

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт информационных технологий и управления в технических системах
(полное название института)

кафедра «Информационные системы»
(полное название кафедры)

Пояснительная записка

к курсовому проекту

на тему _____ Разработка базы данных для многопользовательской игры с
гибкой системой ролей и прав доступа

по дисциплине _____ **Управление данными**

Выполнил: студент III курса, группы: **ИС/6-17-2-о**

Направления подготовки (специальности) 09.03.02

Информационные системы и технологии

(код и наименование направления подготовки (специальности))

профиль (специализация) Информационные системы

Клышко Никита Александрович

(фамилия, имя, отчество студента)

Дата допуска к защите « _____ » _____ 20 19 г.

Руководитель _____

(подпись)

(инициалы, фамилия)

20 19 г.

Аннотация

В данной пояснительной записке содержится проектная документация базы данных для многопользовательской онлайн игры с возможностью регистрации личного аккаунта, разработанной в рамках курсового проекта по дисциплине «Управление данными».

Во введении обозначена тема курсовой проекта, проблема, которая должна быть решена в результате разработки, определены цели и задачи.

В аналитической части производится исследование предметной области, выделение сущностей и связей между ними.

В основной части документа дается описательная постановка задачи, также представлено проектное решение проблемы, в том числе, логическая модель, концептуальная модель, диаграмма «сущность-связь», модель на ключах.. После описания проекта, дается информация о деталях реализации, а также обоснование выбранных технологий и средств разработки.

В заключении делается вывод о результатах проделанной работы и дается оценка разработанного проекта.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ.....	6
1 АНАЛИТИЧЕСКАЯ ЧАСТЬ	8
1.1 Анализ предметной области	8
1.2 Постановка задачи	9
Выводы по разделу 1	10
2 РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ	11
2.1 Построение диаграммы «сущность-связь» в нотации П. Чена	11
2.2 Построение модели, основанной на ключах	13
2.3 Построение полной атрибутивной модели в нотации IDEF1x	14
Выводы по разделу 2	17
3 РАЗРАБОТКА ФИЗИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ	18
3.1 Выбор программной платформы для реализации БД	18
3.2 Реализация базы данных	19
3.3 Тестирование базы данных	20
3.4 Разграничение прав доступа	20
3.5 Расчет информационных параметров базы данных	21
Выводы по разделу 3	23
4 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ	24
4.1 Обоснование выбора языка программирования	24
4.2 Разработка интерфейса пользователя	24
4.3. Тестирование приложения	29
Выводы по разделу 4	31

ЗАКЛЮЧЕНИЕ	32
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	33
ПРИЛОЖЕНИЕ А.....	34
ПРИЛОЖЕНИЕ Б.....	37
ПРИЛОЖЕНИЕ В	39

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

DDD	– Domain-driven design (рус. Предметно-ориентированное проектирование)
IDE	– Integrated development environment (рус. Интегрированная среда разработки)
LAMP	– Linux, Apache, MySQL, PHP
MVC	– Model-View-Controller (рус. Модель-Представление-Контроллер)
ORM	– Object-Relational Mapping (рус. Объектно-реляционное отображение)
XAMPP	– X(cross-platform), Apache, MySQL, PHP, Perl
ПО	– программное обеспечение
ЯП	– язык программирования

ВВЕДЕНИЕ

Актуальность темы. Современные информационные системы часто строятся с применением распределенной архитектуры, позволяющей эффективно и даже автоматически горизонтально масштабировать их. Многопользовательские игры не являются исключением, в большинстве случаев технически очень сложно поддерживать подключения тысяч игроков на одном экземпляре игрового сервера. В связи с этим, в игре используется архитектура, представленная на рисунке 1.

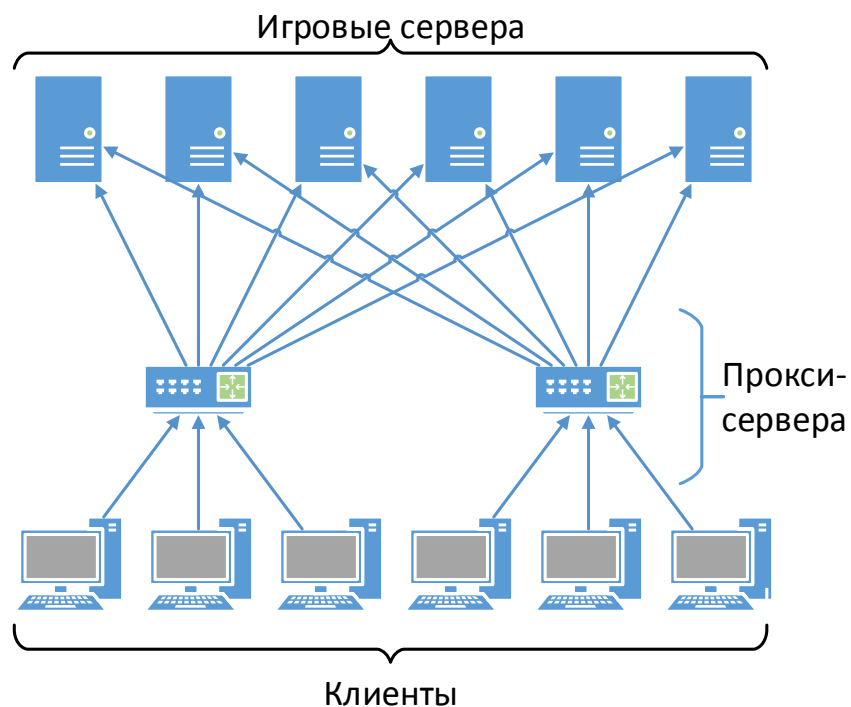


Рисунок 1 – Архитектура игровой сети

Каждый отдельный игровой сервер может иметь систему авторизации и контроля прав доступа, но в качестве средства хранения данных использует текстовые или бинарные файлы. Из этого вытекает проблема необходимости синхронизации данных между экземплярами серверов. Одним из вариантов

решения обозначенной проблемы является создание единой базы данных в игровой сети.

Цель и задачи работы. Целью данного курсового проекта является проектирование и разработка физической модели базы данных для многопользовательской онлайн игры.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- 1) проанализировать предметную область и выделить в ней сущности и связи между ними;
- 2) построить логическую модель данных в нотации Чена;
- 3) построить КВ-модель;
- 4) построить модель в формате IDEF1x;
- 5) разработать физическую базу данных;
- 6) составить запросы к физической реализации БД;
- 7) создать приложение для БД.

Практическое значение работы. Единая база данных в сети игровых серверов позволит пользователям регистрироваться в сети глобально и пользоваться всеми доступными им функциями.

Так как права доступа будут храниться в БД, то администратор получит возможность редактирования групп прав доступа и добавления новых прав в рамках всей сети, без необходимости обновлять конфигурации каждого отдельного сервера.

Очень важным также является требование безопасности личных аккаунтов. Пользователи смогут запрашивать сброс пароля с помощью привязанного адреса электронной почты или изменять привязанную электронную почту с подтверждением отвязки от старого адреса почты.

Несмотря на то, что база данных разрабатывается для конкретной игры, в процессе работы необходимо придерживаться общепринятых требований к построению подобных систем. Это позволит использовать данную БД и в других проектах, не прибегая к масштабным изменениям.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ предметной области

Серверная система многопользовательской онлайн игры это, очевидно, система достаточно большая для того, чтобы декомпозировать процесс ее разработки. Так как основной проект разрабатывается в рамках методологии Domain Driven Design, то и система хранения данных по возможности разрабатывается согласно ей. Таким образом, в рамках данного курсового проекта ставится задача разработки лишь подсистемы аккаунтов и прав доступа, формирующей свой ограниченный контекст.

Предметная область разрабатываемой подсистемы имеет 3 сильных сущности: «User», «Role», «Permission». Очевидно, что пользователь — это независимая сущность, он может иметь отдельные права доступа, а также иметь назначенные роли. Роль также является сильной сущностью и может иметь множество прав доступа. Можно заметить, что как роли, так и пользователи могут иметь набор прав, поэтому право доступа само по себе выделено в отдельную независимую сущность. Таким образом, каждая сущность связана с каждой отношением многие-ко-многим.

Такая структура связей сущностей позволяет реализовать ограничение, определяющее эквивалентность права доступа роли и отдельно взятого пользователя. Эта особенность упрощает проверку прав доступа в связанных с базой данных приложениях.

Полученная модель (рис. 1.1) является сложной сетевой, поэтому она была преобразована в простую сетевую модель, изображенную на рисунке 1.2.

При дальнейшем проектировании добавлены другие зависимые сущности, выделенные в связи с наличием необязательных и многозначных атрибутов исходных сущностей. Добавленные сущности рассмотрены в разделе разработки более подробной логической модели базы данных.

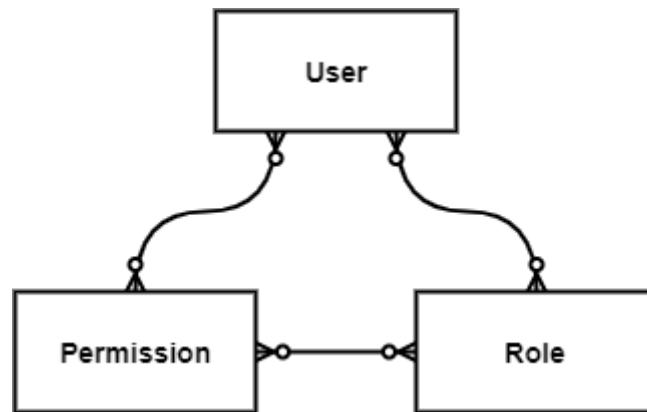


Рисунок 1.1 – Сложная сетевая модель

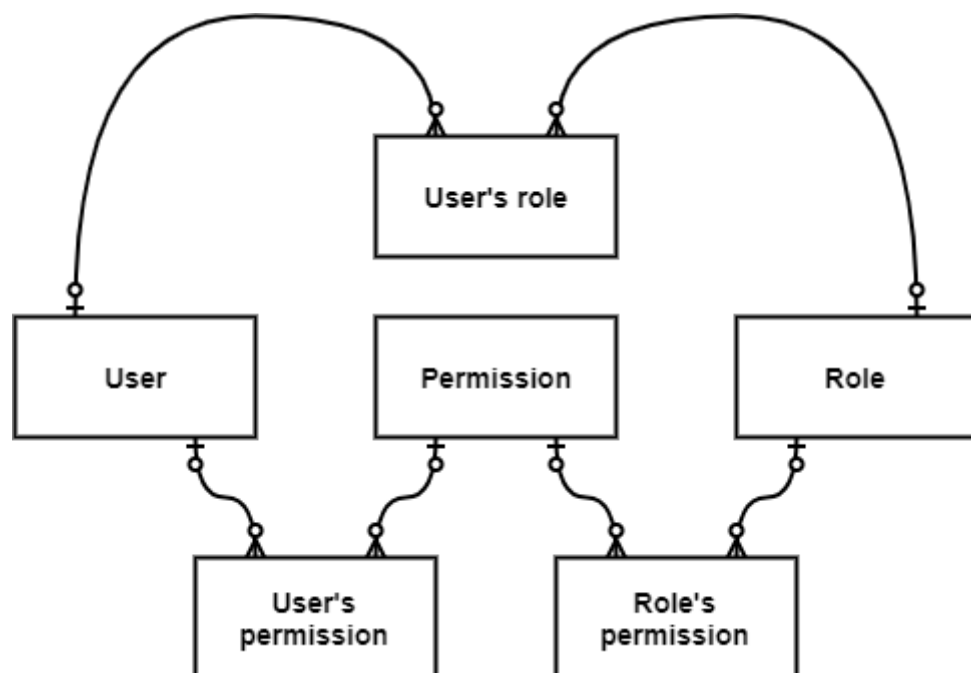


Рисунок 1.2 – Простая сетевая модель

1.2 Постановка задачи

В соответствии с анализом предметной области проекта, задачу разработки можно привести к следующей формулировке.

Разработать базу данных и задокументировать её модель. База данных должна позволять хранить информацию, необходимую для следующих функциональных возможностей:

1. Для всех пользователей:

- регистрация,
- авторизация,
- просмотр профиля:
 - а) просмотр назначенных ролей,
 - б) просмотр выданных прав доступа;
- совершение пожертвований,
- восстановление пароля,
- изменение адреса электронной почты.

2. Для администраторов:

- создание и редактирование прав доступа,
- создание и редактирование ролей в системе,
- редактирование профилей пользователей:
 - а) выдача прав доступа,
 - б) назначение ролей.

Для демонстрации работы системы ролей и прав доступа разработать веб-приложение со следующими возможностями: регистрация, авторизация, просмотр профиля, панель управления администратора.

Выводы по разделу 1

Основными сущностями предметной области, кроме самого пользователя являются права доступа (сущность «Permission») и пользовательские роли (сущность «Role»). Сложность заключается в том, что все эти сущности связаны между собой связями многие-ко-многим.

Поэтому при разработке демонстрационного приложения для базы данных можно ограничиться функциями создания, выборки и обновления этих сущностей, а также создания и удаления связей между ними с контролем целостности, введенных пользователем данных.

2 РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ

2.1 Построение диаграммы «сущность-связь» в нотации П. Чена

Существует множество способов построить одну и ту же логическую модель в нотации П. Чена за счет использования простейших блоков таких диаграмм. Для достижения компактности диаграммы в ней были использованы обозначения многозначных, составных и производных атрибутов.

Многозначные атрибуты в процессе нормализации в большинстве случаев выделяются в отдельную сущность, связанную с родительской сущностью как многие-к-одному, таким образом, на диаграмме нет необходимости обозначения отдельной сущности и ее связи с единственной родительской сущностью.

Составные атрибуты в процессе построения полной атрибутивной модели были преобразованы в несколько атомарных атрибутов для избежания аномалий обновления, когда при обновлении одной части составного поля также будет обновлена другая часть, ранее загруженная в приложение. Значение второй части к этому моменту могло уже быть изменено, поэтому обновление всего составного поля будет некорректным.

Производные атрибуты могут быть реализованы как на уровне базы данных (с помощью вычисляемых (computed) полей отношений или триггеров), так и на уровне прикладной программы. В данном проекте, производные атрибуты были реализованы на уровне приложения с помощью агрегатных функций.

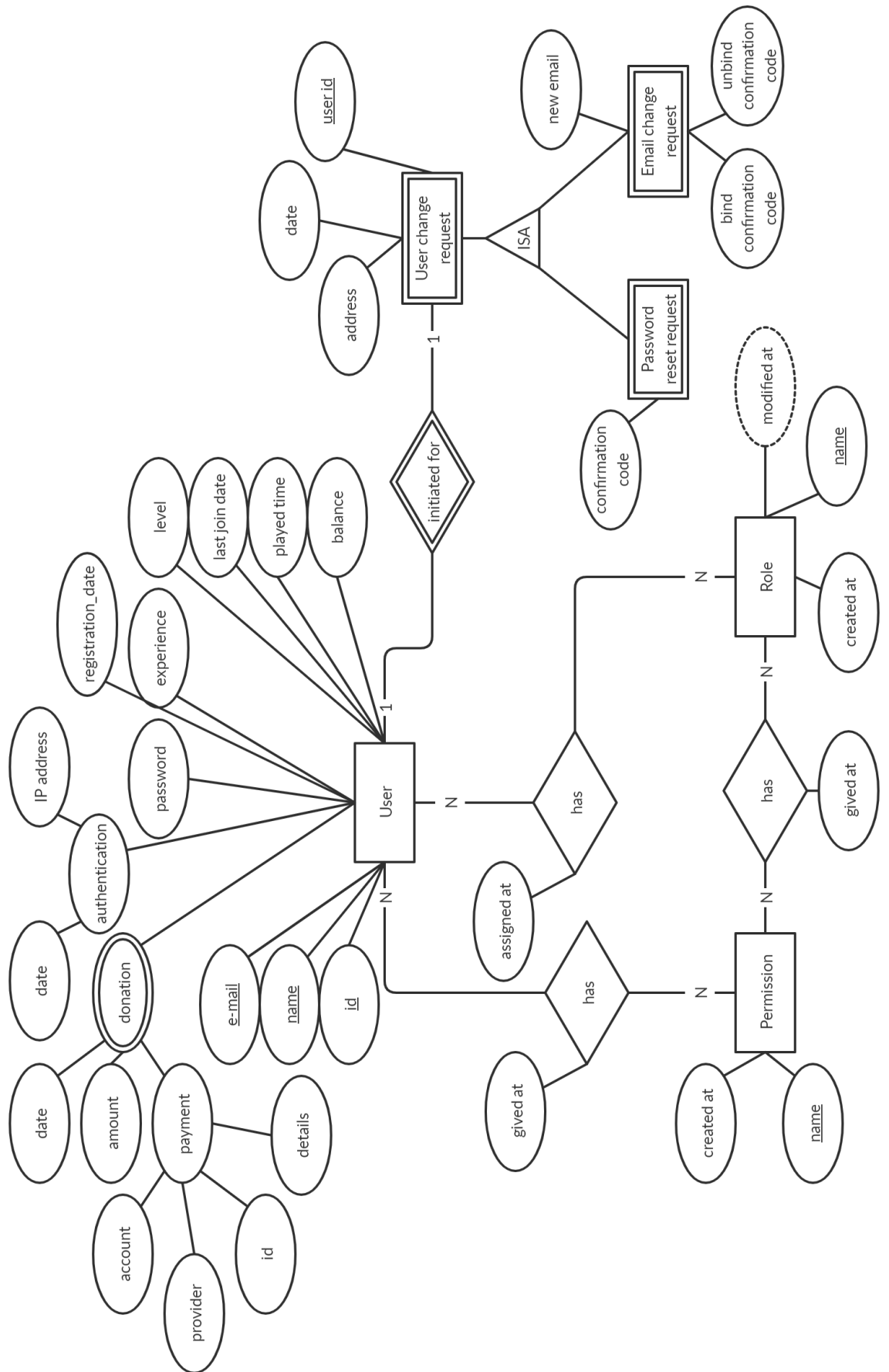


Рисунок 2.1 – Диаграмма «сущность-связь» в нотации Чена

2.2 Построение модели, основанной на ключах

Переход от модели «сущность-связь» к модели, основанной на ключах, позволяет более подробно описать взаимосвязи сущностей посредством первичных и внешних ключей. Данная модель обеспечивает правильность определения первичных ключей в дополнительных таблицах связей многие-ко-многим, потому что первичные ключи связываемых сущностей спускаются по иерархии и образуют составные ключи связующих таблиц.

Построение модели, основанной на ключах, сводится к описанию совокупности уже определенных сущностей с указанием только ключевых полей в виде древовидной структуры. Отсутствие в модели неключевых полей позволяет сосредоточиться на взаимосвязях между сущностями и уменьшает вероятность допустить ошибки при проектировании ограничений целостности физической схемы базы данных.

В случае модели данного курсового проекта связи многие-ко-многим были преобразованы в 3 таблицы с составными первичными ключами, включающими первичные ключи связываемых сущностей в качестве внешних ключей. Это обеспечивает целостность связей в БД.

Как уже было отмечено ранее, многозначные атрибуты обычно выделяются в зависимые сущности, поэтому атрибут «User.donation» выделился в зависимую сущность «Donation».

Полученная KB-модель изображена на рисунке 2.2. По задаче проекта, построение полной атрибутивной модели должно проводиться в соответствии со стандартом IDEF1x, который также описывает нотацию диаграммы для модели, основанной на ключах. Поэтому целесообразным было построение KB-модели в соответствии с этим стандартом.

Стандарт IDEF1x предписывает изображать сильные сущности в фигурах с прямыми углами, а слабые – в фигурах с закругленными углами. В рамках данного стандарта есть возможность обозначить наследование сущностей.

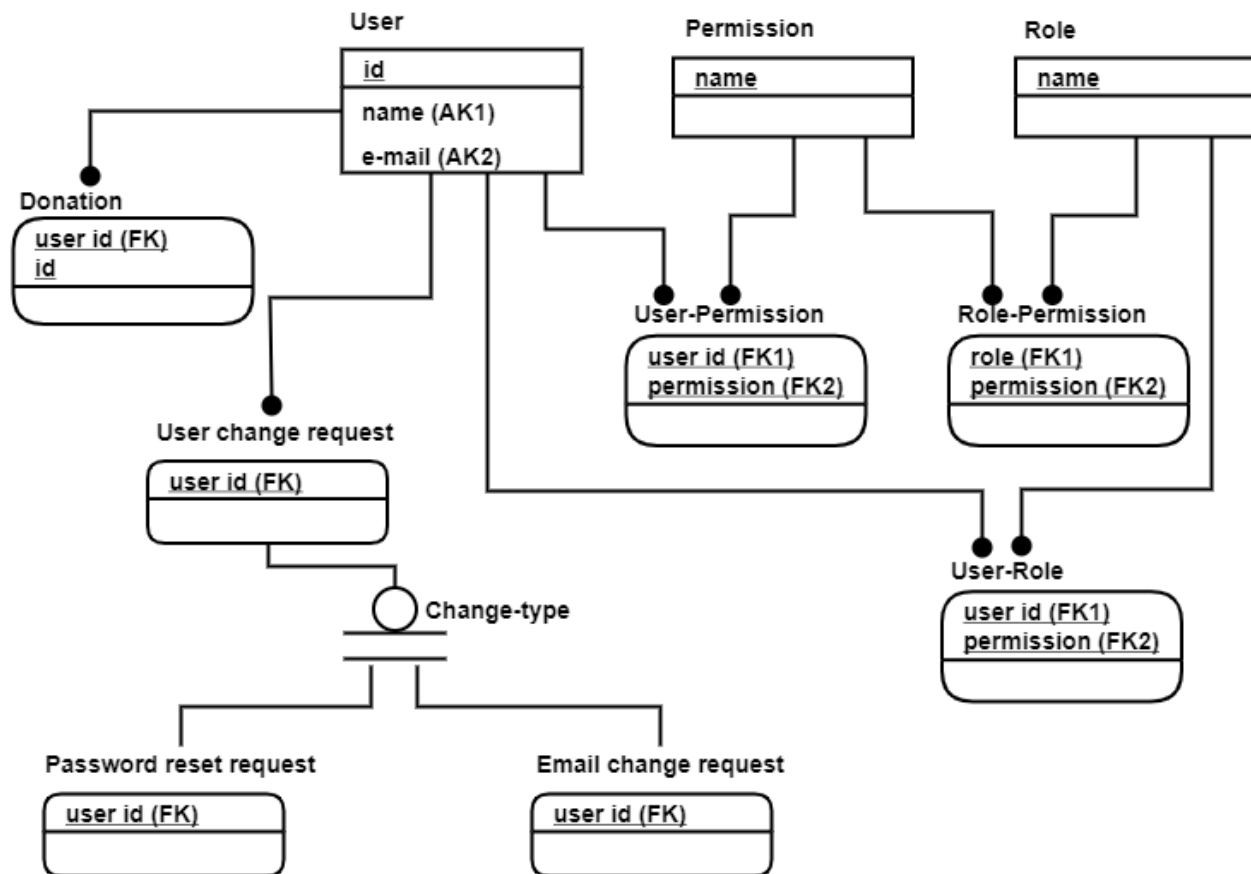


Рисунок 2.2 – Модель, основанная на ключах

Пример наследования можно видеть на диаграмме модели (рис. 2.2): сущности запросов на изменение отдельных, чувствительных к безопасности, атрибутов аккаунта пользователя могут иметь различные параметры, но также имеют множество общих атрибутов, характеризующих все виды запросов.

2.3 Построение полной атрибутивной модели в нотации IDEF1x

Полная атрибутивная модель – очень подробное представление структуры базы данных, близкое к физической модели. В ней должны быть описаны все атрибуты сущностей с указанием типов данных, в том числе и неключевые, в отличие от модели, основанной на ключах.

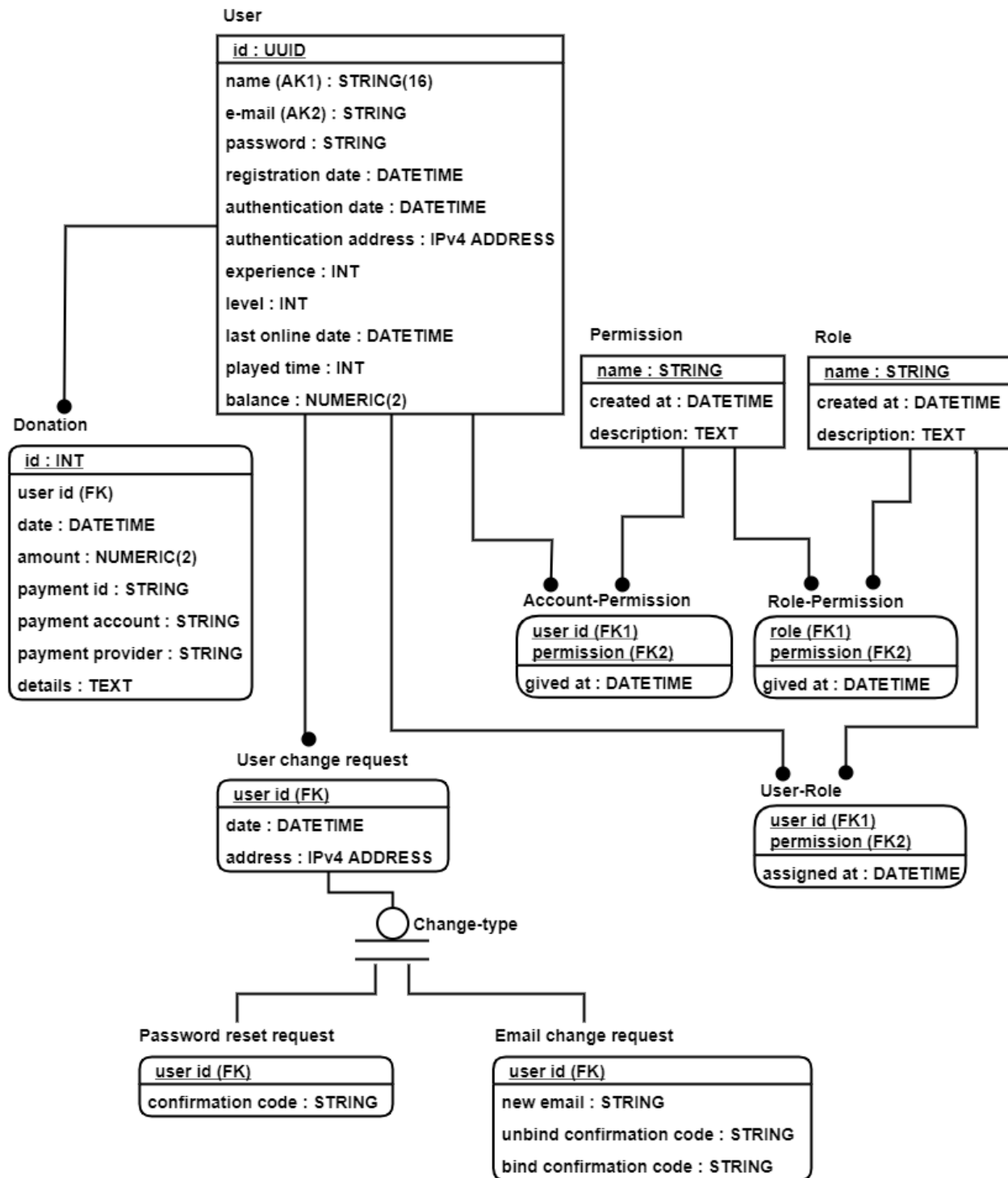


Рисунок 2.3 – Полная атрибутивная модель в нотации IDEF1x

На уровне данной модели может быть проведена нормализация в НФ высокого порядка. В результате преобразований моделей от концептуальной к полной атрибутивной схема была нормализована, как минимум, к нормальной форме Бойса-Кодда. Докажем нахождение полученной реляционной модели в НФБК и более низких нормальных формах, начиная с 1НФ.

Отношение «User» находится в первой нормальной форме, так как каждый кортеж содержит только одно значение для каждого из атрибутов. В процессе приведения этого отношения к первой нормальной форме, схема была составлена так, что каждое значение многозначного атрибута «donation» записывалось в отдельном кортеже отношения «User». Эта схема имела существенные недостатки:

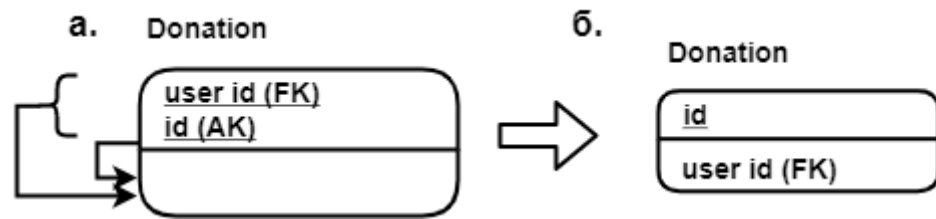
- дублирование информации об аккаунте пользователя для каждого нового значения атрибута «donation»,
- невозможность определения первичного ключа для отношения, так как атрибуты, являющиеся потенциальными ключами также дублировались в множествах кортежей одного пользователя.

Поэтому при дальнейшей нормализации многозначный атрибут «donation» был выделен в отдельное отношение, которое имеет связь многие-к-одному с «User».

Из ранее рассмотренной диаграммы «сущность-связь» (рис. 2.1) видно, что остальные отношения не имеют многозначных атрибутов, и поэтому изначально находятся в первой нормальной форме. Таким образом, можно сказать, что реляционная модель находится в 1НФ.

Приведение ко второй нормальной форме потребовалось только для отношения «Donation». Так как оно имеет связь многие-к-одному с отношением «User», то в состав первичного ключа, содержащего только «user_id», был добавлен уникальный атрибут «id». Это привело к тому, что все функциональные зависимости неключевых атрибутов от первичного ключа стало возможным привести к функциональным зависимостям, в левой части которых оставался только атрибут «id». Преобразование схемы отношения изображено на рисунке 2.4.

Таким образом, атрибут «user_id» был исключен из состава первичного ключа и можно утверждать, что реляционная модель находится во 2НФ.



а – отношение до нормализации; б – отношение во 2НФ

Рисунок 2.4 – Нормализация отношения «Donation»

Модель также находится в третьей нормальной форме и нормальной форме Бойса-Кодда, так как отношения не содержат транзитивных функциональных зависимостей и все существующие функциональные зависимости имеют в своей правой части только соответствующие потенциальные ключи.

Выводы по разделу 2

Разработка логической модели базы данных проводилась в несколько этапов, на каждом из которых требовалось построение диаграмм определенного типа. Каждая последующая диаграмма дополняет или реструктурирует информацию, содержащуюся в диаграмме, созданной на предыдущем этапе.

Таким образом, процесс преобразования логической модели позволил уточнить ее. Полученная на последнем этапе логическая модель была рассмотрена как реляционная и на ней была проведена нормализация. Доказано, что нормализованная модель находится в НФБК.

3 РАЗРАБОТКА ФИЗИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ

3.1 Выбор программной платформы для реализации БД

В современном мире сложно представить крупную информационную систему, работу которой может поддерживать всего один сервер БД, то есть зачастую требуется горизонтальная масштабируемость базы данных. К тому же, в постановке задачи упоминается, что БД разрабатывается как раз для распределенной системы. Хотя это и не означает изначальной потребность в масштабировании базы данных, СУБД должна отвечать современным требованиям масштабируемости. Всем известна популярная открытая система управления базами данных *MySQL*, но не говоря о том, что сегодня все больше применяются нереляционные базы данных, в секторе реляционных СУБД стремительно набирает популярность *PostgreSQL*, которая также распространяется с открытым исходным кодом. *PostgreSQL* лучше *MySQL* по многим параметрам, в том числе, имеет больше возможностей для масштабирования, поэтому в качестве СУБД выбрана *PostgreSQL 12*.

Для внесения изменений в БД в процессе разработки применяется подход «миграций БД». Этот подход позволяет начинать эксплуатацию БД еще до полного завершения ее разработки, что соответствует принципам *Agile* разработки. Также, это позволяет использовать систему контроля версий схемы БД.

В качестве движка миграций использовался *Liquibase Community*, а для контроля версий файлов журналов изменений применялся *Git*. В процессе разработки использовалась IDE *DataGrip* от Jetbrains.

Для реализации автоматических тестов использовался модуль *unittest* языка *Python3*, а также дополнительный модуль *testing.postgresql*, обеспечивающий запуск встроенного тестового сервера *PostgreSQL* с возможностью реинициализации схемы при каждом новом запуске тестов. Это позволяет исключить зависимость результатов тестирования от порядка запуска тестов.

3.2 Реализация базы данных

В процессе преобразования полной атрибутивной модели в физическую модель, реализуемую в конкретной системе управления базами данных, общепринятые типы данных были сопоставлены соответствующим типам выбранной СУБД. Большинство строк, если это не противоречило предметной области, было ограничено длиной в 255 символов для оптимизации объема хранения. Для денежных единиц был использован тип данных с фиксированной точкой для реализации точных арифметических операций без округления.

Полный листинг запросов для инициализации базы данных представлен в приложении А.

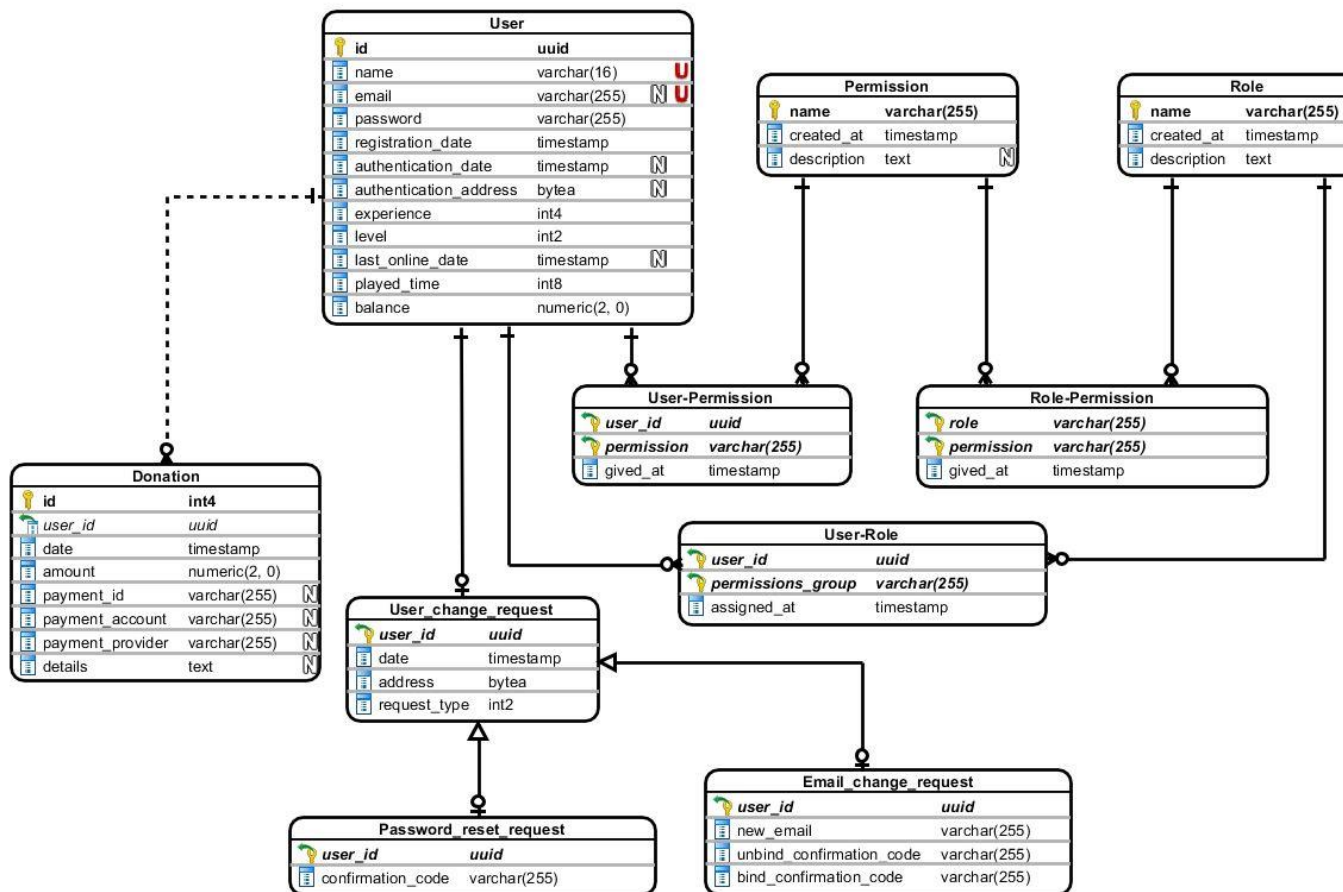


Рисунок 3.1 – Физическая модель базы данных

3.3 Тестирование базы данных

Под тестирование базы данных обычно понимают проверку её схемы, ограничений целостности, триггеров и т.д. Для реализации физической модели базы данных триггеры не использовались, а автоматическое тестирование самой схемы базы нецелесообразно, так как трудозатраты разработки и сложность тестов для схемы сопоставимы с запросами на создание этой схемы. Поэтому тестирование схемы БД производилось после интеграции с приложением, в процессе ручного тестирования самого приложения.

Ограничения целостности являются очень важной составляющей базы данных, поэтому для них были разработаны автоматические тесты. Исходный код скрипта тестирования приведен в приложении Б.

```
vagrant@homestead:~/code/database/src$ ./run_tests.sh
init_file:/home/vagrant/code/database/src/main/changelog/db.changelog-1.sql
....
-----
Ran 4 tests in 0.616s
OK
```

Рисунок 3.2 – Результаты тестирования

3.4 Разграничение прав доступа

По заданию проектируемая система должна обеспечивать управление ролями и правами доступа и возможность их проверки при использовании клиентами функций системы. То есть структура ролей и прав доступа не только очень сложная, но и динамически изменяющаяся. Реализация и поддержка подобных ограничений доступа средствами СУБД не представляются возможными, поэтому разграничение прав доступа целесообразно было реализовать на уровне приложения. Данный подход является достаточно распространен-

ным в современных информационных системах. Он обеспечивает достаточную гибкость и безопасность системы. Таким образом, серверная часть приложения будет иметь полные права доступа к своей базе данных в рамках СУБД.

3.5 Расчет информационных параметров базы данных

Длина одной записи таблицы определяется как сумма длин полей этой записи (формула (3.1)). Некоторые атрибуты имеют типы переменной длины, следовательно, соответствующие поля могут занимать разное количество памяти в зависимости от данных в них. Для таких атрибутов будем использовать, где возможно, оценку средней длины (табл. 3.1) в соответствии с предметной областью. Некоторые оценки определены эмпирическим путем.

Таблица 3.1 – Оценки средней длины значений атрибутов

Отношение	Атрибут	Оценка средней длины
User	email	22
User	password	60
Donation	payment_id	32
Donation	payment_account	32
Donation	payment_provider	16
Donation	details	128
Password_reset_request	confirmation_code	12
Password_change_request	new_email	22
Password_change_request	unbind_confirmation_code	12
Password_change_request	bind_confirmation_code	12
Permission	name	36
Permission	description	64
Role	name	16
Role	description	64

$$L_j = \sum_{i=1}^{M_j} l_{ij} \quad [\text{байт}], \quad (3.1)$$

где M_j – число полей в записях;

l_{ij} – длина поля [байт].

Вычислим:

Для атрибутов переменной длины типа VARCHAR(x_{\max}) длина поля рассчитывается по формуле (3.2).

$$l_{ij} = \begin{cases} 1 + 2 \cdot x, & x_{\max} \leq 255 \\ 2 + 2 \cdot x, & x_{\max} > 255 \end{cases} \quad [\text{байт}] \quad (3.2)$$

где x – фактическое количество символов.

Для атрибутов типа TEXT длина поля рассчитывается по формуле (3.3).

$$l = 2 + 2 \cdot x \quad [\text{байт}] \quad (3.3)$$

где x – фактическое количество символов в поле.

Рассчитаем для каждой таблицы средний объем строки:

$$L_{User} = 16 + (1 + 2 \cdot 16) + (1 + 2 \cdot 22) + (1 + 2 \cdot 60) + 8 + 7 + 4 + 2 + 8 + 8 + 8 = 260 \text{ байт.}$$

$$L_{Donation} = 4 + 16 + 8 + 8 + (1 + 2 \cdot 32) + (1 + 2 \cdot 32) + (1 + 2 \cdot 16) + (2 + 2 \cdot 128) = 457 \text{ байт.}$$

$$L_{User_change_request} = 16 + 8 + 7 + 2 = 33 \text{ байта.}$$

$$L_{Password_reset_request} = 16 + 1 + 2 \cdot 12 = 41 \text{ байт.}$$

$$L_{Email_change_request} = 16 + (1 + 2 \cdot 22) + (1 + 2 \cdot 12) + (1 + 2 \cdot 12) = 111 \text{ байт.}$$

$$L_{Permission} = (1 + 2 \cdot 36) + 8 + (2 + 2 \cdot 64) = 211 \text{ байт.}$$

$$L_{Role} = (1 + 2 \cdot 16) + 8 + (2 + 2 \cdot 64) = 179 \text{ байт.}$$

$$L_{User-Permission} = 16 + (1 + 2 \cdot 36) + 8 = 97 \text{ байт.}$$

$$L_{User-Role} = 16 + (1 + 2 \cdot 16) + 8 = 57 \text{ байт.}$$

$$L_{Role-Permission} = (1 + 2 \cdot 16) + (1 + 2 \cdot 36) + 8 = 114 \text{ байт.}$$

Объем памяти, необходимый для размещения информационного фонда без учёта системных данных и указателей составит:

$$I = \sum_{j=1}^N L_j K_j \quad [\text{байт}], \quad (3.4)$$

где N – число типов записей в информационном фонде;

K_j – количество записей j -го типа.

Предположим, что база данных содержит:

50 000 пользователей, 1000 совершенных пожертвований, 10 ролей, 1000 прав доступа, 1000 связей роль-право доступа, 100 связей пользователь-право

доступа, 1000 связей пользователь-роль. Положим долю пользователей, запросивших изменение данных аккаунта равную 1%.

$$I = 260 * 50000 + 1000 * 457 + 33 * 50000 * 0,1 + 41 * 50000 * 0,05 + \\ + 111 * 50000 * 0,1 + 211 * 1000 + 179 * 10 + 97 * 10 + 57 * 1000 + 114 * 1000 \approx 14 \text{ Мбайт.}$$

Положим приращение информационного фонда $\Delta I = 0,1 \cdot I = 1,4$ Мбайта.

Время заполнения информационного фонда

$$I_{\text{зап}} = \frac{(V_{\text{бд}} - I)}{\Delta I} \quad [\text{с}] \quad (3.4)$$

$$I_{\text{зап}} = \frac{500 - 0,014}{0,0014} \approx 100 \text{ лет}$$

Выводы по разделу 3

Выбор системы управления базами данных для физической реализации спроектированной БД осуществлялся с учетом особенностей системы и предоставляемых функциональных возможностей различными СУБД. В качестве системы управления БД выбрана *PostgreSQL*, которая отвечает главному требованию масштабируемости.

При написании запросов на создание схемы БД использовался распространенный подход «миграций БД», а контроль версий журналов изменений осуществлялся с помощью системы *Git*. Для тестирования ограничений целостности разработаны автоматические модульные тесты на языке *Python*.

4 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

4.1 Обоснование выбора языка программирования

Для разработки приложения к базе данных был выбран MVC-фреймворк *Laravel*. Кроме того, что этот фреймворк является очень мощным и популярным инструментом с подробной документацией, он написан на языке программирования *PHP*. Язык *PHP* очень часто используется при разработке веб-приложение и входит в поставки веб-серверов в комплекте с базами данных (LAMP, XAMPP и т.п.).

Еще одной важной особенностью *Laravel* является то, что в нем используется Object-Relational Mapping (ORM) библиотека *Eloquent*. Эта библиотека позволяет избавиться от рутинной работы написания простых запросов к БД на выборку, добавление, обновление и удаление данных. Тем не менее, сложные запросы требуют подробного анализа и оптимизации, поэтому разрабатываются программистом.

4.2 Разработка интерфейса пользователя

В комплекте поставки фреймворка *Laravel* идет библиотека *Bootstrap 3*. Пользовательский интерфейс был разработан с ее применением, так как она позволяет создавать современные адаптивные веб-интерфейсы.

В соответствии с задачами проекта, в панели администратора для каждой сущности была создана страница для редактирования атрибутов этой сущности, а также ее связей с другими объектами.

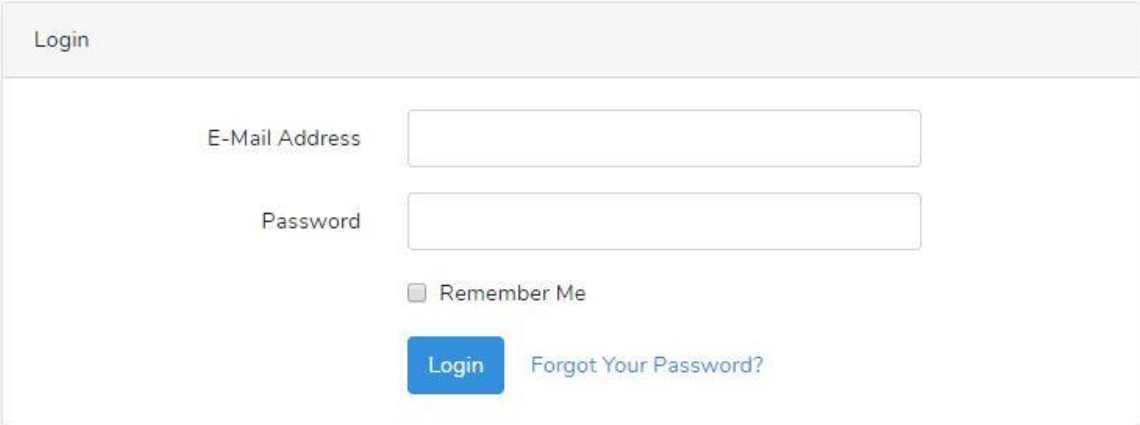
Для регистрации и авторизации пользователей были разработаны единые страницы шлюза авторизации. Чтобы пользовательский интерфейс был однородным, администраторы также авторизуются через этот общий шлюз, а

доступ к панели администратора обеспечивается путем проверки прав доступа пользователя после авторизации.

В панели управления, для операций манипуляции данными была реализована валидация корректности введенных значений на стороне сервера.

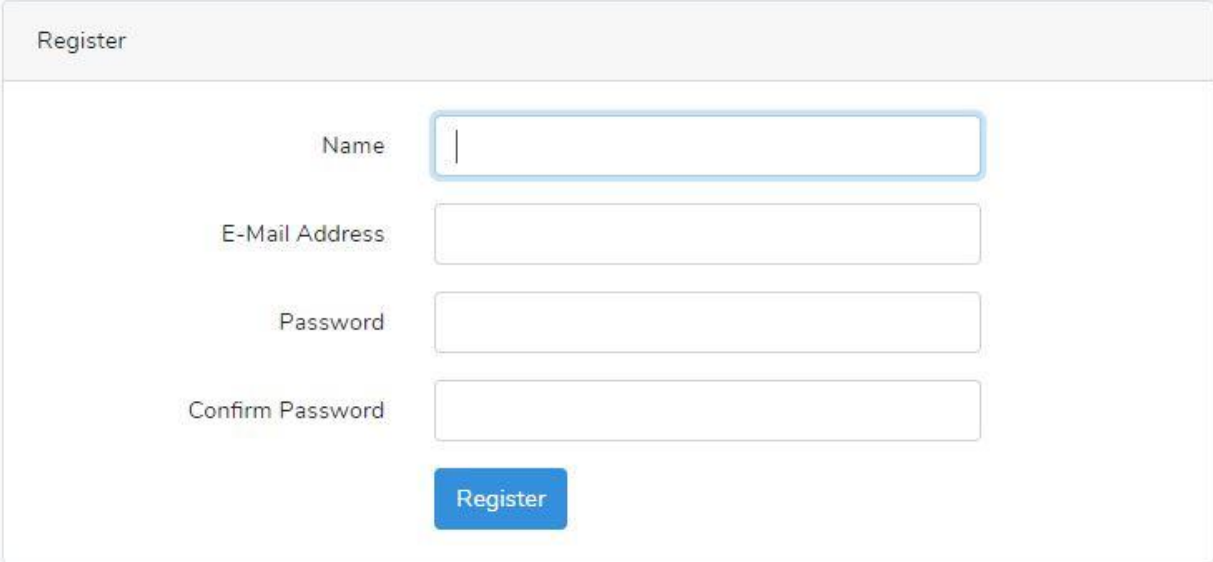
Далее представлены изображения всех разработанных страниц.

На рисунках 4.1 и 4.2 изображены страницы авторизации и регистрации. Если при регистрации пользователь введет недопустимое или уже использованное имя пользователя, то система выведет ошибку (рис. 4.3).



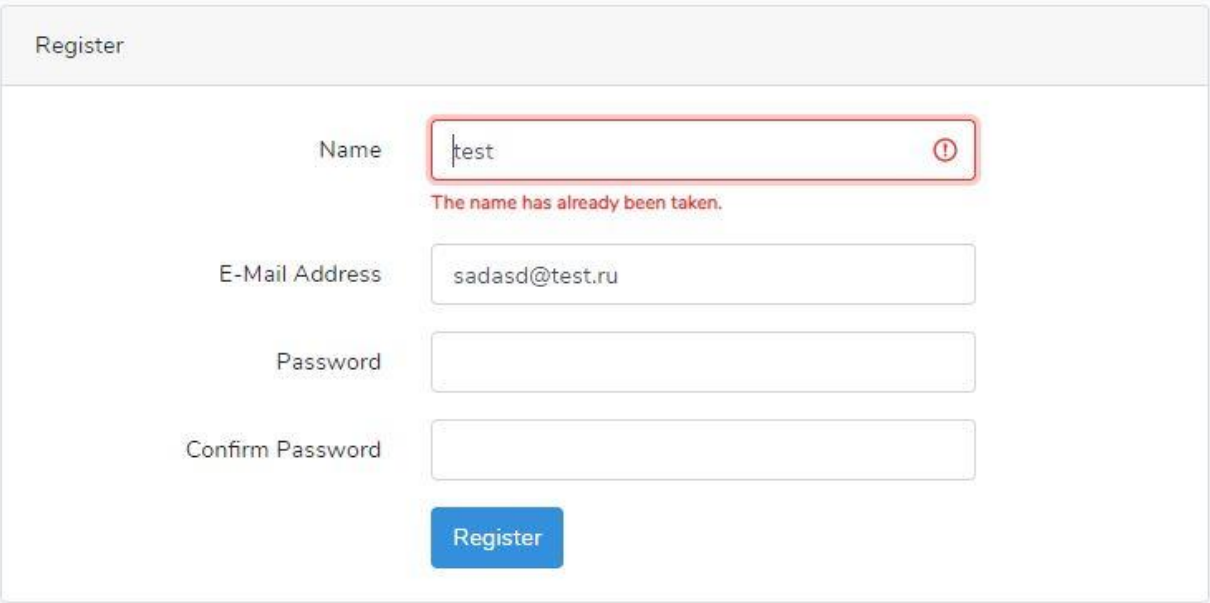
The image shows a web form titled "Login". It contains two input fields: "E-Mail Address" and "Password". Below the password field is a checkbox labeled "Remember Me". At the bottom, there is a blue "Login" button and a link that says "Forgot Your Password?".

Рисунок 4.1 – Страница авторизации



The image shows a web form titled "Register". It contains four input fields: "Name", "E-Mail Address", "Password", and "Confirm Password". At the bottom, there is a blue "Register" button.

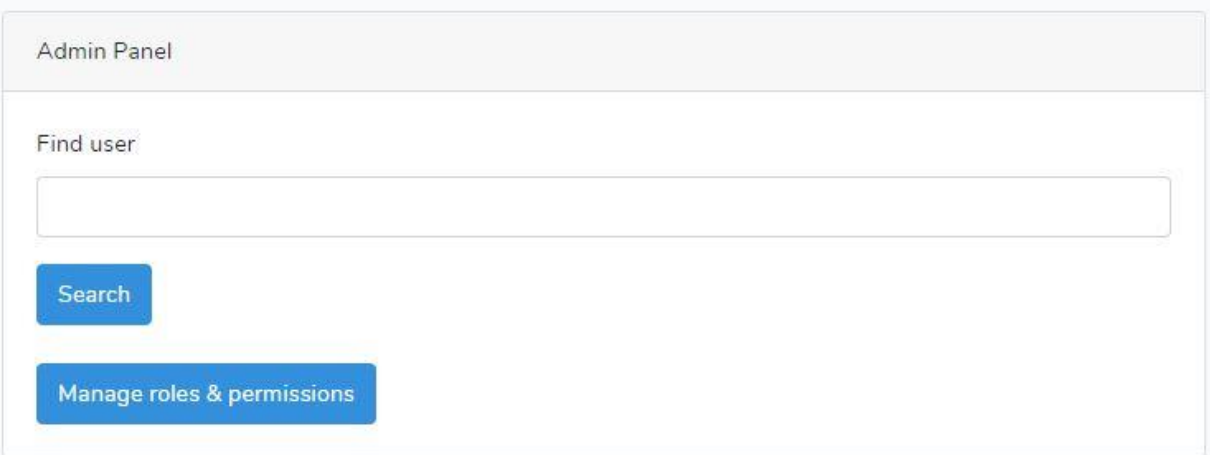
Рисунок 4.2 – Страница регистрации



The image shows a registration form titled "Register". It contains four input fields: "Name", "E-Mail Address", "Password", and "Confirm Password". The "Name" field contains the text "test" and is highlighted with a red border and a red information icon. Below it, a red error message reads "The name has already been taken." The "E-Mail Address" field contains "sadasd@test.ru". The "Password" and "Confirm Password" fields are empty. A blue "Register" button is located at the bottom of the form.

Рисунок 4.3 – Вывод ошибки при регистрации

Главная страница панели администратора изображена на рисунке 4.4. С этой страницы возможен переход на страницу редактирования ролей и прав доступа пользователя или на страницу конфигурирования ролей и прав доступа всей системы.



The image shows the "Admin Panel" interface. It features a "Find user" label above a large text input field. Below the input field is a blue "Search" button. At the bottom of the panel is a blue button labeled "Manage roles & permissions".

Рисунок 4.4 – Главная страница панели администратора

Для перехода на страницу редактирования пользователя (рис. 4.5) необходимо ввести имя пользователя в поле поиска и кликнуть по кнопке «Search».

User Test

Role	Action
Admin	<button>Remove</button>

⌵ Assign

Permission	Action
<input type="text" value="Choose permission..."/> ⌵	<button>Give</button>

Choose permission...

permission.test2

permission.test11233

permission.test1












admin.web.enter

Рисунок 4.5 – Страница редактирования пользователя «Test»

Переход на страницу конфигурирования ролей и прав доступа системы происходит при клике на кнопку «Manage roles & permissions» на главной странице панели администратора (рис 4.4). Вид этой страницы изображен на рисунке 4.6.

Доступные в системе роли и возможные права доступа изображены в виде отдельных таблиц. В конце каждой записи в таблицах расположены кнопки для редактирования и удаления сущностей.

Permissions & roles management

Role	Created at	Changed at	Description	Action
Admin	2019-12-17 15:31:39	2019-12-17 15:32:24	Admin of game network	 
TestRole	2019-12-17 08:37:08	2019-12-19 06:24:19	Special testing role.	 
TestRole3	2019-12-19 07:16:53		sadasa	 
NewRole	2019-12-19 10:14:36	2019-12-19 10:14:45	New role desc.	 
TestRole2	2019-12-19 07:09:31	2019-12-20 09:10:55	Special testing role.wd wsdadsad	 
				










Permission	Created at	Description	Action
permission.test2	2019-12-17 13:29:43	Desc	 
permission.test11233	2019-12-19 07:18:53	knfdjsbfa njwn	 
permission.test1	2019-12-17 08:31:39	Changed description.	 
admin.web.enter	2019-12-17 15:28:18	Changed description for admin permission/	 
			

Рисунок 4.6 – Страница конфигурирования ролей и прав доступа системы

Вид страницы редактирования роли изображен на рисунке 4.7.

Role Admin

Description:
Admin of game network

Permission	Action
admin.web.enter	<button>Remove</button>

Choose permission...

- Choose permission...
- permission.test2
- permission.test11233
- permission.test1
- admin.web.enter

Give Edit

Рисунок 4.7 – Страница редактирования роли «Admin»

На рисунках продемонстрированы основные страницы приложения. Для некоторых сущностей не было необходимости в создании отдельной страницы, поэтому редактирование производится во всплывающем диалоге.

4.3. Тестирование приложения

Тестирование приложения производилось ручным способом. Промежуточные состояния тестируемого экземпляра приложения изображены на рисунках 4.3 – 4.7.

Ручное тестирование могло не выявить скрытых проблем, не видимых через пользовательский интерфейс, поэтому было проверено содержимое таблиц в базе данных.

	user_id	permission	gived_at
1	f0e6baca-f5d3-4b86-b1f2-7049e71f9ccc	permission.test2	2019-12-19 10:14:05.000000

Рисунок 4.8 – Содержимое таблицы «user-permission»

	role	permission	gived_at
1	Admin	admin.web.enter	2019-12-17 15:32:24.000000
2	TestRole	permission.test1	2019-12-19 06:24:19.000000
3	NewRole	permission.test1	2019-12-19 10:14:41.000000
4	NewRole	permission.test11233	2019-12-19 10:14:45.000000
5	TestRole2	permission.test2	2019-12-20 09:10:55.000000

Рисунок 4.9 – Содержимое таблицы «role-permission»

	user_id	role	assigned_at
1	f0e6baca-f5d3-4b86-b1f2-7049e71f9ccc	Admin	2019-12-18 05:45:59.000000
2	88700379-f1db-4c36-8ff6-a4ccdc790965	Admin	2019-12-18 09:28:32.000000
3	ebbac4ac-254b-476a-ae7f-53a5e8d5851b	TestRole	2019-12-19 03:20:41.000000

Рисунок 4.10 – Содержимое таблицы «user-role»

	name	created_at	description
1	permission.test2	2019-12-17 13:29:43.000000	Desc
2	permission.test11233	2019-12-19 07:18:53.000000	knfdjsbfa njwn
3	permission.test1	2019-12-17 08:31:39.000000	Changed description.
4	admin.web.enter	2019-12-17 15:28:18.000000	Changed description for admin permission/

Рисунок 4.11 – Содержимое таблицы «permission»

Для поиска неиспользуемых прав доступа был составлен и выполнен сложный запрос. Текст запроса и результат его выполнения изображены на рисунке 4.12.

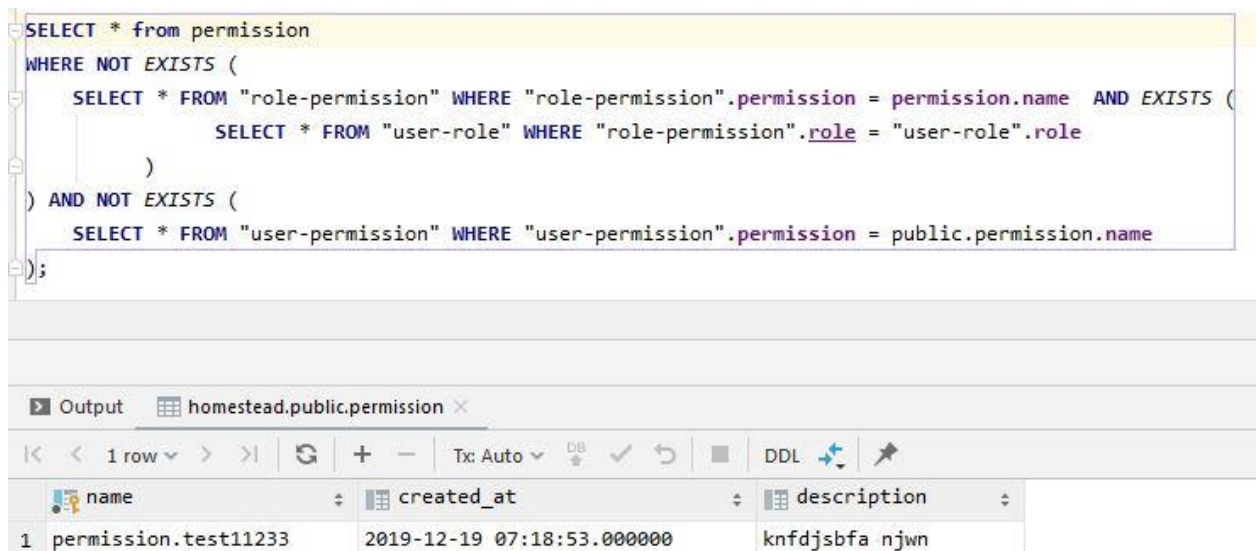


Рисунок 4.12 – Результат выполнения запроса на выборку неиспользуемых прав доступа

Выводы по разделу 4

Приложение для базы данных разработано с применением современных подходов к созданию веб-приложений. Оно основано на MVC-фреймворке *Laravel*. Для основных взаимодействий приложения с базой данных использована ORM-библиотека *Eloquent*.

В целях демонстрации основной функциональности проекта, в приложении реализованы операции создания и модификации основных сущностей, а также операции создания и удаления связей между ними. Разработанная функциональность позволяет гибко настраивать права доступа для определенных ролей и для конкретных пользователей с помощью графического интерфейса. Благодаря этому, нет необходимости постоянного написания запросов во время эксплуатации БД.

ЗАКЛЮЧЕНИЕ

В ходе данного курсового проекта разработана база данных для многопользовательской онлайн игры с возможностью гибкой настройки прав доступа для пользователей, имеющих определенные роли, и для отдельных пользователей системы.

Аналитическая часть работы заключалась в исследовании предметной области и определении объектов и характеров связей между ними. В этой части была построена сложная сетевая модель, которая позже преобразована в простую сетевую.

На основе проведенного анализа, были построены логические модели проектируемой БД. Процесс построения полной атрибутивной модели начинался с построения диаграммы «сущность-связь» в нотации П. Чена. Далее, на основе этой диаграммы была построена модель, основанная на ключах. Итоговая логическая модель была получена путем добавления в модель, основанную на ключах, всех неключевых атрибутов сущностей. После этого, была произведена нормализация модели до нормальной формы Бойса-Кодда.

По полученной логической модели разработаны запросы на создание физической схемы базы данных. Для этой физической модели созданы автоматические тесты ограничений целостности с применением подхода модульного тестирования. Тесты разработаны на языке Python с использованием библиотек `unittest` и `testing.postgresql`.

Для демонстрации работы базы данных было разработано веб-приложение с возможностями администрирования, позволяющее добавлять и редактировать права доступа и роли. Созданные роли и права доступа могут быть назначены пользователям.

Таким образом, в ходе реализации проекта были выполнены все поставленные задачи, цель курсового проекта была достигнута.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Ambler, S. Agile Database Techniques - Effective Strategies for Agile Software Developer [Текст] / S. Ambler. – Wiley Publishing, 2003 – 373 с.
2. Chen, P. The Entity-Relationship Model-Toward a Unified View of Data [Текст] / P. Chen. – 1976 – 36 с.
3. Halpin, T. Information Modeling and Relational Databases [Текст] / T. Halpin, T. Morgan. – 2-е изд. – Elsevier:Burlington, 2008. – 943 с.
4. Дейт К. Дж. Введение в системы баз данных [Текст] / К. Дж. Дейт. – 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.: ил. — Парал. тит. англ.
5. Мейер Д. Теория реляционных баз данных [Текст] / Д. Мейер. – Мир: Москва, 1987 – 608 с.

ПРИЛОЖЕНИЕ А

Листинг запросов инициализации БД

```
--liquibase formatted sql

--changeset klyshko.n:201911290955
CREATE TABLE news(
    id INT,
    author_id INT,
    title VARCHAR(255),
    sub_title VARCHAR(255),
    content text,
    img VARCHAR(255),
    date TIMESTAMP,
    PRIMARY KEY (id)
);

--changeset klyshko.n:201912151136
CREATE TABLE "user" (
    id                                uuid NOT NULL,
    name                             VARCHAR(16) NOT NULL UNIQUE,
    email                            VARCHAR(255) UNIQUE,
    password                         VARCHAR(255) NOT NULL,
    registration_date                TIMESTAMP NOT NULL,
    authentication_date              TIMESTAMP,
    authentication_address cidr,
    experience                        int4 DEFAULT 0 NOT NULL,
    level                            int2 DEFAULT 0 NOT NULL,
    last_online_date                 TIMESTAMP,
    played_time                       int8 DEFAULT 0 NOT NULL,
    balance                          NUMERIC(2) DEFAULT 0 NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (name),
    UNIQUE (email)
);

--changeset klyshko.n:201912151137
CREATE TABLE permission (
    name          VARCHAR(255) NOT NULL,
    created_at    TIMESTAMP,
    description text,
    PRIMARY KEY (name)
);

--changeset klyshko.n:201912151138
CREATE TABLE role (
    name          VARCHAR(255) NOT NULL,
    created_at    TIMESTAMP,
    description text,
    PRIMARY KEY (name)
);

--changeset klyshko.n:201912151139
CREATE TABLE "role-permission" (
    role          VARCHAR(255) NOT NULL,
    permission     VARCHAR(255) NOT NULL,
    gived_at      TIMESTAMP NOT NULL,
    PRIMARY KEY (role, permission),
    CONSTRAINT "permissions_group-permission_fk1" FOREIGN KEY (role) REFERENCES
role(name)
```

```

        ON UPDATE CASCADE ON DELETE RESTRICT,
        CONSTRAINT "permissions_group-permission_fk2" FOREIGN KEY (permission) REFERENCES
permission(name)
        ON UPDATE CASCADE ON DELETE RESTRICT
    );

--changeset klyshko.n:201912151140
CREATE TABLE "user-permission" (
    user_id      uuid NOT NULL,
    permission   VARCHAR(255) NOT NULL,
    gived_at    TIMESTAMP NOT NULL,
    PRIMARY KEY (user_id, permission),
    CONSTRAINT "user-permission_fk1" FOREIGN KEY (user_id) REFERENCES "user"(id)
        ON UPDATE RESTRICT ON DELETE RESTRICT,
    CONSTRAINT "user-permission_fk2" FOREIGN KEY (permission) REFERENCES permis-
sion(name)
        ON UPDATE CASCADE ON DELETE RESTRICT
);

--changeset klyshko.n:201912151205
CREATE TABLE "user-role" (
    user_id      uuid NOT NULL,
    role VARCHAR(255) NOT NULL,
    assigned_at  TIMESTAMP NOT NULL,
    PRIMARY KEY (user_id, role),
    CONSTRAINT "user-permissions_group_fk1" FOREIGN KEY (user_id) REFERENCES
"user"(id)
        ON UPDATE RESTRICT ON DELETE RESTRICT,
    CONSTRAINT "user-permissions_group_fk2" FOREIGN KEY (role) REFERENCES role(name)
        ON UPDATE CASCADE ON DELETE RESTRICT
);

--changeset klyshko.n:201912151206
CREATE SEQUENCE donation_id_generator START 1;

--changeset klyshko.n:201912151207
CREATE TABLE donation (
    id          INT NOT NULL DEFAULT nextval('donation_id_generator'),
    user_id     uuid NOT NULL,
    date        TIMESTAMP NOT NULL,
    amount      NUMERIC(2) NOT NULL,
    payment_id  VARCHAR(255),
    payment_account VARCHAR(255),
    payment_provider VARCHAR(255),
    details     text,
    PRIMARY KEY (id),
    CONSTRAINT "user-donation_fk1" FOREIGN KEY (user_id) REFERENCES "user"(id)
        ON UPDATE RESTRICT ON DELETE RESTRICT
);

--changeset klyshko.n:201912151715
CREATE TABLE "user-change_request" (
    user_id      uuid NOT NULL,
    date         TIMESTAMP NOT NULL,
    address      cidr NOT NULL,
    request_type int2 not NULL,
    PRIMARY KEY (user_id)
);

--changeset klyshko.n:201912151717
CREATE TABLE "user-password_reset_request" (

```

```

        user_id          uuid NOT NULL,
        confirmation_code VARCHAR(255) NOT NULL,
        PRIMARY KEY (user_id),
        CONSTRAINT "password_reset_request_user_id_fk" FOREIGN KEY (user_id) REFERENCES
"user-change_request"(user_id)
        ON UPDATE RESTRICT ON DELETE RESTRICT
    );

--changeset klyshko.n:201912151719
CREATE TABLE "user-email_change_request" (
    user_id          uuid NOT NULL,
    new_email        VARCHAR(255) NOT NULL,
    unbind_confirmation_code VARCHAR(255) NOT NULL,
    bind_confirmation_code  VARCHAR(255) NOT NULL,
    PRIMARY KEY (user_id),
    CONSTRAINT "email_change_request_user_id_fk" FOREIGN KEY (user_id) REFERENCES
"user-change_request"(user_id)
    ON UPDATE RESTRICT ON DELETE RESTRICT
);

--changeset klyshko.n:201912161246
CREATE TABLE "laravel_session" (
    user_id          uuid NOT NULL,
    remember_token VARCHAR(255),
    PRIMARY KEY (user_id),
    CONSTRAINT "laravel_session_user_id_fk" FOREIGN KEY (user_id) REFERENCES
"user"(id)
    ON UPDATE CASCADE ON DELETE CASCADE
);

```

ПРИЛОЖЕНИЕ Б

Исходный код автоматических тестов

```

import unittest
import testing.postgresql
import psycopg2
import sys

init_sql = sys.argv[1]

def initialize_db_schema(postgresql):
    conn = psycopg2.connect(**postgresql.dsn())
    cursor = conn.cursor()
    init_file = open(init_sql, 'r')
    cursor.execute(init_file.read())
    cursor.close()
    conn.commit()
    conn.close()

Postgresql = testing.postgresql.PostgresqlFactory(cache_initialized_db=True,
                                                  on_initialized=initial-
ize_db_schema)

def tearDownModule():
    Postgresql.clear_cache()

class TestDatabaseConstraints(unittest.TestCase):

    @classmethod
    def setUpClass(self):
        self.postgresql = Postgresql()
        self.conn = psycopg2.connect(**self.postgresql.dsn())
        self.cursor = self.conn.cursor()
        self.cursor.execute("""
            INSERT INTO "user"(id, name, email, password, registration_date)
            VALUES('5b92f67a-d21c-4a73-9062-613bc46d0a20', 'test', 'test@mailtest.ru',
'1qaz2wsx', current_timestamp)
            """)
        self.cursor.close()
        self.conn.commit()

    @classmethod
    def tearDownClass(self):
        self.postgresql.stop()

    def setUp(self):
        self.cursor = self.conn.cursor()

    def tearDown(self):
        self.cursor.close()
        self.conn.commit()

    def testUserPrimaryKey(self):
        with self.assertRaises(Exception) as cm:
            self.cursor.execute("""
                INSERT INTO "user"(id, name, email, password, registration_date)
                VALUES('5b92f67a-d21c-4a73-9062-613bc46d0a20', 'test2', 'test2@mail-
test.ru', '1qaz2333wsx', current_timestamp)
            """)

```

```

        """
    self.assertIsNotNone(cm.exception)

    def testUserUniqueName(self):
        with self.assertRaises(Exception) as cm:
            self.cursor.execute("""
                INSERT INTO "user"(id, name, email, password, registration_date)
                VALUES('5b92f67a-d21c-4a73-9062-613bc46d0a21', 'test', 'test3@mail-
test.ru', '1qaz2222wsx', current_timestamp)
            """)
            self.assertIsNotNone(cm.exception)

    def testUserUniqueEmail(self):
        with self.assertRaises(Exception) as cm:
            self.cursor.execute("""
                INSERT INTO "user"(id, name, email, password, registration_date)
                VALUES('5b92f67a-d21c-4a73-9062-613bc46d0a22', 'test123', 'test@mail-
test.ru', '1qaz2222wsx', current_timestamp)
            """)
            self.assertIsNotNone(cm.exception)

    def testNullableUniqueEmail(self):
        self.cursor.execute("""
            INSERT INTO "user"(id, name, email, password, registration_date)
            VALUES('5b92f67a-d21c-4a73-9062-613bc46d0a24', 'testM', NULL, '1qaz2222wsx',
current_timestamp)
        """)
        self.conn.commit()
        self.cursor.execute("""
            INSERT INTO "user"(id, name, email, password, registration_date)
            VALUES('5b92f67a-d21c-4a73-9062-613bc46d0a25', 'testM2', NULL, '1qaz2222wsx',
current_timestamp)
        """)
        self.conn.commit()

if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

```

ПРИЛОЖЕНИЕ В

Исходный код приложения

User.php:

```
class User extends Authenticatable
{
    use Notifiable;

    protected $fillable = ['name', 'email', 'password',];
    protected $hidden = ['password', 'registration_date',];
    protected $casts = ['registration_date' => 'datetime',];

    protected $table = 'user';
    protected $primaryKey = 'id';
    protected $keyType = 'string';
    public $incrementing = false;
    public $timestamps = false;

    protected static function boot()
    {
        parent::boot();

        static::creating(function (Model $model) {
            $model->setAttribute($model->getKeyName(), Uuid::uuid4());
            $model->setAttribute('registration_date', Carbon::now());
        });

        static::saving(function (Model $model) {
            if ($model->rememberToken != null) {
                $model->rememberToken()->save($model->rememberToken);
            }
        });
    }

    /**
     * Has one-to-one relationship with Account\RememberToken because session data placed in separate table.
     */
    public function rememberToken()
    {
        return $this->hasOne('App\Models\RememberToken', 'user_id', 'id');
    }

    public function getRememberToken()
    {
        return optional($this->rememberToken)->remember_token;
    }

    public function setRememberToken($value)
    {
        if (!$this->rememberToken()->exists()) {
            RememberToken::create([
                'user_id' => $this->getAttribute($this->primaryKey),
                'remember_token' => $value
            ]);
        } else {
            $this->load('rememberToken');
            $this->rememberToken->remember_token = $value;
        }
    }

    public function hasVerifiedEmail()
    {
        return true;
    }

    public function getAuthIdentifierName()
    {
        return 'email';
    }

    public function getAuthIdentifier()
```

```

    {
        return $this->email;
    }

    public function roles()
    {
        return $this->belongsToMany('App\Models\Role', 'user-role',
            'user_id', 'role', 'id', 'name', 'role');
    }

    public function permissions()
    {
        return $this->belongsToMany('App\Models\Permission', 'user-permission',
            'user_id', 'permission', 'id', 'name', 'permission');
    }

    public function assignRole($role)
    {
        try {
            $this->roles()->attach([$role => ['assigned_at' => Carbon::now()]]);
            return true;
        } catch (\Exception $e) {
            return false;
        }
    }

    public function removeRole($role)
    {
        $this->roles()->detach($role);
    }

    public function givePermission($permission)
    {
        try {
            $this->permissions()->attach([$permission => ['gived_at' => Carbon::now()]]);
            return true;
        } catch (\Exception $e) {
            return false;
        }
    }

    public function removePermission($permission)
    {
        $this->permissions()->detach($permission);
    }

    public function hasPermission($permission)
    {
        foreach ($this->roles as $role) {
            if($role->hasPermission($permission)) {
                return true;
            }
        }
        return $this->permissions->contains($permission);
    }
}

```

Permission.php:

```

class Permission extends Model
{
    protected $table = 'permission';
    protected $primaryKey = 'name';
    protected $keyType = 'string';
    public $timestamps = false;

    protected $fillable = ['name', 'description'];
    protected $hidden = ['created_at'];

    protected static function boot()
    {
        parent::boot();

        static::creating(function (Model $model) {
            $model->setAttribute('created_at', Carbon::now());
        });
    }
}

```



```

    }
}

```

Role.php:

```

class Role extends Model
{
    protected $table = 'role';
    protected $primaryKey = 'name';
    protected $keyType = 'string';
    public $timestamps = false;

    protected $fillable = ['name', 'description'];
    protected $hidden = ['created_at'];

    protected static function boot()
    {
        parent::boot();

        static::creating(function (Model $model) {
            $model->setAttribute('created_at', Carbon::now());
        });
    }

    public function permissions()
    {
        return $this->belongsToMany('App\Models\Permission', 'role-permission',
            'role', 'permission', 'name', 'name', 'permission');
    }

    public function givePermission($permission)
    {
        try {
            $this->permissions()->attach([$permission => ['gived_at' => Carbon::now()]]);
            return true;
        } catch (\Exception $e) {
            return false;
        }
    }

    public function removePermission($permission)
    {
        $this->permissions()->detach($permission);
    }

    public function hasPermission($permission)
    {
        return $this->permissions->contains($permission);
    }

    public function getChangedAtAttribute()
    {
        return $this->permissions()->max('gived_at');
    }
}

```

UserController.php:

```

class UserController extends Controller
{
    public function find(Request $request)
    {
        $user = User::whereRaw("LOWER(name) = LOWER('". $request->get('username')
            . "')")->first();
        if ($user == null) {
            return redirect()->route('admin')->with('error', 'Not found!');
        }
        return redirect()->route('admin.user.show', $user->id);
    }

    public function show($id)
    {

```

```

    {
        return view('admin.user',
            ['user' => User::findOrFail($id), 'permissions' => Permission::all(),
            'roles' => Role::all()]
        );
    }

    public function removePermission(Request $request, $id)
    {
        $user = User::findOrFail($id);
        $user->removePermission($request->get('permission'));
        return redirect()->route('admin.user.show', $user->id)->with('success', 'Per-
mission removed.');
```

```

    }

    public function givePermission(Request $request, $id)
    {
        $user = User::findOrFail($id);
        if ($user->givePermission($request->get('permission'))) {
            return redirect()->route('admin.user.show', $user->id)->with('success',
            'Permission given.');
```

```

        }
        return redirect()->route('admin.user.show', $user->id)->with('error', 'Per-
mission not found!');
```

```

    }

    public function removeRole(Request $request, $id)
    {
        $user = User::findOrFail($id);
        $user->removeRole($request->get('role'));
        return redirect()->route('admin.user.show', $user->id)->with('success', 'Role
removed.');
```

```

    }

    public function assignRole(Request $request, $id)
    {
        $user = User::findOrFail($id);
        if ($user->assignRole($request->get('role'))) {
            return redirect()->route('admin.user.show', $user->id)->with('success',
            'Role assigned.');
```

```

        }
        return redirect()->route('admin.user.show', $user->id)->with('error', 'Role
not found!');
```

```

    }
}

```

ManageController.php:

```

class ManageController extends Controller
{
    public function index()
    {
        return view('admin.manage', ['permissions' => Permission::all(), 'roles' =>
        Role::all()]);
    }

    public function permission(Request $request)
    {
        $permission = Permission::find($request->get('permission'));
        if ($permission == null) {
            Permission::create([
                'name' => $request->get('permission'),

```

```

        'description' => $request->get('description'),
    ]));
    return redirect()->route('admin.manage')->with('success', 'Permission
created.');
```

```

    } else {
        $permission->name = $request->get('permission');
        $permission->description = $request->get('description');
        $permission->save();
        return redirect()->route('admin.manage')->with('success', 'Permission up-
dated.');
```

```

    }
}

public function role(Request $request)
{
    $role = Role::find($request->get('role'));
    if ($role == null) {
        Role::create([
            'name' => $request->get('role'),
            'description' => $request->get('description'),
        ]);
        return redirect()->route('admin.role.show', $request->get('role'))-
>with('success', 'Role created.');
```

```

    } else {
        $role->name = $request->get('role');
        $role->description = $request->get('description');
        Facade::log('SAVE: ' . $role->save());
        return redirect()->route('admin.role.show', $role->name)->with('success',
'Role updated.');
```

```

    }
}

}

```