МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Выполнил студент

(Ф.И.О.)

Руководитель проекта

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант:

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер:

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2022

Оглавление

[Введение 4](#_Toc134897964)

[1 Анализ требований к программному средству 5](#_Toc134897965)

[1.1 Аналитический обзор аналогов 5](#_Toc134897966)

[1.2 Функциональных требования, определение вариантов использования 6](#_Toc134897967)

[1.3 Вывод по разделу 8](#_Toc134897969)

[2 Разработка архитектуры проекта 9](#_Toc134897971)

[2.1 Обобщенная структура управлением приложения 9](#_Toc134897972)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 9](#_Toc134897973)

[3 Разработка модели базы данных 11](#_Toc134897974)

[3.1 Создание необходимых объектов 11](#_Toc134897975)

[3.1.1 Представления базы данных 11](#_Toc134897976)

[3.1.2 Индексы базы данных 12](#_Toc134897977)

[3.1.3 Триггеры базы данных 13](#_Toc134897978)

[3.2 Описание используемой технологии 14](#_Toc134897979)

[3.3 Вывод 15](#_Toc134897980)

[4 Установка, настройка и использование PosgtreSQL 15.2 16](#_Toc134897981)

[4.1 Установка PostgreSQL 16](#_Toc134897982)

[4.2 Создание таблиц 16](#_Toc134897984)

[4.3 Создание пакетов процедур для базы данных 17](#_Toc134897985)

[4.4.1 Получение данных 17](#_Toc134897987)

[4.4.2 Шифрование данных 18](#_Toc134897988)

[4.4.3 Заполнение таблиц 100 000 строк 19](#_Toc134897989)

[4.4.4 Добавление данных в таблицы 20](#_Toc134897990)

[4.4.5 Удаление данных из таблиц 20](#_Toc134897991)

[4.4.6 Дополнительные функции 21](#_Toc134897992)

[4.5 Описание процедур экспорта и импорта 22](#_Toc134897993)

[4.6 Вывод 23](#_Toc134897994)

[5 Тестирование 24](#_Toc134897995)

[5.1 Тестирование производительности базы данных 24](#_Toc134897996)

[5.2 Вывод 25](#_Toc134897997)

[6 Руководство по использованию программного средства 26](#_Toc134897998)

[6.1 Руководство пользователя 26](#_Toc134897999)

[6.2 Вывод 28](#_Toc134898000)

[Заключение 29](#_Toc134898001)

[Список литературных источников 30](#_Toc134898002)

[Приложение А 31](#_Toc134898003)

# Введение

Целью работы является разработка базы данных для хранения информации о лотах, участниках и торгах на аукционе. Создание такой базы данных позволит эффективно организовать процесс проведения аукциона, обеспечив удобный доступ к информации о лотах и участниках, а также обеспечив точный и надежный учет торгов. Кроме того, целью работы может быть создание системы управления аукционом, которая будет позволять автоматизировать процессы взаимодействия участников аукциона, а также обеспечивать мониторинг и контроль за выполнением правил проведения аукциона.

База данных (БД) - это система, предназначенная для хранения, организации и управления данными в структурированном виде, которая обеспечивает удобный доступ к этим данным и позволяет эффективно обрабатывать и анализировать большие объемы информации. База данных может содержать информацию о различных объектах (например, людях, продуктах, заказах и т.д.), а также связи между ними и правила для доступа и использования этой информации. База данных используется в различных областях, включая бизнес, науку, медицину и другие.

Для обеспечения функциональности приложения используются технология мультимедийных форматов данных при сохранении картинок, а также технология резервного копирования базы данных.

Основные требования к приложению:

* Реализация удобной базы данных.
* Реализация приложения связанного с базой данных.
* Взаимодействие с базой данных при помощи хранимых процедур и функций.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Аукцион - это форма торговли, при которой продавец предлагает на продажу товар или услугу, а покупатели делают конкурентные ставки, чтобы приобрести этот товар или услугу. Аукционы широко используются в различных отраслях и имеют значительную актуальность. В бизнесе аукционы используются для продажи товаров и услуг, привлечения инвесторов, размещения государственных закупок, а также для определения рыночной стоимости товаров и услуг.

**1.1.1 Аналог eBay**

eBay - крупнейший онлайн-аукцион, который предоставляет услуги по продаже и покупке товаров по всему миру. На сайте eBay можно найти широкий ассортимент товаров - от электроники и модной одежды до автомобилей и недвижимости. В качестве продавцов на eBay могут выступать как частные лица, так и бизнесы. Сайт предоставляет широкие возможности для поиска и фильтрации товаров по различным параметрам, а также для общения между продавцами и покупателями. Пример интерфейса данного сервиса представлен на рисунке 1.1.

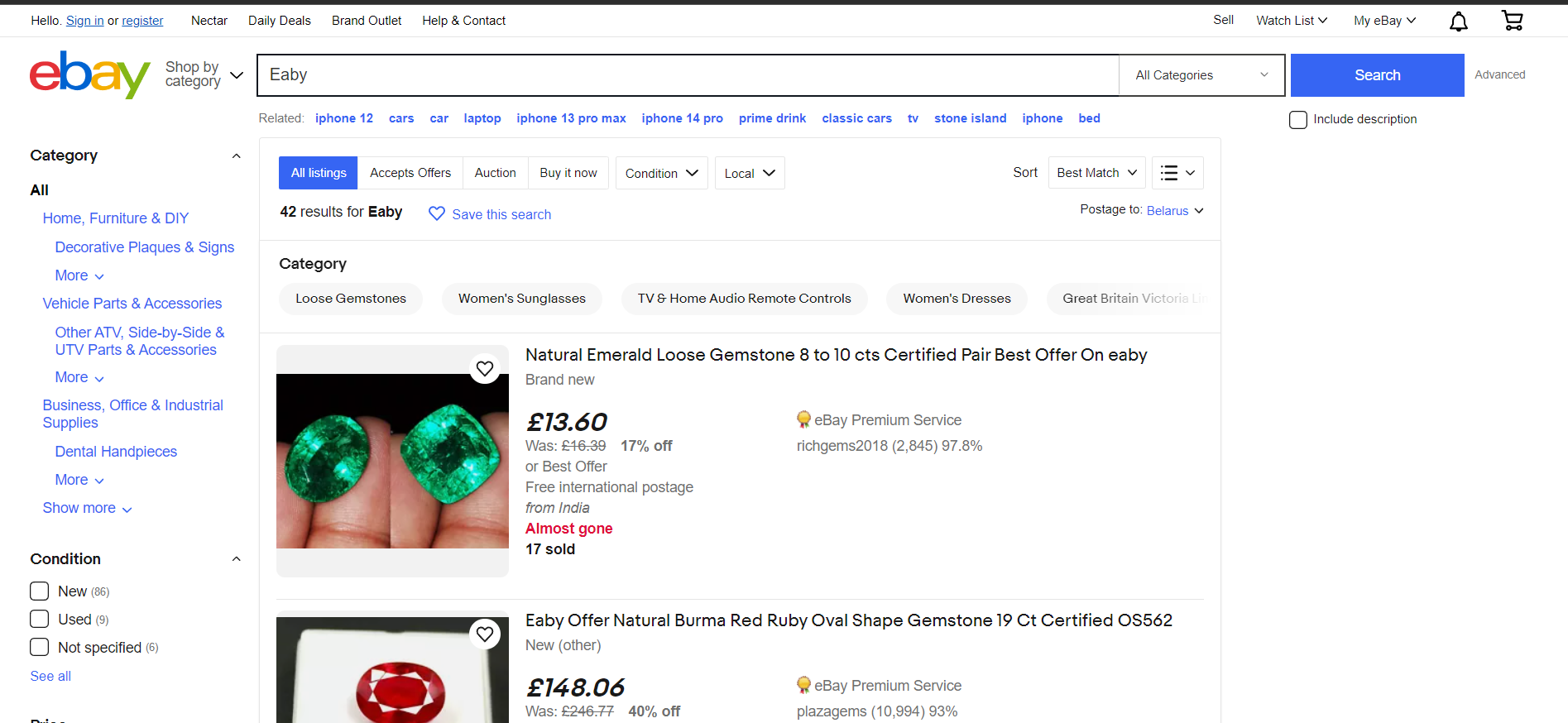


Рисунок 1.1 – Интерфейс сервиса eBay

eBay позволяет продавать и покупать товары во всем мире, что делает его идеальным местом для покупателей и продавцов, которые ищут товары, которые не доступны в их регионе. Безопасность сделок: eBay предлагает множество инструментов для защиты покупателей, например, систему гарантии на определенные виды товаров, возможность возврата товара в случае неудовлетворительности, а также систему обратной связи, которая позволяет продавцам и покупателям оценивать друг друга после завершения сделки.

**1.1.2 Аналог Amazon Auctions**

Amazon Auctions - это онлайн-аукцион, который был запущен Amazon в 1999 году, но в настоящее время он уже не существует. Amazon Auctions предоставлял возможность продавать и покупать товары на аукционе, а также использовать опцию "Купить сейчас" для мгновенной покупки товара. Interface Amazon Auctions был разработан таким образом, чтобы обеспечить простоту использования для пользователей. Сайт был хорошо организован, с отдельными разделами для каждой категории товаров и поиском, который позволял быстро найти нужный товар. Amazon также предоставлял множество информации о каждом товаре, включая описание, фотографии и отзывы покупателей. Пример интерфейса данного сервиса представлен на рисунке 1.2.

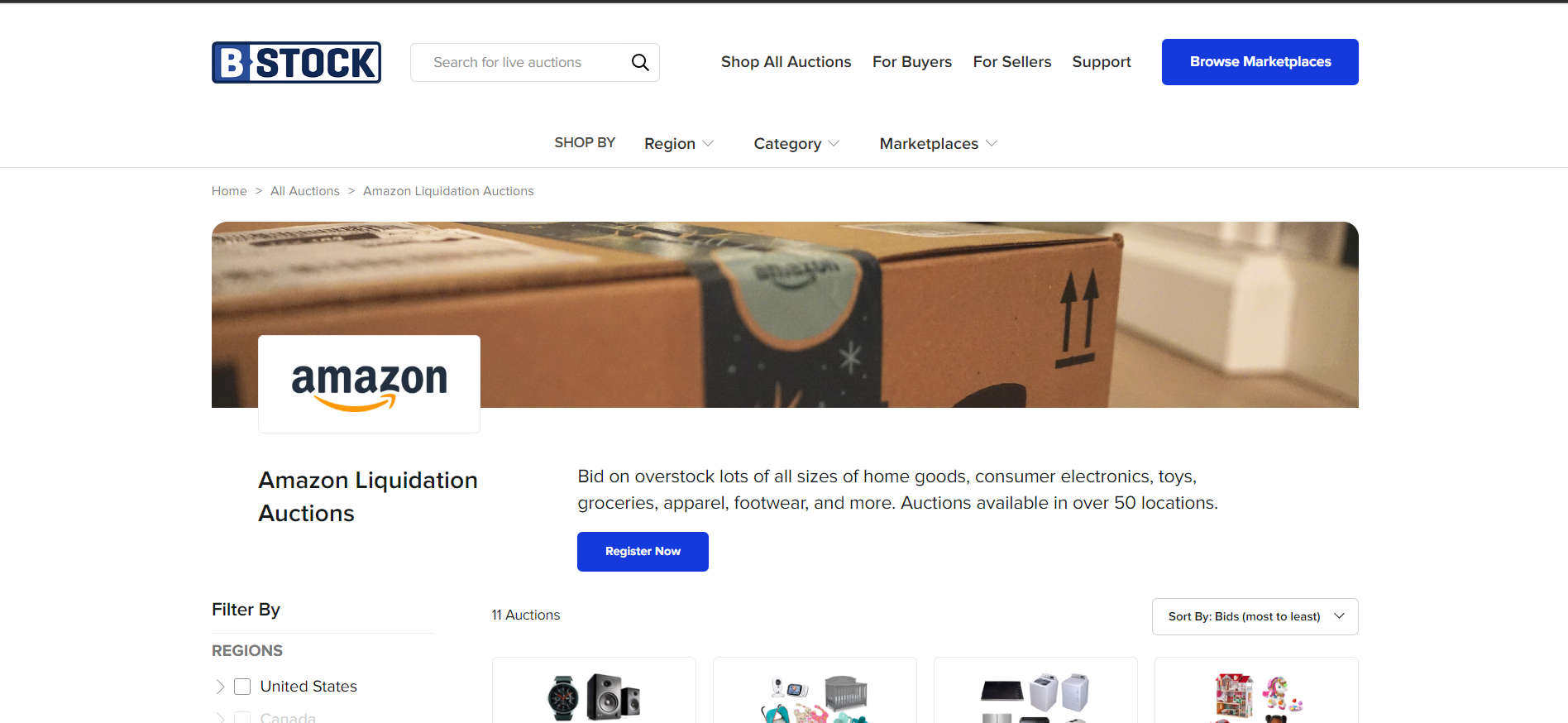


Рисунок 1.2 – Интерфейс сервиса Amazon Auctions

Interface Amazon Auctions был разработан таким образом, чтобы обеспечить простоту использования для пользователей. Сайт был хорошо организован, с отдельными разделами для каждой категории товаров и поиском, который позволял быстро найти нужный товар. Amazon также предоставлял множество информации о каждом товаре, включая описание, фотографии и отзывы покупателей.

## 1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования базы данных определяют, как база данных должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Кроме того, функциональные требования могут определять интеграцию базы данных с другими системами и программами. Например, для площадки аукциона функциональные требования могут включать в себя функции для хранения информации аукционах и их лотах.

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде Use case диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой. На рисунке 1.3 представлена uml диаграмма вариантов использования.

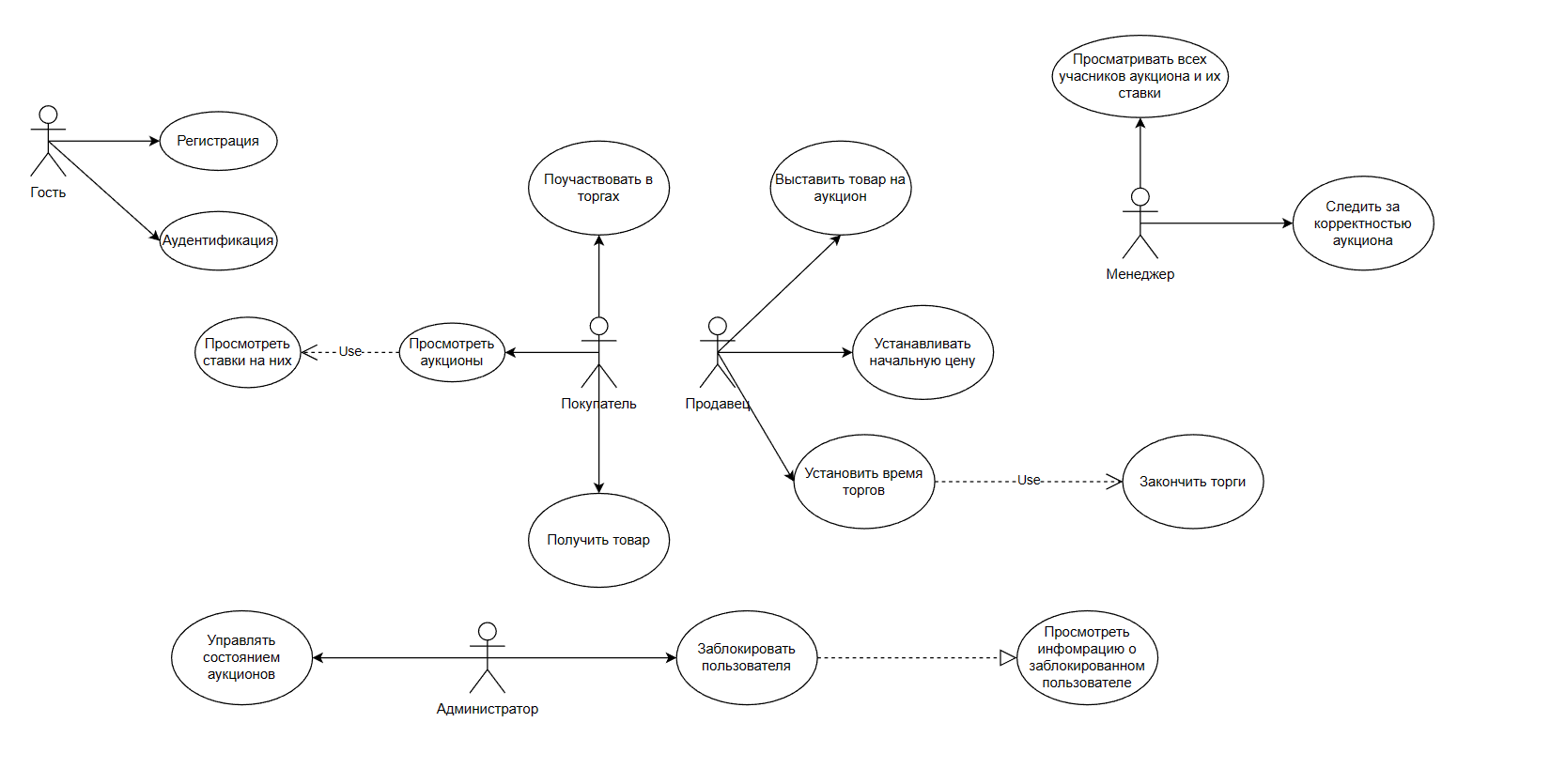


Рисунок 1.3 – Use case диаграмма

В начале работы с приложением пользователь является гостем. Ему будет доступна только просмотр всей информации без возможности участвовать в аукционе. После регистрации гость становится пользователем (User).

После регистрации пользователь (User) получает возможность участвовать в торгах.

Роль администратора (Admin) заключается в управлении площадкой и мониторинге действий аукциона. Администратор имеет доступ к просмотру информации о всех пользователях, в том числе их действиях на площадке.

## В данном разделе были определены роли пользователей и разработаны варианты использования системы в зависимости от этих ролей.

Также присутствует роль Менеджера, который следить за корректностью проведения аукционов.

## 1.3 Вывод по разделу

## По итогу был сделан вывод, что при проектировании базы данных необходимо учитывать функциональные требования, которые определяют, каким образом данные будут храниться, обрабатываться и предоставляться пользовательскому интерфейсу. Кроме того, важно определить роли пользователей и их варианты использования системы, чтобы грамотно организовать функциональность и навигацию в системе. Также важно просмотреть аналогичные решения, и перенять их положительные стороны.

1. Разработка архитектуры проекта

## Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения аукциона будет входить мгновенное повышение ставки на аукционе. Также будет реализована возможность быть учасником ауциона и продавать свой же товар.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) - это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных будет представлена на рисунке 2.1.

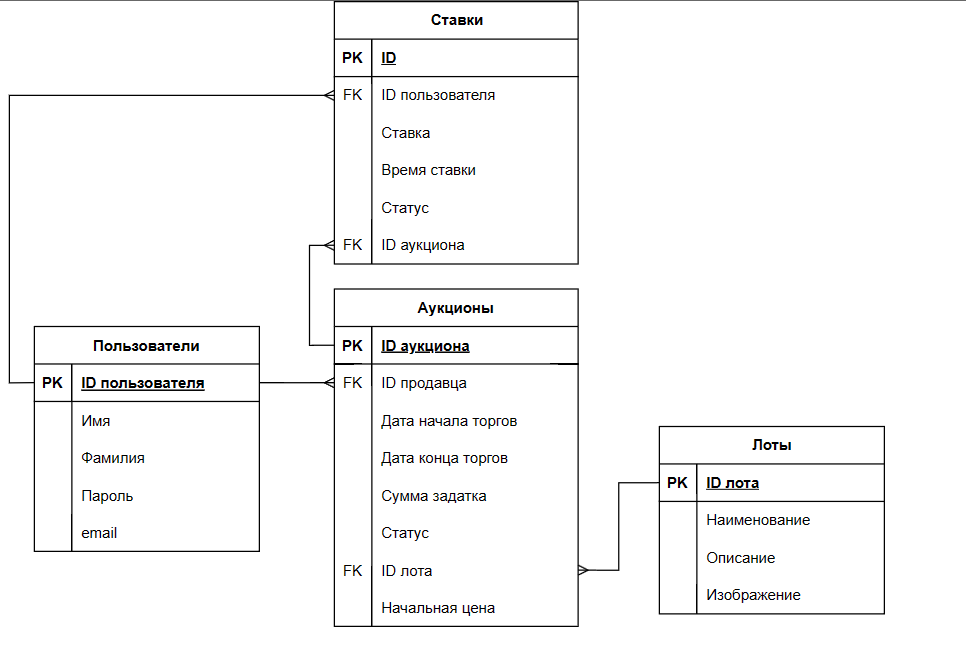


Рисунок 2.1 – Диаграмма базы данных

Видно, что диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-многим", "многие-ко-многим" и "один-к-одному". Например, таблица Ставки связана с таблицами Пользователями, через внешний ключь ID\_Пользователя. Также видно, что таблицы Лоты и Аукционы связаны друг с другом через внешний и первичный ключь ID лота. Все связи представлены на рисунке 2.1, который располагается выше.

**2.3 Описание информационных объектов**

Для реализации базы данных будет разработано 4 таблицы. В структуру схемы базы данных для проекта входят следующие таблицы: Ставки, Пользователи, Аукционы, Лоты. Ниже разобраны каждая из них более подробно.

Таблица Ставки представляет собой перечень ставок, которые делают пользователи. Имеющиеся поля: ID, Id\_Пользователя, Ставка, Время ставки, Статус, ID аукциона.

Таблица Пользователи содержит информацию о пользователях в аукционе. Имеющиеся поля: ID пользователя, Имя, Фамилия, Пароль, email, Роль.

Таблица Аукционы содержит информацию о имеющихся на данный момент аукционах. Имеющиеся поля: ID аукциона, ID пользователя, Дата начала торгов, Дата конца торгов, Статус, ID лота, Начальная цена.

Таблица Лоты содержит информацию о товарах имеющихся на аукционе. Имеющиеся поля: ID лота, Наименование, Описание, Изображение.

**2.4 Вывод**

Проектирование структуры и функциональности приложения требует разработки архитектуры проекта. Создание общей структуры управления приложением позволяет определить компоненты, необходимые для его реализации, и их взаимодействие между собой.

Описание информационных объектов также является важной частью архитектуры проекта, которая позволяет определить, какие данные будут использоваться в приложении, а также их хранение и обработку. Как правило, разработка архитектуры проекта является ключевым этапом в создании функционального и эффективного приложения, так как она помогает определить необходимые компоненты и информационные объекты для достижения целей проекта.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

Для площадки аукциона необходимо создать несколько таблиц, которые будут хранить данные о пользователях, лотах, аукционов, и ставках.

Для реализации данного проекта было принято решение о создании следующих таблиц:

Пользователи – таблица, содержащая инфомарцию о пользователях. В этой таблице будут храниться записи, связанные с данными о пользователях, такиме как: Имя, фамилия, пароль, email, роль.

Ставки – таблица, содержащая данные о ставках, принятых на лоты. В этой таблице будут храниться информация о всех ставках, которые были приняты на определенные лоты: ID Пользователя сделавлешего ставку, ставка, время ставки, статус, ID\_аукциона на который принята ставка.

Аукционы – таблица, содержащая информацию о текущих аукционах, содержит: пользователей, участвующих в аукционе, дата, когда аукцион начнется и закончится, статус данного аукциона, начальную цену при которой можно поучаствовать в аукционе.

Лоты – таблица, содержащая информацию о предметах, выставленных на аукцион. В ней отображены наименование предмета, описание их, а также изображение на товар. В табилце аукцион будет отображена цена на данный лот.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Например, можно создать индексы на столбцах пароль в таблице Пользователи, чтобы быстро находить информацию о пользователе по его паролю.

Для более удобной работы с базой данных можно создать несколько функций и процедур. Например, можно создать функцию для поиска аукциона по заданным критериям, таким как его цена, дата начала и конца аукциона.

### 3.1.1 Представления базы данных

Представление в базе данных - это виртуальная таблица, которая формируется на основе выполнения запроса к одной или нескольким таблицам в базе данных. Представления позволяют объединять данные из нескольких таблиц в одну, не изменяя исходных таблиц и их структуры. Представления используются для обеспечения удобства доступа и управления данными в базе данных, а также для улучшения производительности запросов. В данном проекте были созданы четыре представления:

* num\_lots\_on\_auction, представление, которое показывает количество лотов, выставленных на аукционе;
* last\_bit\_for\_lot, представление, которое показывает последнюю ставку;
* user\_role\_and\_num\_bids, представление которое показывает список пользователей с их ролями и количеством сделанным ими ставок.
* num\_bids\_per\_user, представление которое показывает количество ставок, сделанных каждым пользователем.

Реализация нескольких из представлений представлений представлено в листинге 3.1. Остальные представления будут аналогичны, работая с другими данными.

|  |
| --- |
| CREATE VIEW user\_role\_and\_num\_bids AS  SELECT P.ID\_Пользователя, P.Имя, P.Фамилия, P.Роль, COUNT(\*) AS num\_bids  FROM Пользователи P  JOIN Ставки S ON P.ID\_Пользователя = S.ID\_Пользователя  GROUP BY P.ID\_Пользователя, P.Имя, P.Фамилия, P.Роль;  CREATE VIEW last\_bid\_for\_lot AS  SELECT Л.ID\_Лота, Л.Наименование, S.Ставка, S.Время\_ставки, S.ID\_Пользователя  FROM Лоты Л  LEFT JOIN Ставки S ON Л.ID\_Лота = S.ID\_аукциона  WHERE S.ID\_Ставки = (SELECT MAX(ID\_Ставки) FROM Ставки WHERE ID\_аукциона = Л.ID\_Лота); |

Листинг 3.1 – Представления user\_role\_and\_num\_bids, last\_bid\_for\_lot

Представление user\_role\_and\_num\_bids было создано для того, чтобы объедиить данные из таблицы Пользователи (Users) и Ставки (Bids), чтобы предоставить информацию о количестве ставок, сделанных каждым пользователем, а также их роль в системе.

Представление last\_bid\_for\_lot было создано для того, чтобы получить информацию о лотах, а также объединения ставок на определенные ставки, тем самым вывев все сразу о ставках и лотах в удобной структуре.

### 3.1.2 Индексы базы данных

Индекс в базе данных представляет собой объект, который используется для ускорения поиска данных. Если таблица содержит большое количество строк, то последовательный поиск данных может занимать много времени. Индекс создается на основе значений одного или нескольких столбцов таблицы и указывает на соответствующие строки таблицы. Использование индексов помогает улучшить производительность базы данных, поскольку они имеют оптимизированную структуру для поиска, например, сбалансированное дерево. Один из наиболее распространенных алгоритмов индексации - это B-дерево (B-tree). B-дерево - это сбалансированное дерево поиска, которое обеспечивает эффективный поиск элементов, используя ключи для сортировки данных в индексе.

Для того, чтобы быстро находить пользователей по их параметрам, были созданы четыре индекса: idx\_имя на таблицу Пользователи по полю Имя, idx\_фамилия на таблице Пользователи по полю фамилия. Создание индексов представлено в листинге 3.2.

|  |
| --- |
| CREATE INDEX idx\_имя ON Пользователи (Имя);  CREATE INDEX idx\_фамилия ON Пользователи (Фамилия);  CREATE INDEX idx\_email ON Пользователи (email);  CREATE INDEX idx\_роль ON Пользователи (Роль); |

Листинг 3.2 – Индексы базы данных

Кроме индексов представленных выше, в базе данных присутствуют индексы в таблицах: Ставки, Аукционы, Лоты.

Индекс idx\_описание в таблице Лоты был создан для ускорения поиска лота с помощью его описания.

Индекс idx\_id\_аукцион в таблице Ставки используется для быстрого поиска ставок по ID аукциона.

Индекс idx\_id\_пользователя в таблице Aукцион был создан для ускорения поиска пользователей, участвующих в аукционе по их id.

Применение индексов в базе данных значительно повышает скорость выполнения операций поиска, сортировки и фильтрации данных, особенно в случае больших объемов информации. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах. Поэтому необходимо балансировать количество и типы индексов, чтобы обеспечить оптимальную производительность базы данных.

### 3.1.3 Триггеры базы данных

Триггер в базе данных - это объект, который автоматически выполняет определенные действия при возникновении определенных событий в таблице или представлении базы данных. Триггер может быть настроен на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для поддержки целостности данных, контроля доступа к данным и автоматической обработки данных при выполнении определенных операций в таблице.

Первый триггер, созданный в таблице Auctions, проверяет, дату в аукционе, а именно проверка на то, что дата начала торгов не может быть больше даты окончания торгов. Триггер создается с помощью функции check\_auction\_dates(). Триггер представлен в листинге 3.3.

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_auction\_dates() RETURNS TRIGGER AS $$  BEGIN  IF NEW.Дата\_Начала\_Торгов > NEW.Дата\_Конца\_Торгов THEN  RAISE EXCEPTION 'Дата начала торгов не может быть больше даты окончания торгов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_auction\_dates\_trigger  BEFORE INSERT OR UPDATE ON Аукционы  FOR EACH ROW  EXECUTE FUNCTION check\_auction\_dates(); |

Листинг 3.3 – Скрипт триггера check\_auction\_dates

Второй триггер выполняет проверку, что ставка не меньше максимальной ставки для аукциона. Это нужно для того, чтобы пользователей не смог перекрыть лот ставкой, меньше предыдущей. Скрипт триггера будет представлен на листинге 3.4.

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_bid() RETURNS TRIGGER AS $$  DECLARE  max\_bid numeric;  BEGIN  SELECT MAX(Ставка) INTO max\_bid  FROM Ставки  WHERE ID\_аукциона = NEW.ID\_аукциона;  IF max\_bid IS NULL OR NEW.Ставка >= max\_bid THEN  RETURN NEW;  ELSE  RAISE EXCEPTION 'Ставка должна быть не меньше предыдущей максимальной ставки';  END IF;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_bid\_trigger  BEFORE INSERT ON Ставки  FOR EACH ROW  EXECUTE FUNCTION check\_bid(); |

Листинг 3.4 – Скрипт триггера check\_bid

Таким образом, видно, что проверка различного рода условий поможет наложить ограничения на различные выполнения действий в базе данных.

## 3.2 Описание используемой технологии

Технология мультимедийных типов данных в базах данных (Multimedia Database Technology) – это совокупность методов и средств, которые позволяют хранить и обрабатывать разнообразные типы мультимедийных данных, таких как текст, звук, видео и изображения. В данной базе данных используется технология мультимедийности для изображений. В таблице Лоты в столбце Изображение хранится изображение лота, выставленного на аукцион VARHCAR (255), что будет означать длину ссылку, хранимого изображения. Ссылка, в наиболее обширных базах данных, должна быть расположена на сервере, тем самым каждый запрос к ней будет выполняться быстрее, чем конвертации в битовые значения BYTEA.

Таким образом, использование мультимедийности в базе данных позволяет хранить различные мультимедийные данные, необходимые для работы площадки аукциона.

Хранения картинок в базе данных представлено на рисунке 3.1.



Рисунок 3.1 – Хранение изображений в базе данных

Хранение ссылок и типа данных VARCHAR лучше подходит для хранения мультимедийных объектов в базе данных, чем тип данных BYTEA, по нескольким причинам:

Экономия места. Тип данных VARCHAR использует меньше места для хранения данных, чем тип данных BYTEA, который хранит данные в виде бинарных объектов. При хранении мультимедийных объектов, таких как изображения или видео, это может быть особенно важным, так как такие объекты могут занимать большой объем места.

Простота использования. Хранение ссылок на мультимедийные объекты, а не самих объектов, может упростить доступ к этим объектам, особенно если они хранятся на удаленном сервере. Это также может помочь избежать проблем синхронизации мультимедийных объектов, так как любые изменения, сделанные в исходном объекте, будут автоматически отражаться в базе данных.

Ускорение доступа. Хранение ссылок на мультимедийные объекты может ускорить доступ к этим объектам, особенно если они хранятся на удаленном сервере. Это может быть полезно при обработке большого количества мультимедийных объектов, так как это может уменьшить нагрузку на сервер базы данных и ускорить доступ к объектам.

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для площадки аукциона. Одним из важных аспектов является выбор технологии для хранения мультимедийных данных, таких как изображения, музыка и видео. Для этого была использована технология мультимедийности, которая позволяет хранить мультимедийные данные в формате VARCHAR(255).

4 Установка, настройка и использование PosgtreSQL 15.2

## 4.1 Установка PostgreSQL

PostgreSQL (часто называемый "Postgres") - это мощная, производительная и расширяемая система управления базами данных (СУБД), которая поддерживает широкий диапазон функций и возможностей. Эта СУБД была разработана на языке программирования C и является открытым исходным кодом, что означает, что любой может свободно использовать и изменять ее.

## PostgreSQL поддерживает множество функций, включая многопоточность, транзакционность, внешние ключи, хранимые процедуры, триггеры и многое другое. Он также поддерживает множество языков программирования, включая Java, C++, Perl, Python, Ruby и многие другие.

## 4.2 Создание таблиц

В данном разделе будут созданы таблицы для базы данных. Предед их созданием, для лучшей структуры было принято решение использовать табличные пространства.

Табличное пространство - это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств будет представлен в листинге 4.1.

|  |
| --- |
| CREATE TABLESPACE Пользователи  OWNER postgres  LOCATION 'E:\STUDY\KURS\_DATABASE\Tablespace\USER'    CREATE TABLESPACE Аукционы  OWNER postgres  LOCATION 'E:\STUDY\KURS\_DATABASE\Tablespace\Bet';    CREATE TABLESPACE Ставки  OWNER postgres  LOCATION 'E:\STUDY\KURS\_DATABASE\Tablespace\AUCTION'; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создадим четыре основные таблицы: Пользователи, Аукционы, Лоты, Ставки.

Таблицы Аукционы, Лоты будут храниться в табличном пространстве Аукционы.

Таблица Пользователи будет храниться в табличном пространстве Пользователи.

Таблица Ставки будет храниться в табличном пространстве Ставки.

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. Одна из основных связей - это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице.

|  |
| --- |
| ALTER TABLE Ставки ADD CONSTRAINT fk\_ID\_аукциона FOREIGN KEY (ID\_аукциона) references Аукционы(ID\_аукциона);  ALTER TABLE Аукционы add constraint fk\_id\_лот foreign key (ID\_Лота) references Лоты (ID\_Лота)  Alter Table Аукционы add constraint fk\_id\_Пользователя foreign key(ID\_Пользователя) references Пользователи(ID\_Пользователя); |

Листинг 4.2 – Cкрипт создание ограничений на ключи.

Таким образом, было описано создание табличного пространства для базы данных, а также таблиц, которые будут храниться в этих пространствах. Были созданы три табличных пространства: Пользователи, Аукционы и Лоты, каждом из которых будут храниться соответствующие таблицы.

## 4.3 Создание пакетов процедур для базы данных

Хранимые процедуры и функции представляют собой наборы инструкций SQL, которые могут быть использованы для управления данными через приложение. Хранимые процедуры компилируются один раз и хранятся на сервере, их можно использовать для выполнения сложных операций над данными, таких как массовые изменения в таблицах или оптимизация производительности приложения. Функции, в свою очередь, возвращают значение, которое может быть использовано в других SQL-запросах, например, для подсчета статистики или фильтрации данных.

### Выбор между хранимыми процедурами и функциями зависит от поставленных задач. Если необходимо выполнить сложные операции над данными, то лучше использовать хранимые процедуры.

### 4.4.1 Получение данных

Для вывода данных из таблиц были написаны следующие процедуры и функции: get\_active\_auction, get\_all\_auction, get\_all\_bet, get\_all\_user, get\_auction\_winner, get\_auctions\_for\_user, get\_bids\_for\_lot, get\_current\_auctions, get\_lost\_list, get\_max\_bids.

Главной задачей будет извлечение данных из созданных таблиц.

get\_active\_auction, get\_all\_auction, get\_all\_bet – функции для выборки списка аукционов, и ставок на них, другие процедуры работают по данному прицнипу, только с другими таблицами.

В листинге 4.4 будет функция get\_auction\_winner, которая предназначена для выборки победителя аукциона.

|  |
| --- |
| CREATE FUNCTION get\_auction\_winner(  p\_id\_auction integer  ) RETURNS Пользователи AS $$  DECLARE  v\_winner Пользователи%ROWTYPE;  BEGIN  SELECT Пользователи.\*  INTO v\_winner  FROM Пользователи  JOIN Ставки ON Пользователи.ID\_Пользователя = Ставки.ID\_Пользователя  WHERE Ставки.ID\_аукциона = p\_id\_auction  ORDER BY Ставки.Ставка DESC  LIMIT 1;    RETURN v\_winner;  END;  $$ LANGUAGE plpgsql; |

Листинг 4.4 – Функция get\_auctuon\_winner

Если вы часто выполняете один и тот же запрос для получения данных из базы данных, создание функции позволит вам повторно использовать этот запрос в других частях вашего кода без необходимости повторного написания этого запроса. Все остальные функции и процедуры будут аналогичны, также предназначены для выборки данных из различных таблиц базы данных.

### 4.4.2 Шифрование данных

Для шифрования данных было принято решение создание функций связанных с шифрованием. В качестве примера выберем шифрование пароля, которое было реализовано с помощью функции hash\_password. Шифрование паролей - это важная мера безопасности в информационных системах, поскольку пароли являются ключом к доступу к конфиденциальным данным и аккаунтам пользователей. Предотвращение повторного использования паролей: Шифрование паролей также позволяет предотвратить повторное использование паролей в разных системах. Если пароли не зашифрованы и злоумышленник получает доступ к ним, он может использовать их для доступа к другим системам, где пользователь использует тот же пароль. В случае утечки паролей пользователей организации могут потерять доверие своих клиентов и подвергнуться репутационному ущербу. Тем самым, было решение о шифровании пароля.

В листинге 4.5 представлена функция для шифрования пароля.

|  |
| --- |
| CREATE OR REPLACE FUNCTION hash\_password(password text)  RETURNS text AS $$  BEGIN  RETURN crypt(password, gen\_salt('bf', 8));  END;  $$ LANGUAGE plpgsql; |

Листинг 4.5 – Функция hash\_password

Шифрование необходимо для безопасности данных.

### 4.4.3 Заполнение таблиц 100 000 строк

Для заполнения таблицы Пользователи была разработана функция INSERT\_User, которая вставляет 100000 строк в таблицу. Функция представлена на листинге 4.6.

|  |
| --- |
| CREATE OR REPLACE FUNCTION INSERT\_USER()  RETURNS VOID AS $$  DECLARE  I INTEGER := 1;  BEGIN  WHILE I <= 100000 LOOP  INSERT INTO Пользователи (Имя, Фамилия, Пароль, email, Роль) VALUES ('Имя ' || I, 'Фамилия ' || I, 'Пароль ' || I, 'email ' || I, 'Роль ' || I);  I := I + 1;  END LOOP;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.6 – Функция заполнения таблицы Пользователи

Функция INSERT\_USER была создана для заполнения таблицы Пользователи 100000 строками. В теле функции используется цикл WHILE, который проходит по значениям от 1 до 100000 и для каждого значения выполняет вставку новой строки в таблицу Пользователи с именемами столбцов и порядковым номером из цикла. Функция не возвращает значение и не принимает аргументов. Для выполнения функции необходимо выполнить SELECT INSERT\_User().

Вставка большого количества данных в таблицу может быть полезной для тестирования производительности базы данных и ее возможностей обработки больших объемов данных. Это может помочь определить, как быстро база данных может обрабатывать запросы при работе с большими объемами данных, и позволить выявить узкие места в системе.

### 4.4.4 Добавление данных в таблицы

Были разработаны следующие процедуры и функции для добавления новых строк в основные таблицы базы данных: add\_user, create\_auction, create\_bid, update\_lot, create\_lot, update\_user.

Все эти процедуры и функции могут вставлять, либо редактировать данные в базу данных. Заполнение данных процедурами или функциями может иметь несколько преимуществ по сравнению с заполнением вручную.

Во-первых, процедуры и функции позволяют выполнить сложные операции над данными, которые могут быть трудными или невозможными для выполнения вручную. Например, они могут использоваться для обработки больших объемов данных или для выполнения массовых изменений в базе данных.

Во-вторых, процедуры и функции могут повысить производительность приложения. Это связано с тем, что компиляция и оптимизация хранимых процедур и функций происходят один раз при создании объекта, что уменьшает время выполнения операций и уменьшает нагрузку на сервер базы данных.

В листинге 4.7 представлена процедура add\_user, для создания пользования.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_user (  "Имя" VARCHAR,  "Фамилия" VARCHAR,  "Пароль" VARCHAR,  "email" VARCHAR,  "роль" VARCHAR  ) AS $$  BEGIN  INSERT INTO "Пользователи"("Имя", "Фамилия", "Пароль", "email", "Роль")  VALUES ("Имя", "Фамилия", "Пароль", "email", "роль");  END;  $$ LANGUAGE plpgsql; |

Листинг 4.7 – Функция добавления пользователя add\_user

Остальные функции подобного формата будут являться аналогией. Важная суть, что с помощью подобного рода процедур будет производиться добавление данных в таблицы, вместо постоянного использование Insert.

### 4.4.5 Удаление данных из таблиц

Для удаления данных из базы данных были созданы процедуры: DELETE\_bid, DELETE\_lot, DELETE\_user. Ниже будут описание каждой функции или процедуры.

DELETE\_user - удаляет пользователя из таблицы User, а также удаляет все записи из таблицы Пользователи, в которых присутствует идентификатор удаляемого пользователя.

DELETE\_lot - удаляет лот, а также удаляет все записи из таблицы Лоты. DELETE\_bid - удаляет ставку из таблицы Ставки.

В листинге 4.8 будет процедура DELETE\_user.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE delete\_user(user\_id INT)  AS $$  BEGIN  DELETE FROM Пользователи WHERE ID\_Пользователя = user\_id;  END;  $$ LANGUAGE plpgsql;  drop procedure delete\_user;  CALL delete\_user(1); |

Листинг 4.8 – Процедура для удаления user

Функционал приведенных выше процедур аналогичен. Удаление таким образом, с помощью функций позволит экономить время на выполнения постоянных запросов удаления. Например, если в таблице есть связи с другими таблицами, то удаление данных из таблицы может привести к нарушению целостности данных в базе данных. Функция удаления данных может предоставлять возможность автоматического удаления связанных записей из других таблиц, чтобы избежать этой проблемы.

### 4.4.6 Дополнительные функции

Дополнительные функции в базе данных аукционной площадки будут выполнять действия, неоюхидмые под определенную ситуацию.

Функция get\_bids\_for\_lot password позволяет получить список всех ставок для конкретного лота. Данная функция позволит определить все ставки на определенный товар, это может быть полезно как администратором, так и обычным пользователям. Скрипт представлен в листинге 4.9.

|  |
| --- |
| CREATE FUNCTION get\_bids\_for\_lot(lot\_id INTEGER) RETURNS TABLE (  ID\_Ставки INTEGER,  ID\_Пользователя INTEGER,  Ставка NUMERIC(32,2),  Время\_ставки TIMESTAMP,  Статус INTEGER,  ID\_Аукциона INTEGER  ) AS $$  BEGIN  RETURN QUERY SELECT \* FROM Ставки WHERE ID\_аукциона IN (  SELECT ID\_аукциона FROM Аукционы WHERE ID\_Лота = lot\_id  );  END;  $$ LANGUAGE plpgsql; |

Листинг 4.9 – Функция get\_bids\_for\_lot

Функция get\_auctions\_for\_user получает список всех аукционов для конкретного пользователя. Это полезно для отслеживания в каких аукционах участвует определенный пользователь. Создана для контроля за действиями пользователя. Скрипт данной функции представлен в листенге 4.10.

|  |
| --- |
| CREATE FUNCTION get\_auctions\_for\_user(user\_id INTEGER) RETURNS TABLE (  ID\_Аукциона INTEGER  ) AS $$  BEGIN  RETURN QUERY SELECT ID\_аукциона FROM Аукционы WHERE ID\_Пользователя = user\_id;  END;  $$ LANGUAGE plpgsql; |

Листинг 4.10 – Функция get\_auctuions\_for\_user

Данные функции полезным в основном для остлеживания действий, которые производит на сайте пользователь, обычно используется менеджарами, которые могут передать информацию администрации для принятия мер.

## 4.5 Описание процедур экспорта и импорта

Для таблицы "Пользователи" в базе данных доступна функция экспорта и импорта данных в форматах JSON и XML. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации импорта данных в XML, был выполнен запрос, результат которого стал создание XML файла, с записанными строками из определенной страницы в XML формат. Скрипт запроса представлен в листинге 4.11.

|  |
| --- |
| COPY (SELECT xmlagg(xmlforest(Лоты.\*)) FROM Лоты) TO 'E:/file.xml'; |

Листинг 4.11 – Запрос записывающий данные из таблицы в XML.

Для обратного экспорта данные в XML формат из файла, реализован запросом, который общращаеться к XML файлу, и выбирает данные в обычные строки, это можно реализовать с помощью функции, либо по мере надобности записать данные во временную таблицу, либо таблицу в другой базе данных. Скрипт запроса представлен в листинге 4.12.

|  |
| --- |
| COPY (SELECT xmlagg(xmlforest(Лоты.\*)) FROM Лоты) TO 'E:/file.xml'; |

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах JSON и XML, что делает управление базой данных более удобным и эффективным.

## Вывод

На этом разделе были рассмотрены основные шаги установки, настройки и использования PostgreSQL. Описаны процедуры создания таблиц, ролей для управления доступом к базе данных и пакетов процедур для выполнения различных операций с данными.

Рассмотрены процедуры экспорта и импорта данных в форматах JSON и XML. Также были проведены тесты производительности базы данных, результаты которых показали высокую производительность PostgreSQL при обработке запросов на получение данных.

Из всего этого можно сделать вывод, что PostgreSQL – это мощная и надежная система управления базами данных, которая может использоваться для хранения и обработки больших объемов данных. Правильная установка и настройка PostgreSQL, а также оптимизация запросов, позволяют обеспечить высокую производительность и эффективность работы с базой данных.

5 Тестирование

## 5.1 Тестирование производительности базы данных

Одним из важных этапов разработки является тестирование производительности, которое позволяет оценить способность базы данных обрабатывать запросы и быстро возвращать результаты. Это важно для определения эффективности базы данных и выявления возможных узких мест, которые могут привести к задержкам или проблемам в работе приложения.

Для тестирования производительности базы данных была выбрана таблица Пользователи, содержащая более 100 000 строк. Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| EXPLAIN ANALYZE SELECT email FROM Пользователи WHERE email ILIKE '%email%'; |

Листинг 5.1 – Запрос к таблице Пользователи

Запрос к таблице Пользователи показывает, что его стоимость выполнения с actual time равный 0.502 говорит о том, что выполнение запроса требует значительных затрат времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составляет 502 мс. Результаты запроса будут представлены на рисунке 5.2.

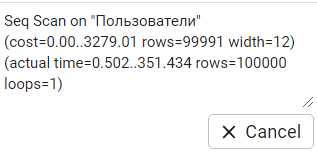


Рисунок 5.2 – Результат выполнения запроса

Для ускорения данного процесса можно создать индекс на поле email так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.3

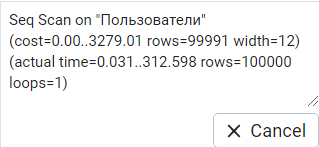


Рисунок 5.3 – Результат выполнения запроса

После создания индекса на поле email запрос к таблице Пользователи имеет стоимость выполнения с actual time равным 0.031, что является невероятным повышением производительности по сравнению с предыдущим запросом.

Тестирование показало, что создание индекса на поле email значительно улучшило производительность запроса к таблице Пользователи. Время выполнения запроса уменьшилась с 0.502 до 0.031, это говорит о том, что сканирование таблицы и применение фильтра стали занимать меньше времени и ресурсов. Поэтому создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

## 5.2 Вывод

В этом разделе было рассмотрено важное понятие тестирования производительности базы данных. Для проведения теста мы выбрали таблицу "Пользователи" с большим количеством данных. Результаты теста показали, что создание индекса на поле "email" значительно улучшило производительность запросов к таблице, сократив время выполнения запросов и уменьшив их стоимость.

Из этого можно сделать вывод, что создание индексов на полях, по которым выполняются частые запросы, имеет большое значение для повышения производительности базы данных, особенно при работе с большими объемами данных. При выборе полей для индексирования необходимо учитывать частоту их использования в запросах, чтобы получить максимальный эффект от создания индексов. Это поможет оптимизировать работу базы данных и обеспечить быстрое выполнение запросов, что важно для эффективной работы приложений.

6 Руководство по использованию программного средства

## 6.1 Руководство пользователя

Руководство пользователя является важным элементом программного обеспечения и необходимо для облегчения использования программы конечным пользователям. Его главная цель - помочь пользователям быстро и эффективно ознакомиться с программой, понять ее возможности и научиться ей пользоваться.

При входе в программу “Art\_INVEST” будет предложена красивая форма регистрация, которая выполнена в матово-сапфировом стиле, с прозрачным фоном. На ней пользователю будет предложено ввести все свои данные и зарегестрироваться. Форма регистрация представлена на рисунке 6.1.

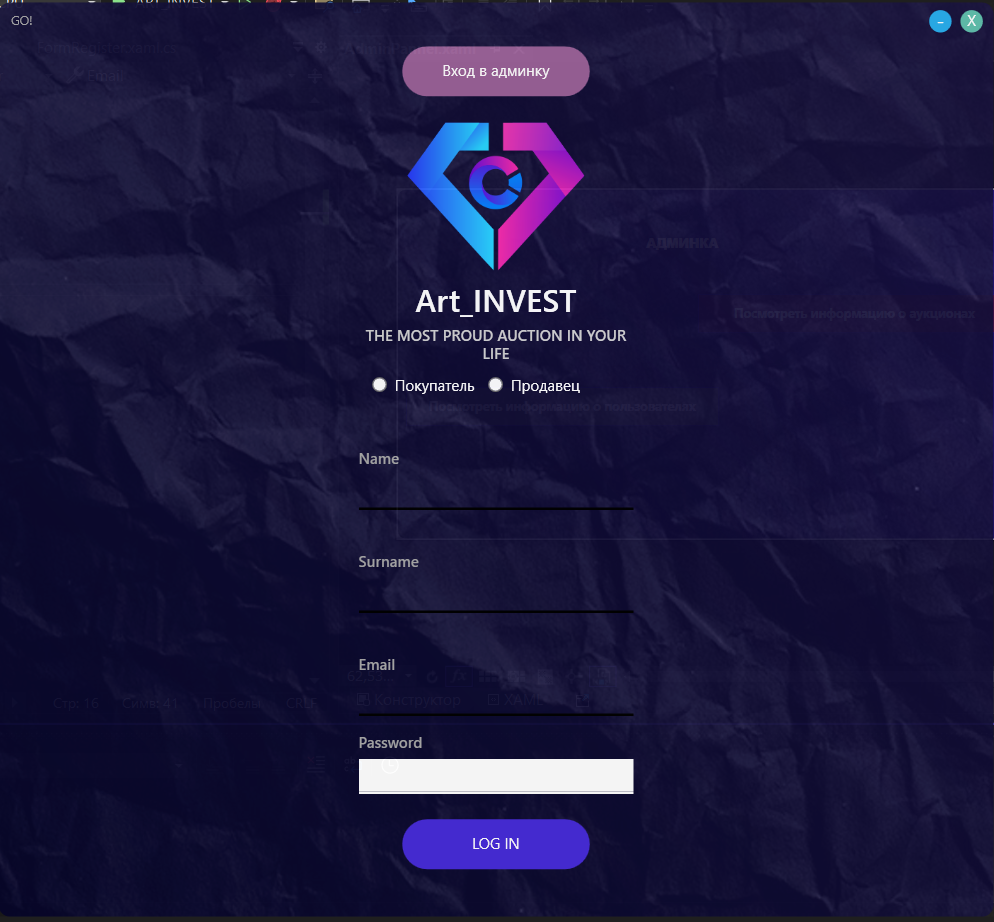


Рисунок 6.1 – Форма регистрации

Введя данные, пользователю стоит нажать на кнопку LOG IN.

Если же пользователь хочет войти как Администратор ему будет преложена отдельная кнопка, для входа в панель администратора. Кнопка расположена в верхней части формы регистрации, прямо над логотипом. Предсатвлена на рисунке 6.2.

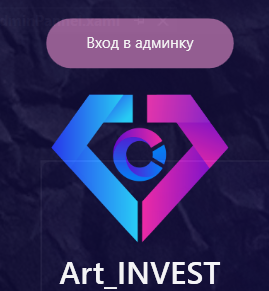


Рисунок 6.2 – Кнопка для входа в Админ Панель.

При нажатии на данную кнопку будет предоставлено окно ввода Токена, который знают исключительно администраторы. Данная форма представлена на рисунке 6.3.

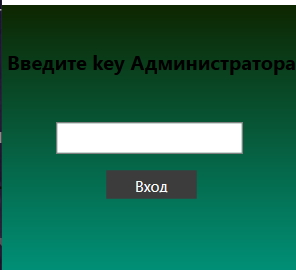


Рисунок 6.3 – Проверка на корректность администратора

Поссле ввода конкретного доступа будет доступна панель администратора, в которой можно ппросмотреть информацию о всех аукционах и пользователей. Панель администратора представлена на рисунке 6.4. Если администратор захочет просмотреть информацию о аукционах ему стоит нажать на Посмотреть информацию о аукционах, если пользователей, то Посмотреть информацию о пользователях.

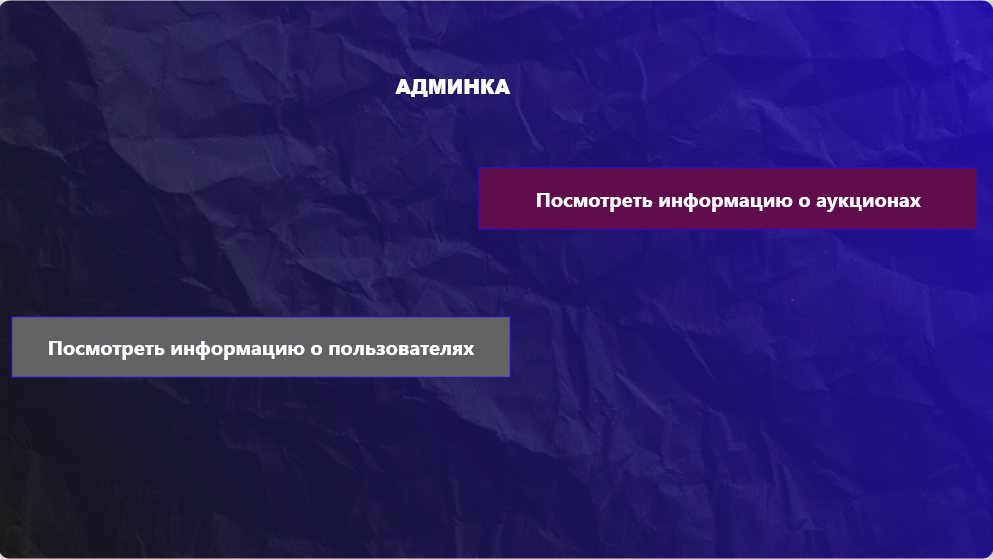


Рисунок 6.4 – Панель Администратора

После входа в основную часть аукциона пользователю станет доступны все основные функции и процедуры по использованию аукциона. К примеру, пользователь сразу может просмотреть все доступные аукционы, это делается с помощью функции get\_all\_auctions(); просмотреть ставки на него с помощью функции get\_all\_bets(). Поучаствовать в самом аукционы делая ставки, с помощью процедуры create\_bid(). Также для удобства вывода информации присутсвуют представления, они все способоствуют пользователю удобно ознокомиться с данными. Также, присутсвует удобная функций get\_active\_auctions(), которая позволяет отслеживать именно активные аукционы.

## 6.2 Вывод

В разделе руководства пользователю, где были рассмотрены различные типы пользователей, было сделано важное заключение о том, что права доступа к функционалу системы должны быть строго разграничены между пользователями с различными ролями. Было подробно описано, что может делать каждый тип пользователей и какие функции доступны для каждой роли.

# Заключение

В данном проекте была поставлена задача разработки базы данных для площадки аукциона с использованием технологии мультимедийных типов данных в СУБД PostgreSQL, а также применением дополнительной технологии резервного копирования.

В ходе выполнения проекта были использованы различные объекты, такие как таблицы, представления, процедуры, индексы триггеры, функции, специальные функции, чтобы обеспечить структурированное хранение данных и удобный доступ к ним. В итоге, поставленная цель была успешно достигнута, и на выходе получена готовая база данных.

В рамках тестирования базы данных был использован поток данных, и результаты тестирования оказались положительными. Также были разработаны процедуры импорта и экспорта данных в формате XML, что значительно облегчило управление данными и повысило удобство использования базы данных.

Разработанная база данных для площадки аукциона имеет ряд важных особенностей, которые позволяют эффективно управлять медиа-файлами и другими данными. Одной из главных особенностей является технология хранения мультимедийных данных, которая обеспечивает надежное и быстрое хранение и доступ к медиа-контенту. В результате, созданная база данных обеспечивает надежное и удобное хранение и управление данными, что является важным фактором для успешного функционирования площадки аукциона.

Делая вывод, можно сказать, что поставленные цели были выполнены. База данных для аукционной площадки выполняет необходимую функцию хранения данных.

# Список литературных источников

1.eBay [Электронный ресурс] / Режим доступа: https://by.ebay.com – Дата доступа: 28.04.2023.

2.Amazon Auction [Электронный ресурс] / Режим доступа: https://bstock.com/auctions/amazon – Дата доступа: 28.04.2023.

3. PostgreSQL Сайт с информацией [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 28.04.2023.

4.Postgrespro.com [Электронный ресурс] / Режим доступа: https://www.postgrespro.ru/ – Дата доступа: 28.04.2023.

# Приложение А

|  |
| --- |
| create table Лоты (  ID\_Лота serial primary key,  Наименование VARCHAR(255) not null,  Описание VARCHAR,  Изображение VARCHAR(255)  )tablespace Аукционы;  create table Пользователи (  ID\_Пользователя serial primary key,  Имя VARCHAR(50) not null,  Фамилия VARCHAR,  Пароль VARCHAR(50) not null,  email VARCHAR not null,  Роль VARCHAR not null  )tablespace Пользователи;  create table Ставки (  ID\_Ставки serial primary key,  ID\_Пользователя serial references Пользователи(ID\_Пользователя),  Ставка numeric (32,2),  Время\_ставки timestamp not null,  Статус integer,  ID\_аукциона serial  )tablespace Ставки;  create table Аукционы (  ID\_аукциона serial primary key,  ID\_Пользователя serial,  Дата\_Начала\_Торгов timestamp not null,  Дата\_Конца\_Торгов timestamp not null,  Статус integer,  ID\_Лота serial references Лоты(Id\_Лота),  Начальная\_цена numeric (32,2)  )tablespace Аукционы;  drop table Ставки  drop table Аукционы  drop table Пользователи  drop table Лоты  ALTER TABLE Ставки ADD CONSTRAINT fk\_ID\_аукциона FOREIGN KEY (ID\_аукциона) references Аукционы(ID\_аукциона);  ALTER TABLE Аукционы add constraint fk\_id\_лот foreign key (ID\_Лота) references Лоты (ID\_Лота)  Alter Table Аукционы add constraint fk\_id\_Пользователя foreign key(ID\_Пользователя) references Пользователи(ID\_Пользователя);  CREATE TABLESPACE Пользователи  OWNER postgres  LOCATION 'E:\STUDY\KURS\_DATABASE\Tablespace\USER'    CREATE TABLESPACE Аукционы  OWNER postgres  LOCATION 'E:\STUDY\KURS\_DATABASE\Tablespace\Bet';    CREATE TABLESPACE Ставки  OWNER postgres  LOCATION 'E:\STUDY\KURS\_DATABASE\Tablespace\AUCTION'; |

Листинг 1 – Скрипты создания таблиц

|  |
| --- |
| ----- Создание функций --------  -- Создание лота --  CREATE OR REPLACE FUNCTION create\_lot(  \_name VARCHAR(255),  \_description VARCHAR,  \_image\_path VARCHAR(255)  ) RETURNS INTEGER AS $$  DECLARE  \_lot\_id INTEGER;  BEGIN  INSERT INTO Лоты (Наименование, Описание, Изображение)  VALUES (\_name, \_description, \_image\_path)  RETURNING ID\_Лота INTO \_lot\_id;  RETURN \_lot\_id;  END;  $$ LANGUAGE plpgsql;  SELECT create\_lot('Изумруд', 'Сапфир', 'https://w.forfun.com/fetch/94/94c56e15f13f1de4740a76742b0b594f.jpeg');  --------- Функции выводов --------  --- Функция для вывода всех лотов в бд:  CREATE OR REPLACE FUNCTION get\_lots\_list() RETURNS TABLE (  lot\_id INTEGER,  name VARCHAR(255),  description VARCHAR,  image VARCHAR(255)  ) AS $$  BEGIN  RETURN QUERY SELECT ID\_Лота, Наименование, Описание, Изображение FROM Лоты;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_lots\_list();  --- Функция для вывода всех пользователей в бд:  CREATE FUNCTION get\_all\_users() RETURNS SETOF Пользователи AS $$  BEGIN  RETURN QUERY SELECT \* FROM Пользователи;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_all\_users();  --- Функция для вывода всех аукционов в бд:  CREATE FUNCTION get\_all\_auctions() RETURNS SETOF Аукционы AS $$  BEGIN  RETURN QUERY SELECT \* FROM Аукционы;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_all\_auctions();  --- Функция для вывода всех ставок в бд:  CREATE FUNCTION get\_all\_bets() RETURNS SETOF Ставки AS $$  BEGIN  RETURN QUERY SELECT \* FROM Ставки;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_all\_bets();  ------- Спец. Функции (Индивидуальные) -------  --- Функция Функция получения списка активных аукционов: данная функция позволит получить список всех активных аукционов. Аукцион считается активным, если его дата окончания еще не наступила и его статус 1.  CREATE FUNCTION get\_active\_auctions() RETURNS SETOF Аукционы AS $$  BEGIN  RETURN QUERY  SELECT \*  FROM Аукционы  WHERE Дата\_Конца\_Торгов > now() AND Статус = 1;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_active\_auctions();  --- Функция определения победителя аукциона:  CREATE FUNCTION get\_auction\_winner(  p\_id\_auction integer  ) RETURNS Пользователи AS $$  DECLARE  v\_winner Пользователи%ROWTYPE;  BEGIN  SELECT Пользователи.\*  INTO v\_winner  FROM Пользователи  JOIN Ставки ON Пользователи.ID\_Пользователя = Ставки.ID\_Пользователя  WHERE Ставки.ID\_аукциона = p\_id\_auction  ORDER BY Ставки.Ставка DESC  LIMIT 1;    RETURN v\_winner;  END;  $$ LANGUAGE plpgsql;  SELECT get\_auction\_winner(2);  --- Функция обновления статуса аукциона:  CREATE FUNCTION update\_auction\_status(  p\_id\_auction integer,  p\_new\_status integer  ) RETURNS VOID AS $$  BEGIN  UPDATE Аукционы  SET Статус = p\_new\_status  WHERE ID\_аукциона = p\_id\_auction;  END;  $$ LANGUAGE plpgsql;  SELECT update\_auction\_status(1, 1);  --- Получение максимальной ставки на аукционе  CREATE FUNCTION get\_max\_bid\_for\_auction(auction\_id INTEGER) RETURNS NUMERIC(32,2) AS $$  DECLARE  max\_bid NUMERIC(32,2);  BEGIN  SELECT MAX(Ставка) INTO max\_bid FROM Ставки WHERE ID\_аукциона = auction\_id;  RETURN max\_bid;  END;  $$ LANGUAGE plpgsql;  SELECT get\_max\_bid\_for\_auction(2);  --- Получение минимальной ставки на аукционе  CREATE FUNCTION get\_min\_bid\_for\_auction(auction\_id INTEGER) RETURNS NUMERIC(32,2) AS $$  DECLARE  min\_bid NUMERIC(32,2);  BEGIN  SELECT MIN(Ставка) INTO min\_bid FROM Ставки WHERE ID\_аукциона = auction\_id;  RETURN min\_bid;  END;  $$ LANGUAGE plpgsql;  SELECT get\_min\_bid\_for\_auction(2);  --- Функция получения списка всех ставок для конкретного лота  CREATE FUNCTION get\_bids\_for\_lot(lot\_id INTEGER) RETURNS TABLE (  ID\_Ставки INTEGER,  ID\_Пользователя INTEGER,  Ставка NUMERIC(32,2),  Время\_ставки TIMESTAMP,  Статус INTEGER,  ID\_Аукциона INTEGER  ) AS $$  BEGIN  RETURN QUERY SELECT \* FROM Ставки WHERE ID\_аукциона IN (  SELECT ID\_аукциона FROM Аукционы WHERE ID\_Лота = lot\_id  );  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_bids\_for\_lot(1);  --- Функция получения списка всех аукционов для конкретного пользователя:  CREATE FUNCTION get\_auctions\_for\_user(user\_id INTEGER) RETURNS TABLE (  ID\_Аукциона INTEGER  ) AS $$  BEGIN  RETURN QUERY SELECT ID\_аукциона FROM Аукционы WHERE ID\_Пользователя = user\_id;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_auctions\_for\_user(1);  --- Функция повышения ставки  CREATE FUNCTION increase\_bid(auction\_id INTEGER, user\_id INTEGER, new\_bid NUMERIC(32,2)) RETURNS VOID AS $$  BEGIN  UPDATE Ставки SET Ставка = Ставка + new\_bid, Время\_ставки = NOW()  WHERE ID\_аукциона = auction\_id AND ID\_Пользователя = user\_id;  END;  $$ LANGUAGE plpgsql;  SELECT increase\_bid(2, 1, 50.00);  select \* from Ставки  --- Функция шифрования пароля  select \* from Пользователи;  CREATE OR REPLACE FUNCTION hash\_password(password text)  RETURNS text AS $$  BEGIN  RETURN crypt(password, gen\_salt('bf', 8));  END;  $$ LANGUAGE plpgsql;  call add\_user ('Кирилл', 'Кузнецов', 'kira123', 'Kira123@mail.ru', 'Покупатель'); |

Листинг 2 – Скрипты создания функций

|  |
| --- |
| -- Для таблицы Лоты:  CREATE INDEX idx\_наименование ON Лоты (Наименование);  CREATE INDEX idx\_описание ON Лоты (Описание);  CREATE INDEX idx\_изображение ON Лоты (Изображение);  -- Для таблицы Пользователи:  CREATE INDEX idx\_имя ON Пользователи (Имя);  CREATE INDEX idx\_фамилия ON Пользователи (Фамилия);  CREATE INDEX idx\_email ON Пользователи (email);  CREATE INDEX idx\_роль ON Пользователи (Роль);  -- Для таблицы Ставки:  CREATE INDEX idx\_id\_пользователя ON Ставки (ID\_Пользователя);  CREATE INDEX idx\_статус ON Ставки (Статус);  CREATE INDEX idx\_id\_аукциона ON Ставки (ID\_аукциона);  -- Для таблицы Аукционы:  CREATE INDEX idx\_id\_лота ON Аукционы (ID\_Лота);  CREATE INDEX idx\_начальная\_цена ON Аукционы (Начальная\_цена);  CREATE INDEX idx\_id\_пользователя ON Аукционы (ID\_Пользователя);  CREATE INDEX idx\_статус ON Аукционы (Статус); |

Листинг 3 – Скрипты создания индексов

|  |
| --- |
| -------- Процедуры с пользователями --------  -- Создание пользователей  CREATE OR REPLACE PROCEDURE Add\_user (  "Имя" VARCHAR,  "Фамилия" VARCHAR,  "Пароль" VARCHAR,  "email" VARCHAR,  "роль" VARCHAR  ) AS $$  BEGIN  INSERT INTO "Пользователи"("Имя", "Фамилия", "Пароль", "email", "Роль")  VALUES ("Имя", "Фамилия", "Пароль", "email", "роль");  END;  $$ LANGUAGE plpgsql;  drop procedure Add\_user;  call Add\_user ('Gena', 'Lehov', '123123123123', 'Gnom123@mail.ru', 'Покупатель');  select \* from Пользователи;  -- Удаление пользователей пользователей  CREATE OR REPLACE PROCEDURE delete\_user(user\_id INT)  AS $$  BEGIN  DELETE FROM Пользователи WHERE ID\_Пользователя = user\_id;  END;  $$ LANGUAGE plpgsql;  drop procedure delete\_user;  CALL delete\_user(1);  -- Редактирование пользователя  CREATE OR REPLACE PROCEDURE update\_user(  user\_id INT,  new\_name VARCHAR(50),  new\_last\_name VARCHAR,  new\_password VARCHAR(50),  new\_email VARCHAR,  new\_role VARCHAR  )  AS $$  BEGIN  UPDATE Пользователи  SET  Имя = new\_name,  Фамилия = new\_last\_name,  Пароль = new\_password,  email = new\_email,  Роль = new\_role  WHERE  ID\_Пользователя = user\_id;  END;  $$ LANGUAGE plpgsql;  CALL update\_user(2, 'Павел', 'Павлов', '123', 'pavel123@mail.ru', 'Продавец');  select \* from Пользователи;  -------- Процедуры с аукционами --------  -- Создание аукциона  CREATE OR REPLACE PROCEDURE create\_auction(  lot\_id INT,  user\_id INT,  start\_time TIMESTAMP,  end\_time TIMESTAMP,  start\_price NUMERIC(32,2),  status INT  )  AS $$  BEGIN  INSERT INTO Аукционы (ID\_Лота, ID\_Пользователя, Дата\_Начала\_Торгов, Дата\_Конца\_Торгов, Начальная\_цена, Статус)  VALUES (lot\_id, user\_id, start\_time, end\_time, start\_price, status);  END;  $$ LANGUAGE plpgsql;  CALL create\_auction(2, 1, '2023-05-01 18:00:00', '2023-05-03 24:00:00', 8000.00,0 );  select \* from Аукционы;  select \* from Лоты;  -- Редактирования аукциона  CREATE OR REPLACE PROCEDURE edit\_auction(  \_id INT,  \_lot\_id INT,  \_user\_id INT,  \_start\_time TIMESTAMP,  \_end\_time TIMESTAMP,  \_start\_price NUMERIC(32,2),  \_status INT  )  AS $$  BEGIN  UPDATE Аукционы  SET ID\_Лота = \_lot\_id,  ID\_Пользователя = \_user\_id,  Дата\_Начала\_Торгов = \_start\_time,  Дата\_Конца\_Торгов = \_end\_time,  Начальная\_цена = \_start\_price,  Статус = \_status  WHERE ID\_аукциона = \_id;  END;  $$ LANGUAGE plpgsql;  CALL edit\_auction(1, 1, 1, '2023-05-01 00:00:00', '2023-05-07 00:00:00', 300.00, 1);  select \* from Аукционы;  -- Удаление аукциона  CREATE OR REPLACE PROCEDURE delete\_auction(auction\_id INT)  AS $$  BEGIN  DELETE FROM Ставки WHERE ID\_аукциона = auction\_id;  DELETE FROM Аукционы WHERE ID\_аукциона = auction\_id;  END;  $$ LANGUAGE plpgsql;  call delete\_auction (1);  ------- Процедуры с лотами --------  -- Cозданиие лота в первой функции.  -- Редактирование лота  CREATE OR REPLACE PROCEDURE update\_lot(  lot\_id INTEGER,  name VARCHAR(255),  description VARCHAR,  image VARCHAR(255)  )  AS $$  BEGIN  UPDATE Лоты  SET Наименование = name, Описание = description, Изображение = image  WHERE ID\_Лота = lot\_id;  END;  $$ LANGUAGE plpgsql;  CALL update\_lot(1, 'Hennesy', 'Выдержка 30 лет', 'https://w.forfun.com/fetch/94/94c56e15f13f1de4740a76742b0b594f.jpeg');  select \* from Лоты;  -- Удаление лота  CREATE OR REPLACE PROCEDURE delete\_lot(  lot\_id INT  )  AS $$  BEGIN  DELETE FROM Лоты WHERE ID\_Лота = lot\_id;  END;  $$ LANGUAGE plpgsql;  CALL delete\_lot(1);  ------- Процедуры со ставками --------  -- Создание ставки  CREATE OR REPLACE PROCEDURE create\_bid(  user\_id INTEGER,  bid\_amount NUMERIC(32,2),  bid\_time TIMESTAMP WITH TIME ZONE,  auction\_id INTEGER,  status INTEGER  ) LANGUAGE plpgsql AS $$  BEGIN  INSERT INTO Ставки (ID\_Пользователя, Ставка, Время\_ставки, Статус, ID\_аукциона)  VALUES (user\_id, bid\_amount, bid\_time, status, auction\_id);  END;  $$;  CALL create\_bid(1, 100.00, CURRENT\_TIMESTAMP, 2, 1);  select \* from Ставки;  -- Удаление ставки  CREATE OR REPLACE PROCEDURE delete\_bid(  bid\_id INTEGER  ) LANGUAGE plpgsql AS $$  BEGIN  DELETE FROM Ставки WHERE ID\_Ставки = bid\_id;  END;  $$;  CALL delete\_bid(1); |

Листинг 4 – Скрипты создание процедур

|  |
| --- |
| -- Триггер на проверку дату в аукционе.  CREATE OR REPLACE FUNCTION check\_auction\_dates() RETURNS TRIGGER AS $$  BEGIN  IF NEW.Дата\_Начала\_Торгов > NEW.Дата\_Конца\_Торгов THEN  RAISE EXCEPTION 'Дата начала торгов не может быть больше даты окончания торгов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_auction\_dates\_trigger  BEFORE INSERT OR UPDATE ON Аукционы  FOR EACH ROW  EXECUTE FUNCTION check\_auction\_dates();  -- Проверка, что ставка не меньше максимальной ставки для аукциона.  CREATE OR REPLACE FUNCTION check\_bid() RETURNS TRIGGER AS $$  DECLARE  max\_bid numeric;  BEGIN  SELECT MAX(Ставка) INTO max\_bid  FROM Ставки  WHERE ID\_аукциона = NEW.ID\_аукциона;  IF max\_bid IS NULL OR NEW.Ставка >= max\_bid THEN  RETURN NEW;  ELSE  RAISE EXCEPTION 'Ставка должна быть не меньше предыдущей максимальной ставки';  END IF;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_bid\_trigger  BEFORE INSERT ON Ставки  FOR EACH ROW  EXECUTE FUNCTION check\_bid();  -- Проверка на то, что нельзя делать ставки на закрытом аукционе.  CREATE OR REPLACE FUNCTION check\_auction\_status() RETURNS TRIGGER AS $$  BEGIN  IF EXISTS (SELECT 1 FROM Аукционы WHERE ID\_аукциона = NEW.ID\_аукциона AND Статус = 0) THEN  RAISE EXCEPTION 'Ставки на закрытом аукционе запрещены!';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_auction\_status\_trigger  BEFORE INSERT ON Ставки  FOR EACH ROW  EXECUTE FUNCTION check\_auction\_status(); |

Листинг 5 – Скрипты создания триггеров

|  |
| --- |
| -- Представление, которое показывает количество лотов, выставленных на аукционе:  CREATE VIEW num\_lots\_on\_auction AS  SELECT COUNT(\*) AS num\_lots, ID\_аукциона  FROM Аукционы  GROUP BY ID\_аукциона;  select \* from num\_lots\_on\_auction  select \* from Пользователи  -- Представление, которое показывает информацию о последней ставке для каждого лота:  CREATE VIEW last\_bid\_for\_lot AS  SELECT Л.ID\_Лота, Л.Наименование, S.Ставка, S.Время\_ставки, S.ID\_Пользователя  FROM Лоты Л  LEFT JOIN Ставки S ON Л.ID\_Лота = S.ID\_аукциона  WHERE S.ID\_Ставки = (SELECT MAX(ID\_Ставки) FROM Ставки WHERE ID\_аукциона = Л.ID\_Лота);  select \* from last\_bid\_for\_lot;  -- Представление, которое показывает количество ставок, сделанных каждым пользователем:  CREATE VIEW num\_bids\_per\_user AS  SELECT COUNT(\*) AS num\_bids, ID\_Пользователя  FROM Ставки  GROUP BY ID\_Пользователя;  select \* from num\_bids\_per\_user;  -- Представление, которое показывает список пользователей с их ролями и количеством сделанных ими ставок:  CREATE VIEW user\_role\_and\_num\_bids AS  SELECT P.ID\_Пользователя, P.Имя, P.Фамилия, P.Роль, COUNT(\*) AS num\_bids  FROM Пользователи P  JOIN Ставки S ON P.ID\_Пользователя = S.ID\_Пользователя  GROUP BY P.ID\_Пользователя, P.Имя, P.Фамилия, P.Роль;  select \* from user\_role\_and\_num\_bids;---------- |

Листинг 6 – Скрипты создания представлений