РЕФЕРАТ

Данная курсовая работа содержит реализацию базы данных и веб-приложения для анализа ежемесячных расходов и доходов, позволяющему вести бюджеты за определенные периоды. Пользователь приложения может просматривать информацию о финансовых кат­­­­егориях, валютах и курсах обмена. Также пользователь может вести учет своих банковских счетов, транзакций, целей и финансовых целей в нескольких валютах.

В качестве системы управления базами данных используется PostgreSQL, веб-приложение реализовано на языке Java версии 11.

Ключевые слова: бюджет, финансовое планирование, финансы, база данных, Java, PostgreSQL.

Расчетно-пояснительная записка к курсовой работе содержит 50 страниц, 21 рисунок, 15 таблиц и 11 источников.

СОДЕРЖАНИЕ

[РЕФЕРАТ 3](#_Toc168560089)

[СОДЕРЖАНИЕ 4](#_Toc168560090)

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 6](#_Toc168560091)

[ВВЕДЕНИЕ 7](#_Toc168560092)

[1 Аналитический раздел 8](#_Toc168560093)

[1.1 Анализ предметной области 8](#_Toc168560094)

[1.2 Анализ существующих решений 9](#_Toc168560095)

[1.3 Формализация задачи 10](#_Toc168560096)

[1.4 Формализация данных 10](#_Toc168560097)

[1.5 Формализация пользователей 12](#_Toc168560098)

[1.6 Анализ моделей баз данных 14](#_Toc168560099)

[1.6.1 Дореляционные модели 14](#_Toc168560100)

[1.6.2 Реляционные модели 15](#_Toc168560101)

[1.6.3 Постреляционные модели 15](#_Toc168560102)

[Вывод из аналитического раздела 16](#_Toc168560103)

[2 Конструкторский раздел 17](#_Toc168560104)

[2.1 Сущности базы данных 17](#_Toc168560105)

[2.2 Роли базы данных 21](#_Toc168560106)

[2.3 Функции и процедуры базы данных 22](#_Toc168560107)

[Вывод из конструкторского раздела 27](#_Toc168560108)

[3 Технологический раздел 28](#_Toc168560109)

[3.1 Выбор СУБД 28](#_Toc168560110)

[3.2 Выбор средств реализации приложения 29](#_Toc168560111)

[3.3 Реализация сущностей и ограничений целостности 30](#_Toc168560112)

[3.4 Реализация функций и процедур 32](#_Toc168560113)

[3.5 Реализация ролевой модели 38](#_Toc168560114)

[3.6 Тестирование функций и процедур 39](#_Toc168560115)

[3.7 Интерфейс доступа к базе данных 42](#_Toc168560116)

[Вывод из технологического раздела 45](#_Toc168560117)

[4 Исследовательский раздел 46](#_Toc168560118)

[4.1 Описание задачи исследования 46](#_Toc168560119)

[4.2 Параметры оборудования 46](#_Toc168560120)

[4.3 Результаты исследования 47](#_Toc168560121)

[Вывод из исследовательского раздела 48](#_Toc168560122)

[ЗАКЛЮЧЕНИЕ 49](#_Toc168560123)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 50](#_Toc168560124)

[ПРИЛОЖЕНИЕ А 51](#_Toc168560125)

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В данной расчетно-пояснительной записке применяются следующие сокращения и обозначения.

БД – база данных.

СУБД – система управления базами данных.

API – (application programming interface) программный интерфейс.

ВВЕДЕНИЕ

На сегодняшний день возможность анализировать и управлять личными финансами стала одним из основных требований успешного ведения домашнего хозяйства. В условиях быстро меняющейся экономической среды необходим удобный и простой инструмент, позволяющий отслеживать состояние банковских счетов. Одним из таких инструментов является приложение для сведения ежемесячного баланса доходов и расходов.

Во-первых, главная потребность в нем обусловлена растущим количеством и сложностью транзакций. Появилось не только огромное количество различных способов оплаты (наличный расчет, внутрибанковские переводы, эквайринг для физических и виртуальных карт, и т.д.), но и множество мелких расходов, что в совокупности приводит к невозможности быстро и точно определить, сколько было потрачено в той или иной категории.

Во-вторых, растущий темп жизни и повсеместная цифровизация общества требуют возможности своевременного онлайн-мониторинга своих финансов, при этом не затрачивающего большого количества времени.

Исходя из вышесказанного, можно сделать вывод о достаточной востребованности и актуальности подобного приложения на текущий момент.

Таким образом, целью данной курсовой работы является разработка программного обеспечения (приложения и базы данных) для анализа ежемесячных доходов и расходов. Для достижения поставленной цели необходимо решить следующие задачи:

1. проанализировать существующие решения;
2. сформулировать требования к разрабатываемому ПО и формализовать задачу;
3. проанализировать существующие базы данных и выбрать подходящие;
4. спроектировать базу данных, описать ее сущности, связи и процедуры;
5. реализовать приложение, позволяющее взаимодействовать с базой данных;
6. провести исследование быстродействия полученного приложения.

1 Аналитический раздел

В данном разделе будет проведен анализ предметной области, сравнительный анализ существующих аналогов, сформулированы требования к разрабатываемой базе данных и приложению, формализована и описана информация, подлежащая хранению в проектируемой базе данных и проведен анализ существующих баз данных на основе формализации данных.

На основе результатов вышеперечисленных действий также будут формализованы и описаны пользователи проектируемого приложения, а также построены ER-диаграмма сущностей в нотации Чена и диаграмма вариантов использования.

* 1. Анализ предметной области

Личный бюджет – персональный план доходов и расходов на определенный период – месяц, квартал, год [1]. Исходя из этого определения, можно считать процесс анализа ежемесячных транзакций процессом планирования бюджета. Тогда основной сущностью, лежащей в основе всей системы, очевидным образом становится транзакция – некоторая запись о факте перемещения денежных средств с одного банковского счета на другой. Также, в контексте анализа личных расходов и доходов кроме исходного счета, целевого счета и суммы, необходима дополнительная информация, включающая время транзакции (для отнесения ее к одному из периодических бюджетов), финансовую категорию, к которой принадлежит транзакция, а также дополнительную пользовательскую информацию.

Следующей важной сущностью является финансовая цель – некоторая цель, требующая наличия определенного количества свободных средств. В нашем случае будет достаточно привязать цель к определенному банковскому счету (т.н. «накопительный» счет) и позволить пользователю задать ожидаемое на нем значение.

Наконец, необходимо помнить о том, что личные накопления и траты могут включать несколько различных валют. Поэтому, для упрощения внесения информации о транзакциях между такими счетами, необходимо хранить также данные о используемых валютах и обменных курсах между ними.

* 1. Анализ существующих решений

Так как спрос на инструменты для анализа расходов и доходов не снижается на протяжении последнего десятилетия, на рынке уже существует множество готовых подходов и решений. Наиболее популярными и распространенными являются:

1. пользовательские реализации в Excel и других табличных процессорах;
2. встроенные аналитические модули в банковских приложениях;
3. Monefy;
4. Дзен-мани.

Сравнение данных решений (см. Таблица 1) будет проводится по следующим критериям:

1. возможность сохранять и анализировать информацию о счетах и транзакциях;
2. возможность запланировать бюджет;
3. возможность запланировать финансовые цели (например, накопление на покупку машины);
4. отсутствие жесткой привязки к платформе (десктоп, веб, мобильные устройства);
5. возможность анализировать транзакции без привязки к платежной системе;
6. возможность онлайн доступа;
7. бесплатность.

Таблица 1. Сравнение существующих решений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Excel и др. | Встроенные модули | Monefy | Дзен-мани |
| 1 | да | да | да | да |
| 2 | только вручную | не во всех | да | да |
| 3 | только вручную | не во всех | нет | да |
| 4 | нет | не во всех | нет | да |
| 5 | да | нет | да | да |
| 6 | нет | да | да | да |
| 7 | зависит от табличного процессора | не все | да | нет |

Ни одно из рассмотренных решений не удовлетворяет в полной мере выбранным критериям. Кроме того, стоит отметить, что на данный момент наиболее подходящее существующее решение (Дзен-мани) фактически находится в неработоспособном состоянии и не получало обновлений с 2017 года. Таким образом, реализуемое приложение выгодно отличается от существующих аналогов, удовлетворяя заявленным критериям.

* 1. Формализация задачи

Необходимо спроектировать базу данных для хранения информации о пользователях, банковских счетах, транзакциях, валютах, обменных курсах, транзакциях, финансовых категориях, бюджетах и целях.

Необходимо разработать приложение (веб-сервис), предоставляющее программный (в виде API) и пользовательский интерфейс для взаимодействия с базой данных, позволяющий просматривать, добавлять, изменять и удалять сущности, хранящиеся в базе данных, а также выполнять над ними дополнительные аналитические операции (подсчет готовности цели, расчет бюджета за период, поиск транзакций по категории).

Необходимо реализовать на стороне приложения механизм авторизации и предусмотреть наличие трех ролей: пользователь, администратор и система.

* 1. Формализация данных

На основе приведенного выше анализа предметной области выделены следующие категории данных:

1. пользователь;
2. банковский счет;
3. финансовая категория;
4. валюта;
5. обменный курс;
6. бюджет;
7. финансовая цель;
8. транзакция.

Подробная информация о каждой категории приведена в таблице 2.

Таблица 2. Категории данных

|  |  |
| --- | --- |
| Категория | Информация |
| Пользователь | Id, логин, хэш пароля, флаг администраторского доступа |
| Банковский счет | Id, id валюты, описание, текущее значение |
| Финансовая категория | Id, название категории |
| Валюта | Id, название валюты |
| Обменный курс | Id, id исходной валюты, id целевой валюты, коэффициент обмена |
| Бюджет | Id, id валюты, начало периода, конец периода, описание, начальный объем средств |
| Финансовая цель | Id, id банковского счета, описание, целевое значение счета |
| Транзакция | Id, id финансовой категории, id исходного счета, id целевого счета, значение транзакции, описание, временная метка |

На рисунке 1 приводится ER-диаграмма сущностей в нотации Чена.

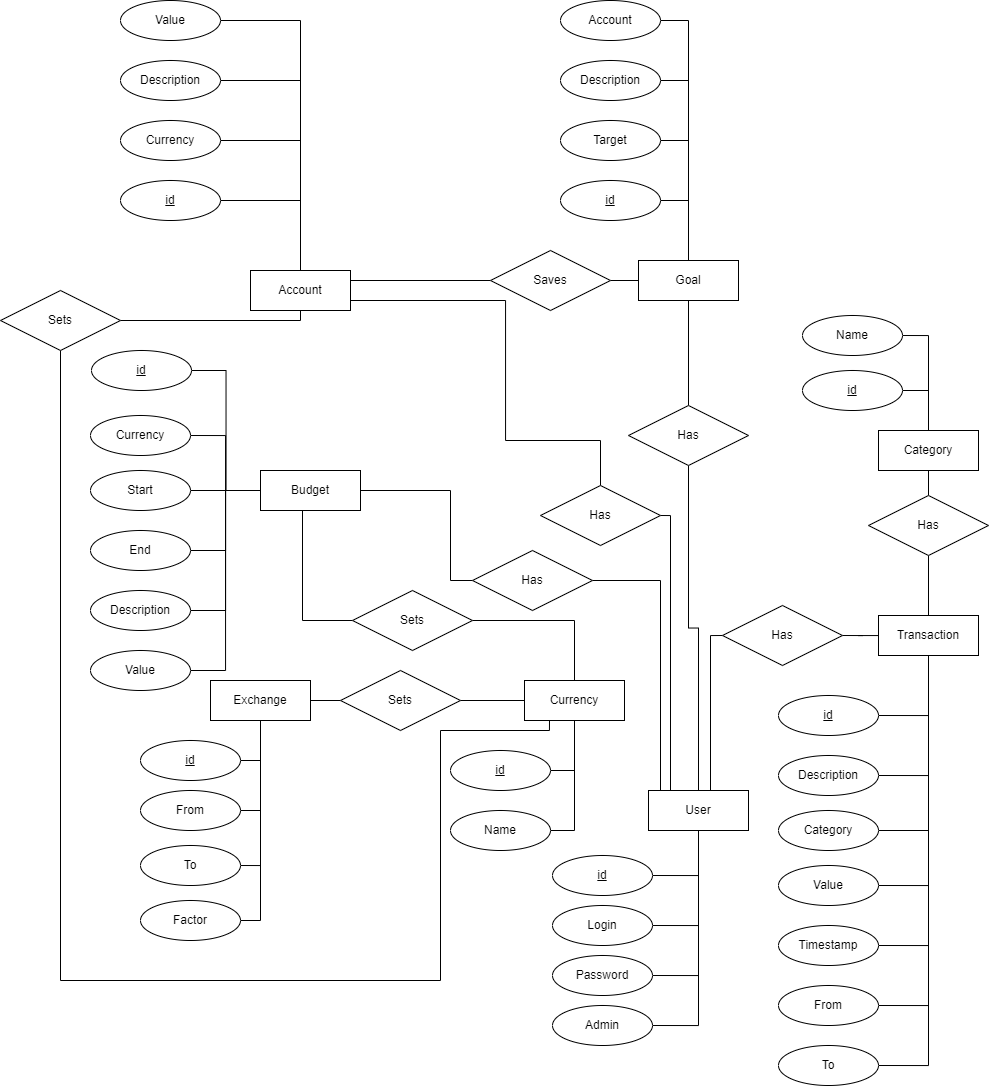


Рисунок 1. ER-диаграмма в нотации Чена

* 1. Формализация пользователей

Выделим категории пользователей, используемые для доступа к приложению:

1. пользователь
2. администратор
3. система (сервис).

Пользователь обладает базовым уровнем доступа, позволяющим вести бюджет: может просматривать валюты, категории и обменные курсы; просматривать и редактировать свою учетную запись; просматривать, создавать, изменять и удалять бюджеты, счета, цели и транзакции.

Администратор обладает всеми возможностями пользователя и дополнительно имеет доступ к созданию, изменению и удалению категорий, валют и обменных курсов. Категория существует для возможности поддержания системы в актуальном состоянии.

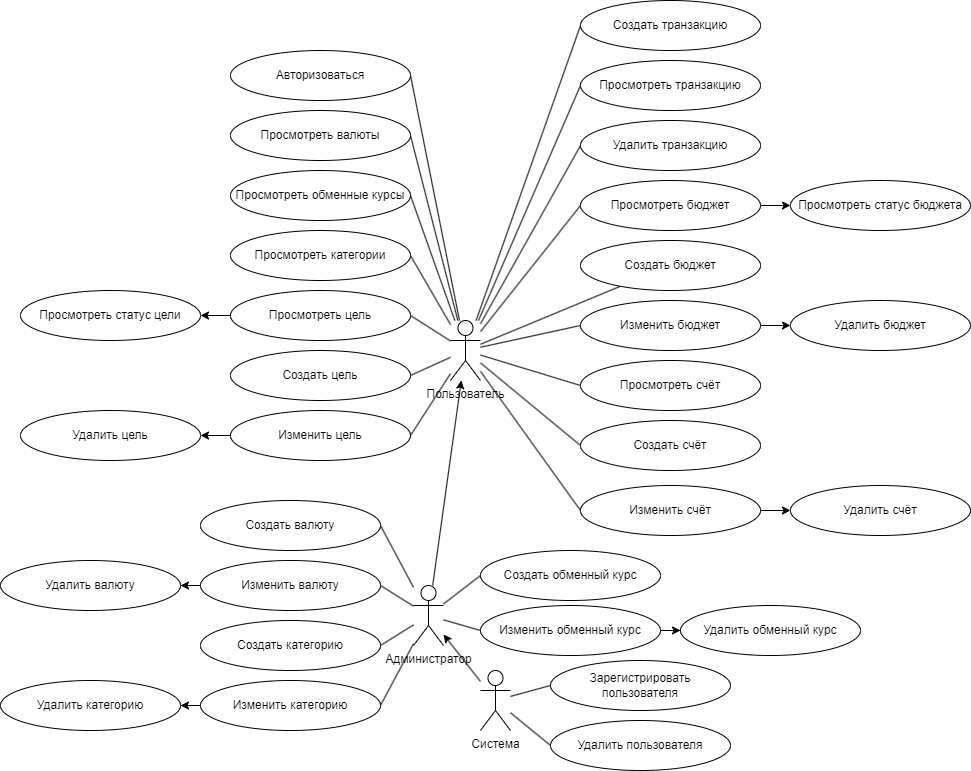
И, наконец, система обладает полным доступом к базе данных, может управлять существующими и создавать новые таблицы, создавать и удалять пользователей. Выполняет техническое обслуживания всей системы и регистрацию учетных записей пользователей. На рисунке 2 для описанных категорий приведена диаграмма вариантов использования.

Рисунок 2. Диаграмма вариантов использования

* 1. Анализ моделей баз данных

Термин «база данных» впервые появился более шестидесяти лет назад, и за это время получил большое количество интерпретаций, в том числе и от многих авторитетных авторов. Несмотря на это, общепризнанная единая формулировка отсутствует. Поэтому, учитывая специфику данной работы и избегая отдавать предпочтение какому-либо стороннему источнику, воспользуемся определением из курса лекций Ю. М. Гавриловой.

База данных – совокупность данных, организованных по определенным правилам, предусматривающая общие принципы описания, хранения и манипулирования, независимая от прикладных программ [2].

Для обеспечения возможности взаимодействия с базой данных используется система управления базами данных (СУБД) – программный компонент, отвечающий за обработку всех пользовательских запросов, составленных на формальном языке, абстрагированным от деталей реализации.

Одним из основных признаков, использующихся для классификации СУБД, является модель данных – абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, составляющих абстрактную машину, с которой взаимодействует пользователь [3].

Существуют следующие основные модели данных:

1. дореляционные;
2. реляционные;
3. постреляционные.
   * 1. Дореляционные модели

Данный тип включает модель на основе инвертированных списков, иерархические и сетевые модели.

Базы данных, организованные с помощью инвертированных списков, напоминают реляционные с тем отличием, что хранимые таблицы и пути доступа к ним видны пользователям [4].

Иерархические базы данных состоят из упорядоченного набора нескольких экземпляров одного типа дерева. Хранимые записи поддерживают связь «предок-потомок», в которой предок может иметь несколько потомков, но потомок может иметь только одного предка [4]. Такой подход не только создает определенные трудности с реализацией различных моделей отношений между сущностями, но и зачастую приводит к массовым изменениям при вставке или удалении.

Сетевая модель является расширением иерархической, позволяя потомку иметь несколько предков.

Главным недостатком дореляционных моделей, полностью нивелирующим их плюсы (эффективное использование памяти и пространство для оптимизаций), является жесткая зависимость от физической организации базы данных, приводящая к высокой сложности внесения каких-либо изменений.

* + 1. Реляционные модели

Реляционная модель обеспечивает представление данных в виде двумерной таблицы-отношения, состоящей из набора столбцов и строк [5]. Столбцы содержат информацию об атрибутах отношения – имя и тип; строки, отличные от заглавной строки, фактически являются кортежами, содержащими один компонент для каждого атрибута [5].

Главными плюсами данного подхода, обеспечившими ему широкую популярность, являются абстрагированность от физической реализации и простота поддержания целостности благодаря хранению информации о типах атрибутов.

Недостатки же включают требование атомарности для атрибутов и трудности с эффективным использованием памяти в отличие от дореляционных моделей.

* + 1. Постреляционные модели

Постреляционные модели представляют собой расширение реляционных, снимая ограничение на неатомарные атрибуты. Это делает данные модели более удобными для хранения составных сущностей, описанных с применением объектно-ориентированного подхода, и значительно повышает эффективность обработки в тех случаях, когда поле сущности является некой последовательностью данных (массив, список, множество и т.д.).

Основным недостатком модели является возросшая сложность поддержания целостности составных атрибутов.

Вывод из аналитического раздела

В данном разделе был проведен анализ предметной области, сравнительный анализ существующих аналогов, в результате которого ни один из них не удовлетворил в полной мере выбранным критериям.

На основе анализа предметной области были сформулированы требования к разрабатываемой базе данных и приложению, формализована и описана информация, подлежащая хранению в проектируемой базе данных и проведен анализ существующих баз данных на основе формализации данных.

По итогам анализа была выбрана реляционная модель, как наиболее соответствующая сформулированным требованиям (нет составных атрибутов, необходимо поддерживать целостность).

Также были формализованы и описаны пользователи проектируемого приложения и построены ER-диаграмма сущностей в нотации Чена и диаграмма вариантов использования.

1. Конструкторский раздел

В данном разделе будут описаны сущности проектируемой базы данных, ограничения целостности, проектируемые процедуры и функции в формате схемы и ролевая модель на уровне базы данных, а также приведена ER-диаграмма проектируемой базы данных.

* 1. Сущности базы данных

На основе результатов формализации категорий данных, приведенных в таблице 2, выделяются следующие таблицы:

1. Currencies – валюты;
2. Exchanges – обменные курсы валют;
3. Categories – финансовые категории;
4. Users – пользователи;
5. Budgets – планируемые бюджеты;
6. Accounts – банковские счета;
7. Goals – финансовые цели;
8. Transactions – транзакции.

Подробная информация для каждой сущности о структуре и типах столбцов, а также ограничениях целостности приведена в таблицах 3, 4, 5, 6, 7, 8, 9, 10.

Таблица 3. Currencies

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор валюты |
| name | text | not null | Название валюты |

Таблица 4. Exchanges

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор обменного курса |
| from | int | foreign key, not null | Исходная валюта |
| to | int | foreign key, not null | Целевая валюта |
| factor | double precision | not null | Обменный коэффициент |

Таблица 5. Categories

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор категории |
| name | text | not null | Название категории |

Таблица 6. Users

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор пользователя |
| login | text | not null | Логин пользователя |
| password | text | not null | Хэш пароля |
| admin | bool | not null | Флаг прав администратора |

Таблица 7. Budgets

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор бюджета |
| owner | int | foreign key, not null | Пользователь, создавший бюджет |
| currency | int | foreign key, not null | Валюта бюджета |
| start | timestamp | not null | Начало бюджетного периода |
| end | timestamp | not null | Конец бюджетного периода |
| description | text | not null | Описание бюджета |
| value | double precision | not null | Начальное значение бюджета (количество средств, доступное с начала периода) |

Таблица 8. Accounts

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор счета |
| owner | int | foreign key, not null | Пользователь, создавший счет |
| currency | int | foreign key, not null | Валюта счета |
| description | text | not null | Описание счета |
| value | double precision | not null | Текущее значение счета |

Таблица 9. Goals

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор цели |
| owner | int | foreign key, not null | Пользователь, создавший цель |
| account | int | foreign key, not null | Счет, к которому привязана цель |
| description | text | not null | Описание цели |
| target | double precision | not null | Сумма, которую необходимо достигнуть на счете |

Таблица 10. Transactions

|  |  |  |  |
| --- | --- | --- | --- |
| Атрибут | Тип | Ограничения | Комментарий |
| id | serial | primary key, not null | Идентификатор транзакции |
| owner | int | foreign key, not null | Пользователь, создавший транзакцию |
| category | int | foreign key, not null | Категория транзакции |
| from | int | foreign key | Исходный счет (null при внешнем поступлении) |
| to | int | foreign key | Целевой счет (null при внешнем переводе) |
| value | double precision | not null | Значение |
| description | text | not null | Описание транзакции |
| timestamp | timestamp | not null | Временная отметка |

Тип serial здесь представляет авто-инкрементируемую версию типа int.

Невозможность создать транзакцию, у которой поля from и to одновременно будут иметь значение null, контролируется приложением.

На рисунке 3 приведена ER-диаграмма проектируемой базы данных.

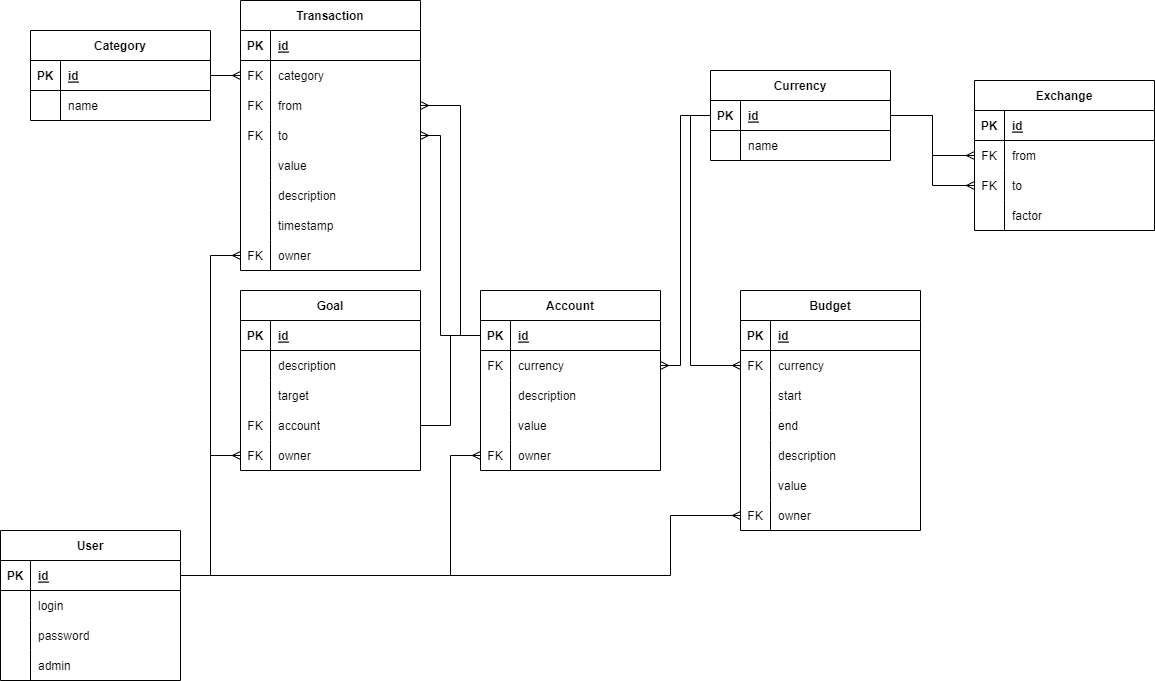


Рисунок 3. ER-диаграмма базы данных

* 1. Роли базы данных

На основе результатов формализации пользователей базы данных, проведённой в аналитическом разделе, определена ролевая модель, включающая три роли:

1. user – имеет право доступа select для таблиц currencies, exchanges, categories, users; имеет право доступа update для таблицы users и имеет полные права доступа к остальным таблицам (не может добавлять и удалять пользователей и изменять общедоступные сущности);
2. admin – имеет права доступа select и update для таблицы users; имеет полные права доступа к остальным таблицам (имеет доступ ко всем действиям, кроме добавления и удаления пользователей);
3. service – имеет полные права доступа ко всем таблицам и базе данных.
   1. Функции и процедуры базы данных

Для удобства работы с базой данных, а также корректной организации вычислений бюджета и целей спроектированы нижеописанные функции и процедуры.

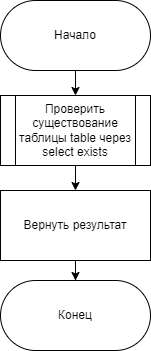
1. check\_table\_exists (table) – функция, проверяющая существование таблицы table. В случае успеха возвращает true, иначе false. Схема приведена на рисунке 4.
2. check\_tables\_exist (tables) – функция, проверяющая существование всех таблиц из массива tables. Если все таблицы существуют, возвращает true, иначе false. Схема приведена на рисунке 5.
3. convert\_value (from, to, value) – функция, преобразующая значение из валюты from в валюту to. Если валюты равны, значение не изменяется. Схема приведена на рисунке 6.
4. get\_budget\_status (id, owner) – функция, вычисляющая статистику для бюджета с идентификатором id, принадлежащего пользователю owner. Определяет, сколько средств за планируемый период было потрачено и получено, а также считает общую сумму. Не учитывает внутренние транзакции (переводы между счетами, принадлежащими пользователю). Схема приведена на рисунке 7.
5. get\_goal\_status (id) – функция, вычисляющая статистику для финансовой цели с идентификатором id. Определяет, сколько процентов цели выполнено, сколько средств накоплено и сколько осталось накопить. Схема приведена на рисунке 8.
6. add\_account\_value (id, currency, value) – процедура, изменяющая значение счёта с идентификатором id на значение value в валюте currency. Если указанная валюта не соответствует валюте счета, функция дополнительно произведёт конвертацию. Схема приведена на рисунке 9.
7. add\_transaction (owner, category, from, to, value, description, timestamp) – функция, выполняющая добавление транзакции с указанными атрибутами и применение её к содержимому счетов from и to. Если транзакция внешняя, её сумма будет считаться указанной в валюте счета, иначе будет произведена конвертация. Функция возвращает идентификатор созданной транзакции. Схема приведена на рисунке 10.
8. del\_transaction (id) – процедура, выполняющая удаление транзакции с идентификатором id и её откат для задействованных счетов. Выполняет преобразования, обратные тем, что делает функция add\_transaction. Схема приведена на рисунке 11.

Рисунок 4. Функция check\_table\_exists

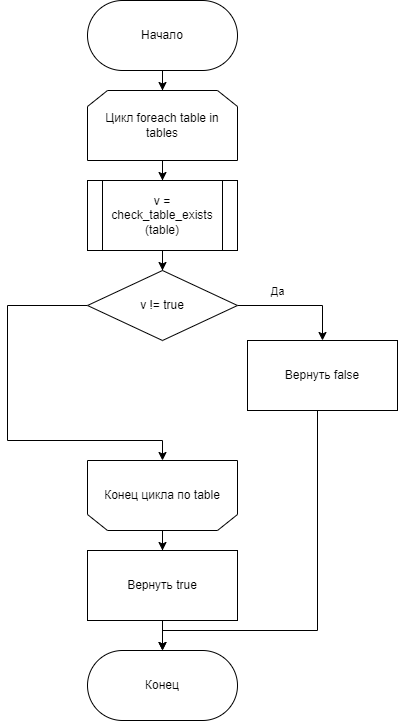
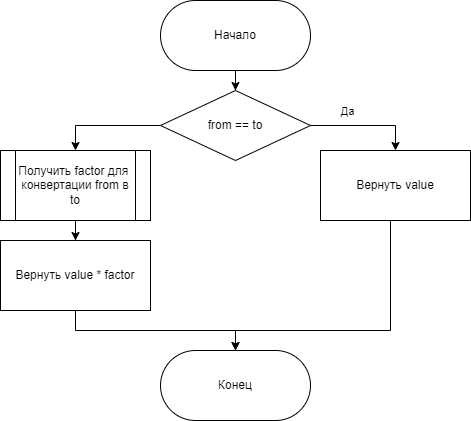


Рисунок 5. Функция check\_tables\_exist

Рисунок 6. Функция convert\_value

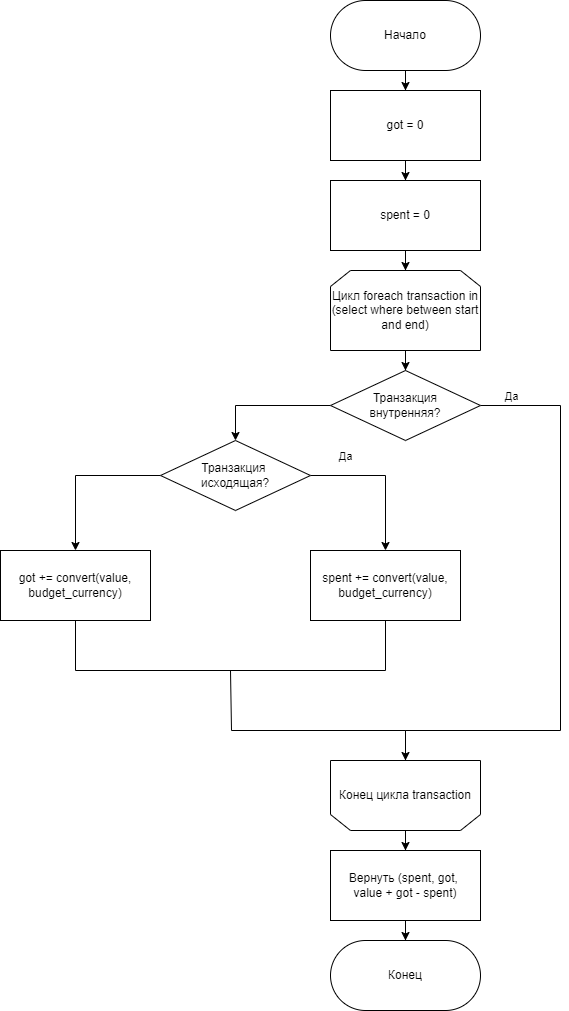
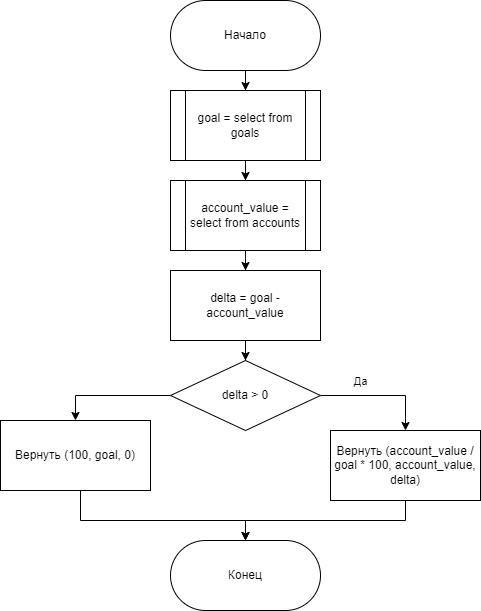


Рисунок 7. Функция get\_budget\_status

Рисунок 8. Функция get\_goal\_status

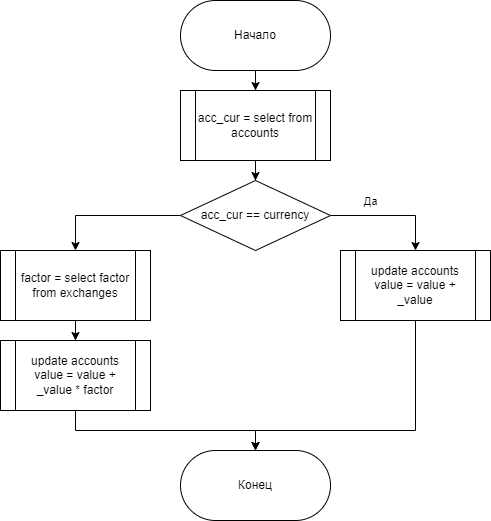


Рисунок 9. Процедура add\_account\_value

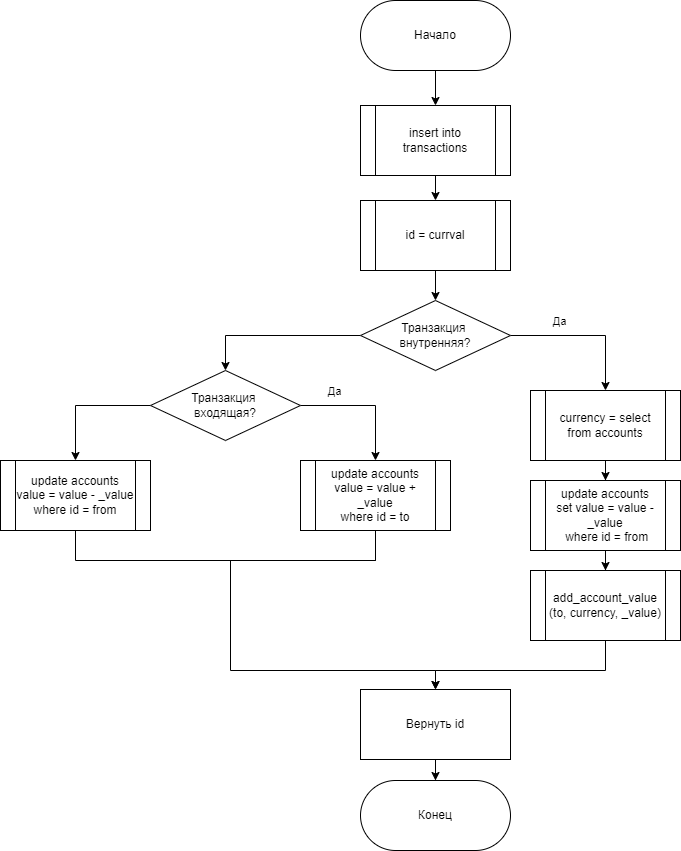
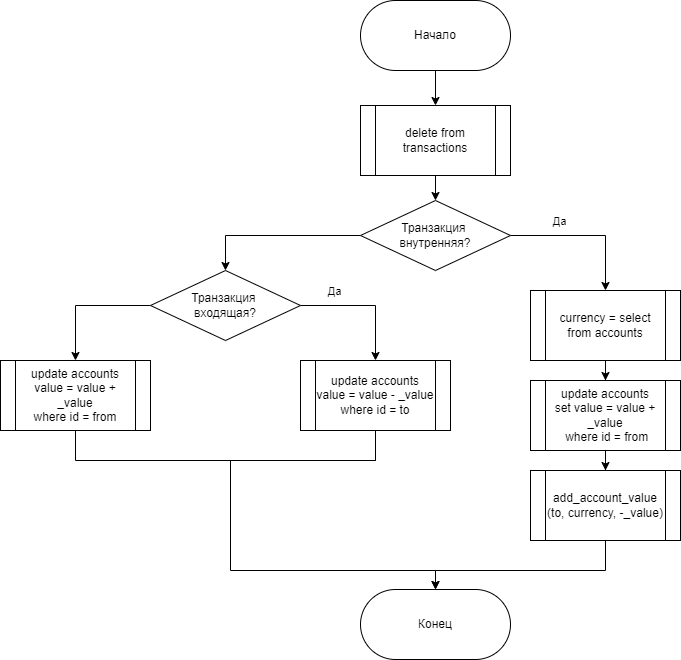
Рисунок 10. Функция add\_transaction

Рисунок 11. Процедура del\_transaction

Вывод из конструкторского раздела

В данном разделе были описаны сущности проектируемой базы данных, прокомментированы их атрибуты и приведены ограничения целостности.

Были перечислены проектируемые процедуры и функции, описаны и представлены в формате блок-схемы.

Была описана ролевая модель на уровне базы данных.

Также была приведена ER-диаграмма проектируемой базы данных.

1. Технологический раздел

В данном разделе будет рассмотрен выбор средств реализации базы данных и приложения, описаны сущности реализованной базы данных, описаны реализованные ограничения целостности базы данных, приведена реализация ролевой модели, приведены все реализованные процедуры и функции, описаны методы тестирования и тесты для всех разработанных на стороне базы функций. Также будет рассмотрен интерфейс доступа к базе данных.

* 1. Выбор СУБД

В результате анализа моделей баз данных была выбрана реляционная модель, поэтому выбор СУБД будет осуществляться среди соответствующих ей решений. На сегодняшний день одними из самых популярных являются:

1) PostgreSQL;

2) MariaDB;

3) Oracle.

Сравним данные СУБД по нижеперечисленным критериям.

1. Распространение под свободной (open-source) лицензией – одно из главных требований, так как в противном случае при изменении условий проприетарной лицензии в одностороннем порядке или других похожих ситуациях возникнет нарушение авторского права.
2. Наличие хорошей документации в свободном доступе – критерий, оказывающий значительное влияние на выбор в пользу того или иного СУБД, так как присутствие качественной документации значительно ускоряет и упрощает разработку.
3. Высокая стабильность и надёжность – также критерий, серьёзно влияющий на выбор.
4. Строгая типизация – необязательный, но очень желательный критерий. Отсутствие неявных преобразований и строгая проверка соответствий типов помогает избежать большого количества ошибок.

Результаты сравнения вышеперечисленных СУБД приведены в таблице 11.

Таблица 11. Сравнение СУБД

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | PostgreSQL | MariaDB | Oracle |
| Open-source | да | да | нет |
| Документация | да | да | часть закрыта |
| Высокая надёжность | да | нет | да |
| Строгая типизация | да | нет | нет |

Таким образом, исходя из вышеприведенных данных, выбран PostgreSQL, как решение, в полной мере удовлетворяющее всем заявленным критериям.

* 1. Выбор средств реализации приложения

В качестве языка программирования, на котором будет реализовано веб-приложение, выбран язык Java [6] версии 11. Этот выбор обусловлен следующими причинами:

1. основной парадигмой языка является ООП;
2. язык и запускающие его виртуальные машины распространяются под свободными лицензиями;
3. наличие встроенного механизма для работы с реляционными базами данных (JDBC);
4. широкий выбор фреймворков для реализации веб-приложения.
5. широкий выбор фреймворков для автоматического тестирования.

В качестве среды разработки была выбрана IDE IntelliJ IDEA [7], на что повлияли следующие причины:

1. данная IDE специально создана для разработки на языке Java;
2. интеграция с большим количеством систем сборки и фреймворков;
3. развитий статический анализатор и продвинутое автодополнение;
4. предоставление графического интерфейса для java отладчика.

Для реализации клиента, предоставляющего интерфейс командной строки, выбран язык Python, как удобный и надежный инструмент для отправки запросов.

* 1. Реализация сущностей и ограничений целостности

В листингах 1, 2, 3, 4, 5, 6, 7 и 8 представлено создание таблиц и задание для них ограничений целостности.

Листинг 1. Создание таблицы currencies

1. create table currencies
2. (
3. id serial not null primary key,
4. name text not null
5. );

Листинг 2. Создание таблицы exchanges

1. create table exchanges
2. (
3. id serial not null primary key,
4. \_from int references currencies (id) on delete cascade not null,
5. \_to int references currencies (id) on delete cascade not null,
6. factor double precision not null
7. );

Листинг 3. Создание таблицы categories

1. create table categories
2. (
3. id serial not null primary key,
4. name text not null
5. );

Листинг 4. Создание таблицы users

1. create table users
2. (
3. id serial not null primary key,
4. login text not null,
5. password text not null,
6. admin bool not null
7. );

Листинг 5. Создание таблицы budgets

1. create table budgets
2. (
3. id serial not null primary key,
4. owner int references currencies (id) on delete cascade not null,
5. currency int references currencies (id) on delete cascade not null,
6. \_start timestamp not null,
7. \_end timestamp not null,
8. description text not null,
9. value double precision not null
10. );

Листинг 6. Создание таблицы accounts

1. create table accounts
2. (
3. id serial not null primary key,
4. owner int references users (id) on delete cascade not null,
5. currency int references currencies (id) on delete cascade not null,
6. description text not null,
7. value double precision not null
8. );

Листинг 7. Создание таблицы goals

1. create table goals
2. (
3. id serial not null primary key,
4. owner int references users (id) on delete cascade not null,
5. account int references accounts (id) on delete cascade not null,
6. description text not null,
7. target double precision not null
8. );

Листинг 8. Создание таблицы transactions

1. create table transactions
2. (
3. id serial not null primary key,
4. owner int references users (id) on delete cascade not null,
5. category int references categories (id) on delete cascade not null,
6. \_from int references accounts (id) on delete cascade,
7. \_to int references accounts (id) on delete cascade,
8. value double precision not null,
9. description text not null,
10. \_timestamp timestamp not null
11. );
    1. Реализация функций и процедур

В листингах 9, 10, 11, 12 представлен исходный код вышеописанных хранимых функций и процедур.

Листинг 9. Функции check\_table\_exists и check\_tables\_exist

1. create function check\_table\_exists(in tbl text)
2. returns bool as $$
3. begin
4. return (select exists (select \* from information\_schema.tables where table\_name = tbl and table\_schema = 'public'));
5. end;
6. $$
7. language plpgsql;
8. create function check\_tables\_exist(in tbls text[])
9. returns bool as $$
10. declare
11. e text;
12. begin
13. foreach e in array tbls
14. loop
15. if not check\_table\_exists(e) then
16. return false;
17. end if;
18. end loop;
19. return true;
20. end;
21. $$
22. language plpgsql;

Листинг 10. Функция get\_goal\_status

1. create type goal\_status as (percents double precision, reached double precision, remained double precision);
2. create function get\_goal\_status(in \_id int)
3. returns goal\_status as $$
4. declare
5. \_acc\_id int;
6. \_goal double precision;
7. \_acc double precision;
8. \_delta double precision;
9. begin
10. \_goal = (select target from goals where id = \_id);
11. \_acc\_id = (select account from goals where id = \_id);
12. \_acc = (select value from accounts where id = \_acc\_id);
13. \_delta = \_goal - \_acc;
14. if (\_delta > 0) then
15. return (select cast(row(\_acc / \_goal \* 100, \_acc, \_delta) as goal\_status));
16. end if;
17. return (select cast(row(100, \_goal, 0) as goal\_status));
18. end;
19. $$
20. language plpgsql;

Листинг 11. Функции convert\_value и get\_budget\_status

1. create function convert\_value(in \_f int, in \_t int, in \_value double precision)
2. returns double precision as $$
3. declare
4. \_factor double precision;
5. begin
6. if \_f = \_t then
7. return \_value;
8. end if;
9. \_factor = (select factor from exchanges where \_from = \_f and \_to = \_t);
10. return \_value \* \_factor;
11. end;
12. $$
13. language plpgsql;

Листинг 11. Продолжение

1. create type budget\_status as (spent double precision, got double precision, total double precision);
2. create function get\_budget\_status(in \_id int, in \_user int)
3. returns budget\_status as $$
4. declare
5. \_t record;
6. \_from int;
7. \_to int;
8. \_value double precision;
9. \_spent double precision;
10. \_got double precision;
11. \_b\_cur int;
12. \_cur int;
13. \_s timestamp;
14. \_e timestamp;
15. begin
16. \_s = (select \_start from budgets where id = \_id);
17. \_e = (select \_end from budgets where id = \_id);
18. \_value = (select value from budgets where id = \_id);
19. \_b\_cur = (select currency from budgets where id = \_id);
20. \_got = 0;
21. \_spent = 0;
22. for \_t in (select \* from transactions where \_timestamp between \_s and \_e)
23. loop
24. \_from = \_t.\_from;
25. \_to = \_t.\_to;
26. -- Skip inner transactions
27. if (\_from is not null) and (\_to is not null) then
28. continue;
29. end if;
30. -- Spent
31. if (\_to is null) then
32. \_cur = (select currency from accounts where id = \_t.\_from);
33. \_spent = \_spent + (select convert\_value(\_cur, \_b\_cur, \_t.value));
34. continue;
35. end if;
36. -- Got
37. \_cur = (select currency from accounts where id = \_t.\_to);
38. \_got = \_got + (select convert\_value(\_cur, \_b\_cur, \_t.value));
39. end loop;
40. return (select cast(row(\_spent, \_got, \_value + \_got - \_spent) as budget\_status));
41. end; $$ language plpgsql;

Листинг 12. Функция add\_transaction и процедуры add\_account\_value, del\_transaction

1. create procedure add\_account\_value(
2. in \_id int,
3. in \_currency int,
4. in \_value double precision
5. ) as $$
6. declare
7. \_factor double precision;
8. \_t\_cur int;
9. begin
10. \_t\_cur = (select currency from accounts where id = \_id);
11. if \_t\_cur = \_currency then
12. update accounts
13. set value = value + \_value
14. where id = \_id;
15. return;
16. end if;
17. \_factor = (select factor from exchanges where \_from = \_currency and \_to = \_t\_cur);
18. update accounts
19. set value = value + \_value \* \_factor
20. where id = \_id;
21. end;
22. $$
23. language plpgsql;

Листинг 12. Продолжение

1. create function add\_transaction(
2. in \_owner int,
3. in \_category int,
4. in \_from int,
5. in \_to int,
6. in \_value double precision,
7. in \_description text,
8. in \_timestamp timestamp
9. )
10. returns int as $$
11. declare
12. \_id int;
13. \_currency int;
14. begin
15. insert into transactions
16. (owner, category, \_from, \_to, value, description, "\_timestamp")
17. values (\_owner, \_category, \_from, \_to, \_value, \_description, \_timestamp);
18. \_id = (select currval('transactions\_id\_seq'));
19. -- If from is not null and to is not null, then use \_from.currency as main currency
20. if (\_from is not null) and (\_to is not null) then
21. \_currency = (select currency from accounts where id = \_from);
22. update accounts
23. set value = value - \_value
24. where id = \_from;
25. call add\_account\_value(\_to, \_currency, \_value);
26. return \_id;
27. end if;
28. -- If from is null and to is not null, then just update to with value
29. if (\_from is null) and (\_to is not null) then
30. update accounts
31. set value = value + \_value
32. where id = \_to;
33. return \_id;
34. end if;
35. -- If from is not null and to is null, then just update from with -value
36. update accounts
37. set value = value - \_value
38. where id = \_from;
39. return \_id;
40. end;
41. $$
42. language plpgsql;

Листинг 12. Продолжение

1. create procedure del\_transaction(in \_id int)
2. as $$
3. declare
4. \_f int;
5. \_t int;
6. \_currency int;
7. \_v double precision;
8. begin
9. \_f = (select \_from from transactions where id = \_id);
10. \_t = (select \_to from transactions where id = \_id);
11. \_v = (select value from transactions where id = \_id);
12. delete from transactions where id = \_id;
13. -- If from is not null and to is not null, then use \_from.currency as main currency
14. if (\_f is not null) and (\_t is not null) then
15. \_currency = (select currency from accounts where id = \_f);
16. update accounts
17. set value = value + \_v
18. where id = \_f;
19. call add\_account\_value(\_t, \_currency, -\_v);
20. return;
21. end if;
22. -- If from is null and to is not null, then sub value from to
23. if (\_f is null) and (\_t is not null) then
24. update accounts
25. set value = value - \_v
26. where id = \_t;
27. return;
28. end if;
29. -- If from is not null and to is null, then add value to from
30. update accounts
31. set value = value + \_v
32. where id = \_f;
33. end;
34. $$
35. language plpgsql;
    1. Реализация ролевой модели

Создание ролей service, admin и user, а также присвоение им соответствующих прав приведено в листингах 13, 14 и 15.

Листинг 13. Создание роли user

1. create role \_user;
2. grant select on currencies to \_user;
3. grant select on exchanges to \_user;
4. grant select on categories to \_user;
5. grant select on users to \_user;
6. grant update on users to \_user;
7. grant all privileges on budgets to \_user;
8. grant all privileges on accounts to \_user;
9. grant all privileges on goals to \_user;
10. grant all privileges on transactions to \_user;
11. grant usage, select on all sequences in schema public to \_user;

Листинг 14. Создание роли admin

1. create role \_admin;
2. grant all privileges on currencies to \_admin;
3. grant all privileges on exchanges to \_admin;
4. grant all privileges on categories to \_admin;
5. grant select on users to \_admin;
6. grant update on users to \_admin;
7. grant all privileges on budgets to \_admin;
8. grant all privileges on accounts to \_admin;
9. grant all privileges on goals to \_admin;
10. grant all privileges on transactions to \_admin;
11. grant usage, select on all sequences in schema public to \_admin;

Листинг 15. Создание роли service

1. create role \_service;
2. grant all privileges on database course to \_service;
3. grant all privileges on currencies to \_service;
4. grant all privileges on exchanges to \_service;
5. grant all privileges on categories to \_service;
6. grant all privileges on users to \_service;
7. grant all privileges on budgets to \_service;
8. grant all privileges on accounts to \_service;
9. grant all privileges on goals to \_service;
10. grant all privileges on transactions to \_service;
11. grant usage, select on all sequences in schema public to \_service;
    1. Тестирование функций и процедур

Реализованные функции и процедуры тестируются с помощью фреймворка JUnit. Перед запуском тестов каждой функции создаётся и инициализируется отдельная база данных, удаляющаяся после завершения. Для взаимодействия с СУБД на стороне Java используется JDBC драйвер, предоставляемый командой разработчиков PostgreSQL [8].

Реализовано 4 группы тестов, находящихся в соответствующих классах.

1. ToolsTest – тесты функций check\_table\_exists и check\_tables\_exist. Описание тестовых кейсов приведено в таблице 12.
2. GoalTest – тесты функции get\_goal\_status. Описание тестовых кейсов приведено в таблице 13.
3. BudgetTest – тесты функций convert\_value и get\_budget\_status. Предварительно создаются тестовая категория, две валюты и обменные курсы между ними. Описание тестовых кейсов приведено в таблице 14.
4. TransactionTest – тесты функции add\_transaction и процедур add\_account\_value и del\_transaction. Подробное описание тестовых кейсов приведено в таблице 15.

Таблица 12. Тесты check\_table\_exists и check\_tables\_exist

|  |  |
| --- | --- |
| Имя кейса | Описание |
| testTableExists | Вызывает функцию check\_table\_exists для заведомо существующей таблицы и ожидает true |
| testTableNotExists | Вызывает функцию check\_table\_exists для заведомо несуществующей таблицы и ожидает false |
| testTablesExist | Вызывает функцию check\_tables\_exist для заведомо существующих таблиц и ожидает true |
| testTablesNotExist | Вызывает функцию check\_tables\_exist для заведомо несуществующих таблиц и ожидает false |

Таблица 13. Тесты get\_goal\_status

|  |  |
| --- | --- |
| Имя кейса | Описание |
| testUnreached | Создает счет и цель, причем значение цели больше содержимого счета, ожидает верный расчет |
| testReached | Создает счет и цель, причем значение цели меньше содержимого счета, ожидает верный расчет |

Таблица 14. Тесты convert\_value и get\_budget\_status

|  |  |
| --- | --- |
| Имя кейса | Описание |
| testConvertToSelf | Вызывает convert\_value, указывая одинаковые исходную и целевую валюты, ожидает конвертируемое значение без изменений |
| testConvert | Вызывает convert\_value, указывая различные исходную и целевую валюты, ожидает корректно конвертированное значение |
| testEmptyBudget | Создает пустой бюджет, вызывает для него get\_budget\_status и ожидает нулевой расчет. |
| testPositiveBudget | Создает бюджет и входящие транзакции за его период, вызывает get\_budget\_status и ожидает сумму значений транзакций (доходов) |
| testNegativeBudget | Создает бюджет и исходящие транзакции за его период, вызывает get\_budget\_status и ожидает сумму значений транзакций (расходов) |
| testBudget | Создает бюджет и исходящие, и входящие транзакции за его период, вызывает get\_budget\_status и ожидает корректную разность доходов и расходов |
| testTransactionsOutOfRange | Создает бюджет и транзакции, часть из которых не попадает в его период, вызывает get\_budget\_status и ожидает, что расчет будет включать только попадающие в период транзакции |

Таблица 15. Тесты add\_transaction, add\_account\_value, del\_transaction

|  |  |
| --- | --- |
| Имя кейса | Описание |
| testAddSameAccountValue | Создает счет, вызывает add\_account\_value для валюты, соответствующей валюте счета, ожидает корректного значения счета |
| testAddDiffAccountValue | Создает счет, вызывает add\_account\_value для валюты, отличающейся от валюты счета, ожидает корректного значения счета |
| testSameInner | Создает 2 счета в одной валюте и транзакцию между ними, ожидает корректного значения счетов, после чего удаляет транзакцию и ожидает исходных значений |
| testDiffInner | Создает 2 счета в разных валютах и транзакцию между ними, ожидает корректного значения счетов, после чего удаляет транзакцию и ожидает исходных значений |
| testToInner | Создает счет и входящую транзакцию на него, ожидает корректного значения счета, после чего удаляет транзакцию и ожидает исходных значений |
| testFromInner | Создает счет и исходящую транзакцию с него, ожидает корректного значения счета, после чего удаляет транзакцию и ожидает исходных значений |

Для запуска тестов необходимо собрать приложение и обеспечить доступ к корректно настроенному серверу СУБД PostgreSQL. Все тесты успешно пройдены.

* 1. Интерфейс доступа к базе данных

Для запуска веб-приложения необходимо воспользоваться подготовленными скриптами для утилиты docker [9], либо развернуть приложение и сервер СУБД PostgreSQL вручную.

Взаимодействие с веб-приложением осуществляется посредством API, реализованным на основе протокола HTTP. В отсутствие графического клиента конечный пользователь может использовать любую утилиту, позволяющую отправлять HTTP-запросы, например, curl [10].

Кроме того, для демонстрации возможностей приложения на языке Python был реализован клиент, предоставляющий интерфейс командной строки. Демонстрация взаимодействия с приложением с его помощью показана ниже на рисунках 12 – 18: вход в аккаунт, создание валюты, создание категории, создание двух счетов, создание нескольких транзакций, создание цели, бюджета и расчет статистики для них.

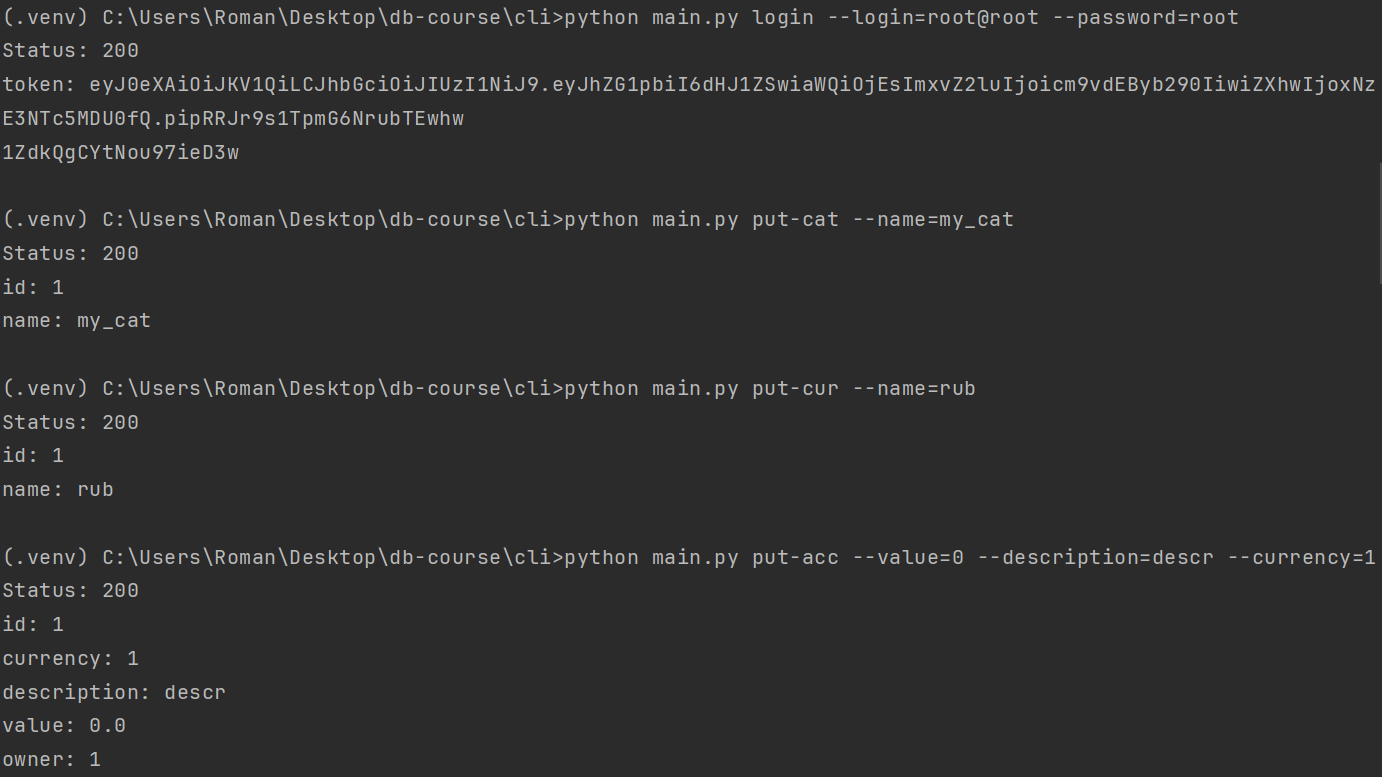


Рисунок 12. Вход, создание категории, валюты и первого счета

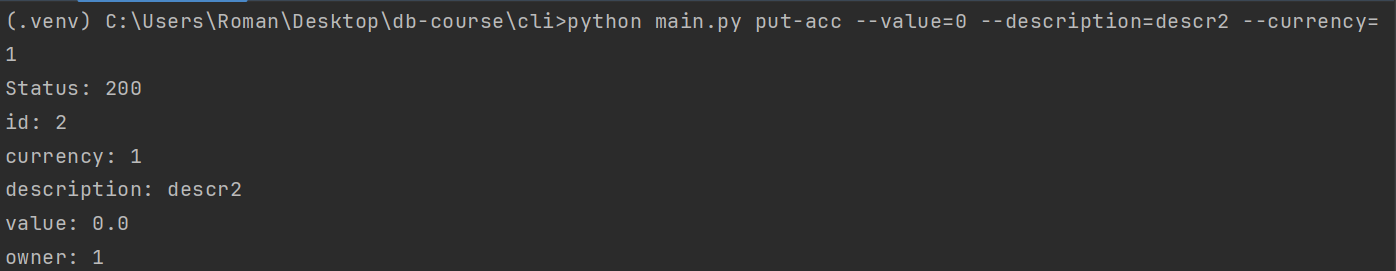


Рисунок 13. Создание второго счета

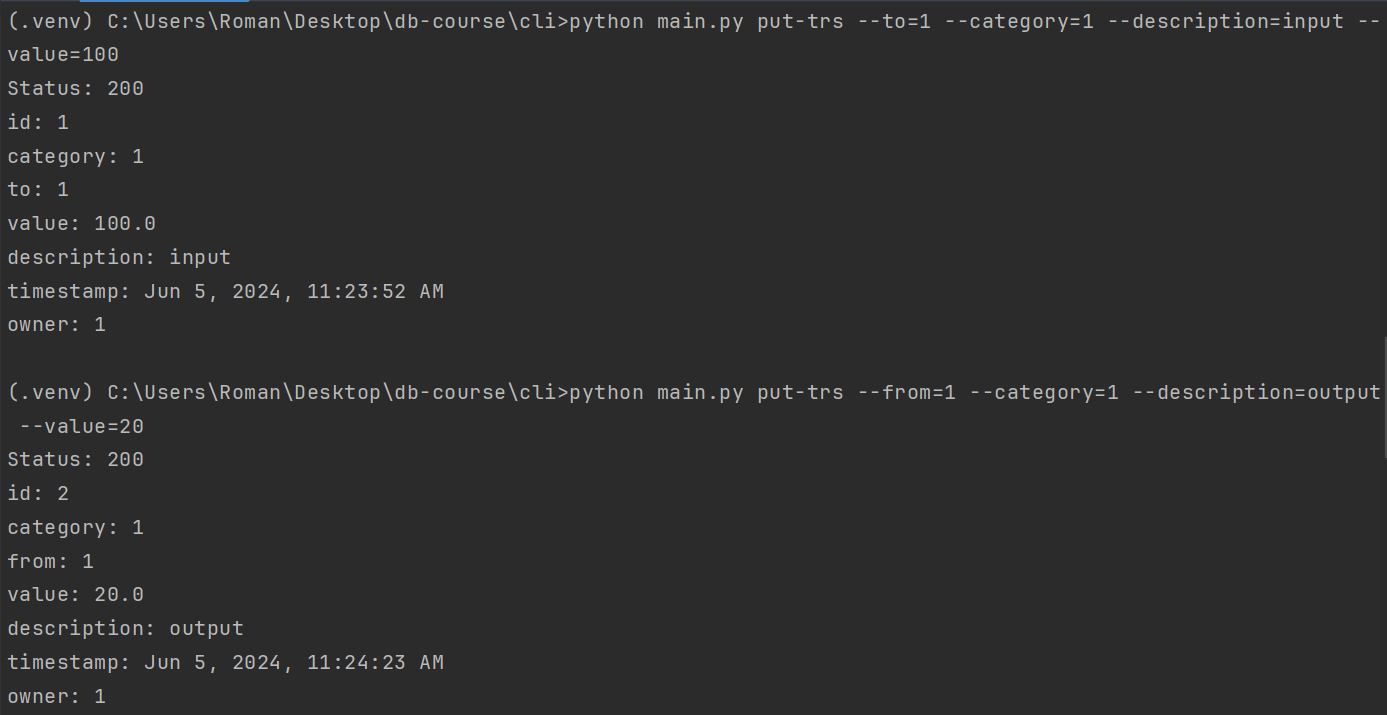


Рисунок 14. Создание транзакций для первого счета

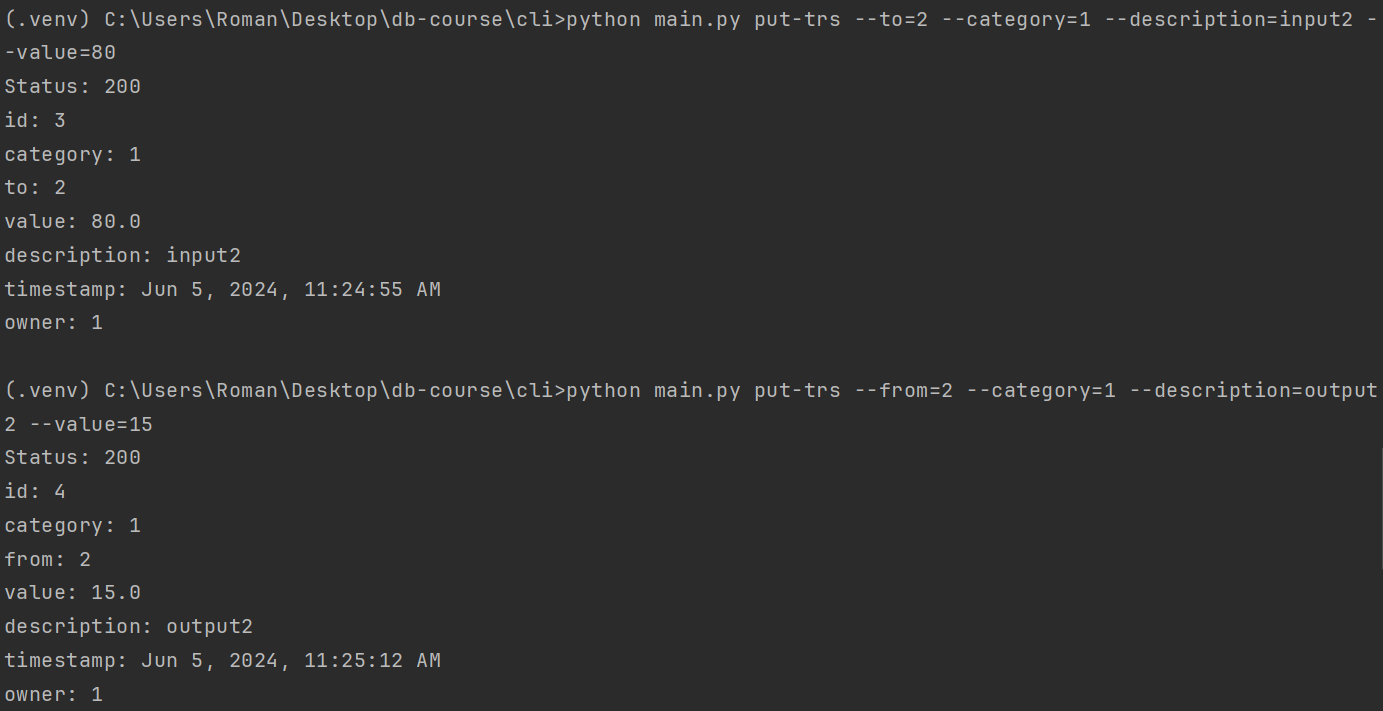


Рисунок 15. Создание транзакций для второго счета

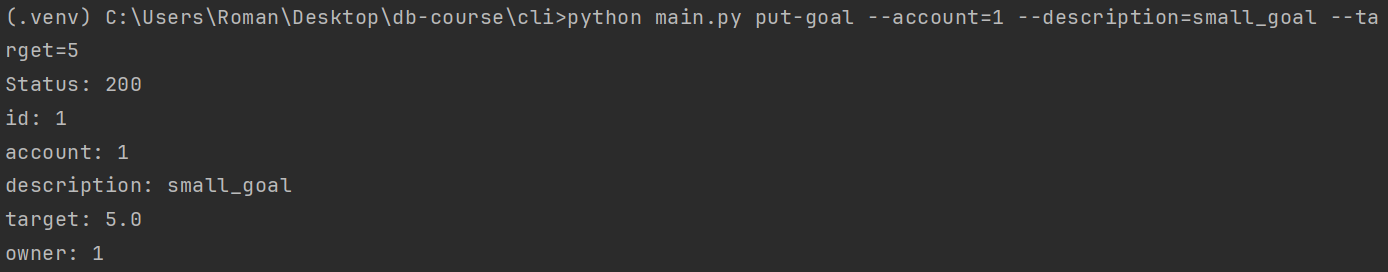


Рисунок 16. Создание цели

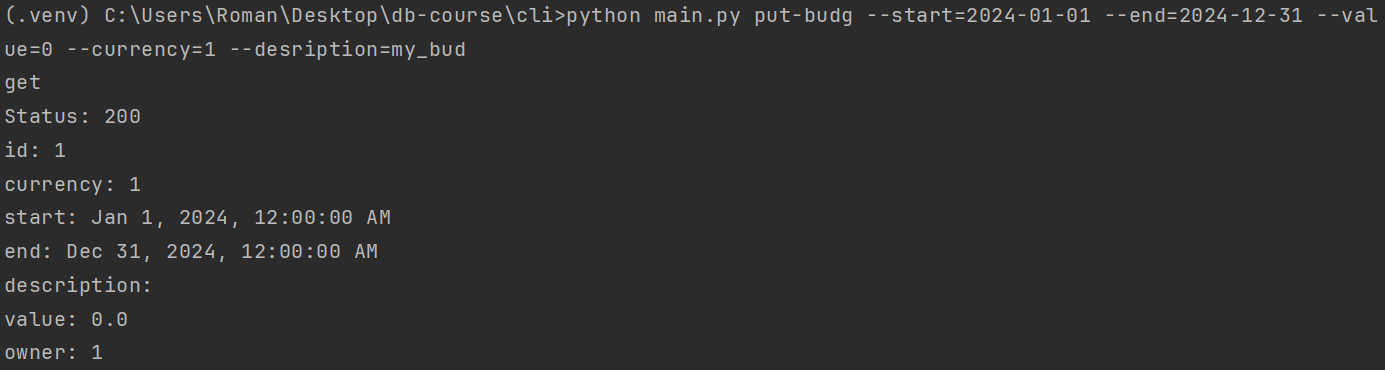


Рисунок 17. Создание бюджета за годовой период

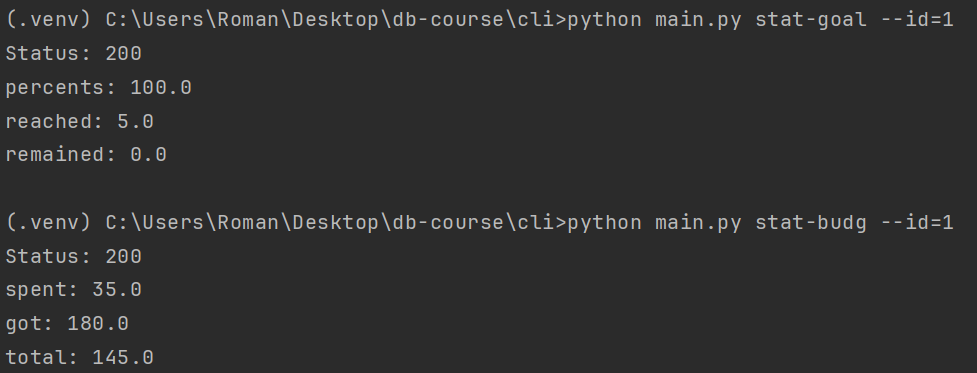


Рисунок 18. Подсчет статистики для цели и бюджета

Вывод из технологического раздела

В данном разделе был рассмотрен выбор средств реализации базы данных и приложения, описаны сущности реализованной базы данных, описаны реализованные ограничения целостности базы данных, приведена реализация ролевой модели, приведен исходный код всех реализованных процедур и функций, описаны методы тестирования и тестовые кейсы для всех разработанных на стороне базы функций. Также было продемонстрировано взаимодействие с базой данных посредством реализованного консольного интерфейса доступа.

1. Исследовательский раздел

В данном разделе будет произведено исследование быстродействия полученного приложения. Будет измерено время выполнения операций над данными в зависимости от количества данных.

* 1. Описание задачи исследования

Так как основной целью приложения для анализа ежемесячных расходов и доходов фактически является вычисление бюджета за искомый период, было решено произвести замеры времени работы функции get\_budget\_status в зависимости от количества сохраненных в базе транзакций.

Эксперимент выполняется следующим образом. Для каждой серии замеров создается и инициализируется отдельная база данных. Затем замеры производятся с использованием партиции по месячному периоду, в который попадают транзакции, и без. Для проведения каждой итерации замеров в базу данных помещается бюджет на 1 месяц и N транзакций, случайным образом распределенных по всему году. После чего вычисляется время вызова функции get\_budget\_status.

Результаты замеров записываются в csv файлы.

* 1. Параметры оборудования

Для проведения исследования быстродействия использовался персональный компьютер со следующими характеристиками:

1. процессор – Intel Core i5-9400F;
2. объём оперативной памяти – 40 ГБ.

Во время проведения замеров компьютер был нагружен только фоновыми процессами системы и непосредственно программой, выполняющей замеры.

Замеры выполнялись с использованием JVM, входящей в состав OpenJDK 11 [6] и докер-образа PostgreSQL версии 16 [11].

* 1. Результаты исследования

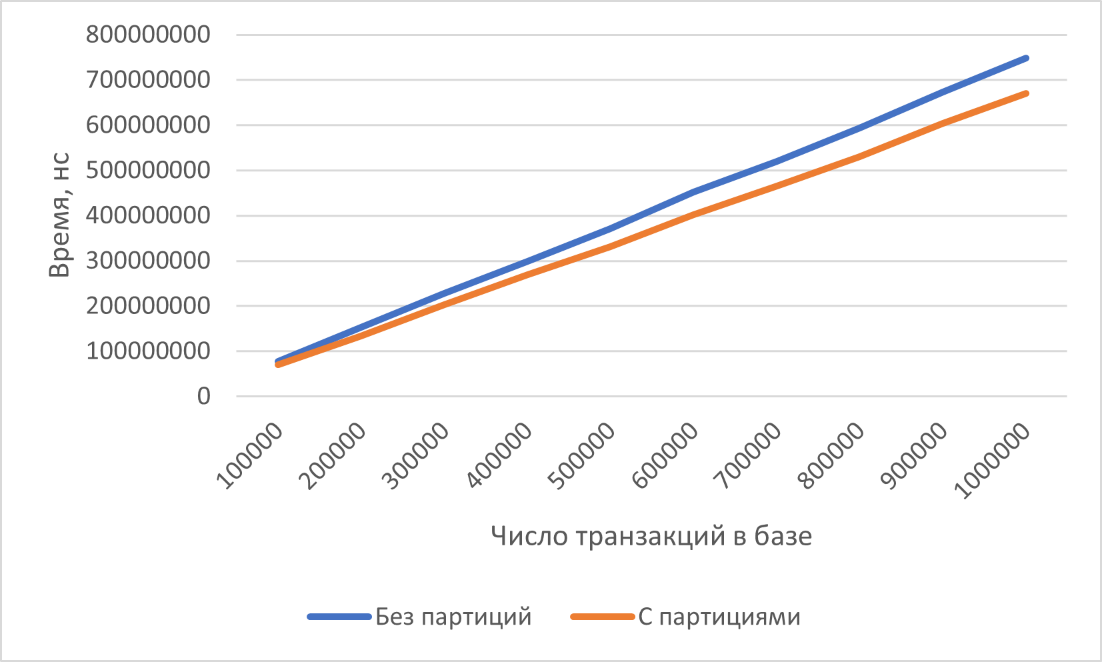
Графики зависимости времени выполнения функции get\_budget\_status от количества транзакций с партициями и без представлены на рисунках 19, 20 и 21.

Рисунок 19. Замеры для N от 100000 до 1000000

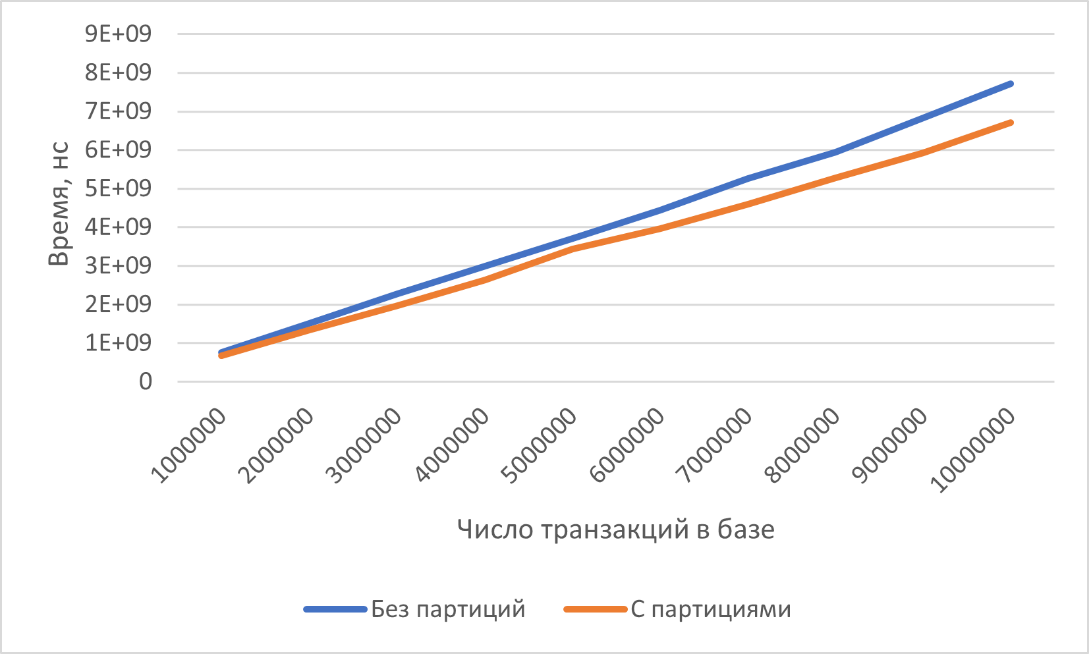


Рисунок 20. Замеры для N от 1000000 до 10000000

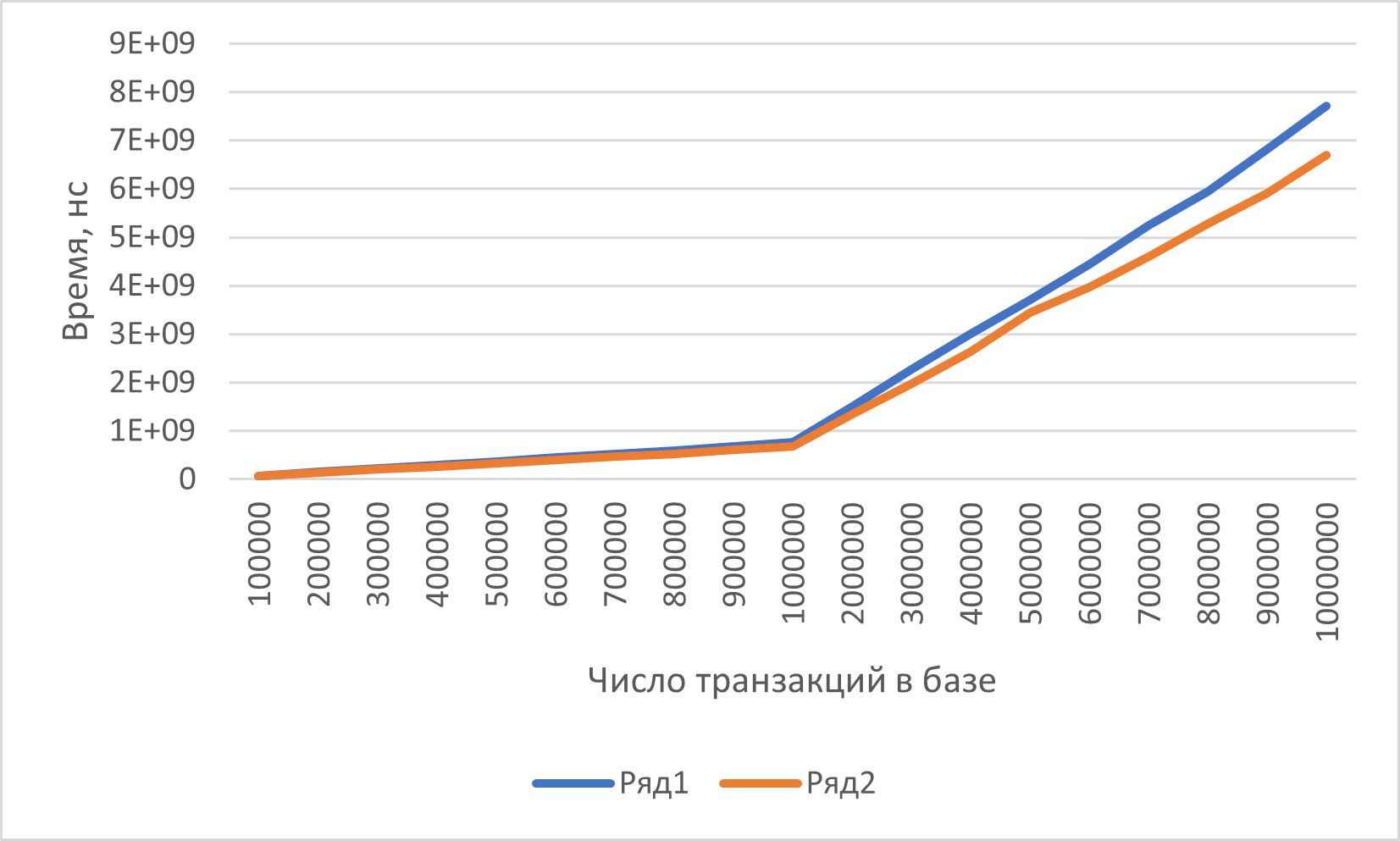


Рисунок 21. Сводный график

Вывод из исследовательского раздела

В данном разделе было произведено исследование быстродействия полученного приложения. Была определена зависимость между количеством транзакций в базе данных и временем, затрачиваемым функцией get\_budget\_status на вычисления статистики бюджета.

Измерения показали, что трудоёмкость вычисления растет линейно. Также можно сделать вывод о том, что на таблицах малого размера (менее двухсот тысяч) партиционирование не дает значимого выигрыша. С ростом количества записей в таблице эффект от применения партиционирования по периоду, соответствующему периоду бюджета, значительно возрастает.

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы была реализована база данных для анализа ежемесячных расходов и доходов, а также приложение для доступа к ней.

Был проведен анализ предметной области, сформулированы требования к базе данных и приложению, проведен анализ существующих баз данных на основе проведенной формализации данных.

На основе результатов формализации были описаны сущности базы данных, ограничения целостности, ролевая модель и спроектированы процедуры и функции.

Разработанная база данных и приложение удовлетворяют всем сформулированным требованиям и обеспечивают возможность анализа ежемесячных расходов и доходов. Подготовлено и успешно пройдено функциональное тестирование реализованных процедур. Также продемонстрировано взаимодействие с приложением посредством реализованного интерфейса командной строки, показаны основные возможности.

В ходе выполнения исследовательской части работы была получена зависимость между количеством транзакций в базе данных и временем на их обработку, а также определена целесообразность применения партиционирования по дате для транзакций.

База данных и приложение, разработанное в рамках курсового проекта, имеют по меньшей мере два направления дальнейшего развития:

1. расширение существующего функционала и подготовка к публикации на общедоступный сервер;
2. разработка графического интерфейса для более удобного взаимодействия с базой данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Финансовая грамотность: личное финансовое планирование [Электронный ресурс] – Режим доступа: <https://provtech.ru/about/finansovaya-gramotnost/Tema-1.4.pdf> (дата обращения: 01.06.2024)
2. Ю. М. Гаврилова. Первая лекция по дисциплине Базы Данных [Электронный ресурс] – Режим доступа: https://lks.bmstu.ru/eufs/4d25b186-bd19-11e6-80c8-005056960017/03-01-2024-Лекция\_01.\_Основные\_понятия\_и\_определения.pdf (дата обращения: 01.06.2024)
3. Дж. Дейт К. Введение в системы баз данных: 7-е издание. Пер. с англ. Ю. Г. Гордиенко, В.В. Репецкого, А.В. Слепцова. – «Вильямс», 2001. – C. 1073.
4. С. Д. Кузнецов. Основы современных баз данных. – Центр Информационных Технологий, 1998.
5. Джеффри Д. Ульман, Дженнифер Уидом. Введение в системы баз данных. – «Лори», 2006. – С. 379.
6. The Java Language Specification [Электронный ресурс] – Режим доступа: <https://docs.oracle.com/javase/specs/jls/se11/html/index.html> (дата обращения 14.11.2023)
7. IntelliJ IDEA [Электронный ресурс] – Режим доступа: <https://www.jetbrains.com/ru-ru/idea/> (дата обращения 14.11.2023)
8. PostgreSQL JDBC Driver [Электронный ресурс] – Режим доступа: https://jdbc.postgresql.org (дата обращения 22.04.2024)
9. Docker [Электронный ресурс] – Режим доступа: https://www.docker.com (дата обращения 22.04.2024)
10. Curl [Электронный ресурс] – Режим доступа: https://curl.se (дата обращения 22.04.2024)
11. PostgreSQL docker image [Электронный ресурс] – Режим доступа: https://hub.docker.com/\_/postgres (дата обращения 22.04.2024)

ПРИЛОЖЕНИЕ А