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

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.

1.1 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.

1.2 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.

1.3 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.

1.4 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.

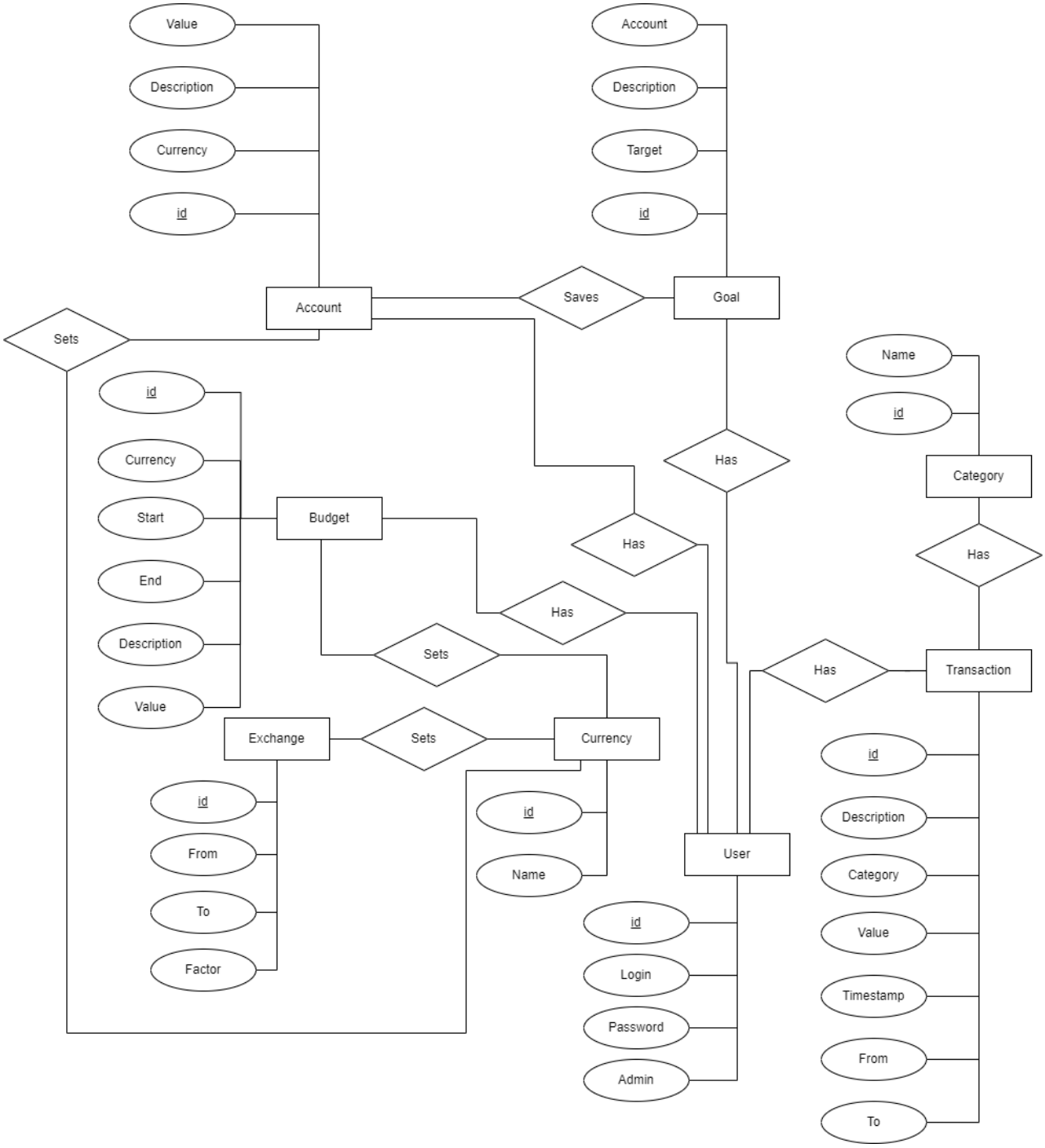

Рисунок 1. ER-диаграмма в нотации Чена

1.5 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.

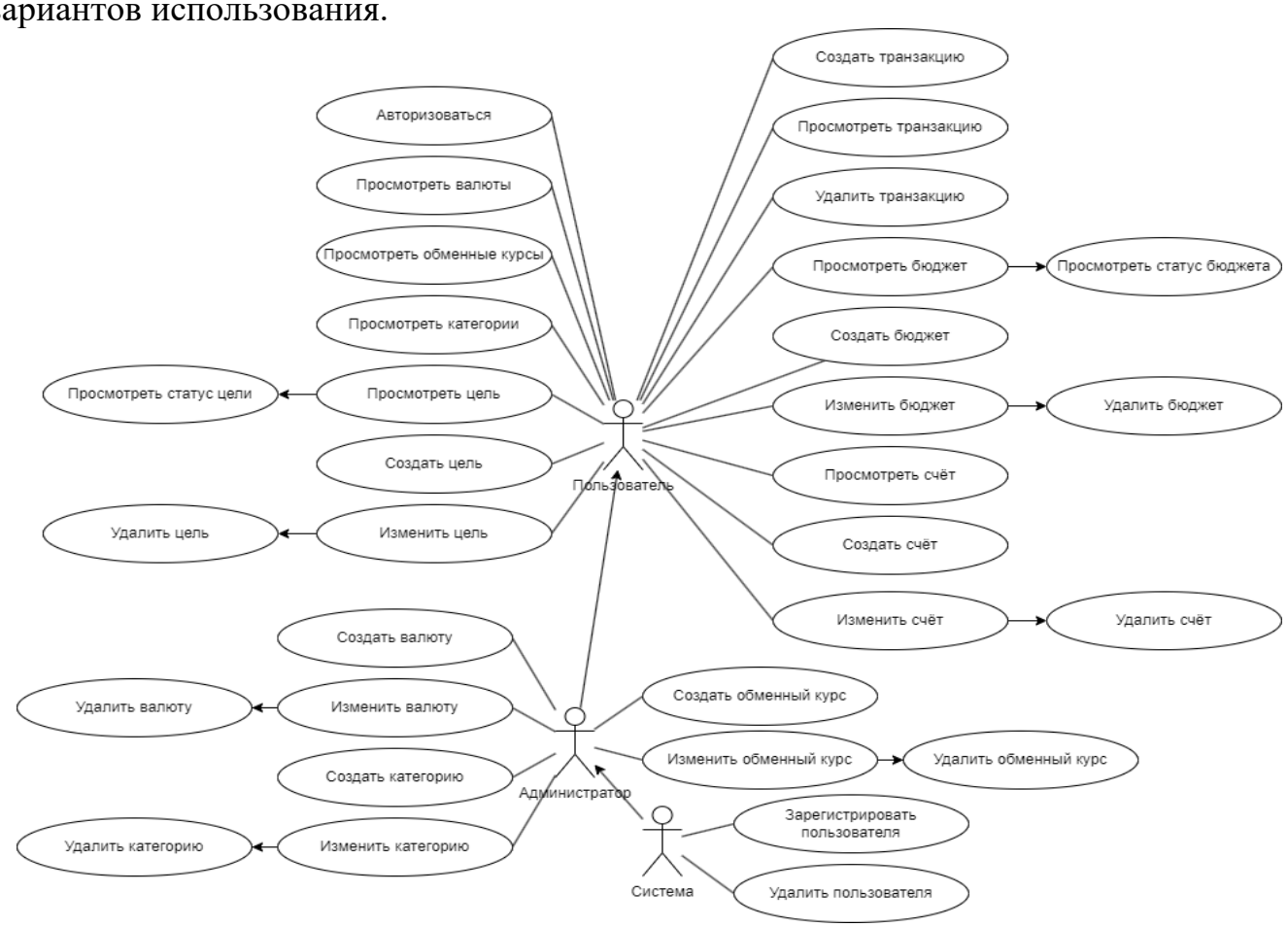

Рисунок 2. Диаграмма вариантов использования

1.6 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.

1.6.1 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.

1.6.2 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.

1.6.3 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.

2 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.

2.1 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.

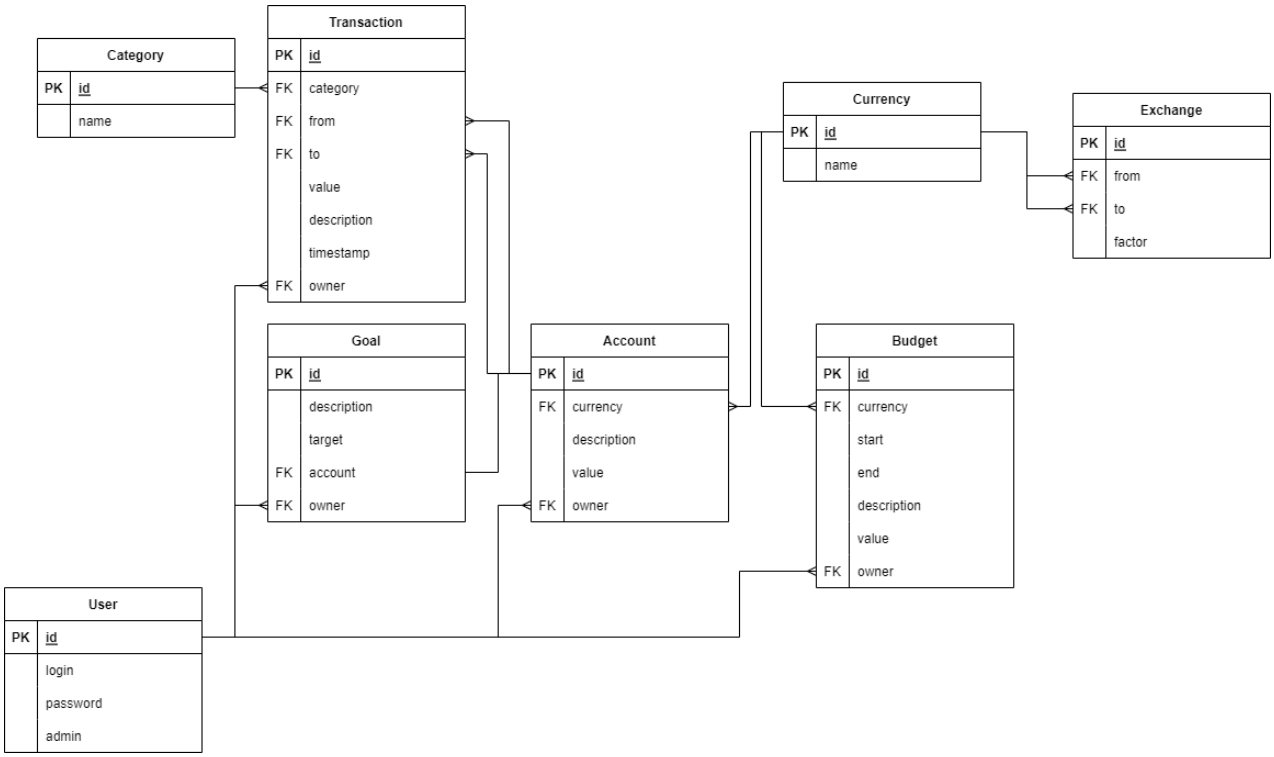

Рисунок 3. ER-диаграмма базы данных

2.2 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.

2.3 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

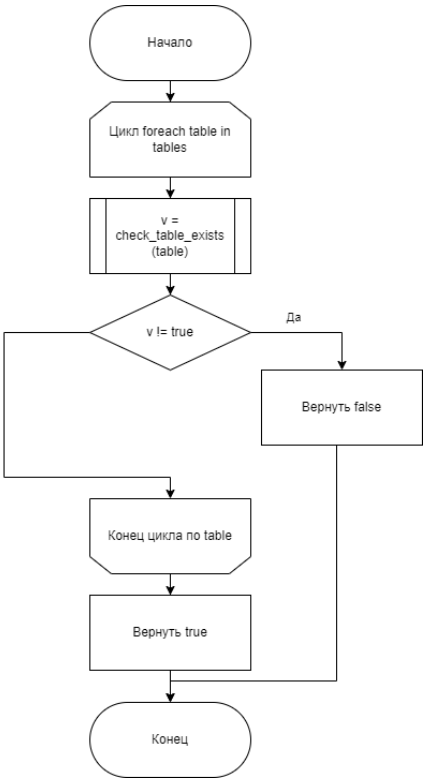

Рисунок 5. Функция check_tables_exist

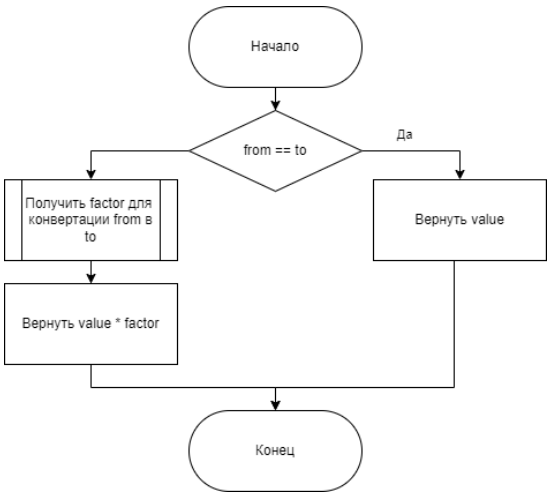

Рисунок 6. Функция convert_value

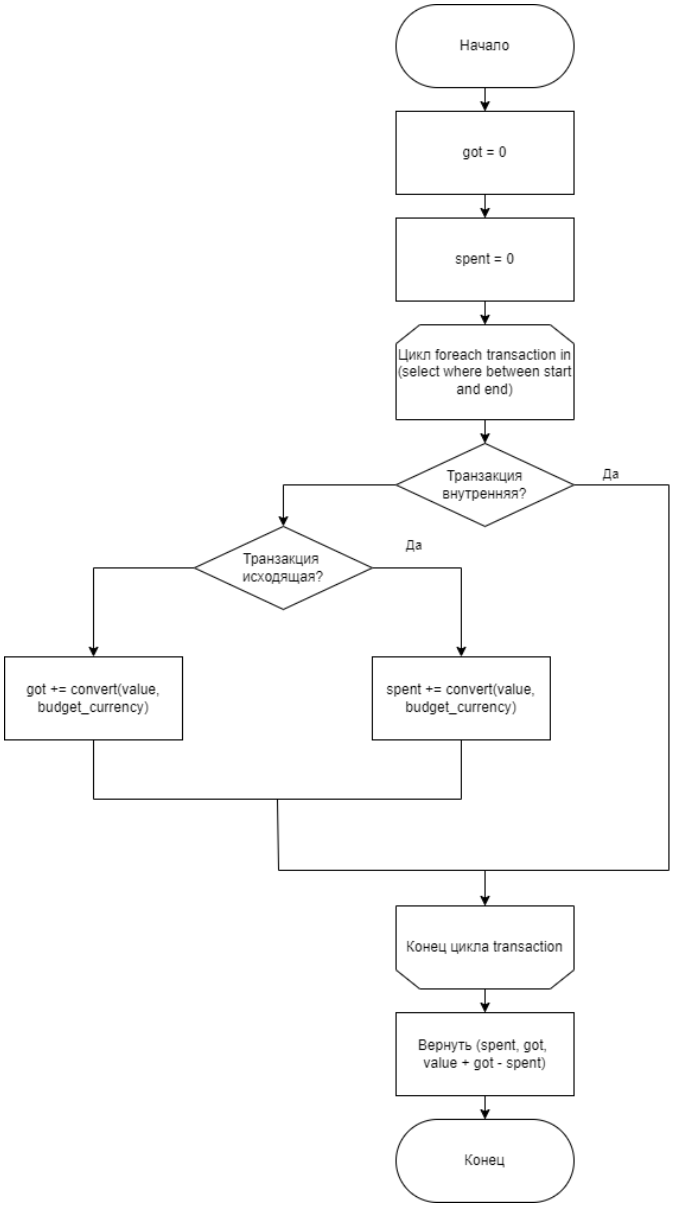

Рисунок 7. Функция get_budget_status

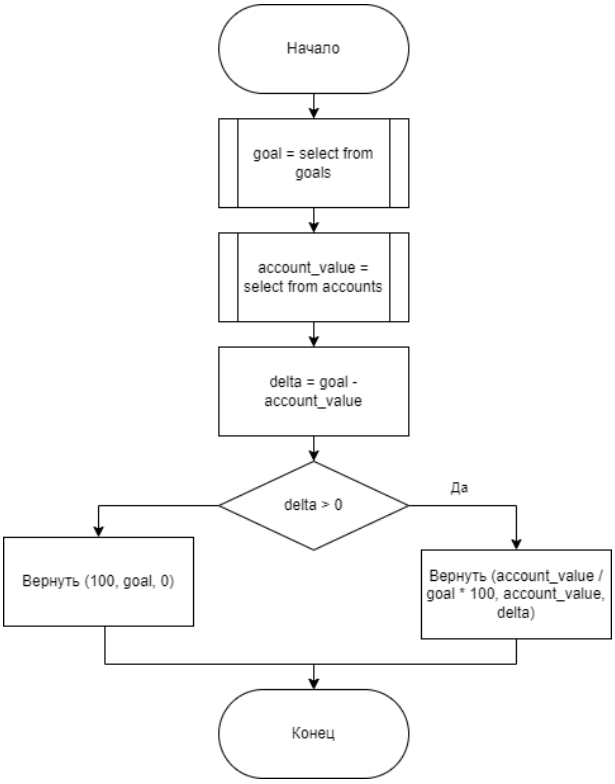

Рисунок 8. Функция get_goal_status

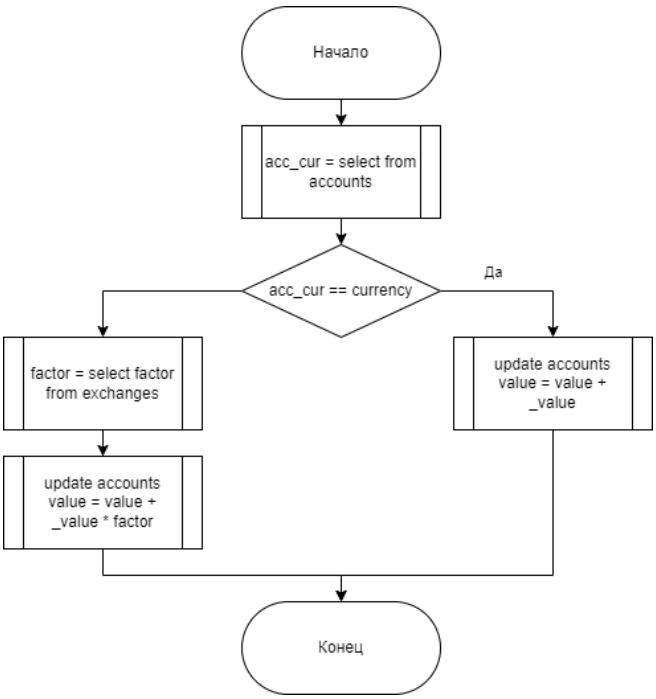

Рисунок 9. Процедура add_account_value

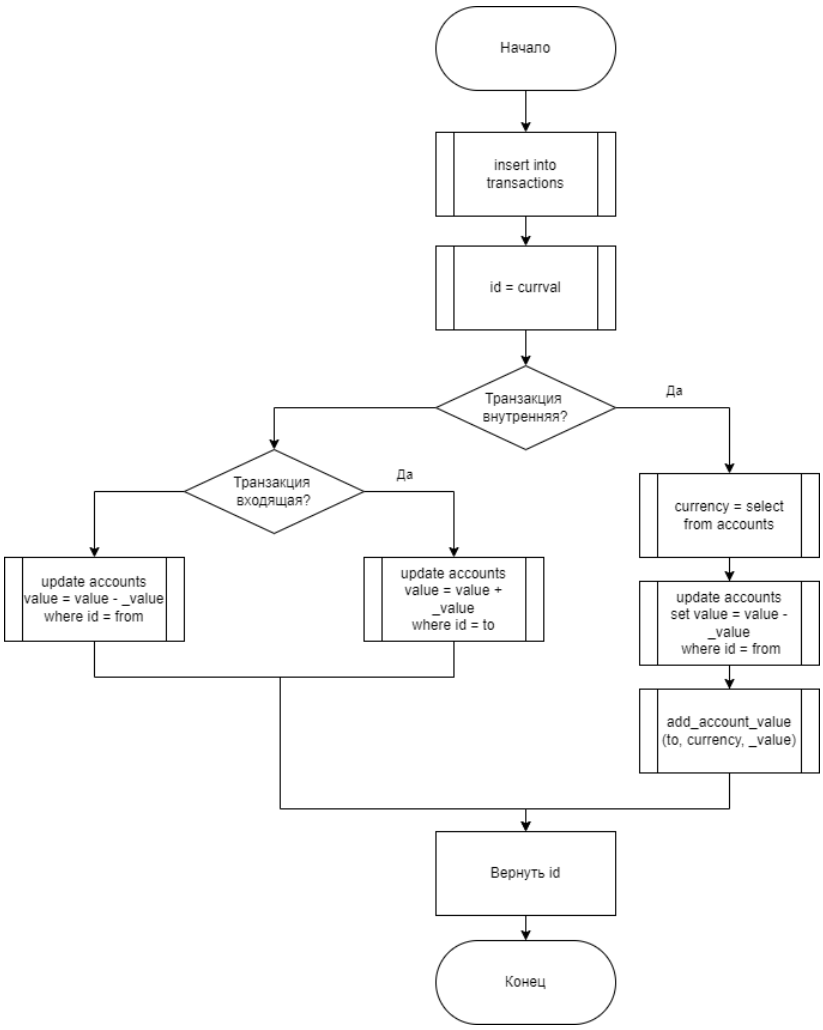

Рисунок 10. Функция add_transaction

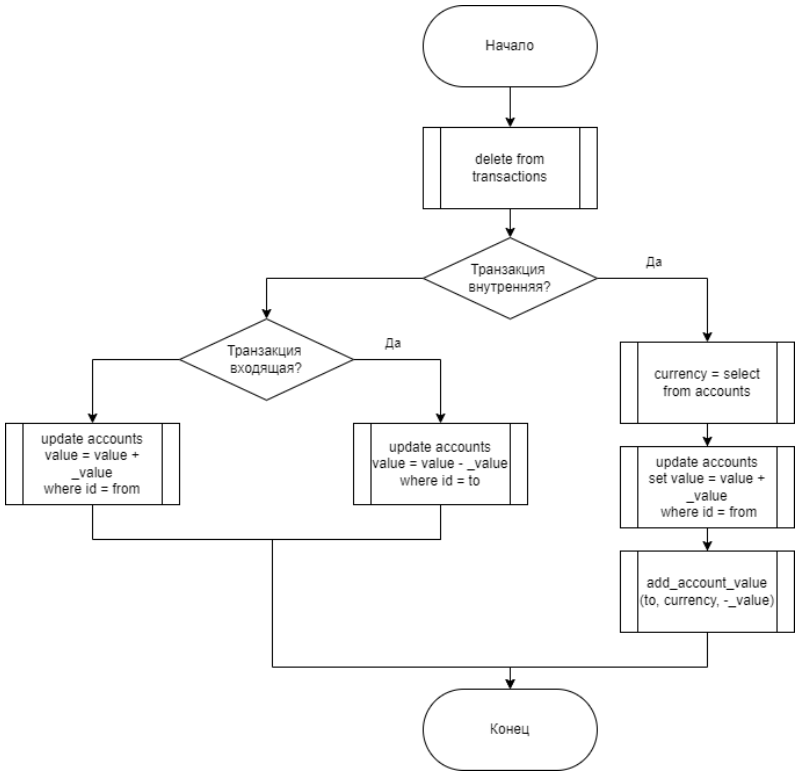

Рисунок 11. Процедура del_transaction

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.

3 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.

3.1 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.

3.2 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.

3.3 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

```
create table currencies
(
    id   serial not null primary key,
    name text not null
);
```

Листинг 2. Создание таблицы exchanges

```
create table exchanges
(
    id     serial not null primary key,
    _from  int references currencies (id) on delete cascade not null,
    _to    int references currencies (id) on delete cascade not null,
    factor double precision not null
);
```

Листинг 3. Создание таблицы categories

```
create table categories
(
    id   serial not null primary key,
    name text not null
);
```

Листинг 4. Создание таблицы users

```
create table users
(
    id       serial not null primary key,
    login    text not null,
    password text not null,
    admin    bool not null
);
```

Листинг 5. Создание таблицы budgets

```
create table budgets
(
    id          serial not null primary key,
    owner       int references currencies (id) on delete cascade not
null,
    currency    int references currencies (id) on delete cascade not
null,
    _start      timestamp not null,
    _end        timestamp not null,
    description text      not null,
    value       double precision not null
);
```

Листинг 6. Создание таблицы accounts

```
create table accounts
(
    id          serial not null primary key,
    owner       int references users (id) on delete cascade not null,
    currency    int references currencies (id) on delete cascade not
null,
    description text not null,
    value       double precision not null
);
```

Листинг 7. Создание таблицы goals

```
create table goals
(
    id          serial not null primary key,
    owner       int references users (id) on delete cascade not null,
    account     int references accounts (id) on delete cascade not null,
    description text not null,
    target      double precision not null
);
```

Листинг 8. Создание таблицы transactions

```
create table transactions
(
    id          serial not null primary key,
    owner       int references users (id) on delete cascade not null,
    category    int references categories (id) on delete cascade not
null,
    _from       int references accounts (id) on delete cascade,
    _to         int references accounts (id) on delete cascade,
    value       double precision not null,
    description text      not null,
    _timestamp  timestamp not null
);
```

3.4 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

```
create function check_table_exists(in tbl text)
returns bool as $$
begin
    return (select exists (select * from information_schema.tables where
table_name = tbl and table_schema = 'public'));
end;
$$
language plpgsql;

create function check_tables_exist(in tbls text[])
returns bool as $$
declare
    e text;
begin
foreach e in array tbls
loop
     if not check_table_exists(e) then
          return false;
        end if;
    end loop;
    return true;
end;
$$
language plpgsql;
```

Листинг 10. Функция get_goal_status

```
create type goal_status as (percents double precision, reached double
precision, remained double precision);

create function get_goal_status(in _id int)
returns goal_status as $$
declare
    _acc_id int;
    _goal double precision;
    _acc double precision;
    _delta double precision;
begin
    _goal = (select target from goals where id = _id);
    _acc_id = (select account from goals where id = _id);
    _acc = (select value from accounts where id = _acc_id);
    _delta = _goal - _acc;
    if (_delta > 0) then
        return (select cast(row(_acc / _goal * 100, _acc, _delta) as
goal_status));
    end if;
    return (select cast(row(100, _goal, 0) as goal_status));
end;
$$
language plpgsql;
```

Листинг 11. Функции convert_value и get_budget_status

```
create function convert_value(in _f int, in _t int, in _value double
precision)
returns double precision as $$
declare
    _factor double precision;
begin
    if _f = _t then
        return _value;
    end if;
    _factor = (select factor from exchanges where _from = _f and _to =
_t);
    return _value * _factor;
end;
$$
language plpgsql;
```

Листинг 11. Продолжение

```
14: create type budget_status as (spent double precision, got double
    precision, total double precision);
15:
16: create function get_budget_status(in _id int, in _user int)
17: returns budget_status as $$
18: declare
19:     _t record;
20:     _from int;
21:     _to int;
22:     _value double precision;
23:     _spent double precision;
24:     _got double precision;
25:     _b_cur int;
26:     _cur int;
27:     _s timestamp;
28:     _e timestamp;
29: begin
30:     _s = (select _start from budgets where id = _id);
31:     _e = (select _end from budgets where id = _id);
32:     _value = (select value from budgets where id = _id);
33:     _b_cur = (select currency from budgets where id = _id);
34:     _got = 0;
35:     _spent = 0;
36:     for _t in (select * from transactions where _timestamp between _s
    and _e)
37: loop
38:       _from = _t._from;
39:       _to = _t._to;
40:       -- Skip inner transactions
41:       if (_from is not null) and (_to is not null) then
42:             continue;
43:          end if;
44:       -- Spent
45:       if (_to is null) then
46:             _cur = (select currency from accounts where id = _t._from);
47:             _spent = _spent + (select convert_value(_cur, _b_cur,
    _t.value));
48:              continue;
49:          end if;
50:       -- Got
51:       _cur = (select currency from accounts where id = _t._to);
52:       _got = _got + (select convert_value(_cur, _b_cur, _t.value));
53:     end loop;
54:     return (select cast(row(_spent, _got, _value + _got - _spent) as
    budget_status));
55: end; $$ language plpgsql;
```

Листинг 12. Функция add_transaction и процедуры add_account_value, del_transaction

```
create procedure add_account_value(
in _id int,
in _currency int,
in _value double precision
) as $$
declare
    _factor double precision;
_t_cur int;
begin
_t_cur = (select currency from accounts where id = _id);
if _t_cur = _currency then
        update accounts
        set value = value + _value
        where id = _id;
        return;
    end if;
_factor = (select factor from exchanges where _from = _currency and _to
= _t_cur);
    update accounts
    set value = value + _value * _factor
    where id = _id;
end;
$$
language plpgsql;
```

Листинг 12. Продолжение

```
create function add_transaction(
in _owner int,
in _category int,
in _from int,
in _to int,
in _value double precision,
in _description text,
in _timestamp timestamp
)
returns int as $$
declare
    _id int;
_currency int;
begin
    insert into transactions
    (owner, category, _from, _to, value, description, "_timestamp")
    values (_owner, _category, _from, _to, _value, _description,
_timestamp);
    _id = (select currval('transactions_id_seq'));
-- If from is not null and to is not null, then use _from.currency as
main currency
if (_from is not null) and (_to is not null) then
     _currency = (select currency from accounts where id = _from);
        update accounts
        set value = value - _value
        where id = _from;
        call add_account_value(_to, _currency, _value);
        return _id;
    end if;
-- If from is null and to is not null, then just update to with value
if (_from is null) and (_to is not null) then
        update accounts
        set value = value + _value
        where id = _to;
        return _id;
    end if;
-- If from is not null and to is null, then just update from with -
value
    update accounts
    set value = value - _value
    where id = _from;
    return _id;
end;
$$
language plpgsql;
```

Листинг 12. Продолжение

```
create procedure del_transaction(in _id int)
as $$
declare
    _f int;
_t int;
_currency int;
_v double precision;
begin
_f = (select _from from transactions where id = _id);
_t = (select _to from transactions where id = _id);
_v = (select value from transactions where id = _id);
    delete from transactions where id = _id;
    -- If from is not null and to is not null, then use _from.currency as main currency
    if (_f is not null) and (_t is not null) then
      _currency = (select currency from accounts where id = _f);
        update accounts
        set value = value + _v
        where id = _f;
        call add_account_value(_t, _currency, -_v);
        return;
    end if;
-- If from is null and to is not null, then sub value from to
if (_f is null) and (_t is not null) then
        update accounts
        set value = value - _v
        where id = _t;
        return;
    end if;
-- If from is not null and to is null, then add value to from
    update accounts
    set value = value + _v
    where id = _f;
end;
$$
language plpgsql;
```

3.5 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

```
create role _user;
grant select on currencies to _user;
grant select on exchanges to _user;
grant select on categories to _user;
grant select on users to _user;
grant update on users to _user;
grant all privileges on budgets to _user;
grant all privileges on accounts to _user;
grant all privileges on goals to _user;
grant all privileges on transactions to _user;
grant usage, select on all sequences in schema public to _user;
```

Листинг 14. Создание роли admin

```
create role _admin;
grant all privileges on currencies to _admin;
grant all privileges on exchanges to _admin;
grant all privileges on categories to _admin;
grant select on users to _admin;
grant update on users to _admin;
grant all privileges on budgets to _admin;
grant all privileges on accounts to _admin;
grant all privileges on goals to _admin;
grant all privileges on transactions to _admin;
grant usage, select on all sequences in schema public to _admin;
```

Листинг 15. Создание роли service

```
create role _service;
grant all privileges on database course to _service;
grant all privileges on currencies to _service;
grant all privileges on exchanges to _service;
grant all privileges on categories to _service;
grant all privileges on users to _service;
grant all privileges on budgets to _service;
grant all privileges on accounts to _service;
grant all privileges on goals to _service;
grant all privileges on transactions to _service;
grant usage, select on all sequences in schema public to _service;
```

3.6 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.

3.7 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.

```
(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py login --login=root@root --password=root
Status: 200
token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhZG1pbiI6dHJ1ZSwiaWQiOjEsImxvZ2luIjoicm9vdEByb290IiwiZXhwIjoxNzE3NTc5MDU0fQ.pipRRJr9s1TpmG6NrubTEwhw1ZdkQgCYtNou97ieD3w

(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-cat --name=my_cat
Status: 200
id: 1
name: my_cat

(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-cur --name=rub
Status: 200
id: 1
name: rub

(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-acc --value=0 --description=descr --currency=1
Status: 200
id: 1
currency: 1
description: descr
value: 0.0
owner: 1
```

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

```
(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-acc --value=0 --description=descr2 --currency=1
Status: 200
id: 2
currency: 1
description: descr2
value: 0.0
owner: 1
```

Рисунок 13. Создание второго счета

```
(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-trs --to=1 --category=1 --description=input --value=100
Status: 200
id: 1
category: 1
to: 1
value: 100.0
description: input
timestamp: Jun 5, 2024, 11:23:52 AM
owner: 1

(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-trs --from=1 --category=1 --description=output --value=20
Status: 200
id: 2
category: 1
from: 1
value: 20.0
description: output
timestamp: Jun 5, 2024, 11:24:23 AM
owner: 1
```

Рисунок 14. Создание транзакций для первого счета

```
(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-trs --to=2 --category=1 --description=input2 --value=80
Status: 200
id: 3
category: 1
to: 2
value: 80.0
description: input2
timestamp: Jun 5, 2024, 11:24:55 AM
owner: 1

(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-trs --from=2 --category=1 --description=output2 --value=15
Status: 200
id: 4
category: 1
from: 2
value: 15.0
description: output2
timestamp: Jun 5, 2024, 11:25:12 AM
owner: 1
```

Рисунок 15. Создание транзакций для второго счета

```
(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-goal --account=1 --description=small_goal --target=5
Status: 200
id: 1
account: 1
description: small_goal
target: 5.0
owner: 1
```

Рисунок 16. Создание цели

```
(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py put-budg --start=2024-01-01 --end=2024-12-31 --value=0 --currency=1 --desription=my_bud
get
Status: 200
id: 1
currency: 1
start: Jan 1, 2024, 12:00:00 AM
end: Dec 31, 2024, 12:00:00 AM
description:
value: 0.0
owner: 1
```

Рисунок 17. Создание бюджета за годовой период

```
(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py stat-goal --id=1
Status: 200
percents: 100.0
reached: 5.0
remained: 0.0

(.venv) C:\Users\Roman\Desktop\db-course\cli>python main.py stat-budg --id=1
Status: 200
spent: 35.0
got: 180.0
total: 145.0
```

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.

4 Исследовательский раздел

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

4.1 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.

4.2 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].

4.3 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

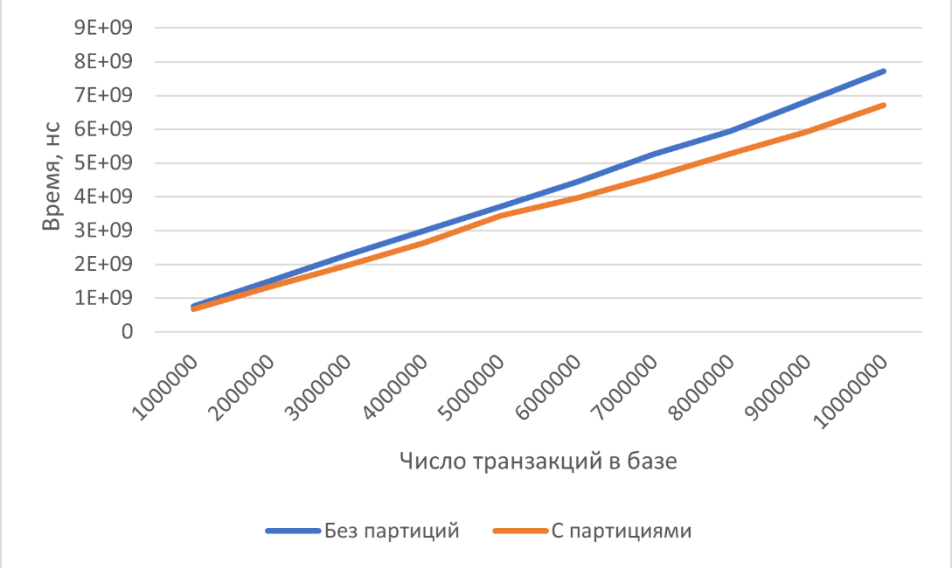

Рисунок 20. Замеры для N от 1000000 до 10000000

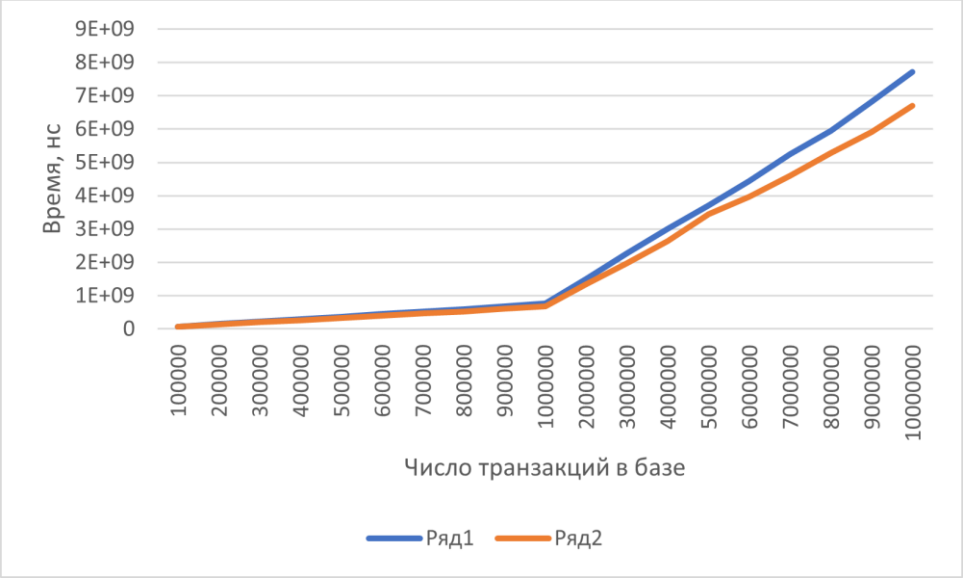

Рисунок 21. Сводный график

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. – С. 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