTIONS\_ID\_SEQ cycle;  create table DEBT\_TRANSACTIONS\_RESTRICTIONS (  id int default DEBT\_TRANSACTIONS\_RESTRICTIONS\_ID\_SEQ.nextval primary key,  transaction\_id int not null  constraint DEBT\_TRANSACTIONS\_RESTRICTIONS\_T\_ID\_FK references DEBT\_TRANSACTIONS(id) on delete cascade,  role\_id int not null constraint DEBT\_TRANSACTIONS\_RESTRICTIONS\_R\_ID\_FK references ROLES(id) on delete cascade  );  create sequence DISBALANCE\_TRANSACTIONS\_ID\_SEQ cycle;  drop sequence DISBALANCE\_TRANSACTIONS\_ID\_SEQ;  create table DISBALANCE\_TRANSACTIONS (  id int default DISBALANCE\_TRANSACTIONS\_ID\_SEQ.nextval primary key,  name nvarchar2(50) not null, -- not unique  amount int not null check ( amount != 0 ),  reason nvarchar2(50) not null,  transaction\_date date not null,  user\_id int not null constraint DISBALANCE\_TRANSACTIONS\_USER\_ID\_FK references USERS(id)  on delete cascade  );  create sequence P2P\_DEBTS\_ID\_SEQ cycle;  create table P2P\_DEBTS (  row\_id int default P2P\_DEBTS\_ID\_SEQ.nextval primary key,  lender int not null constraint P2P\_DEBTS\_LENDER\_FK references DEBTORS(id) on delete cascade,  debtor int not null constraint P2P\_DEBTS\_DEBTOR\_FK references DEBTORS(id) on delete cascade,  amount int default 0 not null  ); |

Приложение Б

Листинг создания процедур

|  |
| --- |
| create or replace procedure GET\_ROLES(p\_cursor out SYS\_REFCURSOR) as  begin  open p\_cursor for select \* from ROLES;  end;  create or replace procedure ADD\_ROLE(p\_id int, role\_name nvarchar2, result out int) as  auxil\_number int := 0;  begin  select count(\*) into auxil\_number from ROLES where name = role\_name;  if auxil\_number = 1 then  result := -1;  else  if p\_id = -1 then  insert into ROLES (name) values (role\_name);  else  insert into ROLES (id, name) values (p\_id, role\_name);  end if;  commit;  result := 0;  end if;  exception  when others then  result := -1;  end;  create or replace procedure DELETE\_ROLE(p\_id int) as  begin  delete ROLES where id = p\_id;  commit;  end;  create or replace procedure IS\_LOGIN\_CORRECT(login nvarchar2, password\_hash nvarchar2, result out int) as  found\_users int;  begin  select count(\*) into found\_users from USERS where name = login and password = password\_hash;  if found\_users = 1 then  result := 1;  else  result := -1;  end if;  end;  create or replace procedure GET\_USERS(p\_cursor out SYS\_REFCURSOR) as  begin  open p\_cursor for select id, name from USERS order by id;  end;  create or replace procedure GET\_USERS\_FOR\_EXPORT(p\_cursor out SYS\_REFCURSOR) as  begin  open p\_cursor for select id, name, password from USERS;  end;  create or replace procedure ADD\_USER(p\_id int, p\_name nvarchar2, p\_password nvarchar2, result out int) as  similar\_usernames int := 0;  number\_of\_existing\_debtors int := 0;  begin  select count(\*) into similar\_usernames from USERS where name = p\_name and (p\_id = -1 or p\_id = id);  if similar\_usernames = 0 then  if p\_id = -1 then  insert into USERS(name, password) values (p\_name, p\_password);--replace needed  else  insert into USERS(id, name, password) values (p\_id, p\_name, p\_password);--replace needed  end if;  select id into result from USERS where name = p\_name and password = p\_password and ROWNUM = 1;  select count(\*) into number\_of\_existing\_debtors from DEBTORS where name = p\_name;  if number\_of\_existing\_debtors = 0 then  ADD\_DEBTOR(-1, p\_name, result, result);  else  update DEBTORS set USER\_ID = result where name = p\_name;  end if;  commit;  else  result := -1;  end if;  end;  create or replace procedure DELETE\_USER(user\_id int) as  begin  delete USERS where id = user\_id;  commit;  end;  create or replace procedure GET\_USER\_ROLES (p\_cursor out SYS\_REFCURSOR, u\_id int) as  begin  open p\_cursor for select role\_id from USERS\_ROLES where user\_id = u\_id;  end;  create or replace procedure ADD\_USER\_ROLE (p\_user\_id int, p\_role\_id int) as  role\_exists int;  user\_exists int;  selected\_role nvarchar2(50);  trying\_to\_assign\_admin exception;  begin  select name into selected\_role from ROLES where id = p\_role\_id;  if selected\_role = 'admin' then raise trying\_to\_assign\_admin; end if;  select count(\*) into user\_exists from USERS where p\_user\_id = id;  if user\_exists = 0 then  return;  end if;  select count(\*) into role\_exists from USERS\_ROLES where user\_id = p\_user\_id and role\_id = p\_role\_id;  if role\_exists = 0 then  insert into USERS\_ROLES (user\_id, role\_id) VALUES (p\_user\_id, p\_role\_id);  commit;  end if;  exception  when trying\_to\_assign\_admin then  RAISE\_APPLICATION\_ERROR(-20002, 'Cannot assign admin role through this procedure');  when others then  RAISE\_APPLICATION\_ERROR(-20001, sqlerrm);  end;  create or replace procedure DELETE\_USER\_ROLE (p\_user\_id int, p\_role\_id int) as  begin  delete USERS\_ROLES where user\_id = p\_user\_id and role\_id = p\_role\_id;  commit;  end;  create or replace procedure GET\_DEBTORS(p\_cursor out SYS\_REFCURSOR) as  begin  open p\_cursor for select \* from DEBTORS order by name;  end;  create or replace procedure ADD\_DEBTOR(p\_id int, p\_name nvarchar2, p\_user\_id int, result out int) as  auxil\_number int := 0;  existing\_debtors int := 0;  begin  if p\_id != -1 then  select count(\*) into existing\_debtors from DEBTORS where id = p\_id;  if existing\_debtors != 0 then  result := -1;  return;  end if;  end if;  select count(\*) into auxil\_number from DEBTORS where name = p\_name;  if auxil\_number = 0 then  if p\_id = -1 then  insert into DEBTORS (name, user\_id) values (p\_name, p\_user\_id);  else  insert into DEBTORS (id, name, user\_id) values (p\_id, p\_name, p\_user\_id);  end if;  select id into auxil\_number from DEBTORS where name = p\_name;  for debtor\_entry in (select id from DEBTORS where id != auxil\_number) loop  insert into P2P\_DEBTS (lender, debtor) values (debtor\_entry.id, auxil\_number);  insert into P2P\_DEBTS (lender, debtor) values (auxil\_number, debtor\_entry.id);  end loop;  result := auxil\_number;  else  result := -1;  end if;  commit;  end;  create or replace procedure DELETE\_DEBTOR(debtor\_id int) as  begin  delete DEBTORS where id = debtor\_id;  commit;  end;  create or replace procedure GET\_DEBT\_TRANSACTIONS(p\_cursor out SYS\_REFCURSOR) as  begin  open p\_cursor for select \* from DEBT\_TRANSACTIONS;  end;  create or replace procedure ADD\_DEBT\_TRANSACTION(p\_id int, p\_name nvarchar2, p\_amount int, p\_lender int,  p\_transaction\_date date, p\_description nvarchar2, result out int) as  existing\_transactions int := 0;  begin  if p\_id != -1 then  select count(\*) into existing\_transactions from DEBT\_TRANSACTIONS where id = p\_id;  if existing\_transactions != 0 then  result := -1;  return;  end if;  end if;  if p\_id = -1 then  insert into DEBT\_TRANSACTIONS (name, amount, lender, transaction\_date, description)  values (p\_name, p\_amount, p\_lender, p\_transaction\_date, p\_description);  else  insert into DEBT\_TRANSACTIONS (id, name, amount, lender, transaction\_date, description)  values (p\_id, p\_name, p\_amount, p\_lender, p\_transaction\_date, p\_description);  end if;  select id into result from DEBT\_TRANSACTIONS where ROWNUM = 1 order by id desc;  commit;  end;  create or replace procedure UPDATE\_DEBT\_TRANSACTION(p\_id int, p\_name nvarchar2, p\_amount int, p\_lender int,  p\_transaction\_date date, p\_description nvarchar2) as  begin  update DEBT\_TRANSACTIONS set name = p\_name, amount = p\_amount, lender = p\_lender,  transaction\_date = p\_transaction\_date, description = p\_description  where id = p\_id;  end;  create or replace procedure DELETE\_DEBT\_TRANSACTION(p\_id int) as  debtors\_cursor SYS\_REFCURSOR;  v\_debtor\_id int;  v\_lender\_id int;  begin  select lender into v\_lender\_id from DEBT\_TRANSACTIONS where id = p\_id;  open debtors\_cursor for select e.debtor\_id from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t  on e.transaction\_id = t.id where t.id = p\_id and e.debtor\_id != t.lender;  loop  fetch debtors\_cursor into v\_debtor\_id;  exit when debtors\_cursor%NOTFOUND;  delete from DEBTORS\_ENTRIES where TRANSACTION\_ID = p\_id and DEBTORS\_ENTRIES.DEBTOR\_ID = v\_debtor\_id;  CALCULATE\_P2P\_DEBTS(v\_lender\_id, v\_debtor\_id);  end loop;  delete DEBT\_TRANSACTIONS where id = p\_id;  commit;  end;  create or replace procedure GET\_DEBTORS\_ENTRIES(p\_cursor out SYS\_REFCURSOR, p\_tran\_id int) as  begin  open p\_cursor for select debtor\_id, multiplier, fulfilled from DEBTORS\_ENTRIES  where transaction\_id = p\_tran\_id;  end;  create or replace procedure ADD\_DEBTOR\_ENTRY(p\_tran\_id int, p\_debtor\_id int, p\_multiplier float, p\_amount int,  p\_fulfilled char) as  auxil\_number int;  lender int;  begin  select count(\*) into auxil\_number from DEBT\_TRANSACTIONS where id = p\_tran\_id;  if auxil\_number = 0 then  RAISE\_APPLICATION\_ERROR(-20001, 'trying to add debtor entry for non-existing debt transaction');  end if;  select count(\*) into auxil\_number from DEBTORS\_ENTRIES where transaction\_id = p\_tran\_id  and debtor\_id = p\_debtor\_id;  if auxil\_number = 0 then  insert into DEBTORS\_ENTRIES(transaction\_id, debtor\_id, multiplier, amount, fulfilled)  values (p\_tran\_id, p\_debtor\_id, p\_multiplier, p\_amount, p\_fulfilled);  end if;  select lender into lender from DEBT\_TRANSACTIONS  where id = p\_tran\_id and ROWNUM = 1;  CALCULATE\_P2P\_DEBTS(p\_debtor\_id, lender);  commit;  end;  create or replace procedure DELETE\_DEBTOR\_ENTRY(p\_transaction\_id int, p\_debtor\_id int) as  lender int;  debtor int;  begin  delete DEBTORS\_ENTRIES where transaction\_id = p\_transaction\_id and debtor\_id = p\_debtor\_id;  select lender into lender from DEBT\_TRANSACTIONS  where id = p\_transaction\_id and ROWNUM = 1;  select id into debtor from DEBTORS where id = p\_debtor\_id;  if debtor is null or lender is null then  RAISE\_APPLICATION\_ERROR(-20001, 'Lender or debtor is null');  end if;  CALCULATE\_P2P\_DEBTS(p\_debtor\_id, lender);  commit;  end;  create or replace procedure GET\_TRANSACTION\_RESTRICTIONS (p\_cursor out SYS\_REFCURSOR, p\_id int) as  begin  open p\_cursor for select role\_id from DEBT\_TRANSACTIONS\_RESTRICTIONS where transaction\_id = p\_id;  end;  create or replace procedure ADD\_TRANSACTION\_RESTRICTION (p\_t\_id int, p\_r\_id int) as  begin  insert into DEBT\_TRANSACTIONS\_RESTRICTIONS (transaction\_id, role\_id) VALUES (p\_t\_id, p\_r\_id);  commit;  exception when others then  RAISE\_APPLICATION\_ERROR(-20002, sqlerrm);  end;  create or replace procedure DELETE\_TRANSACTION\_RESTRICTION (p\_t\_id int, p\_r\_id int) as  begin  delete from DEBT\_TRANSACTIONS\_RESTRICTIONS where transaction\_id = p\_t\_id and role\_id = p\_r\_id;  commit;  end;  create or replace procedure GET\_DISBALANCE\_TRANSACTIONS(p\_cursor out SYS\_REFCURSOR, p\_user\_id int, p\_date date,  p\_mode int, p\_reason nvarchar2) as  begin  commit;  open p\_cursor for select \* from DISBALANCE\_TRANSACTIONS  where (p\_user\_id is null or user\_id = p\_user\_id)  and (p\_date IS NULL OR transaction\_date = p\_date)  and (p\_mode = 0 OR (p\_mode > 0 and amount > 0) or (p\_mode < 0 and amount < 0))  and (p\_reason is null or reason = p\_reason);  end;  create or replace procedure ADD\_DISBALANCE\_TRANSACTION(p\_name nvarchar2, p\_amount float, p\_reason nvarchar2,  p\_transaction\_date date, p\_user\_id int) as  begin  insert into DISBALANCE\_TRANSACTIONS(name, amount, reason, transaction\_date, user\_id) VALUES  (p\_name, p\_amount, p\_reason, p\_transaction\_date, p\_user\_id);  commit;  end;  create or replace procedure UPDATE\_DISBALANCE\_TRANSACTION(p\_id int, p\_name nvarchar2, p\_amount int,  p\_reason nvarchar2, p\_date date) as  begin  update DISBALANCE\_TRANSACTIONS set name=p\_name, amount=p\_amount, reason=p\_reason,  transaction\_date=p\_date where id=p\_id;  commit;  end;  create or replace procedure DELETE\_DISBALANCE\_TRANSACTION(p\_id int) as  begin  delete from DISBALANCE\_TRANSACTIONS where id = p\_id;  commit;  end;  create or replace procedure GET\_P2P\_DEBTS\_BY\_PERSON\_ID(p\_cursor out SYS\_REFCURSOR, p\_id int, is\_debtor char,  start\_date nvarchar2, end\_date nvarchar2) as  begin  if length(start\_date||'q') = 1 and length(end\_date||'q') = 1 then  if is\_debtor = 'Y' then  open p\_cursor for select lender, amount from P2P\_DEBTS where debtor = p\_id and amount > 0;  else  open p\_cursor for select debtor, amount from P2P\_DEBTS where lender = P\_id and amount > 0;  end if;  else  if is\_debtor = 'Y' then  open p\_cursor for select e.debtor\_id, e.amount from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t on  e.transaction\_id = t.id where t.lender != p\_id and e.debtor\_id = p\_id and  (length(start\_date||'q') = 1 or TO\_DATE(start\_date, 'YYYY-MM-DD') <= t.transaction\_date)  and (length(end\_date||'q') = 1 or TO\_DATE(end\_date, 'YYYY-MM-DD') >= t.transaction\_date)  and e.fulfilled = 'N';  else  open p\_cursor for select e.debtor\_id, e.amount from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t on  e.transaction\_id = t.id where t.lender = p\_id and e.debtor\_id != p\_id and  (length(start\_date||'q') = 1 or TO\_DATE(start\_date, 'YYYY-MM-DD') <= t.transaction\_date)  and (length(end\_date||'q') = 1 or TO\_DATE(end\_date, 'YYYY-MM-DD') >= t.transaction\_date)  and e.fulfilled = 'N';  end if;  end if;  end;  create or replace procedure CALCULATE\_P2P\_DEBTS (p\_lender int, p\_debtor int) as  v\_debtors\_debt int := 0;  v\_lenders\_debt int := 0;  begin  select sum(e.amount) into v\_debtors\_debt from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t on  e.transaction\_id = t.id where e.debtor\_id = p\_debtor and e.fulfilled = 'N' and t.lender = p\_lender;  if v\_debtors\_debt is null then  v\_debtors\_debt := 0;  end if;  select sum(e.amount) into v\_lenders\_debt from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t on  e.transaction\_id = t.id where e.debtor\_id = p\_lender and e.fulfilled = 'N' and t.lender = p\_debtor;  if v\_lenders\_debt is null then  v\_lenders\_debt := 0;  end if;  update P2P\_DEBTS set amount = (v\_debtors\_debt - v\_lenders\_debt) where lender = p\_lender and debtor = p\_debtor;  update P2P\_DEBTS set amount = (v\_lenders\_debt - v\_debtors\_debt) where lender = p\_debtor and debtor = p\_lender;  commit;  end;  create or replace procedure GET\_TOTAL\_BALANCE(p\_id int, start\_date nvarchar2, end\_date nvarchar2,  result out int) as  income int;  outcome int;  begin  if length(start\_date||'q') = 1 and length(end\_date||'q') = 1 then  select sum(amount) into result from P2P\_DEBTS where lender = p\_id;  else  select sum(e.amount) into outcome from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t on  e.transaction\_id = t.id where e.debtor\_id = p\_id and t.lender != p\_id and  (length(start\_date||'q') = 1 or TO\_DATE(start\_date, 'YYYY-MM-DD') <= t.transaction\_date) and  (length(end\_date||'q') = 1 or TO\_DATE(end\_date, 'YYYY-MM-DD') >= t.transaction\_date)  and e.fulfilled = 'N';  select sum(e.amount) into income from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t on  e.transaction\_id = t.id where e.debtor\_id != p\_id and t.lender = p\_id and  (length(start\_date||'q') = 1 or TO\_DATE(start\_date, 'YYYY-MM-DD') <= t.transaction\_date) and  (length(end\_date||'q') = 1 or TO\_DATE(end\_date, 'YYYY-MM-DD') >= t.transaction\_date)  and e.fulfilled = 'N';  if income is null then  income := 0;  end if;  if outcome is null then  outcome := 0;  end if;  result := income - outcome;  end if;  if result is null then  result := 0;  end if;  end;  create or replace procedure GET\_TOTAL\_DEBT(p\_id int, start\_date nvarchar2, end\_date nvarchar2,  result out int) as  begin  if length(start\_date||'q') = 1 and length(end\_date||'q') = 1 then  select sum(amount) into result from P2P\_DEBTS where debtor = p\_id and amount > 0;  else  select sum(e.amount) into result from DEBTORS\_ENTRIES e join DEBT\_TRANSACTIONS t on  e.transaction\_id = t.id where e.debtor\_id = p\_id and t.lender != p\_id and e.fulfilled = 'N' and  (length(start\_date||'q') = 1 or TO\_DATE(start\_date, 'YYYY-MM-DD') <= t.transaction\_date) and  (length(end\_date||'q') = 1 or TO\_DATE(end\_date, 'YYYY-MM-DD') >= t.transaction\_date);  end if;  if result is null then  result := 0;  end if;  end;  create or replace procedure WIPE\_DATABASE(redundant\_parameter int) as  begin  delete DEBT\_TRANSACTIONS;  delete DEBT\_TRANSACTIONS\_RESTRICTIONS;  delete DEBTORS;  delete DEBTORS\_ENTRIES;  delete DISBALANCE\_TRANSACTIONS;  delete P2P\_DEBTS;  delete ROLES;  delete USERS;  delete USERS\_ROLES;  commit;  exception when others then  rollback;  RAISE\_APPLICATION\_ERROR(-20003, sqlerrm);  end; |

Приложение В

Листинг создания пользователей и выдачи им необходимых привилегий

|  |
| --- |
| create user GUEST identified by guest\_password;  create user COMMON\_USER identified by user\_password;  create user ADMIN identified by admin\_password;  grant create session, restricted session to GUEST, COMMON\_USER, ADMIN;  grant execute on GET\_ROLES to GUEST, COMMON\_USER, ADMIN;  grant execute on GET\_DEBT\_TRANSACTIONS to GUEST, COMMON\_USER, ADMIN;  grant execute on GET\_DEBTORS to GUEST, COMMON\_USER, ADMIN;  grant execute on GET\_DEBTORS\_ENTRIES to GUEST, COMMON\_USER, ADMIN;  grant execute on GET\_TRANSACTION\_RESTRICTIONS to GUEST, COMMON\_USER, ADMIN;  grant execute on IS\_LOGIN\_CORRECT to GUEST, COMMON\_USER, ADMIN;  grant execute on ADD\_USER to GUEST, COMMON\_USER, ADMIN;  grant execute on GET\_TOTAL\_BALANCE to COMMON\_USER, ADMIN;  grant execute on GET\_DISBALANCE\_TRANSACTIONS to COMMON\_USER, ADMIN;  grant execute on GET\_TOTAL\_DEBT to COMMON\_USER, ADMIN;  grant execute on GET\_P2P\_DEBTS\_BY\_PERSON\_ID to COMMON\_USER, ADMIN;  grant execute on ADD\_DEBT\_TRANSACTION to COMMON\_USER, ADMIN;  grant execute on ADD\_DEBTOR\_ENTRY to COMMON\_USER, ADMIN;  grant execute on ADD\_DISBALANCE\_TRANSACTION to COMMON\_USER, ADMIN;  grant execute on ADD\_TRANSACTION\_RESTRICTION to COMMON\_USER, ADMIN;  grant execute on ADD\_USER\_ROLE to COMMON\_USER, ADMIN;  grant execute on DELETE\_DEBT\_TRANSACTION to COMMON\_USER, ADMIN;  grant execute on DELETE\_DEBTOR\_ENTRY to COMMON\_USER, ADMIN;  grant execute on DELETE\_DISBALANCE\_TRANSACTION to COMMON\_USER, ADMIN;  grant execute on DELETE\_TRANSACTION\_RESTRICTION to COMMON\_USER, ADMIN;  grant execute on DELETE\_USER\_ROLE to COMMON\_USER, ADMIN;  grant execute on GET\_USERS to COMMON\_USER, ADMIN;  grant execute on GET\_USER\_ROLES to COMMON\_USER, ADMIN;  grant execute on UPDATE\_DISBALANCE\_TRANSACTION to COMMON\_USER, ADMIN;  grant execute on UPDATE\_DEBT\_TRANSACTION to COMMON\_USER, ADMIN;  grant execute on ADD\_DEBTOR to ADMIN;  grant execute on ADD\_ROLE to ADMIN;  grant execute on DELETE\_DEBTOR to ADMIN;  grant execute on DELETE\_ROLE to ADMIN;  grant execute on DELETE\_USER to ADMIN;  grant execute on CALCULATE\_P2P\_DEBTS to ADMIN;  grant execute on WIPE\_DATABASE to ADMIN;  grant execute on GET\_USERS\_FOR\_EXPORT to ADMIN; |

Приложение Г

Листинг функций импорта-экспорта приложения

|  |
| --- |
| import json  from .db\_controller import Role, User, DebtTransaction, Debtor, DebtorEntry, DisbalanceTransaction, DbController  class Exporter:  @staticmethod  def export\_roles(roles: [Role]):  filename = 'roles.json'  with open(filename, 'w') as f:  json.dump([{'id': role.id, 'name': role.name} for role in roles], f)  return filename  @staticmethod  def export\_users(users):  filename = 'users.json'  with open(filename, 'w') as f:  json.dump([{'id': user['id'], 'name': user['name'], 'password': user['password'],  'roles': json.dumps([role.id for role in user['roles']])} for user in users], f)  return filename  @staticmethod  def export\_debtors(debtors: [Debtor]):  filename = 'debtors.json'  with open(filename, 'w') as f:  json.dump([{'id': debtor.id, 'name': debtor.name,  'user\_id': debtor.user\_id} for debtor in debtors], f)  return filename  @staticmethod  def export\_debt\_transactions(transactions: [DebtTransaction]):  filename = 'debt\_transactions.json'  with open(filename, 'w') as f:  json.dump([{'id': t.id, 'name': t.name, 'amount': t.amount, 'lender': t.lender.id,  'date': t.date, 'description': t.description, 'debtors\_entries': json.dumps(  [{'debtor\_id': entry.debtor.id, 'multiplier': entry.multiplier,  'fulfilled': entry.fulfilled} for entry in t.debtors\_entries]  ), 'restrictions': json.dumps([role.id for role in t.restrictions])} for t in transactions], f)  return filename  @staticmethod  def export\_disbalance\_transactions(transactions: [DisbalanceTransaction]):  filename = 'disbalance\_transactions.json'  with open(filename, 'w') as f:  json.dump([{'id': t.id, 'name': t.name, 'amount': t.amount, 'reason': t.reason,  'date': t.date, 'user\_id': t.user.id} for t in transactions], f)  return filename  class Importer:  @staticmethod  def import\_roles(db\_controller: DbController):  filename = 'roles.json'  with open(filename, 'r') as f:  new\_roles = json.loads(f.read())  for role in new\_roles:  db\_controller.add\_role(p\_id=role['id'], name=role['name'])  return filename  @staticmethod  def import\_debtors(db\_controller: DbController):  filename = 'debtors.json'  with open(filename, 'r') as f:  new\_debtors = json.loads(f.read())  for debtor in new\_debtors:  db\_controller.add\_debtor(p\_id=debtor['id'], user\_id=None, name=debtor['name'])  return filename  @staticmethod  def import\_users(db\_controller: DbController, db\_storage):  filename = 'users.json'  with open(filename, 'r') as f:  new\_users = json.loads(f.read())  for user in new\_users:  user\_roles = list(map(lambda x: list(filter(lambda y: y.id == int(x), db\_storage['roles']))[0],  json.loads(user['roles'])))  db\_controller.add\_user(p\_id=user['id'],  name=user['name'],  password=user['password'],  roles=user\_roles)  return filename  @staticmethod  def import\_debt\_transactions(db\_controller: DbController, db\_storage):  filename = 'debt\_transactions.json'  with open(filename, 'r') as f:  new\_transactions = json.loads(f.read())  for t in new\_transactions:  print(json.loads(t['restrictions']))  db\_controller.add\_debt\_transaction(p\_id=int(t['id']), name=t['name'],  amount=float(t['amount']), lender=list(filter(lambda x: x.id == int(t['lender']), db\_storage['debtors']))[0], date=t['date'],  description=t['description'], restrictions=list(map(lambda x: Role(role\_id=int(x), name=''), json.loads(t['restrictions']))))  debtors\_entries = json.loads(t['debtors\_entries'])  sum\_multiplier = sum(map(lambda x: float(x['multiplier']), debtors\_entries))  for entry in debtors\_entries: db\_controller.add\_debtor\_entry\_by\_transaction\_id(transaction\_id=int(t['id']), debtor\_id=int(entry['debtor\_id']), multiplier=float(entry['multiplier']),  amount=float(t['amount']) \*float(entry['multiplier']) / sum\_multiplier, fulfilled=entry['fulfilled'])  return filename  @staticmethod  def import\_disbalance\_transactions(db\_controller: DbController, db\_storage):  filename = 'disbalance\_transactions.json'  with open(filename, 'r') as f:  new\_transactions = json.loads(f.read())  for t in new\_transactions: db\_controller.add\_disbalance\_transaction(name=t['name'], amount=float(t['amount']), reason=t['reason'], date=t['date'], user=list(filter(lambda x: x.id == int(t['user\_id']), db\_storage['users']))[0])  return filename |