Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

|  |  |
| --- | --- |
|  | *К защите допустить:*  Руководитель курсового проекта ассистент кафедры информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. С. Плиска |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**СИСТЕМА ИНТЕРНЕТ-БАНКИНГ**

БГУИР КП 1-40-04-01 009 ПЗ

|  |  |
| --- | --- |
| Студент  Руководитель | Довгун В.А.  Плиска В.С. |

Минск 2023

**СОДРЖАНИЕ**

[Введение 5](#_Toc153777731)

[1 Анализ литературных источников и обзор аналогов 6](#_Toc153777732)

[1.1 Анализ литературных источников 6](#_Toc153777733)

[1.2 Обзор существующих аналогов 12](#_Toc153777736)

[2 Формирование функциональных требований и выбор инструментов разработки 21](#_Toc153777740)

[2.1 Формирование функциональных требований 21](#_Toc153777741)

[2.2 Выбор инструментов разработки 21](#_Toc153777742)

[3 проектирование базы данных 29](#_Toc153777746)

[3.1 Инфологическая модель 29](#_Toc153777747)

[3.2 Даталогическая модель 32](#_Toc153777748)

[4 Разработка базы данных 34](#_Toc153777749)

[4.1 Создание сущностей базы данных 34](#_Toc153777750)

[4.2 Создание хранимых процедур и функций 34](#_Toc153777751)

[4.3 Создание триггеров 38](#_Toc153777759)

[5 Тестирование базы данных 45](#_Toc153777769)

[Заключение 48](#_Toc153777770)

[Список использованных источников 49](#_Toc153777771)

[Приложение А (обязательное) Листинг кода 50](#_Toc153777772)

# ВВЕДЕНИЕ

В банковском секторе, как и в экономике в целом, на сегодняшний день наблюдается ожесточенная конкуренция. Банки в полной мере используют все доступные ресурсы для того, чтобы увеличить свою долю на рынке и удержать клиентов, что становится все тяжелее в эпоху быстроразвивающихся цифровых технологий. Клиенты все реже лично посещают отделения банков, поэтому сегодня внедрение цифровых каналов обслуживания является популярной тенденцией в банковской сфере.

Система Интернет-банкинга является очень удобной, так как это позволяет клиентам быстрее и эффективнее осуществлять банковские операции любом удобном месте, где есть интернет, без посещения банковского отделения. В результате финансовые институты повышают удовлетворенность и лояльность клиентов, что является конечной целью всех коммерческих банков.

Из вышесказанного можно сделать вывод, что банки в определенной мере вынуждены внедрять сервисы Интернет-банкинга, чтобы не терять свою клиентскую базу и повысить свою конкурентоспособность, поэтому реализация такой системы является достаточно актуальной задачей.

Цель курсового проекта – разработка базы данных согласно выбранной теме, создать пул-запросов, триггеров, хранимых процедур, индексов для этой базы данных.

Задачи курсового проекта:

– Определить сущности проектируемой БД и их связи.

– Нормализовать БД до 3НФ.

– Создать индексы для наиболее часто используемых сущностей.

– Создать пул триггеров и хранимых процедур для работы с БД.

# 1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ОБЗОР АНАЛОГОВ

## 1.1 Анализ литературных источников

### 1.1.1 Анализ системы интернет-банкинга

Анализируя литературные источники по темам «база данных для системы интернет-банкинга» и «системы интернет-банкинга» были сделаны следующие выводы. Интернет-банкинг – это одна из форм дистанционного банковского обслуживания, технология, которая позволяет клиенту получить доступ ко всем его банковским счетам и управлять ими самостоятельно с компьютера, планшета, смартфона, имеющего доступ в интернет [1].

Интернет-банкинг позволяет банкам оставаться конкурентоспособными. Также банкинг может помочь банкам привлекать новых клиентов и удерживать существующих, предлагая удобные и доступные услуги.

Ниже представлены преимущество использования интернет-банкингов:

1 Экономия времени. Это самое очевидное преимущество использования интернет-банкинга. Пользователям не нужно добираться в банк, стоять в очереди, тратить время на обратную дорогу. В любой удобный момент можно получить доступ к счетам и совершить любую операцию, не выходя из дома.

2 Экономия денег. Благодаря интернет-банкингу банки экономят на расходах на аренду помещений, зарплату сотрудников, технику, канцелярию. Поэтому банки заинтересованы в распространении интернет-банкинга – и поэтому они предлагают «самостоятельным» клиентам всевозможные бонусы. Если, к примеру, в отделении банка клиент должен заплатить комиссию за какой-либо платеж, то, возможно, в интернет-банкинге такой комиссии не будет, либо она будет заметно ниже. Если через интернет-банкинг открывается депозит – то более привлекательной будет доходность. Если самостоятельно оформляется заявка на выпуск новой карточки – это тоже обойдется дешевле, чем через отделение. И так по многим другим банковским услугам.

3 Круглосуточный доступ. Получить доступ ко всем своим счетам и операциям можно в режиме 24/7. Понадобилось заплатить за телефон в новогоднюю ночь? Без проблем. Интернет-банкинг работает без праздников и выходных.

Системы интернет-банкингов крайне полезны для крупных коммерческих банков с большим количеством сотрудников, для агрегации больших данных в одном месте так и могут быть полезны для небольших частных банков, которые хотят улучшить эффективность управления данными внутри своей системы.

### 1.1.2 Проектирование баз данных

База данных является неотъемлемой частью такой системы, так как с помощью ее можно автоматизировать множество процессов.

База данных – представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью ЭВМ.

В современных приложениях работа с базой данных ведется посредством готового программного обеспечения, которое содержит в себе все необходимые инструменты и компоненты для успешного взаимодействия с хранилищем данных. Такое программное обеспечение называется СУБД – система управления базой данных. Основными функциями СУБД являются:

– создание баз данных, изменение, удаление и объединение их по определённым признакам

– хранение данных, в том числе больших массивов, в структурированном виде и нужном формате;

– защита данных от взлома и нежелательных изменений при помощи распределённого доступа: когда разным группам пользователей доступны разный объём и сегменты данных.;

– выгрузка и сортировка данных по заданным фильтрам при помощи SQL-запросов.;

– поддержка целостности баз данных, резервное копирование и восстановление после сбоев.

СУБД классифицируются на разные типы в зависимости от моделей используемых данных, способов предоставления доступа к БД, а также по уровню распределенности. Классификация СУБД по признакам:

– модель данных (иерархические, сетевые, реляционные, объектно-ориентированные, объектно-реляционные);

− степень распределенности (локальные, распределенные);

− способ доступа к базе данных (файл-серверные, клиент-серверные, встраиваемые).

Логическое и физическое представление баз данных — это два разных способа описания и организации данных в базе данных. Логическое представление отражает, как данные понимаются и используются пользователями и приложениями. Физическое представление отражает, как данные хранятся и обрабатываются системой управления базой данных.

Логическое представление базы данных состоит из логических объектов, таких как таблицы, представления, индексы и т.д. Эти объекты определяют структуру, свойства и отношения данных, а также правила и ограничения, которые обеспечивают целостность и согласованность данных. Логическое представление базы данных может быть описано с помощью концептуальной схемы, которая показывает все элементы данных и их связи в виде графического диаграммы. Логическое представление базы данных может быть разделено на несколько уровней абстракции, таких как внешний, логический и внутренний.

Внешний уровень представляет данные в формате, понятном конкретному пользователю или приложению. Логический уровень представляет данные в формате, понятном всем пользователям и приложениям. Внутренний уровень представляет данные в формате, понятном системе управления базой данных. Пример логического представления базы данных представлен на рисунке 1.1:

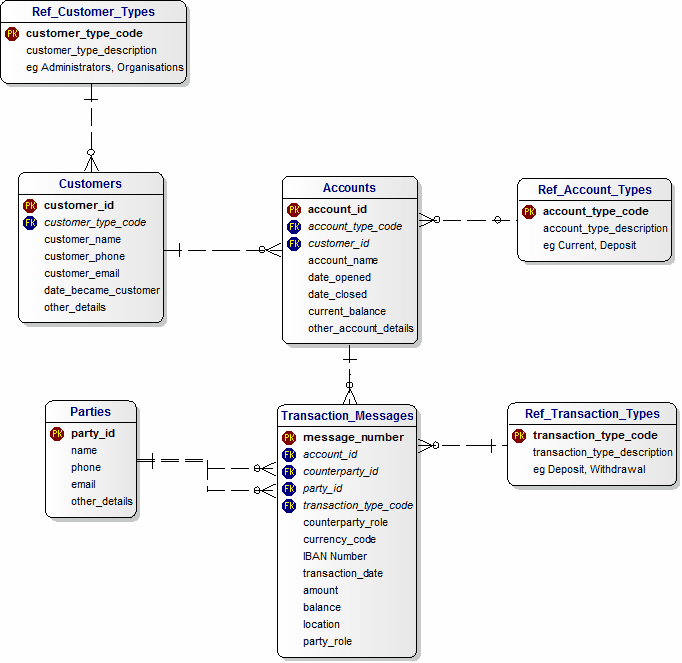


Рисунок 1.1 – Логическое представление базы данных

Физическое представление базы данных состоит из физических объектов, таких как файлы, блоки, страницы, сегменты и т.д. Эти объекты определяют способы хранения, доступа и обработки данных на физическом носителе, таком как диск, память или сеть.

Физическое представление базы данных может быть описано с помощью физической схемы, которая показывает расположение и размер физических объектов, а также параметры и настройки, которые влияют на производительность и эффективность базы данных.

Физическое представление базы данных может быть оптимизировано с помощью различных методов и техник, таких как сжатие, шардирование, кэширование, индексирование и т.д. Схема организации базы данных с физическим уровнем представлена на рисунке 1.2:

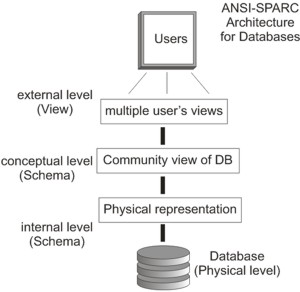


Рисунок 1.2 – Схема организации базы данных с физическим уровнем

Разделение логического и физического представления базы данных позволяет достичь логической и физической независимости данных. Логическая независимость означает, что изменения в логической структуре данных не влияют на прикладные программы и пользователей. Физическая независимость означает, что изменения в физической организации данных не влияют на логическую структуру данных и прикладные программы.

В процессе работы с базой данных могут возникать т.н. аномалии – ошибки, определяющие несоответствие модели данных и модели предметной области. Выделяются три вида аномалий:

– аномалия вставки определяется, когда при добавлении новой записи в таблицу отсутствует часть данных и появляется необходимость либо оставить поле пустым, либо заполнить фиктивными данными, либо не добавлять запись вовсе;

– аномалия обновления определяется, когда происходит сбой обновления данных в таблице, т.е. не проходит корректное обновление всех данных, либо не проходит обновление данных вовсе;

– аномалия удаления определяется, когда при удалении ненужного блока данных происходит потеря полезных данных, все еще необходимых приложению.

Для устранения аномалий принято производить нормализацию таблиц базы данных. Нормализация – группировка и/или распределение атрибутов по отношениям с целью устранения аномалий операций с БД, обеспечения целостности данных и оптимизации модели БД.

Существуют несколько требований к нормализации, которые, в свою очередь, могут противоречить друг другу. Для этого необходимо выбирать тот минимальный набор требований, который поможет решить проблему в конкретном случае и не повлечет серьезных изменений архитектуры приложения.

Первая нормальная форма. В базе данных не должно быть дубликатов и составных данных. Элементы составных данных лучше разнести по разным полям, иначе в процессе работы с данными могут появиться ошибки и аномалии.

Вторая нормальная форма. Говорят, что таблица приведена ко второй нормальной форме, если она приведена к 1НФ и в ней отсутствуют частичные зависимости. Другими словами, у такой таблицы не должно быть атрибутов, зависящих только от части первичного ключа. Отказ от составного первичного ключа имеет один неоспоримый плюс – упрощается процесс приведения таблицы к 2НФ. Доказательство на поверхности: если первичный ключ не является составным, то нет и частичных зависимостей.

Третья нормальная форма. Таблица приведена к третьей нормальной форме, если она соответствует 2НФ и в ней отсутствуют транзитивные зависимости. Транзитивная зависимость A → B → С устраняется одним-единственным способом – за счет выделения атрибутов B → С (или A → B) в другую таблицу. Связи между такими таблицами строятся по ключевым столбцам: первичный ключ главной таблицы соединяется с внешним ключом подчиненной таблицы.

Нормальная форма Бойса-Кодда. Помимо 3НФ, специалисты выделяют усиленную разновидность 3НФ нормальную форму Бойса-Кодда. Смысл усиления в том, что во время формулирования первоначальных требования к 3НФ Кодд не предусмотрел вероятность того, что в нормализуемом отношении может существовать более одного потенциального ключа, указанные ключи окажутся составными и эти ключи станут обладателями хотя бы одного общего атрибута. Вероятность совместного возникновения перечисленных событий крайне невысока, но все-таки не исключена.

Четвертая нормальная форма. Если третья нормальная форма призвана для борьбы с транзитивными зависимостями, то 4НФ состоит в конфронтации с другим злом реляционной модели – многозначными зависимостями. Многозначная зависимость – это не что иное, как связь «многие ко многим».

Итак, база данных, соответствующая четвертой ступени нормализации, обязана избавиться от многозначных зависимостей между атрибутами отношений.

Пятая нормальная форма. Для обычного разработчика БД пятая нормальная форма представляет скорее теоретический, нежели практический интерес. 5НФ требует обеспечения беспрепятственной возможности перестройки данных в нормализованных таблицах. Приведение таблицы к высшей степени нормализации – крайне редкий случай. Это действие имеет смысл, только если таблица содержит так называемые зависимые сочетания.

Зависимые сочетания – это свойство декомпозиции, которое вызывает генерацию ложных строк при обратном соединении декомпозированных отношений с по мощью операции естественного соединения.

Резюме. Нормализация осуществляется на этапе логического проектирования БД и представляет собой вариант восходящего подхода, который начинается с установления связей между атрибутами. На практике для построения приемлемой логической модели БД следует пройти только 3 или 4 ступени нормальных форм:

1 Все поля в таблицах неделимы и не содержат повторяющихся групп.

2 Все неключевые поля в таблицах зависят от первичного ключа.

3 В таблицах отсутствуют избыточные неключевые поля.

4 В таблицах устранены многозначные зависимости.

Таким образом, из сказанного выше можно определить, что проектирование базы данных является, наряду с выбором платформы и архитектуры, ключевым элементом в планировании и разработке приложения, а к общим задачам проектирования БД можно отнести следующее:

– обеспечение хранения в БД всей необходимой информации;

– обеспечение возможности получения данных по всем необходимым

запросам;

– сокращение избыточности и дублирования данных;

– обеспечение целостности базы данных.

## 1.2 Обзор существующих аналогов

Анализ существующих аналогов на рынке является важным этапом в разработке любого ПО. На сегодняшний день существует множество готовых банковских приложений с различным функционалом. Ниже приведен обзор наиболее популярных систем в данной области.

### 1.2.1 Система «Интернет-Банкинг» от Беларусбанка

Система «Интернет-Банкинг» – это услуга, предоставляемая ОАО «АСБ Беларусбанк», по мониторингу, управлению счетами и осуществлению банковских транзакций через сеть Интернет [2]. На рисунке 1.3 представлен графический интерфейс приложения:

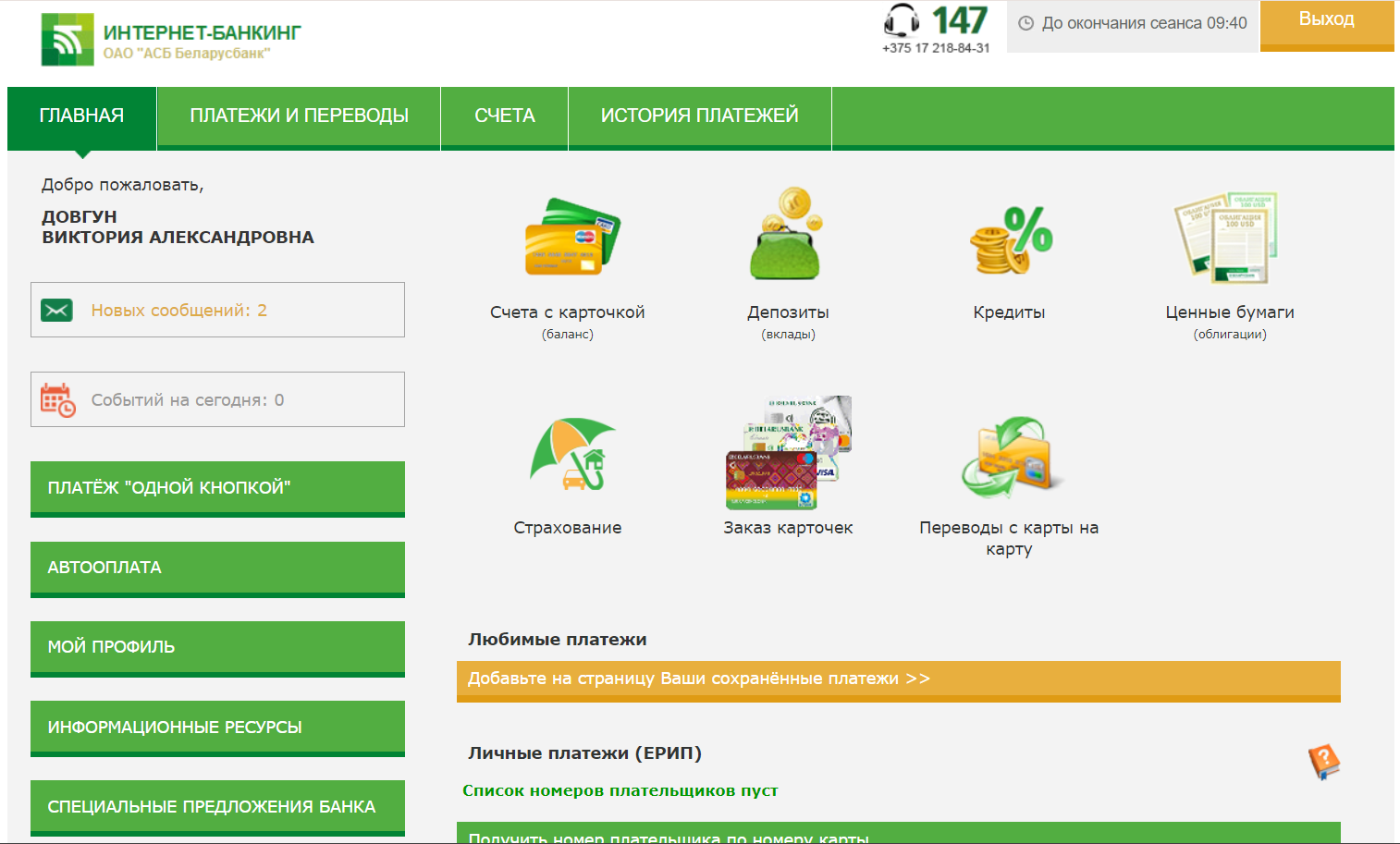


Рисунок 1.3 – Графический интерфейс системы «Интернет-Банкинг»

Из рисунка видно, что у системы достаточно широкий функционал. В системе «Интернет-банкинг» пользователь может просматривать баланс своих счетов, совершать перевод средств с карту на карту, производить различные платежи, в том числе платежи в системе «Расчет» (АИС ЕРИП). На рисунке 1.4 представлены возможные платежи, которые можно произвести:

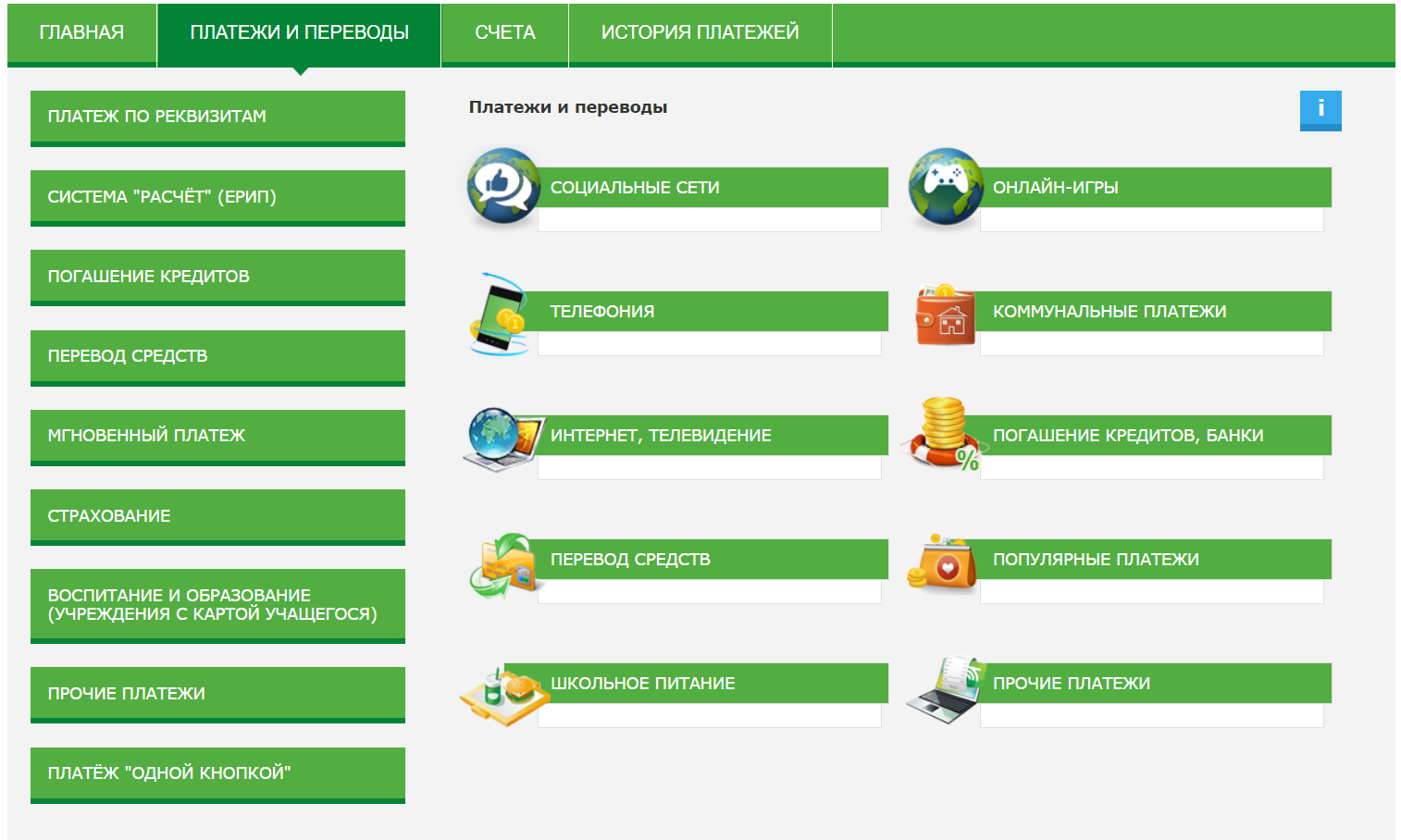


Рисунок 1.4 – Страница платежей

Так же система позволяет открывать различные счета, кредиты и вклады, а также совершать различные операции с ними. На рисунке 1.5 показаны возможные счета и как предоставляется информация по ним:

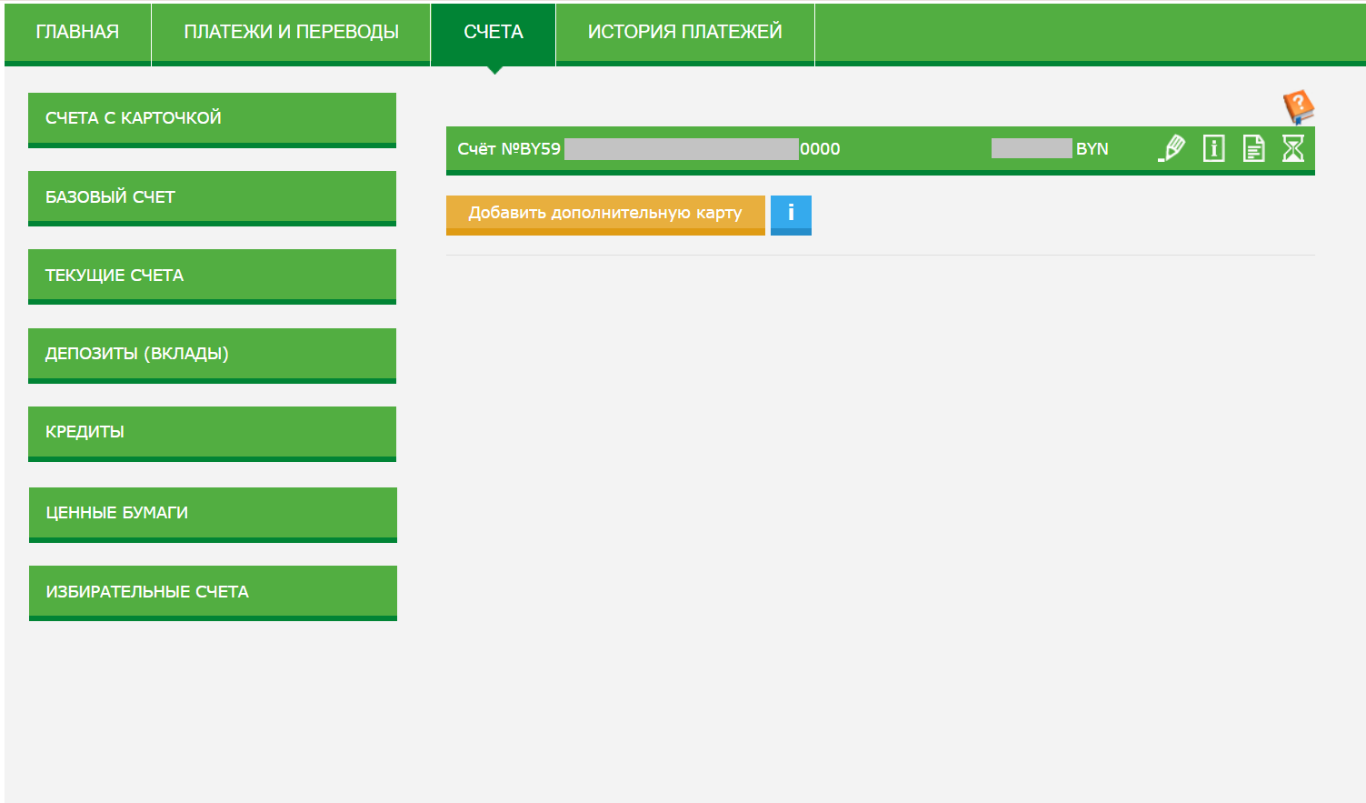


Рисунок 1.5 Страница счетов

Еще одной хорошей функцией является возможность просмотра истории платежей (см. рисунок 1.6).

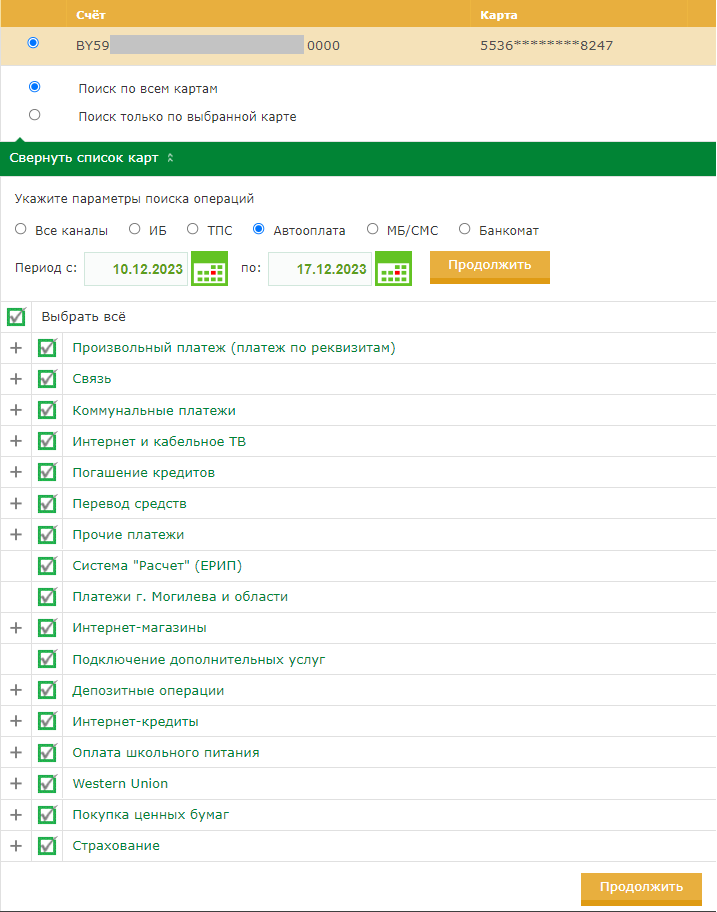


Рисунок 1.6 – Фильтры для просмотра истории платежей

Исходя из рассмотренных функций системы «Интернет-банкинг» можно выделить некоторые сущности, которые должны быть в разрабатываемой базу данных: сущность банковского счета, сущность кредитов, сущность вкладов, сущность переводов.

Таким образом к основным преимуществам рассматриваемой системы можно отнести достаточно широкий функционал и простоту в использовании. К недостаткам можно отнести не очень привлекательный внешний вид, все выглядит очень грубо и несовременно. Также отсутствует функция категоризации транзакций, из-за чего затруднительно просматривать статистку расходов по определенным категориям.

### 1.2.2 Интернет-Банкинг от Белинвестбанка

Данный сервис позволяет держателям банковских платежных карточек ОАО «Белинвестбанк» дистанционно совершать платежи, получать справочную информацию по счетам и совершать множество других операций [3]. На рисунке 1.7 представлен графический интерфейс приложения:

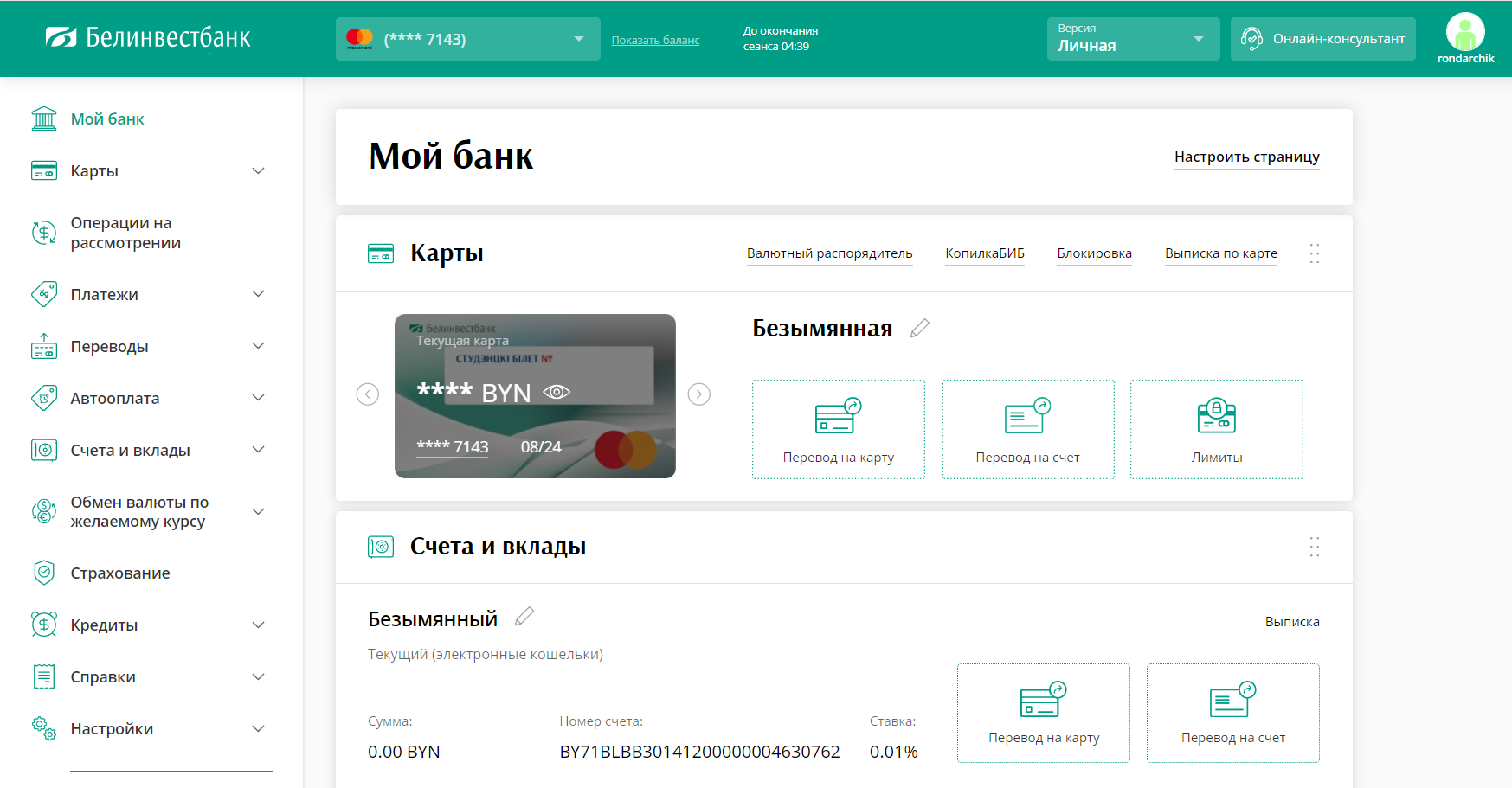


Рисунок 1.7 – Графический интерфейс приложения

Из рисунка видно, что основной функционал примерно такой же как у системы от Беларубанка: просмотр информации о картах и счетах, проведение платежей, оформление кредитов и вкладов и др.

Отличительной особенностью является просмотр курса валют по картам, счетам и просмотр курса установленного Нацбанком (см. рисунок 1.8).

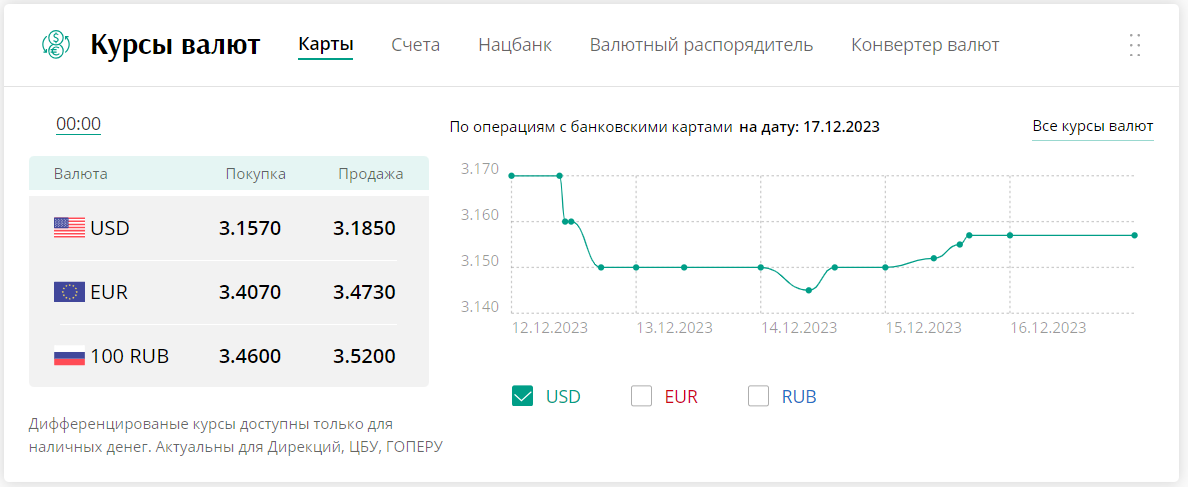


Рисунок 1.8 – Курсы валют

Так же пользователь может пользоваться встроенным конвертером валют по курсу банка. Конвертер валют представлен на рисунке 1.9:

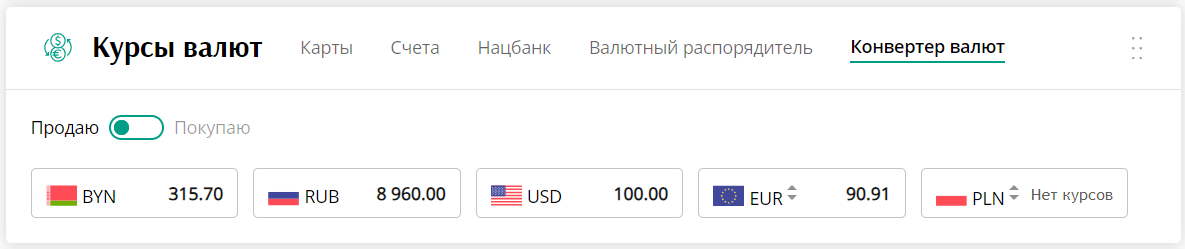


Рисунок 1.9 – Конвертер валют

Таким образом мы можем выделить еще две основные сущности для нашей базы данных: сущность валюты, которая будет хранить в себе код валюты и полное название, а также сущность курсов, которая будет содержать курс обмена и 2 валюты, по которым производится обмен.

К основным преимуществам рассматриваемой системы можно отнести достаточно широкий функционал, простой и красивый внешний вид и наличие информации о курсах валют. К недостаткам можно отнести тоже, что и у системы от Беларусбанка: отсутствие функции категоризации транзакций, из-за чего затруднительно просматривать статистку расходов по определенным категориям.

### 1.2.3 Myfin.by

Это не совсем система интернет-банкинга. Myfin.by – это портал, на котором собрана актуальная информация по курсам валют, кредитам, вкладам, банковским картам, криптовалютам. Данный портал помогает решать все денежные вопросы максимально быстро, удобно и просто. Также Myfin.by – это сайт №1 по охвату аудитории среди всех финансовых порталов Беларуси, а также самая полная база данных по всем банкам Беларуси и их услугам [4]. На рисунке 1.10 представлена страница с предоставляемыми функциями приложения:

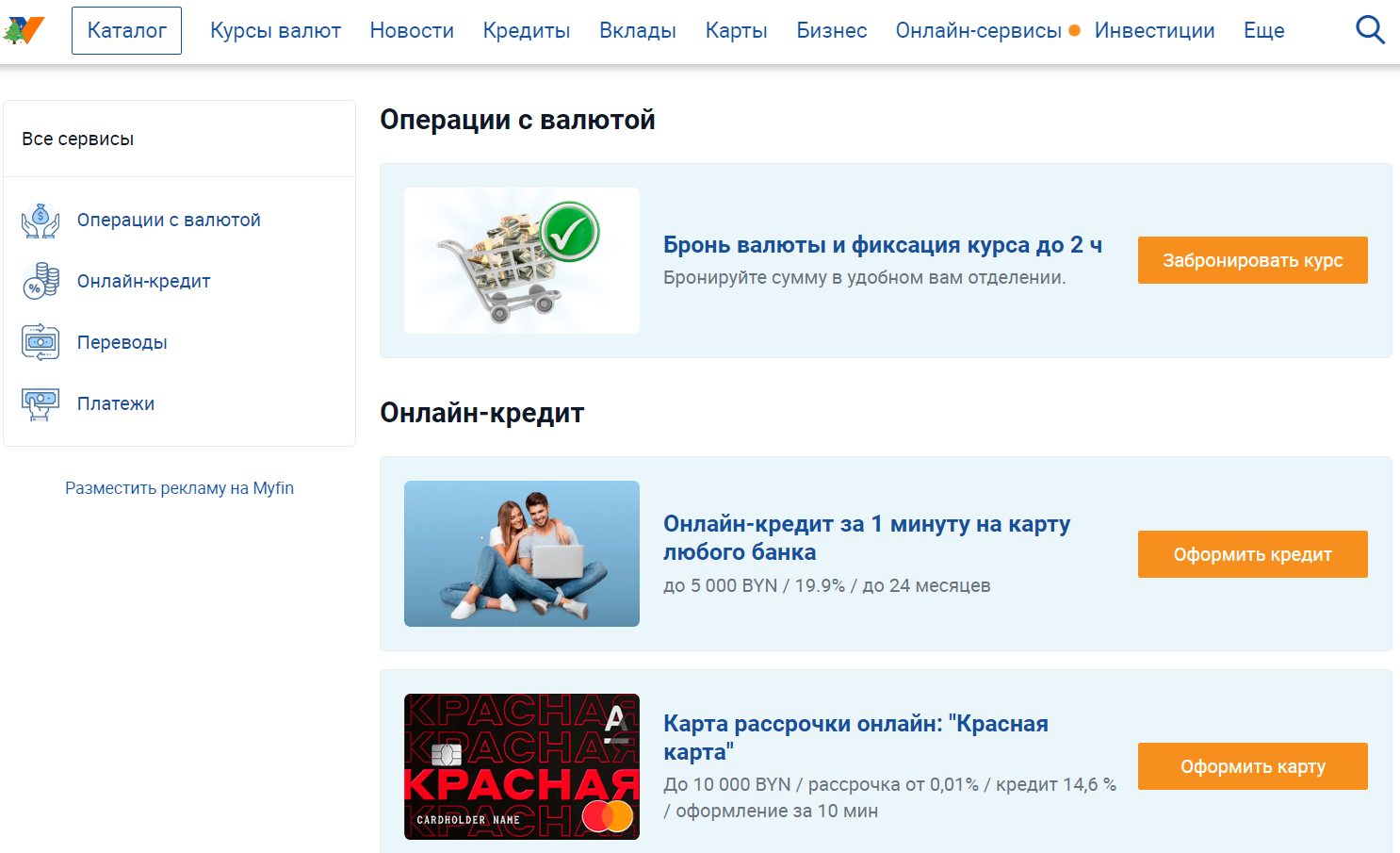


Рисунок 1.10 – Страница онлайн-сервисов портала

Очень удобной функцией является бронирование курса. Как это работает: пользователь может зафиксировать выгодный курс обмена валюты до 2 часов в одном из отделений МТБанка, для этого необходимо заполнить поля в калькуляторе, выбрать отделение, где есть нужная сумма, и подтвердить оформление сделки. Курс будет зафиксирован для пользователя до 2 часов и не изменится за время пути в Банк (см. рисунок 1.11).

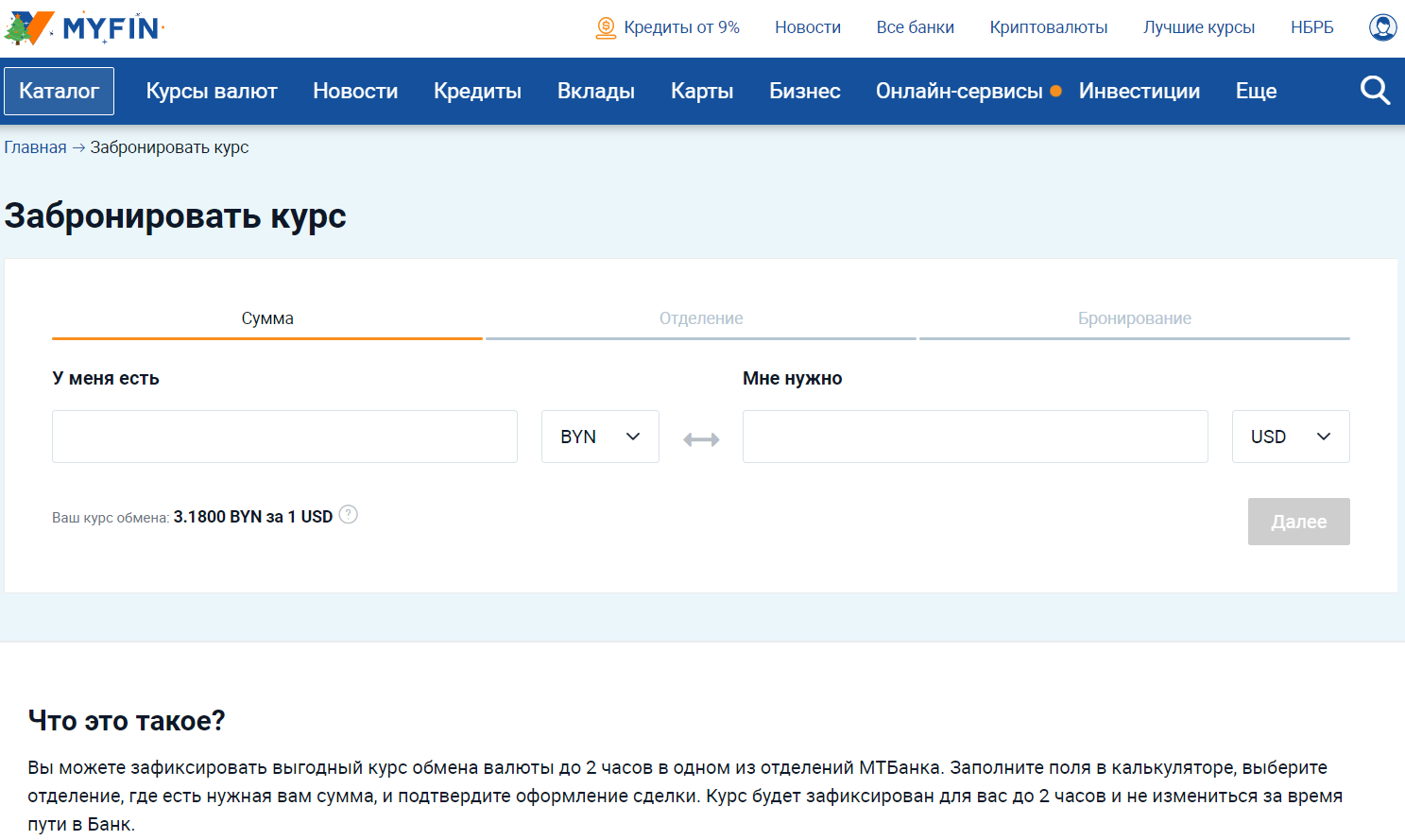


Рисунок 1.11 – Бронирование курса валюты

Так же на сайте представлена информация о кредитах, доступных в большинстве банков Беларуси. Для выбора кредита есть удобные фильтры. На рисунке 1.12 представлена страница с различными видами кредитов:

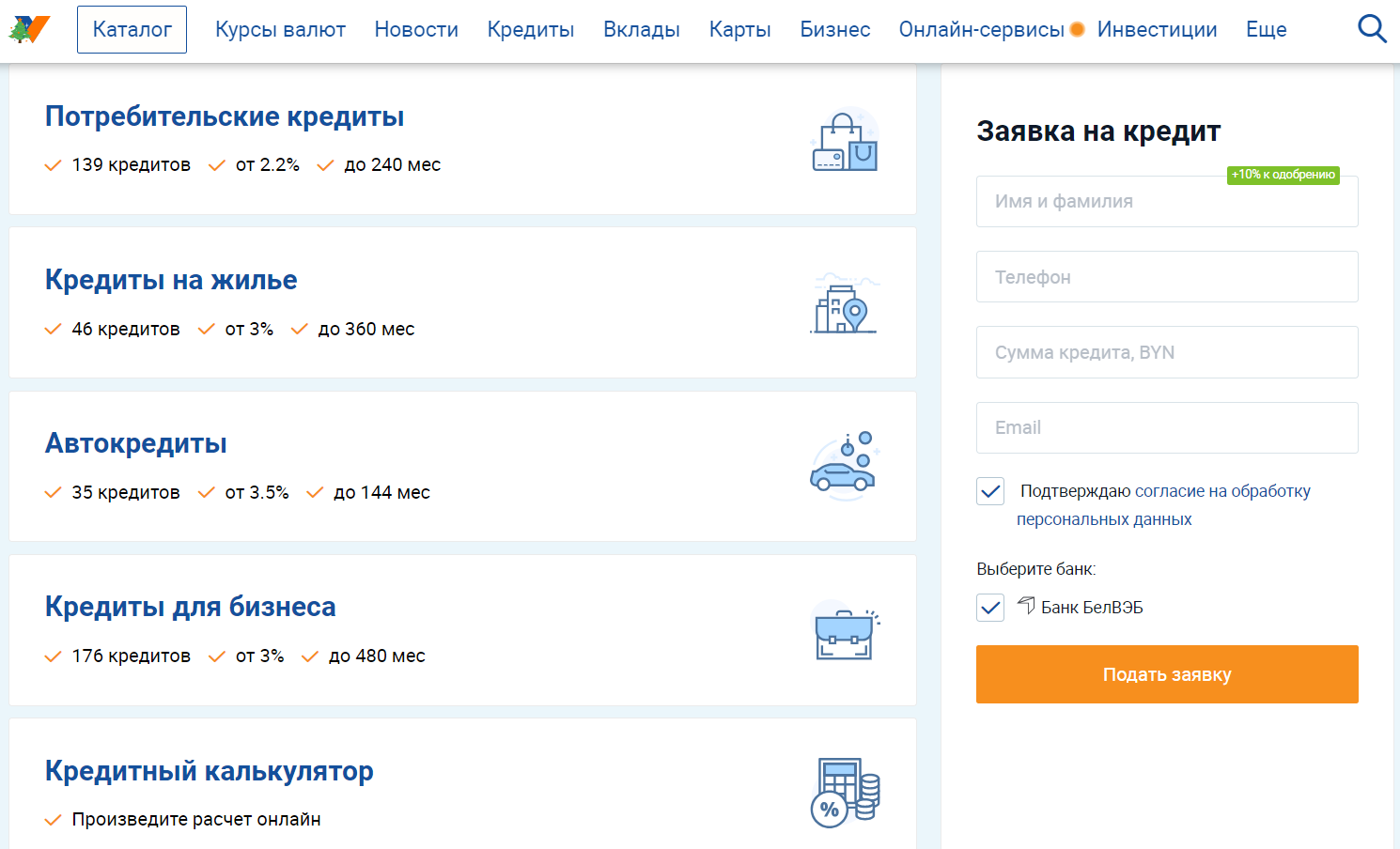


Рисунок 1.12 – Виды кредитов

На рисунке 1.13 представлены возможные фильтры по кредитам и список возможных кредитов:

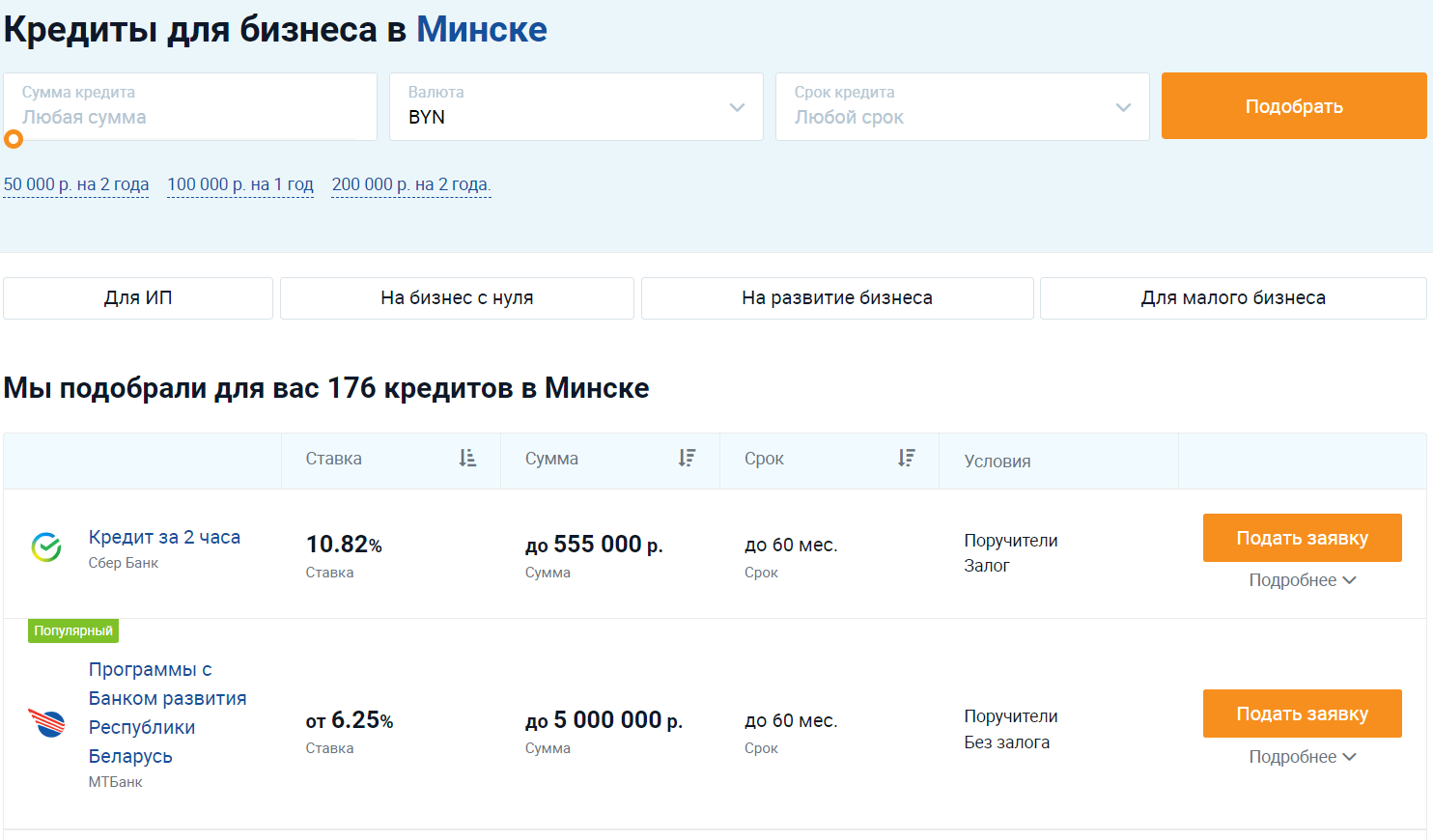


Рисунок 1.13 – Фильтры и список кредитов

Также отличительной особенностью является наличие курсов и калькулятора по криптовалютам (см. рисунок 1.14).

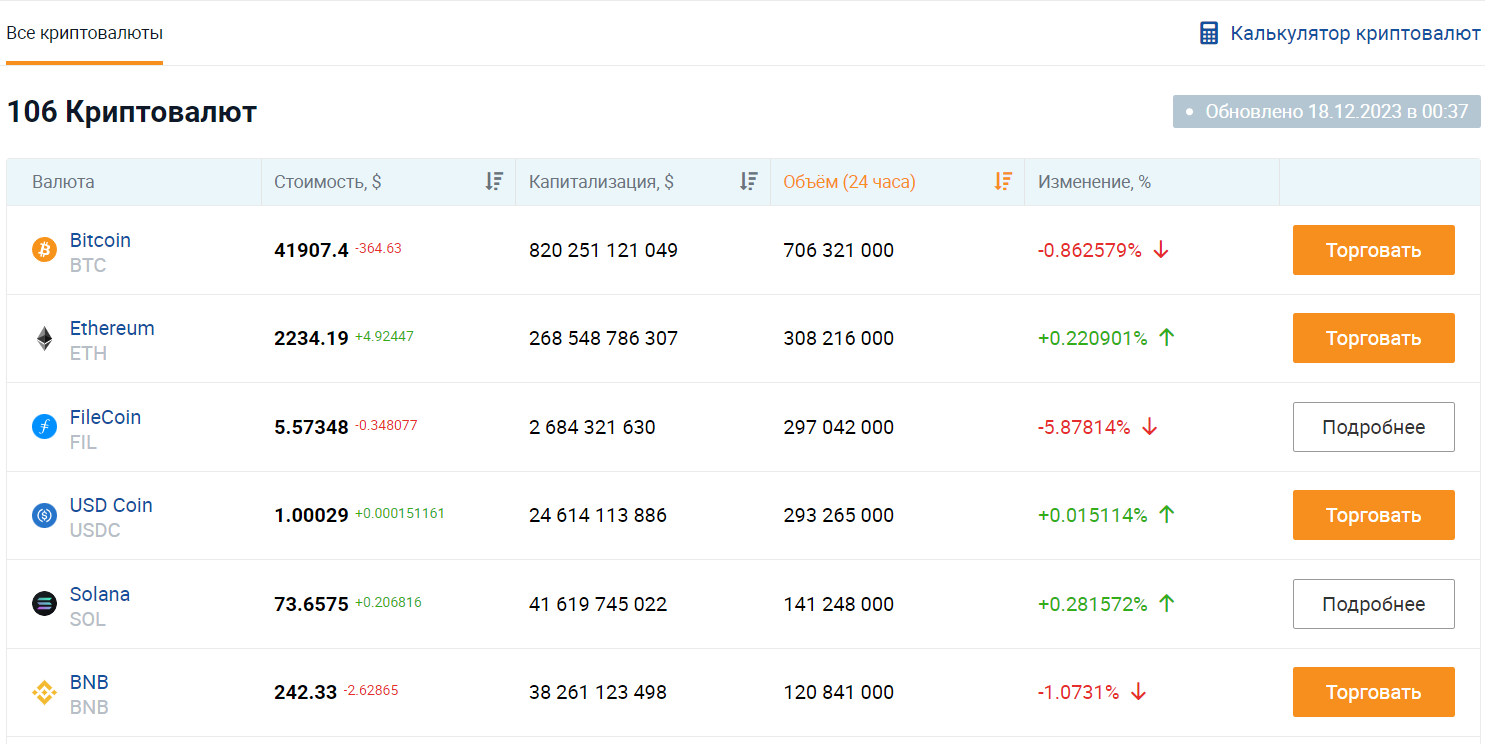


Рисунок 1.14 – Курсы криптовалюты

Таким образом Myfin.by является отличным порталом по финансам, так как здесь можно просмотреть основные новости, курсы валют и криптовалют, доступные кредиты и вклады по банкам и многое другое. Так же у приложения достаточно простой и удобный интерфейс. Однако это не является полноценной банковской системой, где можно привязать карту, проводить платежи, это является основным недостатком сайта.

# 2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ И ВЫБОР ИНСТРУМЕНТОВ РАЗРАБОТКИ

## 2.1 Формирование функциональных требований

На основе анализа литературы на тематику систем интернет-банкингов и на основе анализа существующих аналогов выдвинем следующие функциональные требования, к разрабатываемой системе:

1 Возможность к добавлению и удалению различного рода информации, такой как: информация о пользователях, кредитах и их счетах, а вкладах.

2 Возможность пользователю с ролью «Клиент» обращаться в службу поддержки с интересующими вопросами.

3 Возможность пользователю с ролью «Клиент» открывать счета и переводить средства между ними.

4 Возможность пользователю с ролью «Клиент» проводить различные типы транзакций и выбирать для них определенные категории.

5 Возможность пользователю с ролью «Клиент» просматривать курсы валют и открывать счет в иностранной валюте.

6 Возможность пользователю с ролью «Клиент» подать заявку на кредит или вклад и оформить его после того, как пользователь с ролью «Менеджер» одобрит ее.

7 Возможность пользователям с ролью «Менеджер» одобрять или отклонять заявки на кредиты и вклады пользователей с ролью «Клиент», в случае если недостаточно данных, данные недостоверны и т.д.

8 Возможность пользователям с ролью «Специалист технической поддержки» отвечать пользователям с ролью «Клиент» на заданные ими вопросы.

## 2.2 Выбор инструментов разработки

Всего в современном проектировании и разработке баз данных выделяют 2 основных вида: SQL-, NoSQL- базы данных. Но это ни в коем случае не говорит о том, что для первого вида используются SQL-запросы, а для второй нет, здесь скорее это относится к тому, что SQL базы данных используют реляционную теорию, а NoSQL – нет. Для более явного визуального представления рассмотрим рисунок 2.1, на котором изображены основные подвиды баз данных:

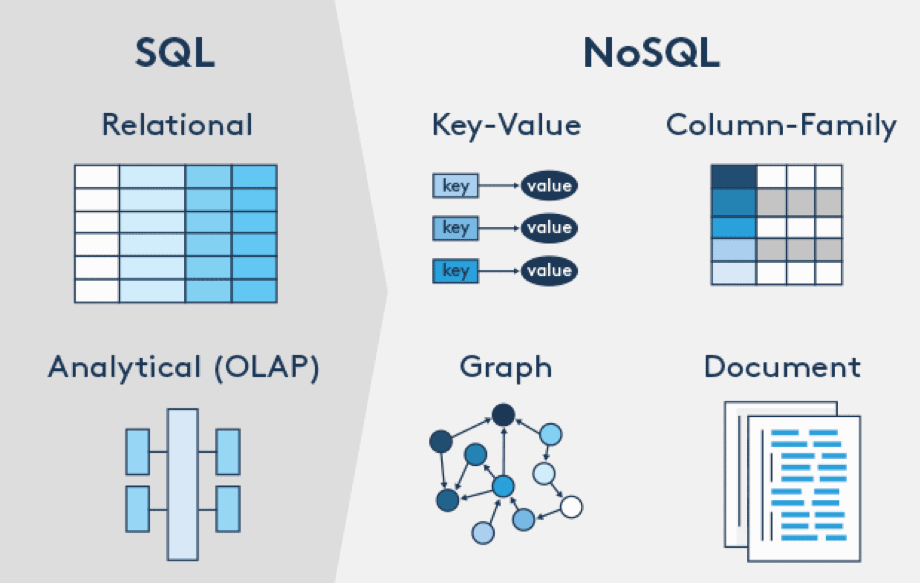


Рисунок 2.1 – Основные виды SQL и NoSQL баз данных

Теперь более явным образом рассмотрим основные отличия реляционных и нереляционных баз данных.

### 2.2.1 Нереляционные базы данных

Документо-ориентированные базы данных хранят данные в виде документов, которые имеют свою структуру и могут содержать разные типы данных. Документы группируются в коллекции, которые могут иметь разные схемы. Для работы с документо-ориентированными базами данных используются специальные языки запросов, которые позволяют обращаться к данным по их атрибутам. Рассмотрим подробнее примеры документо-ориентированных хранилищ данных:

1 MongoDB — одна из самых популярных и мощных документо-ориентированных баз данных, которая поддерживает разные форматы документов, такие как JSON, BSON и XML. MongoDB обладает высокой производительностью, масштабируемостью и гибкостью, а также предоставляет различные функции и инструменты для работы с данными, такие как агрегация, индексация, шардирование, репликация и т.д.

2 Firebase — облачная платформа, которая предоставляет документо-ориентированную базу данных в реальном времени, называемую Cloud Firestore. Firebase позволяет хранить и синхронизировать данные между разными клиентами и серверами, а также предлагает различные сервисы для разработки мобильных и веб-приложений, такие как аутентификация, хостинг, аналитика, машинное обучение и т.д.

Далее рассмотрим базы данных ключ-значение. Базы данных ключ-значение хранят данные в виде пар ключ-значение, где ключ является уникальным идентификатором, а значение может быть любым типом данных. Базы данных ключ-значение обеспечивают быстрый доступ к данным по ключу, но не поддерживают сложные запросы и связи между данными. Обладают они следующими особенностями:

1 Легко масштабируются по горизонтали. Достигается это благодаря тому, что данные могут быть распределены по разным узлам или серверам без необходимости объединения таблиц или синхронизации схем.

2 Подходят для хранения и обработки неструктурированных или полуструктурированных данных. Такими данными как правило выступают текст, изображения или видео, которые могут иметь разные размеры или формат.

3 Они позволяют гибко изменять структуру и свойства данных, так как не требуют жесткой схемы или типизации данных.

4 Они обладают высокой производительностью и низкой задержкой, так как обрабатывают данные в оперативной памяти или на быстрых носителях.

Отличными примерами базы данных ключ-значения являются:

Redis — одна из самых популярных и мощных баз данных ключ-значение, которая хранит данные в оперативной памяти и поддерживает разные типы значений, такие как строки, списки, множества, хеши, битовые массивы и т.д. Redis также предоставляет различные функции и инструменты для работы с данными, такие как транзакции, репликация, шардирование, кэширование, публикация-подписка и т.д.

DynamoDB — облачная база данных ключ-значение, предоставляемая Amazon Web Services. DynamoDB хранит данные на твердотельных накопителях и обеспечивает высокую доступность, надежность и масштабируемость данных. DynamoDB поддерживает разные типы значений, такие как строки, числа, бинарные данные, списки и карты, а также позволяет выполнять условные запросы и обновления данных.

Графовые базы данных хранят данные в виде узлов и ребер, которые представляют собой сущности и связи между ними. Графовые базы данных подходят для моделирования сложных сетей и отношений, таких как социальные сети, рекомендательные системы, маршрутизация и т.д. Для работы с графовыми базами данных используются специальные языки запросов, которые позволяют искать пути и паттерны в графе. Примеры графовых баз данных: Neo4j, OrientDB, ArangoDB и др.

Они лучше отражают реальную структуру и семантику данных, которые часто имеют сложные и динамические взаимосвязи, такие как социальные сети, рекомендательные системы, биоинформатика и т.д.

Графовые базы данных — это базы данных, которые используют математическую теорию графов для отображения и обработки связей между данными. В графовых базах данных данные представлены в виде узлов и ребер, которые обозначают сущности и отношения между ними. Графовые базы данных имеют ряд преимуществ перед реляционными и другими видами нереляционных баз данных, таких как:

Они позволяют быстро и эффективно выполнять запросы, которые требуют обхода и анализа связей в графе, такие как поиск кратчайшего пути, обнаружение сообществ, выявление аномалий и т.д.

Они обеспечивают высокую гибкость и масштабируемость, так как не требуют жесткой схемы данных и позволяют добавлять, удалять и изменять узлы и ребра без нарушения целостности данных.

Графовые базы данных могут быть разделены на два основных типа: базы данных свойственных графов и базы данных знаний. Базы данных свойственных графов хранят данные в виде графов со свойствами, то есть узлы и ребра имеют атрибуты, которые описывают их характеристики.

Базы данных знаний хранят данные в виде графов онтологий, то есть узлы и ребра имеют семантические метки, которые определяют их типы и смысл. Примеры баз данных свойственных графов: Neo4j, ArangoDB, JanusGraphи др. Примеры баз данных знаний: AllegroGraph, Datomic, TerminusDB и др.

Для работы с графовыми базами данных используются специальные языки запросов, которые позволяют обращаться к данным по их узлам, ребрам и свойствам. Некоторые известные языки запросов для графовых баз данных: Cypher, Gremlin, SPARQL и др.

**2.2.2** Реляционные базы данных

Oracle Database — это объектно-реляционная система управления базами данных (СУБД) от компании Oracle. Она используется для создания структуры новой базы, ее наполнения, редактирования содержимого. Oracle Database поддерживает очень большие базы данных и может обслуживать большое количество пользователей.Oracle Database – это объектно-реляционная система управления базами данных (СУБД) от компании Oracle. Она используется для создания структуры новой базы, ее наполнения, редактирования содержимого. Oracle Database поддерживает очень большие базы данных и может обслуживать большое количество пользователей [5].

Преимущества Oracle Database:

1 Производительность и масштабируемость. Oracle Database способна обрабатывать огромные объемы данных и обслуживать большое количество пользователей одновременно. Это делает ее идеальным выбором для крупных корпоративных приложений, таких как системы управления ресурсами предприятия (ERP), системы управления цепочками поставок (SCM) и системы управления клиентами (CRM);

2 Безопасность. Oracle Database обеспечивает безопасный и легкий доступ к базе данных. Это важно для защиты конфиденциальной информации и обеспечения надежности системы. Oracle Database поддерживает множество функций безопасности, таких как аутентификация, авторизация, шифрование и аудит;

3 Поддержка. Oracle предлагает обширную поддержку и обучение, что может быть полезно для компаний с большими командами разработчиков. Oracle предлагает различные варианты поддержки, включая техническую поддержку по телефону, онлайн-поддержку и личное обучение.

К недостаткам Oracle Database можно отнести высокие технические требования: Oracle Database требует производительного сервера для работы3. Это может быть проблемой для небольших компаний или стартапов с ограниченными ресурсами. Также стоимость Oracle Database может быть достаточно высокой, особенно для малых и средних предприятий [5].

Структура СУБД Oracle Database представлена на рисунке 2.2:

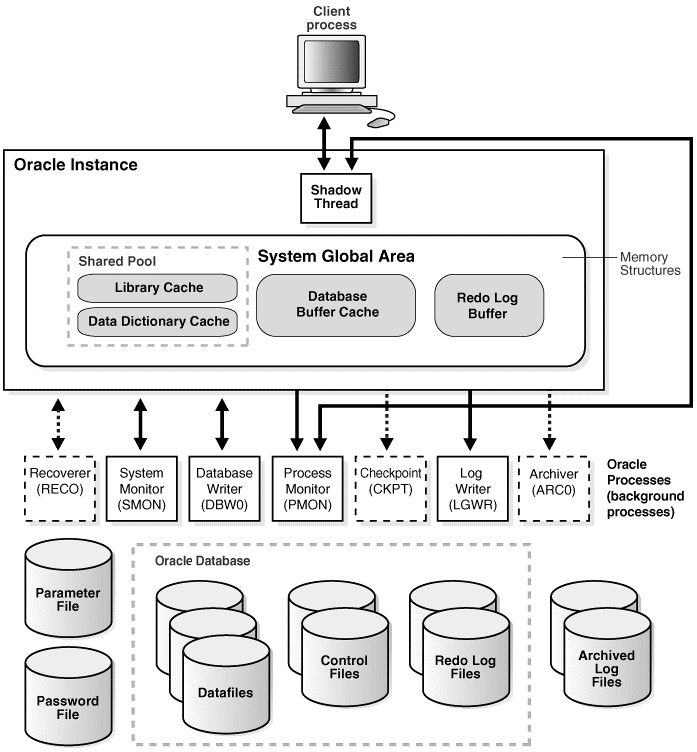


Рисунок 2.2 – Структура СУБД Oracle Database

Следующая известная СУБД PostgreSQL — это свободная и открытая система управления базами данных, которая поддерживает широкий спектр функций и возможностей. Она является популярной альтернативой коммерческим СУБД, таким как Oracle Database и Microsoft SQL Server [6].

PostgreSQL предлагает множество преимуществ, которые делают ее привлекательным выбором для различных приложений и предприятий. PostgreSQL – это высокопроизводительная СУБД, которая способна обрабатывать большие объемы данных и обслуживать большое количество пользователей одновременно. Это достигается за счет нескольких факторов. В PostgreSQL используется оптимизированный движок запросов, который помогает эффективно обрабатывать запросы к базе данных. Поддерживает параллельные операции, что позволяет выполнять несколько запросов одновременно. PostgreSQL может масштабироваться по вертикали, добавляя дополнительные ресурсы, или по горизонтали, распределяя нагрузку между несколькими серверами.

PostgreSQL может использоваться для различных приложений, включая:

1. Крупные корпоративные приложения: PostgreSQL является хорошим выбором для крупных предприятий, которым требуется высокая производительность, масштабируемость и безопасность. Она может использоваться для таких приложений, как системы управления ресурсами предприятия (ERP), системы управления цепочками поставок (SCM) и системы управления клиентами (CRM);
2. Веб-приложения: PostgreSQL является популярной СУБД для веб-приложений. Она поддерживает функции, необходимые для разработки масштабируемых и надежных веб-приложений;
3. Интернет-магазины: PostgreSQL может использоваться для создания и управления интернет-магазинами. Она поддерживает функции, необходимые для обработки транзакций, хранения товаров и управления клиентами;
4. Социальные сети: PostgreSQL может использоваться для создания и управления социальными сетями. Она поддерживает функции, необходимые для хранения данных пользователей, обработки сообщений и обеспечения безопасности.

PostgreSQL поддерживает множество функций безопасности, которые помогают защитить данные от несанкционированного доступа и использования. Поддерживаются различные методы аутентификации, такие как пароли, сертификаты и двухфакторная аутентификация. Это помогает гарантировать, что только авторизованные пользователи могут получить доступ к базе данных. PostgreSQL поддерживает различные модели авторизации, такие как роль-основанная и объектно-основанная. Это позволяет администраторам базы данных контролировать, какие пользователи имеют доступ к каким данным и функциям.

На рисунке 2.3 изображена структура процессов в PostgreSQL.

PostgreSQL поддерживает различные методы шифрования, такие как шифрование данных в покое и шифрование данных в пути. Это помогает защитить данные от несанкционированного доступа во время хранения и передачи. Так же поддерживается аудит действий пользователей, что позволяет администраторам базы данных отслеживать активность пользователей и выявлять потенциальные угрозы безопасности.

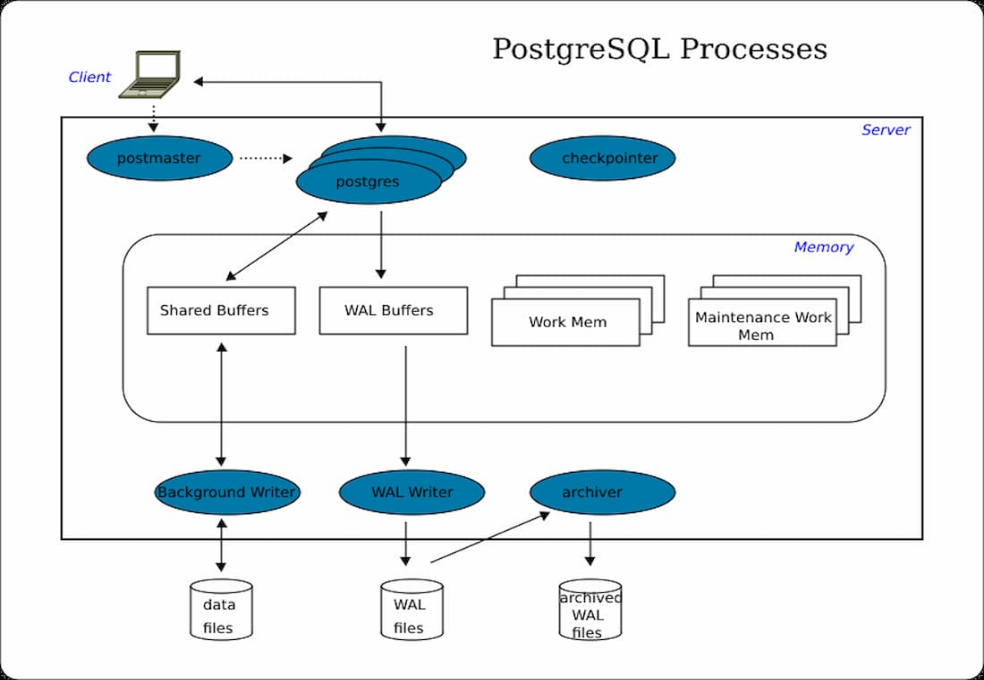


Рисунок 2.3 – Структура процессов в PostgreSQL

PostgreSQL является гибкой СУБД, которая может адаптироваться к различным потребностям бизнеса. Присутствует поддержка расширения с помощью дополнительных модулей, которые могут быть добавлены для добавления новых функций и возможностей. Это позволяет PostgreSQL адаптироваться к меняющимся потребностям бизнеса.

PostgreSQL также является хорошим выбором для небольших предприятий и стартапов. Она предлагает множество функций и возможностей по более доступной цене, чем коммерческие СУБД.

**2.2.3** Выбор базы данных для разрабатываемой системы

Исходя из анализа подходов в предыдущем пункте, следует сделать вывод, что предметная область системы интернет-банкинга будет реализована с помощью реляционной базы данных. Потому что выбранная предметная область имеет четкие атрибуты, сущности и связи, которые достаточно просто можно представить в виде таблиц и ключей в реляционной теории разработке баз данных.

В качестве базы данных для разработки была выбрана PostgreSQL, которая уже давно стала рыночным стандартом разработки веб-приложений, поскольку хорошо зарекомендовала себя в данной сфере.

# 3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

## 3.1 Инфологическая модель

Начальной точкой в проектировании базы данных является создание инфологической модели [7]. Сама инфологическая модель представляет собой только описание сущностей и связанный с нею атрибутов, без конкретизации типов данных и подробностей реализации на серверной части базы данных.

Предметная область разрабатываемой системы включает в себя следующие сущности и атрибуты:

а) пользователь:

– логин;

– адрес электронной почты;

– имя;

– фамилия;

– хешированный пароль;

– идентификатор роли;

б) роль:

– название роли;

в) клиент:

– номер телефона;

– дата рождения;

г) специалист технической поддержки:

– статус специалиста;

– время последней авторизации;

– идентификатор расписания специалиста;

д) менеджер:

– идентификатор отделения;

е) расписание специалистов технической поддержки:

– название расписания;

ж) запрос в техническую поддержку:

– идентификатор клиента;

– идентификатор специалиста;

– дата и время отправления запроса;

– сообщение;

– идентификатор статуса запроса;

з) статус запроса:

– название статуса запроса;

и) ответ специалиста технической поддержки:

– идентификатор специалиста;

– идентификатор запроса;

– дата и время отправления ответа;

– сообщение;

к) банковский счет:

– название счета;

– идентификатор клиента;

– баланс;

– идентификатор валюты;

– дата создания;

– идентификатор статуса счета;

л) статус счета:

– название статуса счета;

м) валюта:

– короткое название валюты;

– полное название валюты;

н) курс валют:

– идентификатор валюты для перевода;

– идентификатор валюты, в которую переводят;

– курс обмена;

– последнее время обновления;

о) транзакция:

– идентификатор счета отправителя;

– идентификатор счета получателя;

– сумма;

– идентификатор категории;

– идентификатор типа транзакции;

– дата и время проведения транзакции;

п) тип транзакции:

– название типа;

р) категория транзакции:

– название категории;

– идентификатор типа транзакции;

с) город:

– название города;

т) отделение:

– название отделения;

– идентификатор города;

– адрес отделения;

у) запрос на кредит:

– идентификатор города;

– идентификатор типа кредита;

– сумма кредита;

– идентификатор клиента;

– идентификатор менеджера;

– дата запроса;

– статус;

ф) тип кредита:

– название кредита;

– минимальная сумма;

– максимальная сумма;

– кредитная ставка;

– длительность в месяцах;

х) счет кредита:

– идентификатор клиента;

– сумма;

– выплаченная сумма;

– кредитная ставка;

– дата открытия;

– дата закрытия;

ц) кредитная транзакция:

– идентификатор кредитного счета;

– сумма;

– идентификатор типа кредитной транзакции;

– дата проведения транзакции;

ч) тип кредитной транзакции

– название транзакции;

ш) запрос на вклад:

– идентификатор города;

– идентификатор типа вклада;

– сумма вклада;

– идентификатор клиента;

– идентификатор менеджера;

– дата запроса;

– статус;

щ) тип вклада:

– название вклада;

– минимальная сумма;

– максимальная сумма;

– депозитная ставка;

– длительность в месяцах;

э) счет вклада:

– идентификатор клиента;

– сумма;

– накопленная сумма;

– депозитная ставка;

– дата открытия;

– дата закрытия;

ю) депозитная транзакция:

– идентификатор депозитного счета;

– сумма;

– идентификатор типа депозитной транзакции;

– дата проведения транзакции;

я) тип депозитной транзакции

– название транзакции;

## 3.2 Даталогическая модель

По результатам разработки инфологической модели базы данных построим даталогическую модель базы данных, в которой будут определены типы данных для каждого поля, участвующего в описании разрабатываемой предметной области. ER-диаграмма базы данных [8] представлена на рисунке 3.1:

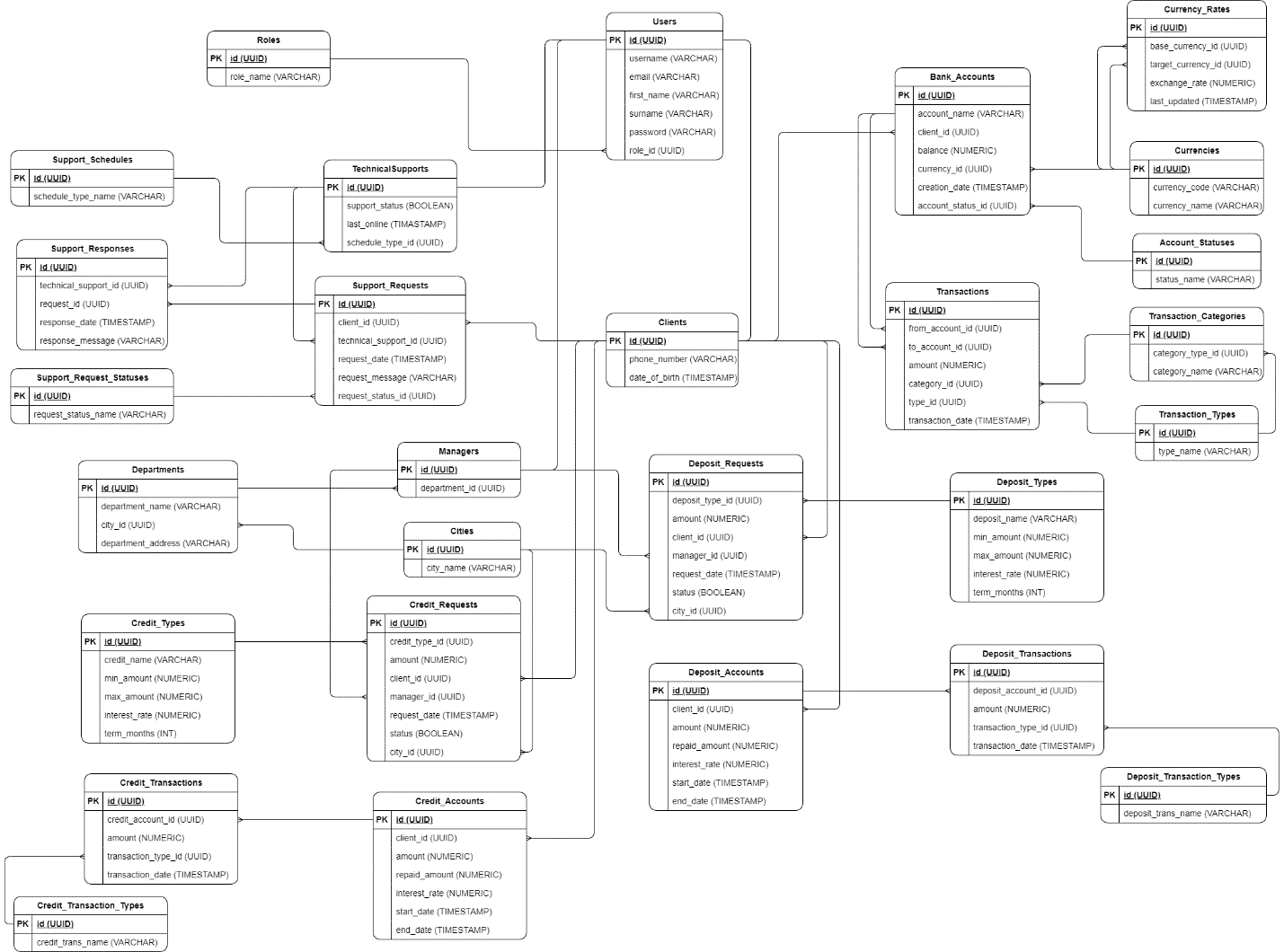


Рисунок 3.1 – ER-диаграмма базы данных

Таким образом, в результате проектирования схем базы данных мы получаем готовые модели, которые остаётся перенести в физическую реализацию средствами выбранной СУБД.

# 4 РАЗРАБОТКА БАЗЫ ДАННЫХ

## 4.1 Создание сущностей базы данных

После того, как были написаны инфологические и даталогические модели базы данных, выбрана база данных и язык, можно приступать к непосредственной реализации сущностей базы данных.

Программный SQL-код, реализующий данные сущности представлен в приложении А. Он создает сущности согласно инфологической и даталогической моделям данных.

## 4.2 Создание хранимых процедур и функций

Программный SQL-код всех хранимым процедур и функций, описанных в данном разделе, представлен в приложении А.

### 4.2.1 Функция insert\_new\_user

Данная функция предназначена для вставки нового пользователя в таблицу пользователей, а также, в зависимости от выбранной роли для пользователя, вставляется новый пользователь в таблицу с соответствующей ролью. Для выполнения такой сложной вставки была создана вспомогательная таблица, принимающая всевозможные атрибуты таблиц пользователей и ролей пользователей. Реализация функции представлена на рисунке 4.1:

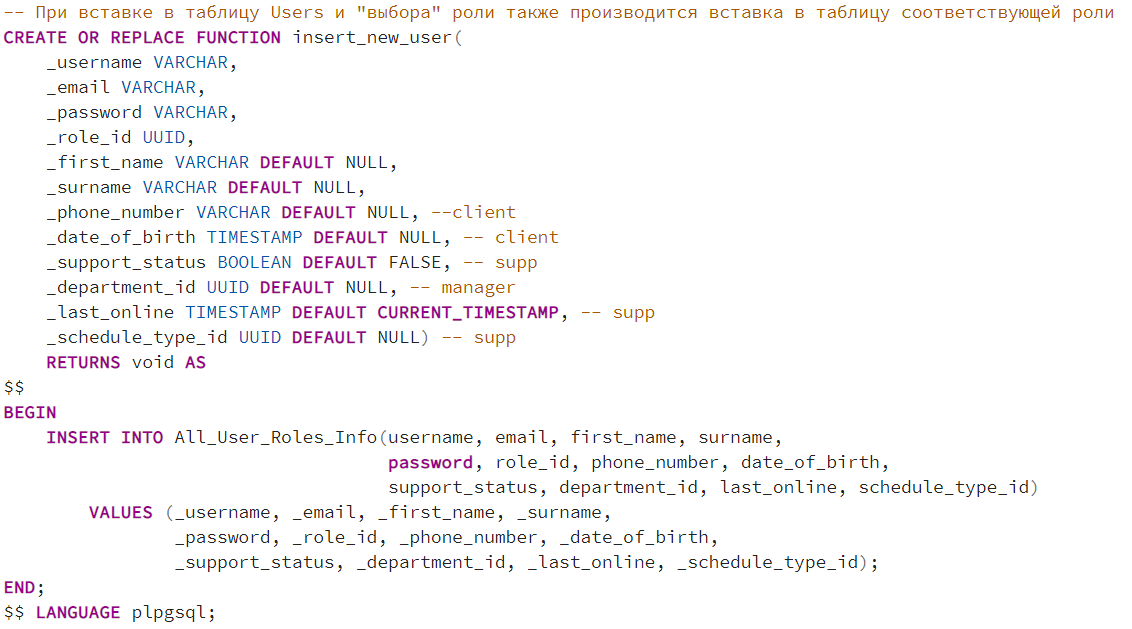


Рисунок 4.1 – Код функции insert\_new\_user

### 4.2.2 Функция generate\_uuid

Данная функция предназначена для генерации уникальных идентификаторов для каждой таблицы. Код функции представлен на рисунке 4.2:

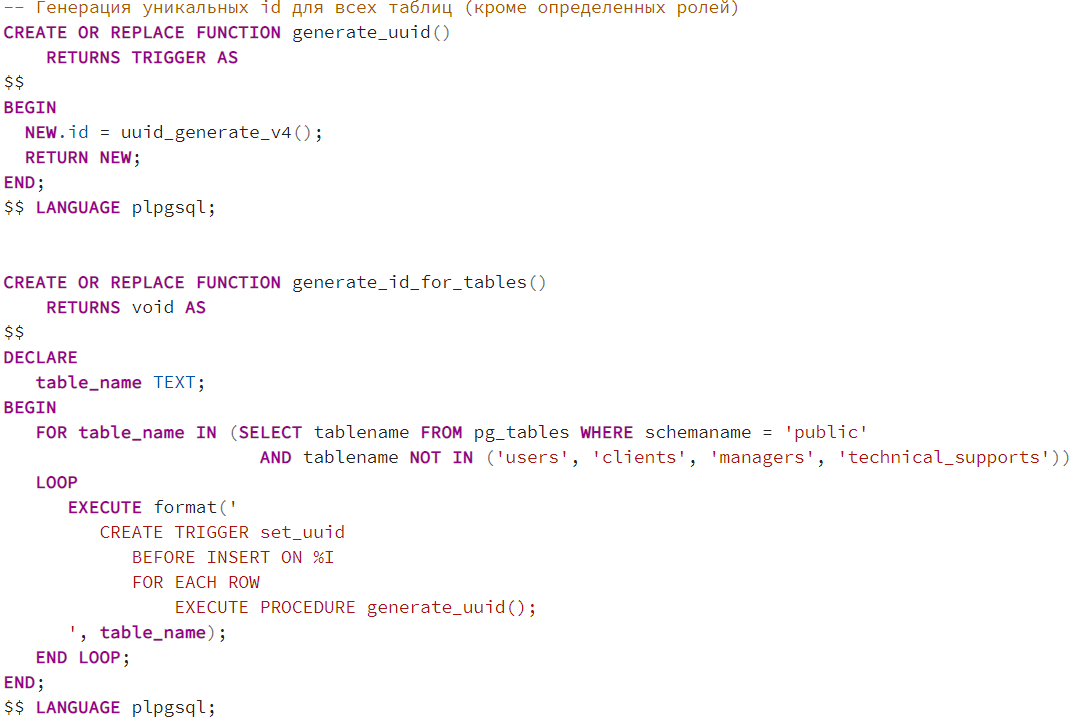


Рисунок 4.2 – Код функции generate\_uuid

### 4.2.3 Функция hash\_password

Данная функция предназначена для хеширования значений паролей. Код функции представлен на рисунке 4.3:

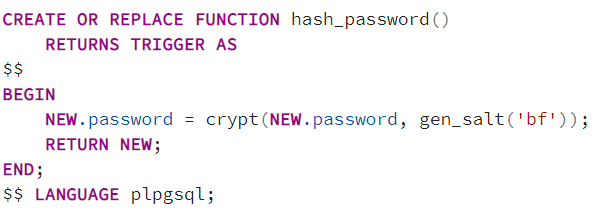


Рисунок 4.3 – Код функции hash\_password

### 4.2.4 Процедура add\_currency

Данная процедура принимает на вход короткое и полное названия валюты и затем вставляет запись в таблицу валют. Код процедуры представлен на рисунке 4.4:

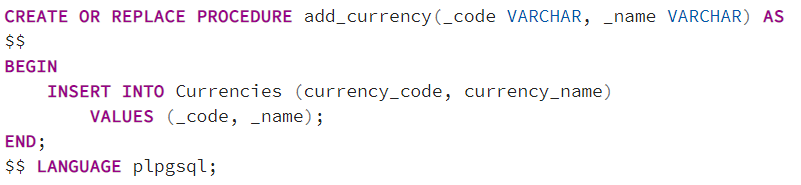


Рисунок 4.4 – Процедура add\_currency

### 4.2.5 Процедура add\_credit\_type

Данная процедура принимает на вход название кредита, минимальную и максимальную возможные суммы, кредитную ставку и длительность в месяцах и затем вставляет запись в таблицу типов кредитов. Код реализации представлен на рисунке 4.5:

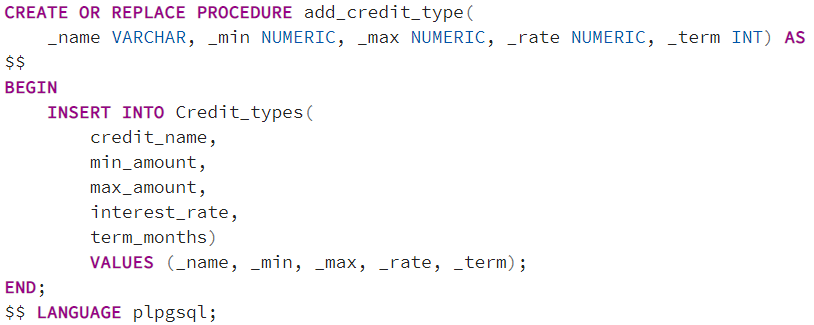


Рисунок 4.5 – Процедура add\_credit\_type

Процедура add\_deposit\_type реализована по такому же принципу. Код ее реализации представлен в приложении А.

### 4.2.6 Функции вставки и обновления курсов валют

Данные функции в отличии от других реализованы не в PostgreSQL, а на языке Python.

Функция create\_exchanges принимает на вход идентификаторы базовой валюты и валюты, в которую необходимо конвертировать значение, и выполняет вставку записи в таблице, причем курс обмена изначально установлен в 0. Затем эта функция вызывается в цикле для того, чтобы перебрать все возможные не повторяющиеся пары валют. Реализация представлена на рисунке 4.6:

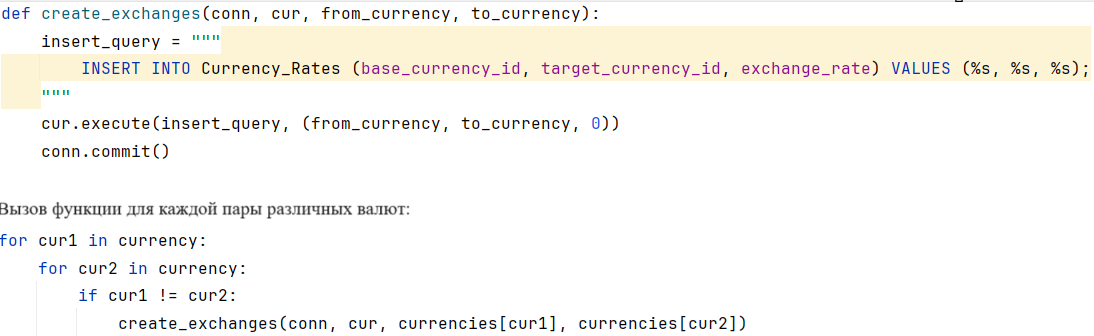


Рисунок 4.6 – Функция create\_exchanges

Функция update\_exchange\_rate принимает на вход идентификаторы базовой валюты и валюты, в которую необходимо конвертировать значение, и курс и выполняет обновление значения столбца курса в таблице, причем время последнего обновления ставится текущее. Затем, как и в предыдущей функции она вызывается в цикле, чтобы изменить все записи в таблице с курсами валют. Код функции представлен на рисунке 4.7:

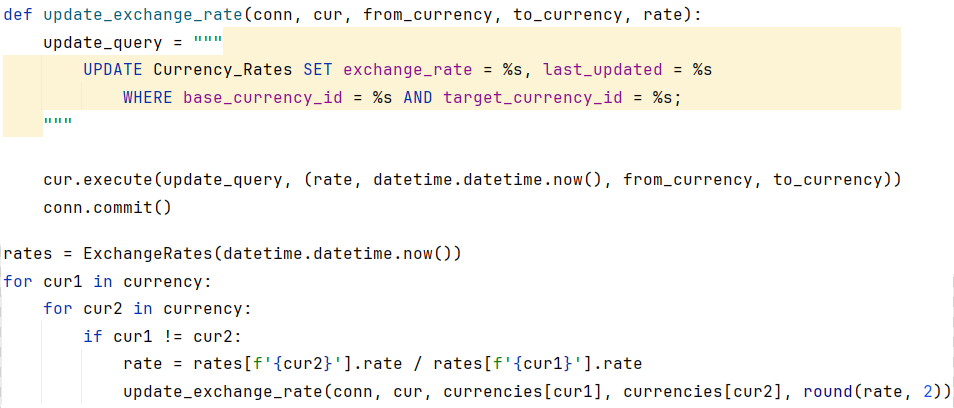


Рисунок 4.7 – Функция update\_exchange\_rate

### 4.2.7 Функция update\_schedule

Она также реализована на языке Python. Данная функция принимает на вход идентификатор специалиста технической поддержки и его название расписания. Данная функция обновляет статус специалиста в зависимости от его расписания и текущего времени. Так например специалист, у которого расписание «Без перерывов» будет в сети с 9 утра до 17 вечера, а специалист, у которого расписание «С перерывами» будет в сети в промежутки времени: с 10 до 12, с 14 до 16, с 18 до 22. Затем она также как и предыдущие функции вызывается в цикле для каждого существующего специалиста. Реализация данной функции представлена на рисунке 4.8:

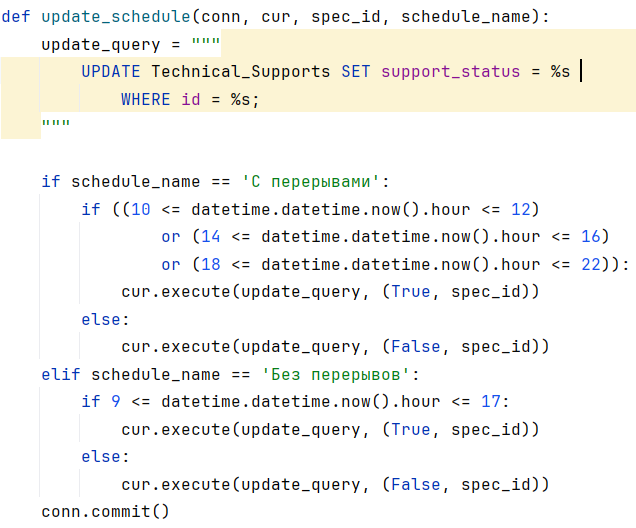


Рисунок 4.8 – Функция update\_schedule

## 4.3 Создание триггеров

Программный SQL-код всех триггеров и их функций, описанных в данном разделе, представлен в приложении А.

### 4.3.1 Триггер support\_request\_insert

Данный триггер нужен для того, чтобы, когда у специалиста менялся статус (т.е. специалист заходит в сеть), у пользовательского запроса менялся статус со значения «Отправлен» на «Принят». Также этот триггер назначает специалиста в сети, либо специалиста, который недавнее всего был в сети, для ответа на запрос. Реализация представлена на рисунке 4.9:

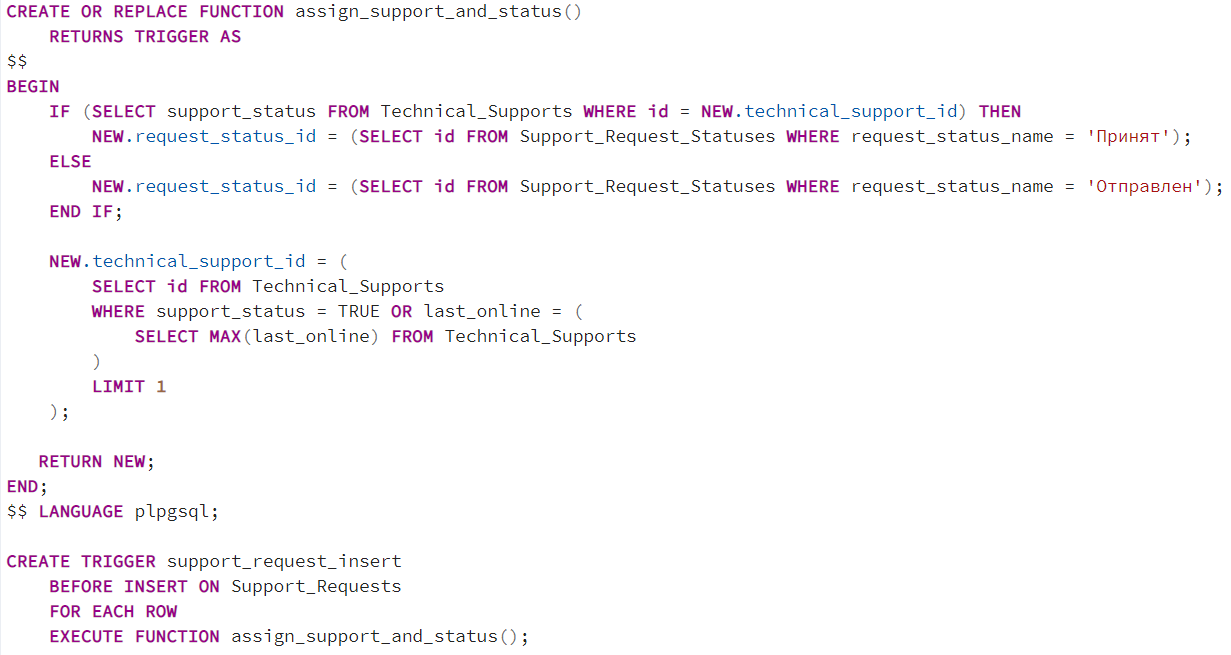


Рисунок 4.9 – Триггер support\_request\_insert

### 4.3.2 Триггер support\_response\_insert

Данный триггер срабатывает после вставки нового значения в таблицу ответов службы поддержки. Он обновляет статус запроса на значение «Закрыт» в таблице запросов, что означает, что специалист отправил ответ на запрос пользователя. Реализация представлена на рисунке 4.10:

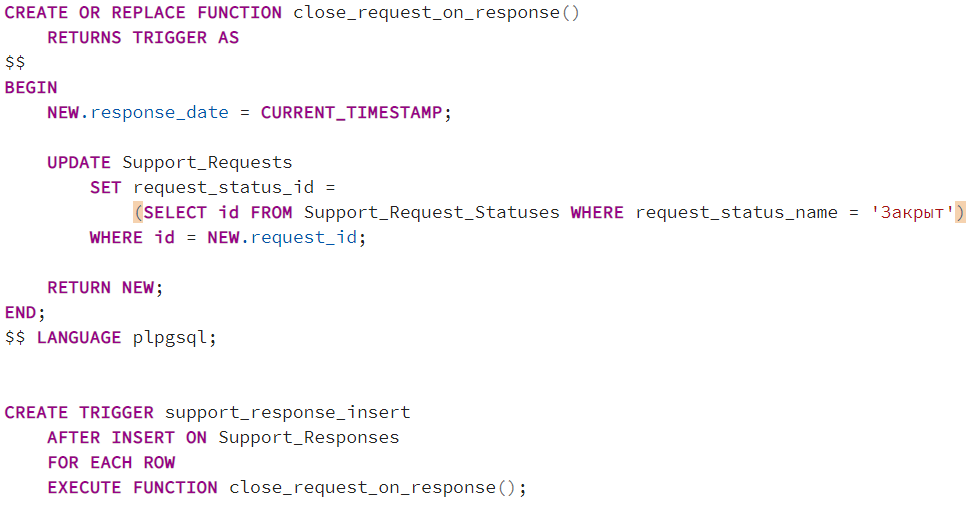


Рисунок 4.10 – Триггер support\_response\_insert

### 4.3.3 Триггер create\_new\_transaction

Данный триггер срабатывает до вставки нового значения в таблицу транзакций. Он проверяет валюты счетов отправителя и получателя, и если они отличны, то конвертирует переводимую сумму в валюту счета получателя по установленному курсу. Реализация представлена на рисунке 4.11:

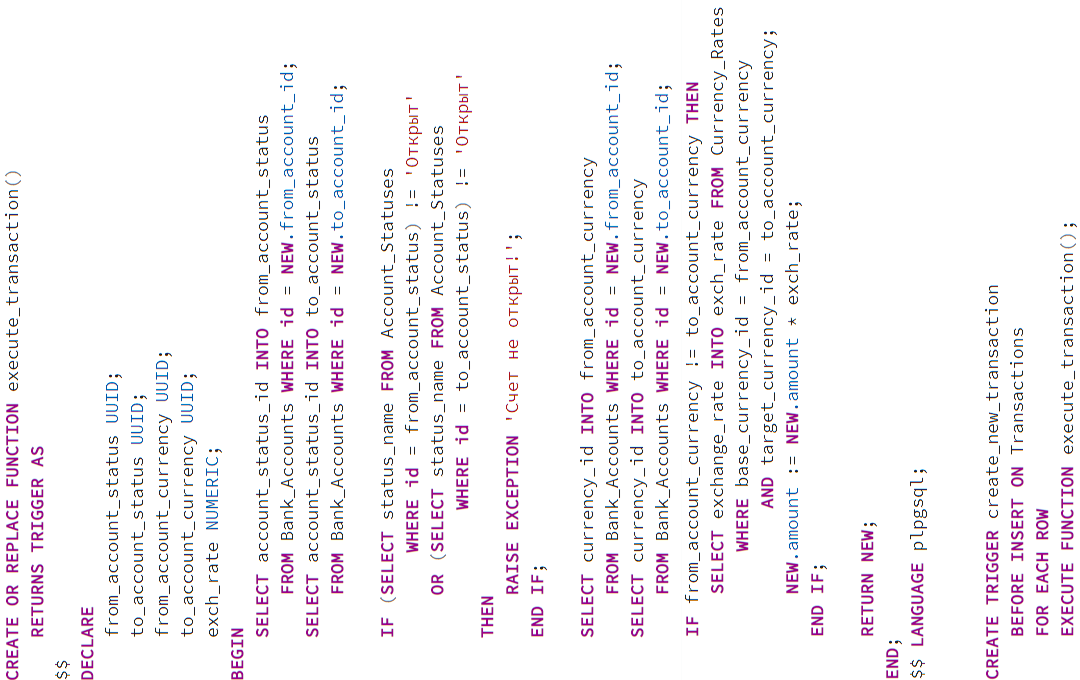


Рисунок 4.11 – Триггер create\_new\_transaction

### 4.3.4 Триггер update\_balances

Данный триггер срабатывает уже после вставки значений в таблицу транзакций. Он отнимает значение переводимой суммы со счета отправителя и прибавляет ее на счет получателя. Реализация на рисунке 4.12:

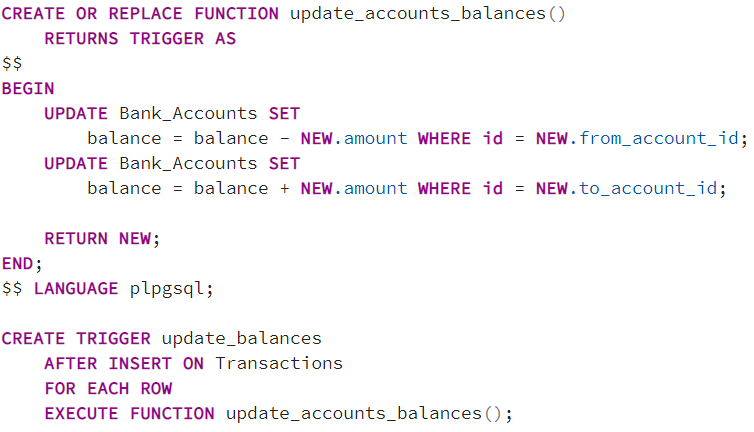


Рисунок 4.12 – Триггер update\_balances

### 4.3.5 Триггер credit\_request

Этот триггер срабатывает до вставки значения в таблицу запросов на кредит. Он проверят соответствует ли запрошенная сумма возможной кредитной сумме и назначает менеджера по кредиту, который работает в том же городе, что запросил пользователь. Реализация представлена на рисунке 4.13:

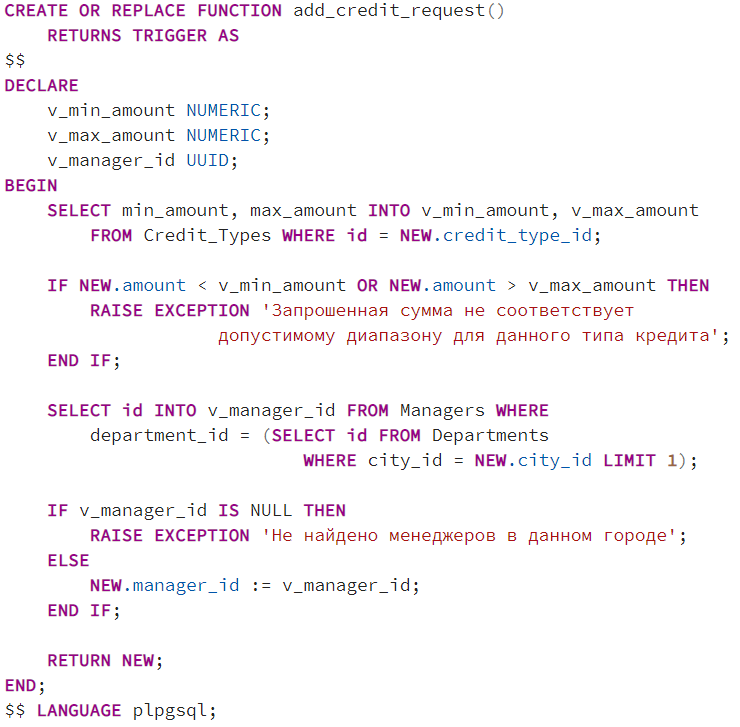


Рисунок 4.13 – Триггер credit\_request

### 4.3.6 Триггер update\_credit\_request\_status

Триггер срабатывает до вставки или обновления в таблицу заявок на кредит. Он проверяет заполнена ли вся информация о клиенте, если она не заполнена, или клиенту меньше 18 лет, то менеджер должен отклонить заявку, поэтому обновляется статус заявки в таблице заявок на кредит. Реализация представлена на рисунке 4.14:

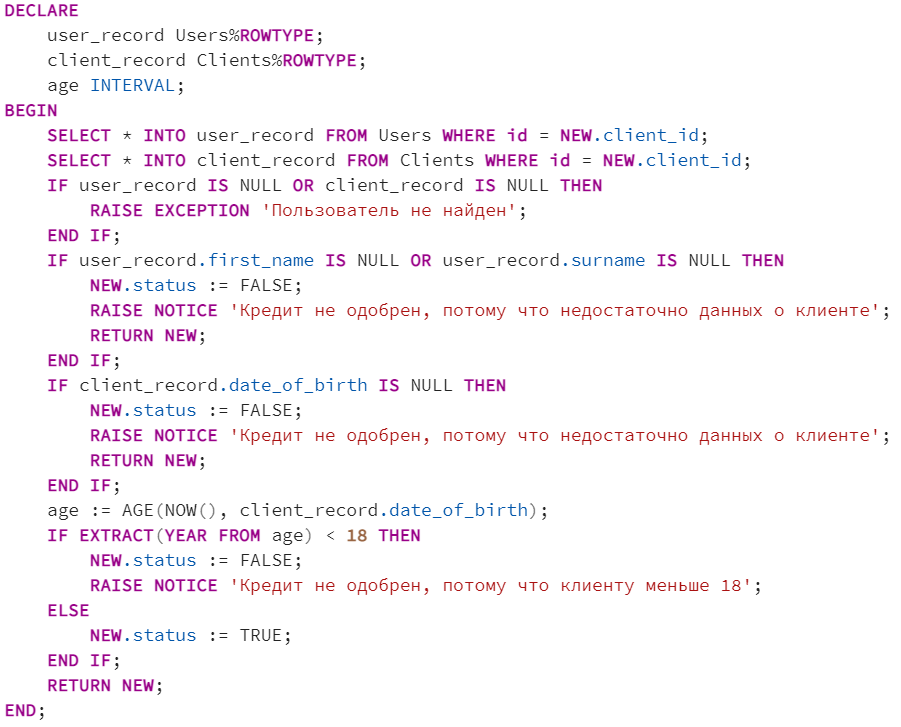


Рисунок 4.14 – Триггер update\_credit\_request\_status

### 4.3.7 Триггер create\_credit\_account

Данный триггер срабатывает после вставки или обновлений в таблице запросов на кредит. Если заявка одобрена, то создается счет для кредита, причем дата открытия кредита берется сегодняшняя, а дата закрытия пересчитывается как сегодняшняя плюс срок кредита. Реализация на рисунке 4.15:

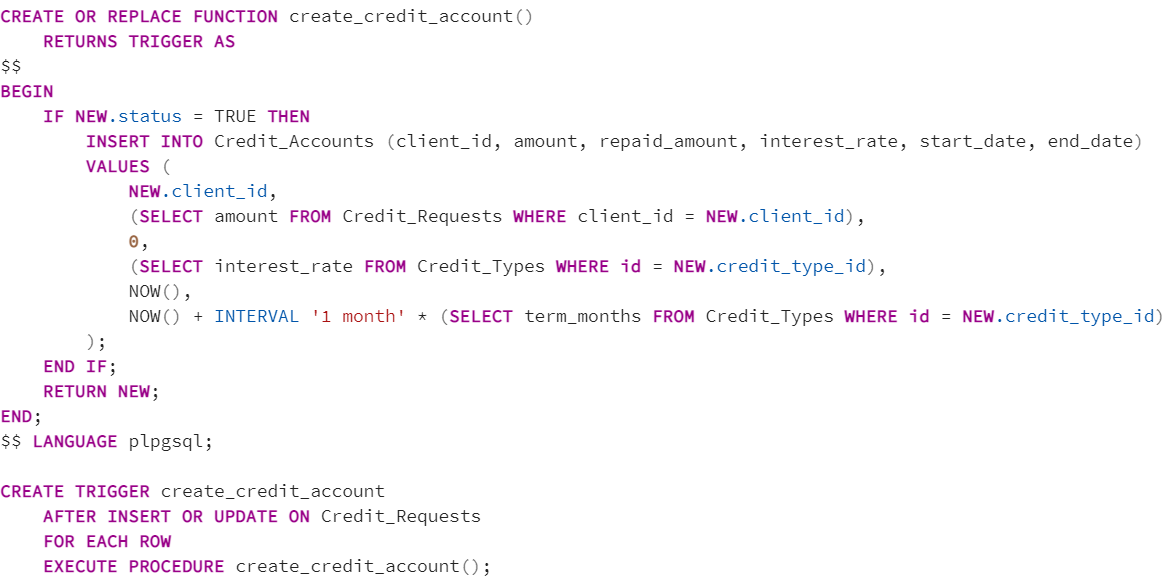


Рисунок 4.15 – Триггер create\_credit\_account

### 4.3.8 Триггер create\_credit\_transaction

Этот триггер срабатывает после вставки значения в таблицу кредитных счетов. Создается новая транзакция «Выдача кредита». Реализация представлена на рисунке 4.16:

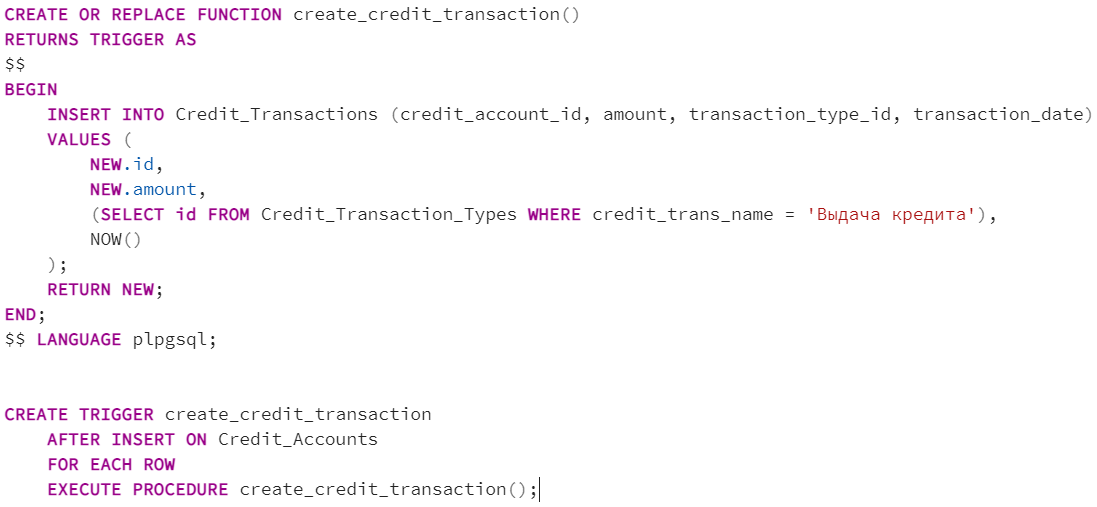


Рисунок 4.16 – Триггер create\_credit\_transaction

### 4.3.9 Триггер new\_credit\_transaction

Триггер срабатывает после вставки в таблицу кредитных транзакций. У нас создается новая категория транзакций, если ее еще не существует. Затем выбирается банковский счет с наибольшим балансом и производится списание в пользу погашения кредита. Если что-то не проходит, то выводятся соответствующие сообщения. Реализация представлена на рисунке 4.17:

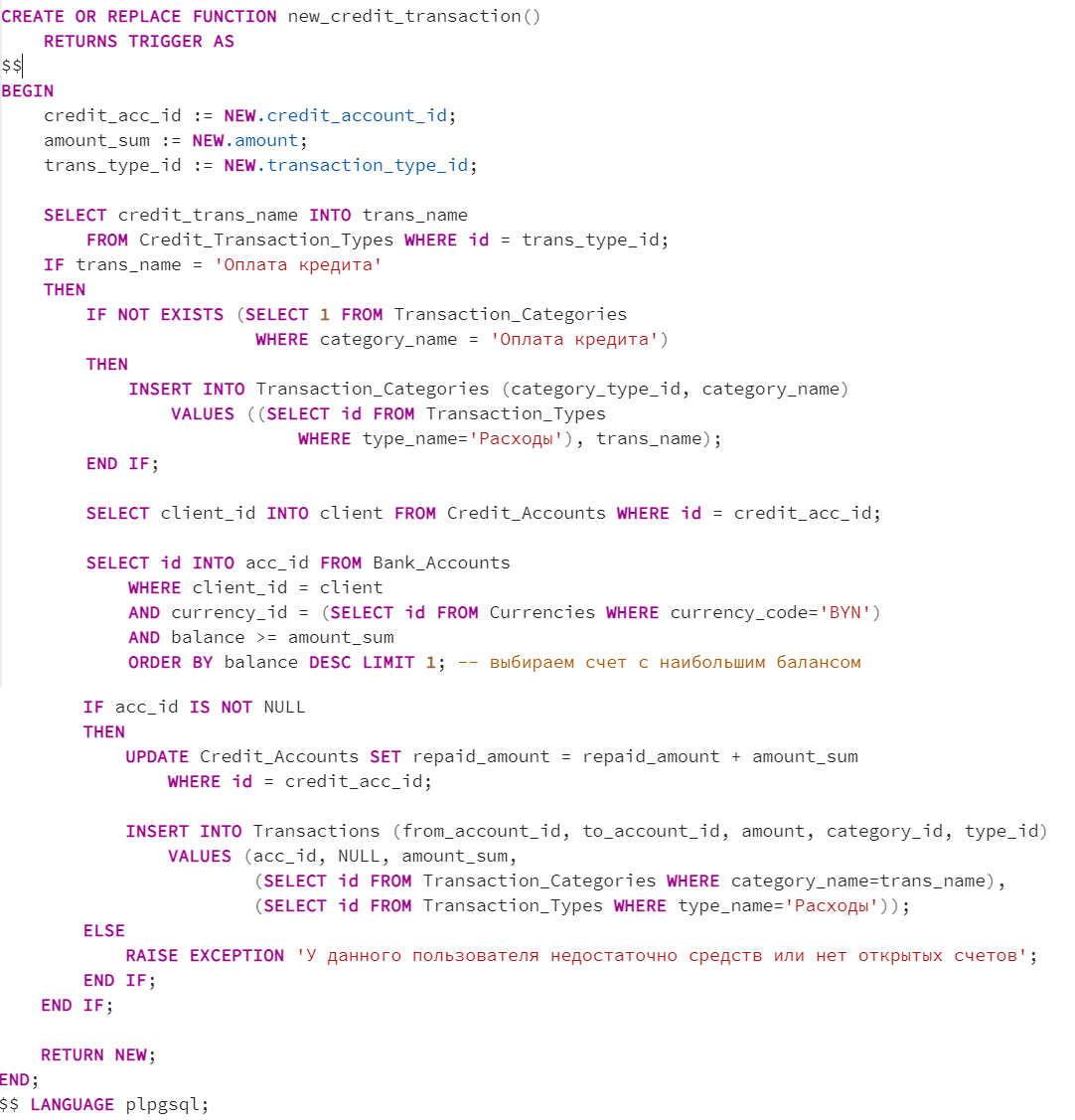


Рисунок 4.17 – Триггер new\_credit\_transaction

# 5 ТЕСТИРОВАНИЕ БАЗЫ ДАННЫХ

После реализации физической модели базы данных путём написания sql-скриптов, необходимо протестировать базу данных и убедиться, что всё работает корректно. Для этого запустим некоторые процедуры и посмотрим на результат.

Результат вызова функции вставки нового пользователя представлен на рисунках 5.1-5.2:

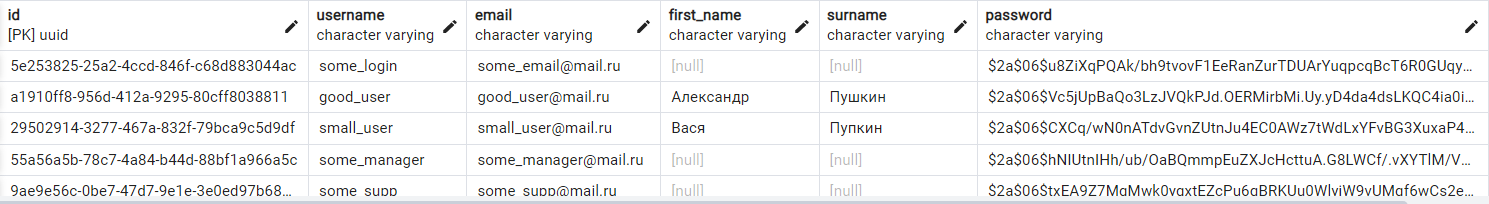


Рисунок 5.1 – Результат вызова insert\_new\_user в таблице Users

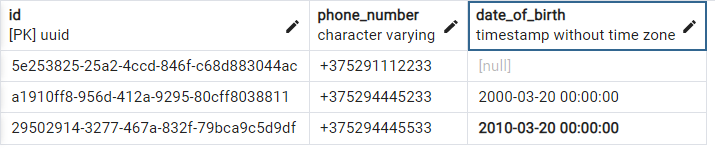


Рисунок 5.2 – Результат вызова insert\_new\_user в таблице Clients

Можно заметить, что у нас также работают наши функции и триггеры, один из которых генерирует уникальный идентификатор при вставке в любую таблицу значений, а второй генерирует хеш-значение для пароля.

На рисунке 5.3 представлен результат работы функции обновления курса валют:

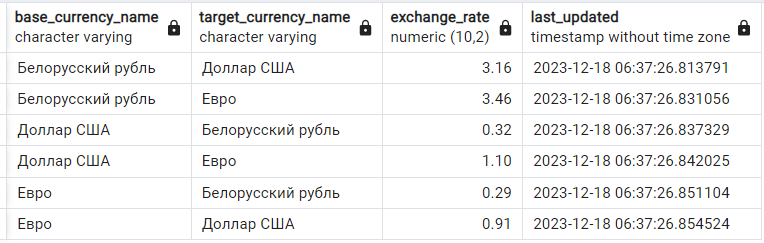


Рисунок 5.3 – Результат выполнения update\_exchange\_rate

На рисунке 5.4 представлен результат работы функций update\_accounts\_balances и execute\_transaction:

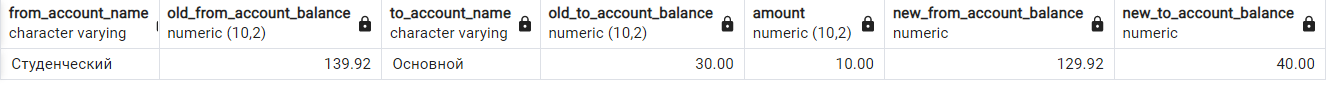


Рисунок 5.4 – Результат выполнения execute\_transaction

На рисунке 5.5 представлен результат работы тех же функций, но при переводе между счетами с разными валютами:

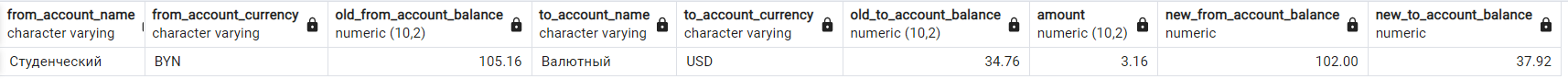


Рисунок 5.5 – Результат выполнения execute\_transaction с разными валютами

Проверим функцию add\_credit\_request с разными пользователями. Результаты работы представлены на рисунках 5.6-5.8:

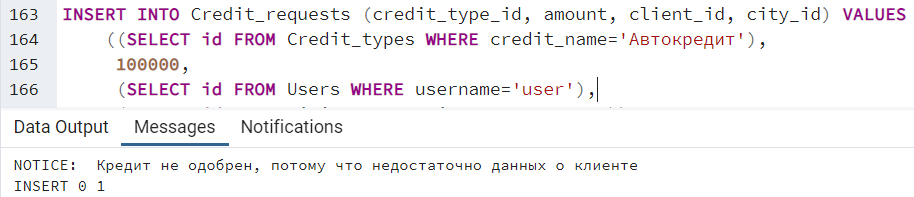


Рисунок 5.6 – Результат выполнения add\_credit\_request

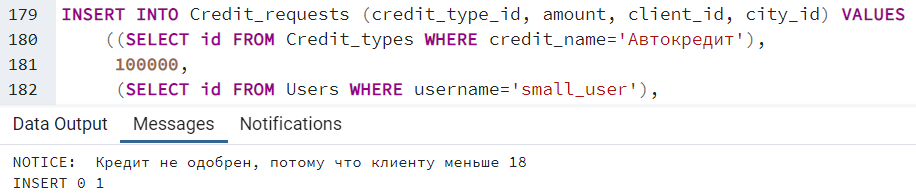


Рисунок 5.7 – Результат выполнения add\_credit\_request

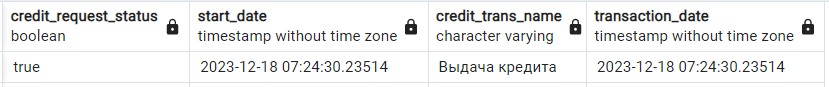


Рисунок 5.8 – Успешный результат выполнения add\_credit\_request

На рисунке 5.8 видно, что после создания заявки на кредит, она сразу была одобрена и создался кредитный счет и соответствующая транзакция.

Проверим функцию new\_credit\_transaction. Результат работы представлен на рисунке 5.9:

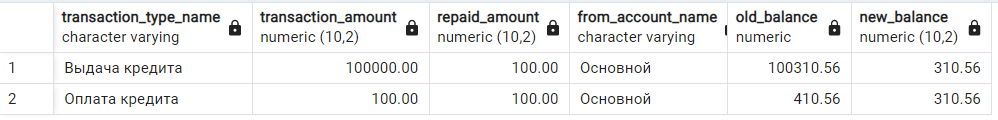


Рисунок 5.9 – Результат выполнения new\_credit\_transaction

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была спроектирована и создана база данных в рамках предметной области для приложения «Система интерент-банкинг».

В ходе выполнения работы было произведено сравнение основных аспектов реляционных и не реляционных баз данных, основных их свойств и особенностей. Была выбрана база данных для реализации данного проекта – PostgreSQL, которая уже не первый год является стандартом разработки различных веб-приложений в компаниях по всему миру.

Помимо всего, был разработан список запросов к базе данных, реализованы триггеры, процедуры, функции, индексы, ограничения и другие вещи, поддерживающие работу базы данных в исправном состоянии, а также реализующие основной функционал.

Таким образом, цели данной курсовой работу могут считаться достигнутыми, а задачи – выполненными.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Интернет-банкинг – НБРБ [Электронный ресурс]. – Режим доступа: <http://fingramota.by/ru/guide/cashless-payments/online-banking>

[2] Интернет-банкинг Беларусбанк [Электронный ресурс]. – Режим доступа: <https://belarusbank.by/ru/fizicheskim_licam/31886/internet_banking>

[3] "Интернет-банкинг" Белинвестбанка [Электронный ресурс]. – Режим доступа: <https://www.belinvestbank.by/individual/page/internet-banking>

[4] Myfin.by | Банки Беларуси. Кредиты. Вклады. Курсы валют [Электронный ресурс]. – Режим доступа: <https://myfin.by/>

[5] База данных Oracle. Структура и основные понятия СУБД Oracle [Электронный ресурс]. – Режим доступа: <https://otus.ru/nest/post/1577/>

[6] PostgreSQL: что это за СУБД, основы и преимущества [Электронный ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/postgresql/>

[7] Инфологическая модель данных [Электронный ресурс]. – Режим доступа: <https://studbooks.net/2256941/informatika/infologicheskaya_model_dannyh>

[8] Что такое ER-диаграмма и как ее создать? [Электронный ресурс]. – Режим доступа: <https://www.lucidchart.com/pages/er-diagrams>

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

-- create tables scripts

CREATE TABLE IF NOT EXISTS Roles (

id UUID PRIMARY KEY,

role\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Users (

id UUID PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE,

email VARCHAR(50) NOT NULL UNIQUE,

first\_name VARCHAR(50),

surname VARCHAR(50),

password VARCHAR(200) NOT NULL,

role\_id UUID NOT NULL,

CONSTRAINT FK\_on\_Role FOREIGN KEY (role\_id) REFERENCES Roles(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Clients (

id UUID NOT NULL PRIMARY KEY REFERENCES Users(id),

phone\_number VARCHAR(17) NOT NULL UNIQUE,

date\_of\_birth TIMESTAMP

);

CREATE TABLE IF NOT EXISTS Support\_Schedules (

id UUID PRIMARY KEY,

schedule\_type\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Technical\_Supports (

id UUID NOT NULL PRIMARY KEY REFERENCES Users(id),

support\_status BOOLEAN DEFAULT FALSE,

last\_online TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

schedule\_type\_id UUID NOT NULL,

CONSTRAINT FK\_on\_Schedules FOREIGN KEY (schedule\_type\_id) REFERENCES Support\_Schedules(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Cities (

id UUID PRIMARY KEY,

city\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Departments (

id UUID PRIMARY KEY,

department\_name VARCHAR(50) NOT NULL UNIQUE,

city\_id UUID NOT NULL,

department\_address VARCHAR(200) NOT NULL UNIQUE,

CONSTRAINT FK\_on\_Cities FOREIGN KEY (city\_id) REFERENCES Cities(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Managers (

id UUID NOT NULL PRIMARY KEY REFERENCES Users(id),

department\_id UUID NOT NULL,

CONSTRAINT FK\_on\_Departments FOREIGN KEY (department\_id) REFERENCES Departments(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Support\_Request\_Statuses (

id UUID PRIMARY KEY,

request\_status\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Support\_Requests (

id UUID PRIMARY KEY,

client\_id UUID NOT NULL,

technical\_support\_id UUID NOT NULL,

request\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

request\_message VARCHAR(500) NOT NULL,

request\_status\_id UUID NOT NULL,

CONSTRAINT FK\_on\_Clients FOREIGN KEY (client\_id) REFERENCES Clients(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Technical\_Supports FOREIGN KEY (technical\_support\_id) REFERENCES Technical\_Supports(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Request\_Status FOREIGN KEY (request\_status\_id) REFERENCES Support\_Request\_Statuses(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Support\_Responses (

id UUID PRIMARY KEY,

technical\_support\_id UUID NOT NULL,

request\_id UUID NOT NULL,

response\_date TIMESTAMP,

response\_message VARCHAR(500),

CONSTRAINT FK\_on\_Technical\_Supports FOREIGN KEY (technical\_support\_id) REFERENCES Technical\_Supports(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Support\_Requests FOREIGN KEY (request\_id) REFERENCES Support\_Requests(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Credit\_Types (

id UUID PRIMARY KEY,

credit\_name VARCHAR(50) NOT NULL,

min\_amount NUMERIC(10, 2) NOT NULL,

max\_amount NUMERIC(10, 2) NOT NULL,

interest\_rate NUMERIC(10, 2) NOT NULL,

term\_months INT NOT NULL

);

CREATE TABLE IF NOT EXISTS Credit\_Requests (

id UUID PRIMARY KEY,

credit\_type\_id UUID NOT NULL,

amount NUMERIC(10, 2) NOT NULL,

client\_id UUID NOT NULL,

manager\_id UUID NOT NULL,

request\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status BOOLEAN DEFAULT FALSE,

city\_id UUID NOT NULL,

CONSTRAINT FK\_on\_Credit\_Types FOREIGN KEY (credit\_type\_id) REFERENCES Credit\_Types(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Clients FOREIGN KEY (client\_id) REFERENCES Clients(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Managers FOREIGN KEY (manager\_id) REFERENCES Managers(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Credit\_Accounts (

id UUID PRIMARY KEY,

client\_id UUID NOT NULL,

amount NUMERIC(10, 2) NOT NULL,

repaid\_amount NUMERIC(10, 2) NOT NULL,

interest\_rate NUMERIC(10, 2) NOT NULL,

start\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

end\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_on\_Clients FOREIGN KEY (client\_id) REFERENCES Clients(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Credit\_Transaction\_Types (

id UUID PRIMARY KEY,

credit\_trans\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Credit\_Transactions (

id UUID PRIMARY KEY,

credit\_account\_id UUID NOT NULL,

amount NUMERIC(10, 2) NOT NULL,

transaction\_type\_id UUID NOT NULL,

transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_on\_Credit\_Accounts FOREIGN KEY (credit\_account\_id) REFERENCES Credit\_Accounts(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Credit\_Transaction\_Types FOREIGN KEY (transaction\_type\_id) REFERENCES Credit\_Transaction\_Types(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Deposit\_Types (

id UUID PRIMARY KEY,

deposit\_name VARCHAR(50) NOT NULL,

min\_amount NUMERIC(10, 2) NOT NULL,

max\_amount NUMERIC(10, 2) NOT NULL,

interest\_rate NUMERIC(10, 2) NOT NULL,

term\_months INT NOT NULL

);

CREATE TABLE IF NOT EXISTS Deposit\_Requests (

id UUID PRIMARY KEY,

deposit\_type\_id UUID NOT NULL,

amount NUMERIC(10, 2) NOT NULL,

client\_id UUID NOT NULL,

manager\_id UUID NOT NULL,

request\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status BOOLEAN DEFAULT FALSE,

city\_id UUID NOT NULL,

CONSTRAINT FK\_on\_Deposit\_Types FOREIGN KEY (deposit\_type\_id) REFERENCES Deposit\_Types(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Clients FOREIGN KEY (client\_id) REFERENCES Clients(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Managers FOREIGN KEY (manager\_id) REFERENCES Managers(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Deposit\_Accounts (

id UUID PRIMARY KEY,

client\_id UUID NOT NULL,

amount NUMERIC(10, 2) NOT NULL,

repaid\_amount NUMERIC(10, 2) NOT NULL,

interest\_rate NUMERIC(10, 2) NOT NULL,

start\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

end\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_on\_Clients FOREIGN KEY (client\_id) REFERENCES Clients(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Deposit\_Transaction\_Types (

id UUID PRIMARY KEY,

deposit\_trans\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Deposit\_Transactions (

id UUID PRIMARY KEY,

deposit\_account\_id UUID NOT NULL,

amount NUMERIC(10, 2) NOT NULL,

transaction\_type\_id UUID NOT NULL,

transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_on\_Deposit\_Accounts FOREIGN KEY (deposit\_account\_id) REFERENCES Deposit\_Accounts(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Deposit\_Transaction\_Types FOREIGN KEY (transaction\_type\_id) REFERENCES Deposit\_Transaction\_Types(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Currencies (

id UUID PRIMARY KEY,

currency\_code VARCHAR(3) NOT NULL,

currency\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Currency\_Rates (

id UUID PRIMARY KEY,

base\_currency\_id UUID NOT NULL,

target\_currency\_id UUID NOT NULL,

exchange\_rate NUMERIC(10, 2) NOT NULL,

last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_on\_Currency1 FOREIGN KEY (base\_currency\_id) REFERENCES Currencies(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Currency2 FOREIGN KEY (target\_currency\_id) REFERENCES Currencies(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Account\_Statuses (

id UUID PRIMARY KEY,

status\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Bank\_Accounts (

id UUID PRIMARY KEY,

account\_name VARCHAR(50) NOT NULL,

client\_id UUID NOT NULL,

balance NUMERIC(10, 2) NOT NULL,

currency\_id UUID NOT NULL,

creation\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

account\_status\_id UUID NOT NULL,

CONSTRAINT FK\_on\_Clients FOREIGN KEY (client\_id) REFERENCES Clients(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Currencies FOREIGN KEY (currency\_id) REFERENCES Currencies(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Account\_Statuses FOREIGN KEY (account\_status\_id) REFERENCES Account\_Statuses(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Transaction\_Types (

id UUID PRIMARY KEY,

type\_name VARCHAR(50) NOT NULL

);

CREATE TABLE IF NOT EXISTS Transaction\_Categories (

id UUID PRIMARY KEY,

category\_type\_id UUID NOT NULL,

category\_name VARCHAR(50) NOT NULL,

CONSTRAINT FK\_on\_Trans\_types FOREIGN KEY (category\_type\_id) REFERENCES Transaction\_Types(id) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Transactions (

id UUID PRIMARY KEY,

from\_account\_id UUID NOT NULL,

to\_account\_id UUID,

amount NUMERIC(10, 2) NOT NULL,

category\_id UUID NOT NULL,

type\_id UUID NOT NULL,

transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT FK\_on\_Account1 FOREIGN KEY (from\_account\_id) REFERENCES Bank\_Accounts(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Account2 FOREIGN KEY (to\_account\_id) REFERENCES Bank\_Accounts(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Categories FOREIGN KEY (category\_id) REFERENCES Transaction\_Categories(id) ON DELETE CASCADE,

CONSTRAINT FK\_on\_Trans\_types FOREIGN KEY (type\_id) REFERENCES Transaction\_Types(id) ON DELETE CASCADE

);

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- Генерация уникальных id для всех таблиц (кроме определенных ролей)

CREATE OR REPLACE FUNCTION generate\_uuid()

RETURNS TRIGGER AS

$$

BEGIN

NEW.id = uuid\_generate\_v4();

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION generate\_id\_for\_tables()

RETURNS void AS

$$

DECLARE

table\_name TEXT;

BEGIN

FOR table\_name IN (SELECT tablename FROM pg\_tables WHERE schemaname = 'public'

AND tablename NOT IN ('users', 'clients', 'managers', 'technical\_supports'))

LOOP

EXECUTE format('

CREATE TRIGGER set\_uuid

BEFORE INSERT ON %I

FOR EACH ROW

EXECUTE PROCEDURE generate\_uuid();

', table\_name);

END LOOP;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION generate\_id\_for\_one\_new\_table(\_table\_name VARCHAR)

RETURNS void AS

$$

DECLARE

BEGIN

EXECUTE format('

CREATE TRIGGER set\_uuid

BEFORE INSERT ON %I

FOR EACH ROW

EXECUTE PROCEDURE generate\_uuid();

', \_table\_name);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION hash\_password()

RETURNS TRIGGER AS

$$

BEGIN

NEW.password = crypt(NEW.password, gen\_salt('bf'));

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

SELECT generate\_id\_for\_tables(); -- единичный вызов после создания

CREATE OR REPLACE PROCEDURE add\_currency(\_code VARCHAR, \_name VARCHAR) AS

$$

BEGIN

INSERT INTO Currencies (currency\_code, currency\_name)

VALUES (\_code, \_name);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE add\_credit\_type(

\_name VARCHAR, \_min NUMERIC, \_max NUMERIC, \_rate NUMERIC, \_term INT) AS

$$

BEGIN

INSERT INTO Credit\_types(

credit\_name,

min\_amount,

max\_amount,

interest\_rate,

term\_months)

VALUES (\_name, \_min, \_max, \_rate, \_term);

END;

$$ LANGUAGE plpgsql;

-- При вставке в таблицу Users и "выбора" роли также производится вставка в таблицу соответствующей роли

CREATE OR REPLACE FUNCTION insert\_new\_user(

\_username VARCHAR,

\_email VARCHAR,

\_password VARCHAR,

\_role\_id UUID,

\_first\_name VARCHAR DEFAULT NULL,

\_surname VARCHAR DEFAULT NULL,

\_phone\_number VARCHAR DEFAULT NULL, --client

\_date\_of\_birth TIMESTAMP DEFAULT NULL, -- client

\_support\_status BOOLEAN DEFAULT FALSE, -- supp

\_department\_id UUID DEFAULT NULL, -- manager

\_last\_online TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- supp

\_schedule\_type\_id UUID DEFAULT NULL) -- supp

RETURNS void AS

$$

BEGIN

INSERT INTO All\_User\_Roles\_Info(username, email, first\_name, surname,

password, role\_id, phone\_number, date\_of\_birth,

support\_status, department\_id, last\_online, schedule\_type\_id)

VALUES (\_username, \_email, \_first\_name, \_surname,

\_password, \_role\_id, \_phone\_number, \_date\_of\_birth,

\_support\_status, \_department\_id, \_last\_online, \_schedule\_type\_id);

END;

$$ LANGUAGE plpgsql;

DROP FUNCTION insert\_new\_user;

CREATE OR REPLACE FUNCTION distribute\_user\_info()

RETURNS TRIGGER AS

$$

BEGIN

INSERT INTO Users(id, username, email, first\_name, surname, password, role\_id)

VALUES (NEW.id, NEW.username, NEW.email, NEW.first\_name, NEW.surname, NEW.password, NEW.role\_id);

IF NEW.role\_id = (SELECT id FROM Roles WHERE role\_name = 'Клиент') THEN

INSERT INTO Clients(id, phone\_number, date\_of\_birth)

VALUES (NEW.id, NEW.phone\_number, NEW.date\_of\_birth);

ELSIF NEW.role\_id = (SELECT id FROM Roles WHERE role\_name = 'Менеджер') THEN

INSERT INTO Managers(id, department\_id)

VALUES (NEW.id, NEW.department\_id);

ELSIF NEW.role\_id = (SELECT id FROM Roles WHERE role\_name = 'Специалист технической поддержки') THEN

INSERT INTO Technical\_Supports(id, support\_status, last\_online, schedule\_type\_id)

VALUES (NEW.id, NEW.support\_status, NEW.last\_online, NEW.schedule\_type\_id);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Функция генерирующая название отдела

CREATE OR REPLACE FUNCTION generate\_department\_name()

RETURNS TRIGGER AS

$$

DECLARE

city\_rank INT;

department\_number INT;

new\_department\_name VARCHAR;

BEGIN

SELECT dept\_number INTO city\_rank

FROM City\_Department\_Number WHERE city\_id = NEW.city\_id;

SELECT last\_department\_number + 1 INTO department\_number

FROM City\_Department\_Counters WHERE city\_id = NEW.city\_id;

IF department\_number IS NULL THEN

department\_number := 1;

END IF;

INSERT INTO City\_Department\_Counters(city\_id, last\_department\_number)

VALUES (NEW.city\_id, department\_number)

ON CONFLICT (city\_id) DO UPDATE SET last\_department\_number = department\_number;

new\_department\_name := 'Отделение №' || city\_rank || '00/' || city\_rank || '00' || department\_number;

NEW.department\_name := new\_department\_name;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- обновление времени последнего пребывания в сети

CREATE OR REPLACE FUNCTION update\_last\_online()

RETURNS TRIGGER AS

$$

BEGIN

IF NEW.support\_status = FALSE THEN

NEW.last\_online = CURRENT\_TIMESTAMP;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- запросы и их статус

CREATE OR REPLACE FUNCTION assign\_support\_and\_status()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT support\_status FROM Technical\_Supports WHERE id = NEW.technical\_support\_id) THEN

NEW.request\_status\_id = (SELECT id FROM Support\_Request\_Statuses WHERE request\_status\_name = 'Принят');

ELSE

NEW.request\_status\_id = (SELECT id FROM Support\_Request\_Statuses WHERE request\_status\_name = 'Отправлен');

END IF;

NEW.technical\_support\_id = (

SELECT id FROM Technical\_Supports

WHERE support\_status = TRUE OR last\_online = (

SELECT MAX(last\_online) FROM Technical\_Supports

)

LIMIT 1

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION insert\_request\_response()

RETURNS TRIGGER AS

$$

BEGIN

INSERT INTO Support\_Responses (technical\_support\_id, request\_id)

VALUES (NEW.technical\_support\_id, NEW.id);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_request\_status()

RETURNS TRIGGER AS

$$

BEGIN

IF NEW.support\_status = TRUE THEN

UPDATE Support\_Requests

SET request\_status\_id = (SELECT id FROM Support\_Request\_Statuses WHERE request\_status\_name = 'Принят')

WHERE technical\_support\_id = NEW.id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION close\_request\_on\_response()

RETURNS TRIGGER AS

$$

BEGIN

NEW.response\_date = CURRENT\_TIMESTAMP;

UPDATE Support\_Requests

SET request\_status\_id = (SELECT id FROM Support\_Request\_Statuses WHERE request\_status\_name = 'Закрыт')

WHERE id = NEW.request\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION set\_open\_status\_on\_insert()

RETURNS TRIGGER AS

$$

BEGIN

NEW.account\_status\_id := (SELECT id FROM Account\_Statuses WHERE status\_name = 'Открыт');

NEW.balance := 0;

NEW.creation\_date := CURRENT\_TIMESTAMP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION execute\_transaction()

RETURNS TRIGGER AS

$$

DECLARE

from\_account\_status UUID;

to\_account\_status UUID;

from\_account\_currency UUID;

to\_account\_currency UUID;

exch\_rate NUMERIC;

BEGIN

SELECT account\_status\_id INTO from\_account\_status FROM Bank\_Accounts WHERE id = NEW.from\_account\_id;

SELECT account\_status\_id INTO to\_account\_status FROM Bank\_Accounts WHERE id = NEW.to\_account\_id;

IF (SELECT status\_name FROM Account\_Statuses WHERE id = from\_account\_status) != 'Открыт'

OR (SELECT status\_name FROM Account\_Statuses WHERE id = to\_account\_status) != 'Открыт'

THEN

RAISE EXCEPTION 'Счет не открыт!';

END IF;

SELECT currency\_id INTO from\_account\_currency FROM Bank\_Accounts WHERE id = NEW.from\_account\_id;

SELECT currency\_id INTO to\_account\_currency FROM Bank\_Accounts WHERE id = NEW.to\_account\_id;

IF from\_account\_currency != to\_account\_currency THEN

SELECT exchange\_rate INTO exch\_rate FROM Currency\_Rates

WHERE base\_currency\_id = from\_account\_currency AND target\_currency\_id = to\_account\_currency;

NEW.amount := NEW.amount \* exch\_rate;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_accounts\_balances()

RETURNS TRIGGER AS

$$

BEGIN

UPDATE Bank\_Accounts SET balance = balance - NEW.amount WHERE id = NEW.from\_account\_id;

UPDATE Bank\_Accounts SET balance = balance + NEW.amount WHERE id = NEW.to\_account\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION add\_credit\_request()

RETURNS TRIGGER AS

$$

DECLARE

v\_min\_amount NUMERIC;

v\_max\_amount NUMERIC;

v\_manager\_id UUID;

BEGIN

SELECT min\_amount, max\_amount INTO v\_min\_amount, v\_max\_amount

FROM Credit\_Types WHERE id = NEW.credit\_type\_id;

IF NEW.amount < v\_min\_amount OR NEW.amount > v\_max\_amount THEN

RAISE EXCEPTION 'Запрошенная сумма не соответствует допустимому диапазону для данного типа кредита';

END IF;

SELECT id INTO v\_manager\_id FROM Managers WHERE

department\_id = (SELECT id FROM Departments WHERE city\_id = NEW.city\_id LIMIT 1);

IF v\_manager\_id IS NULL THEN

RAISE EXCEPTION 'Не найдено менеджеров в данном городе';

ELSE

NEW.manager\_id := v\_manager\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_credit\_request\_status()

RETURNS TRIGGER AS

$$

DECLARE

user\_record Users%ROWTYPE;

client\_record Clients%ROWTYPE;

age INTERVAL;

BEGIN

SELECT \* INTO user\_record FROM Users WHERE id = NEW.client\_id;

SELECT \* INTO client\_record FROM Clients WHERE id = NEW.client\_id;

IF user\_record IS NULL OR client\_record IS NULL THEN

RAISE EXCEPTION 'Пользователь не найден';

END IF;

-- Проверка на NULL для first\_name и surname в Users

IF user\_record.first\_name IS NULL OR user\_record.surname IS NULL THEN

NEW.status := FALSE;

RAISE NOTICE 'Кредит не одобрен, потому что недостаточно данных о клиенте';

RETURN NEW;

END IF;

-- Проверка на NULL для date\_of\_birth в Clients

IF client\_record.date\_of\_birth IS NULL THEN

NEW.status := FALSE;

RAISE NOTICE 'Кредит не одобрен, потому что недостаточно данных о клиенте';

RETURN NEW;

END IF;

age := AGE(NOW(), client\_record.date\_of\_birth);

IF EXTRACT(YEAR FROM age) < 18 THEN

NEW.status := FALSE;

RAISE NOTICE 'Кредит не одобрен, потому что клиенту меньше 18';

ELSE

NEW.status := TRUE;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION create\_credit\_account()

RETURNS TRIGGER AS

$$

BEGIN

IF NEW.status = TRUE THEN

INSERT INTO Credit\_Accounts (client\_id, amount, repaid\_amount, interest\_rate, start\_date, end\_date)

VALUES (

NEW.client\_id,

(SELECT amount FROM Credit\_Requests WHERE client\_id = NEW.client\_id),

0,

(SELECT interest\_rate FROM Credit\_Types WHERE id = NEW.credit\_type\_id),

NOW(),

NOW() + INTERVAL '1 month' \* (SELECT term\_months FROM Credit\_Types WHERE id = NEW.credit\_type\_id)

);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION create\_credit\_transaction()

RETURNS TRIGGER AS

$$

BEGIN

INSERT INTO Credit\_Transactions (credit\_account\_id, amount, transaction\_type\_id, transaction\_date)

VALUES (

NEW.id,

NEW.amount,

(SELECT id FROM Credit\_Transaction\_Types WHERE credit\_trans\_name = 'Выдача кредита'),

NOW()

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION delete\_credit\_account()

RETURNS TRIGGER AS

$$

BEGIN

DELETE FROM Credit\_Accounts WHERE client\_id = OLD.client\_id;

DELETE FROM Credit\_Transactions WHERE credit\_account\_id = OLD.id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION new\_credit\_transaction()

RETURNS TRIGGER AS

$$

DECLARE

credit\_acc\_id UUID;

amount\_sum NUMERIC(10, 2);

trans\_type\_id UUID;

trans\_name VARCHAR;

client UUID;

acc\_id UUID;

BEGIN

credit\_acc\_id := NEW.credit\_account\_id;

amount\_sum := NEW.amount;

trans\_type\_id := NEW.transaction\_type\_id;

SELECT credit\_trans\_name INTO trans\_name FROM Credit\_Transaction\_Types WHERE id = trans\_type\_id;

IF trans\_name = 'Оплата кредита'

THEN

IF NOT EXISTS (SELECT 1 FROM Transaction\_Categories WHERE category\_name = 'Оплата кредита')

THEN

INSERT INTO Transaction\_Categories (category\_type\_id, category\_name)

VALUES ((SELECT id FROM Transaction\_Types WHERE type\_name='Расходы'), trans\_name);

END IF;

SELECT client\_id INTO client FROM Credit\_Accounts WHERE id = credit\_acc\_id;

SELECT id INTO acc\_id FROM Bank\_Accounts

WHERE client\_id = client AND currency\_id = (SELECT id FROM Currencies WHERE currency\_code='BYN')

AND balance >= amount\_sum

ORDER BY balance DESC LIMIT 1; -- выбираем счет с наибольшим балансом

IF acc\_id IS NOT NULL

THEN

-- UPDATE Bank\_Accounts SET balance = balance - amount\_sum WHERE id = acc\_id;

UPDATE Credit\_Accounts SET repaid\_amount = repaid\_amount + amount\_sum WHERE id = credit\_acc\_id;

INSERT INTO Transactions (from\_account\_id, to\_account\_id, amount, category\_id, type\_id)

VALUES (acc\_id, NULL, amount\_sum,

(SELECT id FROM Transaction\_Categories WHERE category\_name=trans\_name),

(SELECT id FROM Transaction\_Types WHERE type\_name='Расходы'));

ELSE

RAISE EXCEPTION 'У данного пользователя недостаточно средств или нет открытых счетов';

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION delete\_deposit\_account()

RETURNS TRIGGER AS

$$

BEGIN

DELETE FROM Deposit\_Accounts WHERE client\_id = OLD.client\_id;

DELETE FROM Deposit\_Transactions WHERE deposit\_account\_id = OLD.id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION add\_deposit\_request()

RETURNS TRIGGER AS

$$

DECLARE

v\_min\_amount NUMERIC;

v\_max\_amount NUMERIC;

v\_manager\_id UUID;

BEGIN

SELECT min\_amount, max\_amount INTO v\_min\_amount, v\_max\_amount

FROM Deposit\_Types WHERE id = NEW.deposit\_type\_id;

IF NEW.amount < v\_min\_amount OR NEW.amount > v\_max\_amount THEN

RAISE EXCEPTION 'Запрошенная сумма не соответствует допустимому диапазону для данного типа кредита';

END IF;

SELECT id INTO v\_manager\_id FROM Managers WHERE

department\_id = (SELECT id FROM Departments WHERE city\_id = NEW.city\_id LIMIT 1);

IF v\_manager\_id IS NULL THEN

RAISE EXCEPTION 'Не найдено менеджеров в данном городе';

ELSE

NEW.manager\_id := v\_manager\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update\_deposit\_request\_status()

RETURNS TRIGGER AS

$$

DECLARE

user\_record Users%ROWTYPE;

client\_record Clients%ROWTYPE;

age INTERVAL;

BEGIN

SELECT \* INTO user\_record FROM Users WHERE id = NEW.client\_id;

SELECT \* INTO client\_record FROM Clients WHERE id = NEW.client\_id;

IF user\_record IS NULL OR client\_record IS NULL THEN

RAISE EXCEPTION 'Пользователь не найден';

END IF;

-- Проверка на NULL для first\_name и surname в Users

IF user\_record.first\_name IS NULL OR user\_record.surname IS NULL THEN

NEW.status := FALSE;

RAISE NOTICE 'Кредит не одобрен, потому что недостаточно данных о клиенте';

RETURN NEW;

END IF;

-- Проверка на NULL для date\_of\_birth в Clients

IF client\_record.date\_of\_birth IS NULL THEN

NEW.status := FALSE;

RAISE NOTICE 'Вклад не одобрен, потому что недостаточно данных о клиенте';

RETURN NEW;

END IF;

age := AGE(NOW(), client\_record.date\_of\_birth);

IF EXTRACT(YEAR FROM age) < 18 THEN

NEW.status := FALSE;

RAISE NOTICE 'Вклад не одобрен, потому что клиенту меньше 18';

ELSE

NEW.status := TRUE;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION create\_deposit\_account()

RETURNS TRIGGER AS

$$

BEGIN

IF NEW.status = TRUE THEN

INSERT INTO Deposit\_Accounts (client\_id, amount, repaid\_amount, interest\_rate, start\_date, end\_date)

VALUES (

NEW.client\_id,

(SELECT amount FROM Deposit\_Requests WHERE client\_id = NEW.client\_id),

0,

(SELECT interest\_rate FROM Deposit\_Types WHERE id = NEW.deposit\_type\_id),

NOW(),

NOW() + INTERVAL '1 month' \* (SELECT term\_months FROM Deposit\_Types WHERE id = NEW.deposit\_type\_id)

);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION create\_deposit\_transaction()

RETURNS TRIGGER AS

$$

BEGIN

INSERT INTO Deposit\_Transactions (deposit\_account\_id, amount, transaction\_type\_id, transaction\_date)

VALUES (

NEW.id,

NEW.amount,

(SELECT id FROM Deposit\_Transaction\_Types WHERE deposit\_trans\_name = 'Открытие вклада'),

NOW()

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION new\_deposit\_transaction()

RETURNS TRIGGER AS

$$

DECLARE

deposit\_acc\_id UUID;

amount\_sum NUMERIC(10, 2);

trans\_type\_id UUID;

trans\_name VARCHAR;

client UUID;

acc\_id UUID;

BEGIN

deposit\_acc\_id := NEW.deposit\_account\_id;

amount\_sum := NEW.amount;

trans\_type\_id := NEW.transaction\_type\_id;

SELECT deposit\_trans\_name INTO trans\_name FROM Deposit\_Transaction\_Types WHERE id = trans\_type\_id;

IF trans\_name = 'Пополнение вклада'

THEN

IF NOT EXISTS (SELECT 1 FROM Transaction\_Categories WHERE category\_name = 'Пополнение вклада')

THEN

INSERT INTO Transaction\_Categories (category\_type\_id, category\_name)

VALUES ((SELECT id FROM Transaction\_Types WHERE type\_name='Расходы'), trans\_name);

END IF;

SELECT client\_id INTO client FROM Deposit\_Accounts WHERE id = deposit\_acc\_id;

SELECT id INTO acc\_id FROM Bank\_Accounts

WHERE client\_id = client AND currency\_id = (SELECT id FROM Currencies WHERE currency\_code='BYN')

AND balance >= amount\_sum

ORDER BY balance DESC LIMIT 1; -- выбираем счет с наибольшим балансом

IF acc\_id IS NOT NULL

THEN

UPDATE Deposit\_Accounts SET amount = amount + amount\_sum WHERE id = deposit\_acc\_id;

INSERT INTO Transactions (from\_account\_id, to\_account\_id, amount, category\_id, type\_id)

VALUES (acc\_id, NULL, amount\_sum,

(SELECT id FROM Transaction\_Categories WHERE category\_name=trans\_name),

(SELECT id FROM Transaction\_Types WHERE type\_name='Расходы'));

ELSE

RAISE EXCEPTION 'У данного пользователя недостаточно средств или нет открытых счетов';

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER after\_user\_insert

AFTER INSERT ON All\_User\_Roles\_Info

FOR EACH ROW

EXECUTE FUNCTION distribute\_user\_info();

CREATE TRIGGER hash\_password\_trigger

BEFORE INSERT ON All\_User\_Roles\_Info

FOR EACH ROW

EXECUTE FUNCTION hash\_password();

CREATE TRIGGER before\_department\_insert

BEFORE INSERT ON Departments

FOR EACH ROW

EXECUTE PROCEDURE generate\_department\_name();

CREATE TRIGGER support\_status\_update\_online

BEFORE UPDATE ON Technical\_Supports

FOR EACH ROW

EXECUTE FUNCTION update\_last\_online();

CREATE TRIGGER support\_request\_insert

BEFORE INSERT ON Support\_Requests

FOR EACH ROW

EXECUTE FUNCTION assign\_support\_and\_status();

CREATE TRIGGER insert\_response

AFTER INSERT ON Support\_Requests

FOR EACH ROW

EXECUTE FUNCTION insert\_request\_response();

CREATE TRIGGER support\_status\_update

AFTER UPDATE ON Technical\_Supports

FOR EACH ROW

EXECUTE FUNCTION update\_request\_status();

CREATE TRIGGER support\_response\_insert

AFTER UPDATE ON Support\_Responses

FOR EACH ROW

EXECUTE FUNCTION close\_request\_on\_response();

CREATE TRIGGER set\_open\_status\_on\_insert

BEFORE INSERT ON Bank\_Accounts

FOR EACH ROW

EXECUTE FUNCTION set\_open\_status\_on\_insert();

CREATE TRIGGER create\_new\_transaction

BEFORE INSERT ON Transactions

FOR EACH ROW

EXECUTE FUNCTION execute\_transaction();

CREATE TRIGGER update\_balances

AFTER INSERT ON Transactions

FOR EACH ROW

EXECUTE FUNCTION update\_accounts\_balances();

CREATE TRIGGER credit\_request

BEFORE INSERT ON Credit\_Requests

FOR EACH ROW

EXECUTE FUNCTION add\_credit\_request();

CREATE TRIGGER update\_credit\_request\_status

BEFORE INSERT OR UPDATE ON Credit\_Requests

FOR EACH ROW

EXECUTE PROCEDURE update\_credit\_request\_status();

CREATE TRIGGER create\_credit\_account

AFTER INSERT OR UPDATE ON Credit\_Requests

FOR EACH ROW

EXECUTE PROCEDURE create\_credit\_account();

CREATE TRIGGER create\_credit\_transaction

AFTER INSERT ON Credit\_Accounts

FOR EACH ROW

EXECUTE PROCEDURE create\_credit\_transaction();

CREATE TRIGGER delete\_credit\_account

AFTER DELETE ON Credit\_Requests

FOR EACH ROW

EXECUTE PROCEDURE delete\_credit\_account();

CREATE TRIGGER new\_credit\_transaction

AFTER INSERT ON Credit\_Transactions

FOR EACH ROW

EXECUTE PROCEDURE new\_credit\_transaction();

CREATE TRIGGER delete\_deposit\_account

AFTER DELETE ON Deposit\_Requests

FOR EACH ROW

EXECUTE PROCEDURE delete\_deposit\_account();

CREATE TRIGGER deposit\_request

BEFORE INSERT ON Deposit\_Requests

FOR EACH ROW

EXECUTE FUNCTION add\_deposit\_request();

CREATE TRIGGER update\_deposit\_request\_status

BEFORE INSERT OR UPDATE ON Deposit\_Requests

FOR EACH ROW

EXECUTE PROCEDURE update\_deposit\_request\_status();

CREATE TRIGGER create\_deposit\_account

AFTER INSERT OR UPDATE ON Deposit\_Requests

FOR EACH ROW

EXECUTE PROCEDURE create\_deposit\_account();

CREATE TRIGGER create\_deposit\_transaction

AFTER INSERT ON Deposit\_Accounts

FOR EACH ROW

EXECUTE PROCEDURE create\_deposit\_transaction();

CREATE TRIGGER new\_deposit\_transaction

AFTER INSERT ON Deposit\_Transactions

FOR EACH ROW

EXECUTE PROCEDURE new\_deposit\_transaction();