МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных аукциона с применением технологии резервного копирования и восстановления»

Выполнил студент Трубач Дмитрий Сергеевич

(Ф.И.О.)

Руководитель работы асс. Нистюк Ольга Александровна

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

И.о. зав. кафедрой ст. преп. Блинова Е.А.

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

Курсовая работа защищена с оценкой

Минск 2023

Содержание

[Введение 3](#_Toc153984086)

[1 Постановка задачи 4](#_Toc153984087)

[2 Проектирование базы данных 5](#_Toc153984088)

[2.1 Определение вариантов использования 5](#_Toc153984089)

[2.2 Диаграммы UML, взаимодействие всех компонентов 6](#_Toc153984090)

[2.3 Вывод 6](#_Toc153984091)

[3 Разработка объектов базы данных 7](#_Toc153984092)

[3.1 Разработка таблиц базы данных 7](#_Toc153984093)

[3.2 Разработка представлений базы данных 9](#_Toc153984094)

[3.3 Разработка индексов базы данных 10](#_Toc153984095)

[3.4 Разработка функций базы данных 10](#_Toc153984096)

[3.5 Разработка процедур базы данных 11](#_Toc153984097)

[3.6 Разработка триггеров базы данных 12](#_Toc153984098)

[3.7 Разработка пользователей 13](#_Toc153984099)

[3.8 Вывод 13](#_Toc153984100)

[4 Описание процедур импорта и экспорта 14](#_Toc153984101)

[5 Тестирование производительности 15](#_Toc153984102)

[6 Описание технологии и ее применение в базе данных 17](#_Toc153984103)

[7 Руководство пользователя 18](#_Toc153984104)

[Заключение 19](#_Toc153984105)

[Список используемых источников 20](#_Toc153984106)

[Приложение А 21](#_Toc153984107)

[Приложение Б 22](#_Toc153984108)

[Приложение В 27](#_Toc153984109)

[Приложение Г 29](#_Toc153984110)

[Приложение Д 30](#_Toc153984111)

# 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 резервного копирования базы данных, а также технология мультимедийных форматов данных при сохранении картинок.

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

# 1 Постановка задачи

Важным этапом разработки любого нового продукта является анализ существующих на рынке аналогов. Он позволяет определить, какие решения уже представлены потребителям, выявить их сильные и слабые стороны, а также определиться с направлениями для усовершенствований.

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.

На текущий момент рынок онлайн-аукционов представляет собой множество платформ с различным уровнем функционала и безопасности. Однако, многие из них не могут предложить полноценного комплекса инструментов, сочетающего в себе удобство, гибкость и высокие стандарты безопасности. Это оставляет простор для новых разработок в этой сфере.

Задача проекта: разработать архитектуру приложения, взаимодействие с которой будет понятно любому пользователю. Построить базу данных и выполнить тестирование готового продукта.

Должны быть выполнены следующие требования:

˗ база данных должна быть реализована в СУБД PostgreSQL.

˗ доступ к данным должен осуществляться только через соответствующие процедуры;

˗ должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON;

˗ необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице;

˗ применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.

# 2 Проектирование базы данных

## 2.1 Определение вариантов использования

Главное начало функциональных требований базы данных заключается в определении способа обработки данных и обеспечении необходимой функциональности пользовательскому интерфейсу. Важно описать, как данные должны быть хранены и организованы, как осуществляется поиск и выборка данных, как обновляются данные и как обеспечивается безопасность информации. Кроме того, следует учесть взаимодействие 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.

Кроме 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 2.1 представлена UML диаграмма вариантов использования.

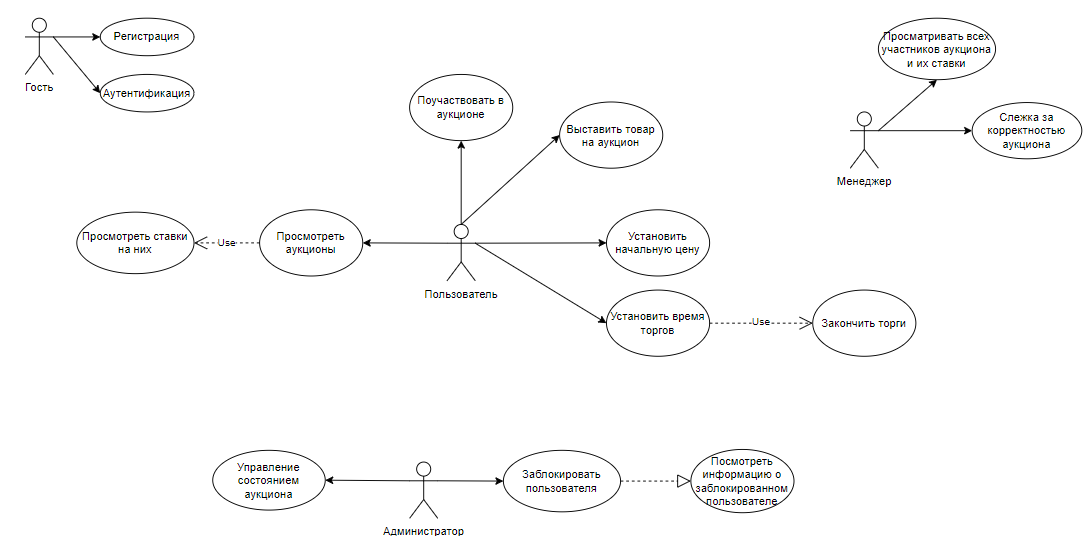


Рисунок 2.1 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. 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.

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

## 2.2 Диаграммы UML, взаимодействие 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 представлена на рисунке 2.2.

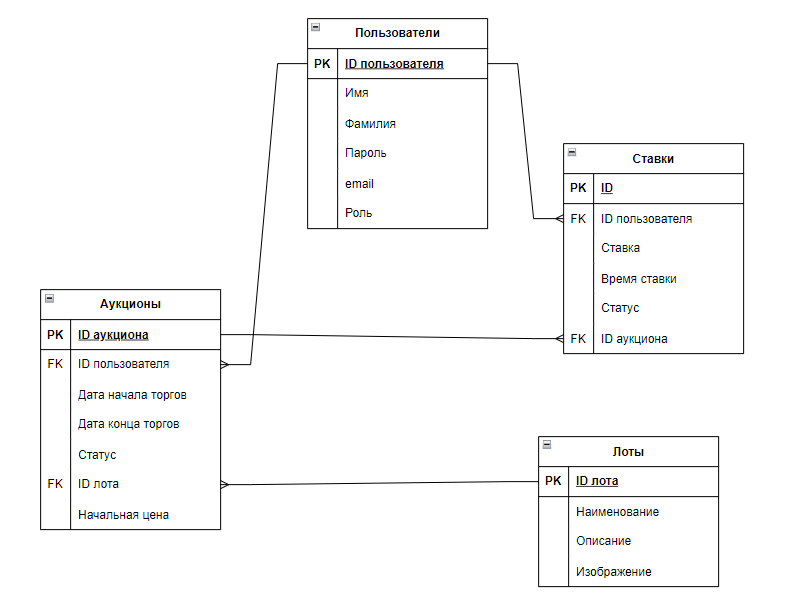


Рисунок 2.2 – Диаграмма базы данных

Видно, что диаграмма показывает связи между таблицами и полями, а также отношения между ними. 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.

Например, таблица Ставки связана с таблицами Пользователями, через внешний ключ ID\_Пользователя. Также видно, что таблицы Лоты и Аукционы связаны друг с другом через внешний и первичный ключ ID лота. Все связи представлены на рисунке 2.2, который располагается выше.

## 2.3 Вывод

Такая архитектура базы данных предоставляет гибкость в управлении данными, высокую производительность и безопасность. Структурированное разделение на авторизацию и доменную логику позволяет оптимально использовать ресурсы системы и обеспечивать эффективное взаимодействие с данными.

# 3 Разработка объектов базы данных

## 3.1 Разработка таблиц базы данных

При разработке приложения для курсового проекта была использована база данных PostgreSQL.

Для реализации базы данных было разработано 4 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Пользователи, Лоты, Аукционы и Ставки. Ниже будет описание про каждую из них более подробно.

Таблица Пользователи содержит информацию о пользователях. Состоит из следующих столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы Пользователи

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Пользователя | Уникальный идентификатор пользователя, первичный ключ | SERIAL |
| Имя | Имя пользователя | VARCHAR(50) |
| Фамилия | Фамилия пользователя | VARCHAR |
| Пароль | Пароль пользователя | VARCHAR(50) |
| email | Электронная почта пользователя | VARCHAR |
| Роль | Роль пользователя в системе | VARCHAR |

Таблица Лоты содержит информацию о предметах, выставленных на аукцион. В таблице Аукцион будет отображена цена на данный лот. Состоит из следующих столбцов (таблица 3.2):

Таблица 3.2 – Столбцы таблицы Лоты

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Лота | Уникальный идентификатор лота, первичный ключ | SERIAL |
| Наименование | Имя лота | VARCHAR(255) |
| Описание | Описание лота | VARCHAR |
| Изображения | Изображение товара | VARCHAR(255) |

Таблица Аукционы хранит информацию о текущих аукционах, содержит следующие столбцы (таблица 3.3):

Таблица 3.3 – Столбцы таблицы Аукционы

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_аукциона | Уникальный идентификатор лота, первичный ключ | SERIAL |
| ID\_Пользователя | Имя лота | VARCHAR(255) |
| Дата\_Начала\_Торгов | Дата начала торгов за лот | VARCHAR |
| Дата\_Конца\_Торгов | Дата конца торгов за лот | VARCHAR(255) |
| Статус | Статус аукциона | INTEGER |
| ID\_Лота | Уникальный идентификатор лота, вторичный ключ | SERIAL |

Продолжение таблицы 3.3

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Начальная цена | Начальная цена при которой можно поучаствовать в аукционе | NUMERIC(32,2) |

Таблица Ставки содержит данные о ставках, принятых на лоты. Содержит в себе такие столбцы, такие как (таблица 3.4):

Таблица 3.4 – Столбцы таблицы Ставки

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Ставки | Уникальный идентификатор ставки, первичный ключ | SERIAL |
| ID\_Пользователя | Идентификатор пользователя, который будет делать ставку | SERIAL |
| Ставка | Числовое значение ставки | NUMERIC(32,2) |
| Время\_ставки | Время, когда была произведена ставка | TIMESTAMP |
| Статус | Статус ставки | INTEGER |
| ID\_аукциона | Идентификатор аукциона, на который будет сделана ставка | SERIAL |

Перед 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 3.1.

|  |
| --- |
| CREATE TABLESPACE Пользователи  OWNER postgres  LOCATION 'E:\TradeX\Tablespaces\Users';  CREATE TABLESPACE Аукционы  OWNER postgres  LOCATION 'E:\TradeX\Tablespaces\Auctions';  CREATE TABLESPACE Ставки  OWNER postgres  LOCATION 'E:\TradeX\Tablespaces\Bets'; |

Листинг 3.1 – Cкрипт для создания табличных пространств

В табличном пространстве Пользователи будут храниться данные из таблицы Пользователи. Тут собраны данные связанные напрямую с пользователем. В табличном пространстве Аукционы будут хранится данные из таблиц Аукционы и Лоты. В табличном пространстве Ставки будут хранится данные из таблицы Лоты.

Листинг SQL-кода для создания таблиц находится в [приложении А](#_Приложение_А).

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.

## 3.2 Разработка представлений 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\_bid\_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 представлено в листинге 3.2. Остальные представления будут аналогичны, работая с другими данными.

|  |
| --- |
| -- 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;  -- 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; |

Листинг 3.2 – Представления num\_lots\_on\_auction и last\_bid\_for\_lot

Представление num\_lots\_on\_auction было создано для того, чтобы показать количество лотов, выставленных на аукционе. Оно объединяет данные из таблицы Аукционы и использует агрегатную функцию COUNT(\*), чтобы подсчитать количество лотов для каждого ID\_аукциона.

Представление last\_bid\_for\_lot было создано для того, чтобы показать информацию о последней ставке для каждого лота. Оно объединяет данные из таблицы Лоты и Ставки, используя оператор LEFT JOIN, чтобы получить все лоты, даже если для них не было сделано ставок. Затем представление выбирает последнюю ставку для каждого лота, используя подзапрос (SELECT MAX(ID\_Ставки) FROM Ставки WHERE ID\_аукциона = Л.ID\_Лота).

## 3.3 Разработка индексов 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.

Например, для того, чтобы быстро находить лоты по их параметрам, были созданы три индекса: idx\_наименование по полю Наименование, idx\_описание по полю Описание и idx\_изображение по полю Изображение. Создание индексов представлено в листинге 3.3.

|  |
| --- |
| CREATE INDEX idx\_наименование ON Лоты (Наименование);  CREATE INDEX idx\_описание ON Лоты (Описание);  CREATE INDEX idx\_изображение ON Лоты (Изображение); |

Листинг 3.3 – Индексы таблицы Лоты

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

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.

## 3.4 Разработка функций базы данных

Для управления данными через приложение все пользователи использует функции. Функция – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Стоит отметить, что функции не могут содержать группы операторов DML и DDL. В моём приложении реализовано 19 функций. Пример создания функции представлен на листинге 3.4.

|  |
| --- |
| CREATE OR REPLACE FUNCTION create\_lot(  \_name VARCHAR(255),  \_description VARCHAR,  \_image\_path VARCHAR(255)  ) RETURN INTEGER AS $$  DECLARE  lot\_id INTEGER;  BEGIN  IF \_name IS NULL OR \_name = '' OR \_description IS NULL OR \_description = '' THEN  RAISE EXCEPTION 'Наименование и Описание не могут быть NULL или пустыми строками';  END IF;  IF \_image\_path !~\* '(https?://)?([\\da-z.-]+)\\.([a-z.]{2,6})[/\\w .-]\*/?' THEN  RAISE EXCEPTION 'Поле Изображение должно быть ссылкой';  END IF;  BEGIN  INSERT INTO Лоты (Наименование, Описание, Изображение)  VALUES (\_name, \_description, \_image\_path)  RETURNING ID\_Лота INTO \_lot\_id;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN NULL;  END;  RETURN \_lot\_id;  END;  $$ LANGUAGE plpgsql; |

Листинг 3.4 – Функция создания лота

Внутри функции происходит операция вставки (INSERT) новой записи в таблицу Лоты с использованием переданных параметров \_name, \_description и \_image\_path. Значение идентификатора (ID) нового лота сохраняется в переменную \_lot\_id с помощью ключевого слова RETURNING. Затем функция возвращает значение \_lot\_id, которое является идентификатором (ID) созданного лота.

Взаимодействие с аукционами реализовано с помощью функций, код приведен в [приложении Б](#_Приложение_Б).

## 3.5 Разработка процедур базы данных

По большому счёту, процедуры выполняют аналогичные действия с функциями, за тем исключением, что они не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Всего было разработано мною 10 процедур. На листинге 3.5 представлен пример реализации одной из процедур.

|  |
| --- |
| 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 3.5 – Процедура создания пользователя

Данная процедура создает пользователя, в качестве параметров принимает имя, фамилию, пароль, email и роль. Скрипт других процедур, которые связаны с пользователем, приведен в [приложении В](#_Приложение_В).

## 3.6 Разработка триггеров 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.

Всего было сделано 4 триггера, в листинге 3.6 приведен код одного из триггеров.

|  |
| --- |
| 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 3.6 – Скрипт триггера check\_auction\_dates

Данный триггер создается в таблице Аукционы, который проверяет дату в аукционе, а именно проверка на то, что дата начала торгов не может быть больше даты окончания торгов.

## 3.7 Разработка пользователей

Пользователь базы данных — это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

При проектировании базы данных было создано 4 типа пользователей: гость, пользователь, модератор и администратор.

Создание пользователей представлено в листинге 3.7.

|  |
| --- |
| CREATE USER admin\_user WITH PASSWORD '111';  ALTER ROLE admin\_user SET ROLE Администратор;  CREATE USER regular\_user WITH PASSWORD '111';  ALTER ROLE regular\_user SET ROLE Пользователь;  CREATE USER manager\_user WITH PASSWORD '111';  ALTER ROLE manager\_user SET ROLE Менеджер;  CREATE USER guest\_user WITH PASSWORD '111';  ALTER ROLE guest\_user SET ROLE Гость; |

Листинг 3.7 – Создание пользователей

Далее для созданного пользователя базы данных были выданы необходимые привилегии необходимые для функционала клиента.

## 3.8 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для площадки аукциона. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря хорошо спроектированным объектам базы данных система способна обеспечивать надежность, своевременное быстродействие и целостность данных.

# 4 Описание процедур импорта и экспорта

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

Для реализации экспорта данных в JSON, была разработана функция, результатом которой стало создание JSON файла, с записанными строками из определенной таблицы в JSON формат. Скрипт запроса представлен в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE FUNCTION EXPORT\_LOTS\_TO\_JSON\_FILE(FILE\_PATH TEXT)  RETURNS VOID AS  $$  DECLARE  JSON\_DATA JSON;  BEGIN  BEGIN  SELECT JSON\_AGG(ROW\_TO\_JSON(Лоты)) INTO JSON\_DATA FROM Лоты;  PERFORM PG\_FILE\_WRITE(FILE\_PATH, JSON\_DATA::TEXT,true);  EXCEPTION WHEN OTHERS THEN  RAISE 'Произошла ошибка: %', SQLERRM;  END;  END;  $$  LANGUAGE PLPGSQL; |

Листинг 4.1 – Функция EXPORT\_LOTS\_TO\_JSON\_FILE

Для обратного экспорта данные в JSON формат из файла, реализован запросом, который обращается к JSON 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 [приложении Г](#_Приложение_Г).

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

# 5 Тестирование производительности

Одной из ключевых задач в процессе разработки является тестирование производительности базы данных. Такое тестирование позволяет оценить способность базы данных обрабатывать запросы и возвращать результаты с высокой скоростью. Оценка производительности важна для определения эффективности базы данных и выявления возможных узких мест, которые могут вызывать задержки или проблемы в работе приложения.

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для данной задачи мы создали анонимный блок, и так как работаем именно с действующей базой данных, то можем использовать готовую процедуру для вставки данных в таблицу. Разработанный анонимный блок позволяет добавить большое количество строк за одно выполнение (листинг 5.1).

|  |
| --- |
| CREATE OR REPLACE FUNCTION INSERT\_LOTS() RETURNS VOID AS $$  BEGIN FOR i IN 1..100000 LOOP INSERT INTO Лоты (Наименование, Описание, Изображение)  VALUES ('Наименование' || i, 'Описание' || i, 'Изображение' || i  );  END LOOP;  END;  $$ LANGUAGE plpgsql; |

Листинг 5.1 – Заполнение таблицы 100000 строк

Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| EXPLAIN ANALYZE SELECT Наименование FROM Лоты WHERE Наименование = 'Наименование 1024'; |

Листинг 5.2 – Запрос к таблице Лоты

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составило 38.366 мс, а время планирования – 0.506 мс. Результаты запроса будут представлены на рисунке 5.1.

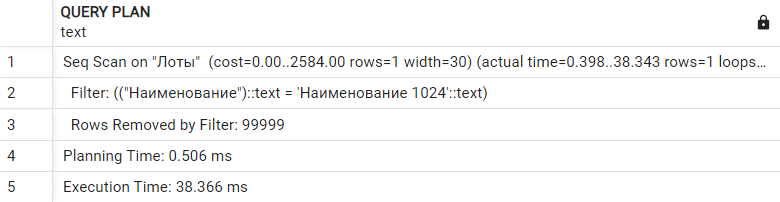


Рисунок 5.1 – Результат выполнения запроса

Для ускорения данного процесса создадим индекс на поле Наименование 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 5.2.

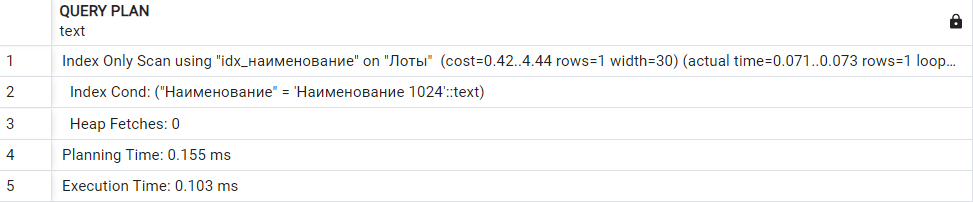


Рисунок 5.3 – Результат выполнения запроса

После создания индексов видно, что лоты стали генерировать намного быстрее. Время планирования заняло 0.155 мс, а время выполнения 0.103 мс, что более чем в 300 раз быстрее.

Результаты тестирования говорят о том, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

# 6 Описание технологии и ее применение в базе данных

Резервное копирование и восстановление в базах данных представляет собой систему методов и инструментов, предназначенных для сохранения и восстановления данных в случае их потери или повреждения. В данной базе данных используется технология резервного копирования для обеспечения надежности и безопасности хранения информации.

На рисунке 6.1 иллюстрируется процесс резервного копирования данных в базе.

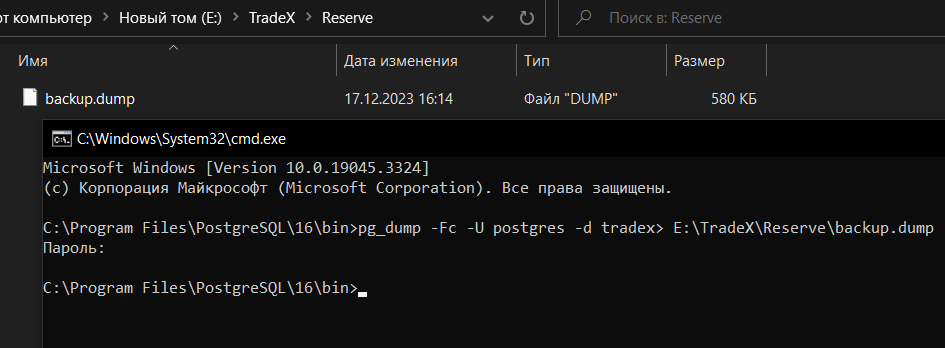


Рисунок 6.1 – Процесс резервного копирования данных

Применение резервного копирования и восстановления обладает несколькими ключевыми преимуществами:

* 1. Сохранение целостности данных. Регулярные резервные копии помогают предотвратить потерю информации в случае аварийных ситуаций, таких как сбои оборудования или программного обеспечения.
  2. Обеспечение безопасности. Резервные копии данных могут служить важным инструментом в защите от угроз безопасности, таких как вирусы или злонамеренные атаки, позволяя быстро восстановить информацию после инцидента.
  3. Восстановление в случае необходимости. Благодаря наличию резервных копий базы данных возможно оперативное восстановление данных до момента их утраты или повреждения.
  4. Эффективное управление данными. Возможность регулярного создания резервных копий и их последующего использования дает возможность контролировать и управлять данными с большей уверенностью.

Технология резервного копирования позволяет обеспечить сохранность и доступность данных, играя ключевую роль в обеспечении непрерывности работы базы данных и предоставлении средств для восстановления информации в случае необходимости.

# 7 Руководство пользователя

При входе на аукционную площадку, мы будем иметь статус гостя, это значит – ограниченный функционал. Мы сможем лишь просмотреть, какие лоты доступны на аукционе в данный момент (представление num\_lots\_on\_auction), поиск лотов (функция get\_lots\_list) и последняя ставка, которая произведена на лот (представление last\_bid\_for\_lot). Главная функция у гостя – регистрация (процедура Add\_user) и авторизация (функция authenticate\_user).

После регистрации и последующей авторизации, пользователь уже имеет одноименный статус – Пользователь. У него открываются новые возможности, он может создавать лоты для последующих аукционов на них (функция create\_lot), также он имеет возможность участвовать в других аукционах (процедура create\_bid, фунцкия increase\_bid), посмотреть на статистику аукциона (функции get\_active\_auctions, get\_max\_bid\_for\_auction, get\_max\_bid\_for\_auction, get\_bids\_for\_lot, get\_auctions\_for\_user, get\_users\_with\_won\_lots).

Если вход на площадку был произведен с аккаунта менеджера, то мы сможем просматривать и редактировать все лоты и аукционы других пользователей, однако сами создать что-то мы не можем (процедуры update\_auction\_status, update\_user, edit\_auction, update\_lot; функции check\_bid).

Вход с учетной записью администратора дает нам полное управление базой данных аукционной площадки.

Реализация каждой из вышеупомянутой роли приведена в [приложении Д](#_Приложение_Д).

# Заключение

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

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 JSON, 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 ей. Одной из главных особенностей является технология резервного копирования и восстановления, которая обеспечивает сохранность данных в случае любых сбоев или ошибок. Это означает, что все данные аукциона, включая информацию о пользователях, лотах, ставках и аукционах, могут быть быстро восстановлены, минимизируя потерю информации и обеспечивая непрерывность работы площадки.

Кроме того, база данных обладает гибкой системой привилегий, которая позволяет разграничивать доступ к различным функциям и данным в зависимости от роли пользователя. Это обеспечивает безопасность данных и предотвращает несанкционированный доступ.

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

# Список используемых источников

1. eBay [Электронный ресурс] – https://www.ebay.com/ – Дата доступа 27.09.2023
2. Amazon Liquidation Auctions [Электронный ресурс] – https://bstock.com/auctions/amazon/ – Дата доступа 28.09.2023
3. Ay.by Аукционы Беларуси [Электронный ресурс] – http://ay.by/ – Дата доступа 28.09.2023
4. PostgreSQL Сайт с информацией [Электронный ресурс] – https://postgrespro.ru/docs/postgresql.com/ – Дата доступа 10.10.2023
5. Технология резервного копирования и восстановления в PostgreSQL [Электронный ресурс] – https://postgrespro.ru/docs/postgresql/9.6/backup-dump/ – Дата доступа 13.11.2023

# Приложение А

|  |
| --- |
| CREATE TABLE Лоты (  ID\_Лота SERIAL PRIMARY KEY,  Наименование VARCHAR(255) NOT NULL,  Описание VARCHAR NOT NULL,  Изображение VARCHAR(255)  ) TABLESPACE Аукционы;  CREATE TABLE Пользователи (  ID\_Пользователя SERIAL PRIMARY KEY,  Имя VARCHAR(50) NOT NULL,  Фамилия VARCHAR NOT NULL,  Пароль VARCHAR(100) NOT NULL,  email VARCHAR NOT NULL,  Роль VARCHAR NOT NULL  ) TABLESPACE Пользователи;  CREATE TABLE Аукционы (  ID\_аукциона SERIAL PRIMARY KEY,  ID\_Пользователя SERIAL REFERENCES Пользователи(ID\_Пользователя),  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) CHECK (Начальная\_цена > 0)  ) TABLESPACE Аукционы;  CREATE TABLE Ставки (  ID\_Ставки SERIAL PRIMARY KEY,  ID\_Пользователя SERIAL REFERENCES Пользователи(ID\_Пользователя),  Ставка NUMERIC(32,2) CHECK (Ставка > 0),  Время\_ставки TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,  Статус INTEGER,  ID\_аукциона SERIAL REFERENCES Аукционы(ID\_аукциона)  ) TABLESPACE Ставки;  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 1 – Создание таблиц

# Приложение Б

|  |
| --- |
| -- Функция для вывода всех пользователей в бд  CREATE OR REPLACE FUNCTION get\_all\_users() RETURNS SETOF Пользователи AS $$  BEGIN  BEGIN  RETURN QUERY SELECT \* FROM Пользователи;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN;  END;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_all\_users();  -- Функция для вывода всех аукционов в бд  CREATE OR REPLACE FUNCTION get\_all\_auctions() RETURNS SETOF Аукционы AS $$  BEGIN  BEGIN  RETURN QUERY SELECT \* FROM Аукционы;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN;  END;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_all\_auctions();  -- Функция для вывода всех ставок в бд  CREATE OR REPLACE FUNCTION get\_all\_bets() RETURNS SETOF Ставки AS $$  BEGIN  BEGIN  RETURN QUERY SELECT \* FROM Ставки;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN;  END;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_all\_bets();  -- 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 OR REPLACE FUNCTION get\_active\_auctions() RETURNS SETOF Аукционы AS $$  BEGIN  BEGIN  RETURN QUERY SELECT \* FROM Аукционы WHERE Дата\_Конца\_Торгов > now() AND Статус = 1;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN;  END;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_active\_auctions();  -- Функция определения победителя аукциона  CREATE OR REPLACE FUNCTION get\_auction\_winner(  p\_id\_auction integer  ) RETURNS Пользователи AS $$  DECLARE  v\_winner Пользователи%ROWTYPE;  BEGIN  IF p\_id\_auction <= 0 THEN  RAISE EXCEPTION 'ID аукциона должен быть положительным числом';  END IF;  BEGIN  SELECT Пользователи.\*  INTO v\_winner  FROM Пользователи  JOIN Ставки ON Пользователи.ID\_Пользователя = Ставки.ID\_Пользователя  WHERE Ставки.ID\_аукциона = p\_id\_auction  ORDER BY Ставки.Ставка DESC  LIMIT 1;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN NULL;  END;  RETURN v\_winner;  END;  $$ LANGUAGE plpgsql;  SELECT get\_auction\_winner(2);  -- Функция обновления статуса аукциона  CREATE OR REPLACE FUNCTION update\_auction\_status(  p\_id\_auction integer,  p\_new\_status integer  ) RETURNS VOID AS $$  BEGIN  IF p\_id\_auction <= 0 THEN  RAISE EXCEPTION 'ID аукциона должен быть положительным числом';  END IF;  IF p\_new\_status != 0 AND p\_new\_status != 1 THEN  RAISE EXCEPTION 'Новый статус должен быть равен 0 или 1';  END IF;  BEGIN  UPDATE Аукционы  SET Статус = p\_new\_status  WHERE ID\_аукциона = p\_id\_auction;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  END;  END;  $$ LANGUAGE plpgsql;  SELECT update\_auction\_status(1, 1);  -- Получение максимальной ставки на аукционе  CREATE OR REPLACE FUNCTION get\_max\_bid\_for\_auction(auction\_id INTEGER) RETURNS NUMERIC(32,2) AS $$  DECLARE  max\_bid NUMERIC(32,2);  BEGIN  IF auction\_id <= 0 THEN  RAISE EXCEPTION 'ID аукциона должен быть положительным числом';  END IF;  BEGIN  SELECT MAX(Ставка) INTO max\_bid FROM Ставки WHERE ID\_аукциона = auction\_id;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN NULL;  END;  RETURN max\_bid;  END;  $$ LANGUAGE plpgsql;  SELECT get\_max\_bid\_for\_auction(2);  -- Получение минимальной ставки на аукционе  CREATE OR REPLACE FUNCTION get\_min\_bid\_for\_auction(auction\_id INTEGER) RETURNS NUMERIC(32,2) AS $$  DECLARE  min\_bid NUMERIC(32,2);  BEGIN  IF auction\_id <= 0 THEN  RAISE EXCEPTION 'ID аукциона должен быть положительным числом';  END IF;  BEGIN  SELECT MIN(Ставка) INTO min\_bid FROM Ставки WHERE ID\_аукциона = auction\_id;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN NULL;  END;  RETURN min\_bid;  END;  $$ LANGUAGE plpgsql;  SELECT get\_min\_bid\_for\_auction(2);  -- Функция получения списка всех ставок для конкретного лота  CREATE OR REPLACE 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 $$  BEGIN  IF lot\_id <= 0 THEN  RAISE EXCEPTION 'ID лота должен быть положительным числом';  END IF;  BEGIN  RETURN QUERY SELECT \* FROM Ставки WHERE ID\_аукциона IN (  SELECT ID\_аукциона FROM Аукционы WHERE ID\_Лота = lot\_id  );  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN;  END;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_bids\_for\_lot(1);  -- Функция получения списка всех аукционов для конкретного пользователя  CREATE OR REPLACE FUNCTION get\_auctions\_for\_user(user\_id INTEGER) RETURNS TABLE (  ID\_Аукциона INTEGER  ) AS $$  BEGIN  IF user\_id <= 0 THEN  RAISE EXCEPTION 'ID пользователя должен быть положительным числом';  END IF;  BEGIN  RETURN QUERY SELECT ID\_аукциона FROM Аукционы WHERE ID\_Пользователя = user\_id;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  RETURN;  END;  END;  $$ LANGUAGE plpgsql;  SELECT \* FROM get\_auctions\_for\_user(1);  -- Функция повышения ставки  CREATE OR REPLACE FUNCTION increase\_bid(  auction\_id INTEGER,  user\_id INTEGER,  new\_bid NUMERIC(32,2)  ) RETURNS VOID AS $$  BEGIN  IF auction\_id <= 0 OR user\_id <= 0 OR new\_bid <= 0 THEN  RAISE EXCEPTION 'ID аукциона, ID пользователя и новая ставка должны быть положительными числами';  END IF;  BEGIN  UPDATE Ставки SET Ставка = Ставка + new\_bid, Время\_ставки = NOW()  WHERE ID\_аукциона = auction\_id AND ID\_Пользователя = user\_id;  EXCEPTION WHEN others THEN  RAISE NOTICE 'Произошла ошибка: %', SQLERRM;  END;  END;  $$ LANGUAGE plpgsql;  SELECT increase\_bid(2, 1, 50.00); |

Листинг 1 – Функции, связанные непосредственно с аукционами

# Приложение В

|  |
| --- |
| -- Удаление пользователей  CREATE OR REPLACE PROCEDURE delete\_user(user\_id INT)  AS $$  DECLARE  user\_exists INT;  BEGIN  SELECT COUNT(\*) INTO user\_exists FROM Пользователи WHERE ID\_Пользователя = user\_id;  IF user\_exists = 0 THEN  RAISE EXCEPTION 'Пользователь с ID % не найден', user\_id;  ELSE  DELETE FROM Пользователи WHERE ID\_Пользователя = user\_id;  RAISE NOTICE 'Пользователь с ID % успешно удален', user\_id;  END IF;  EXCEPTION WHEN others THEN  RAISE EXCEPTION 'Произошла ошибка: %', SQLERRM;  END;  $$ LANGUAGE plpgsql;  CALL delete\_user(31); |

Листинг 1 – Процедура удаления пользователя

|  |
| --- |
| -- Редактирование пользователя  CREATE OR REPLACE PROCEDURE update\_user(  user\_id INT,  new\_name VARCHAR(50),  new\_last\_name VARCHAR,  new\_password VARCHAR(100),  new\_email VARCHAR,  new\_role VARCHAR  )  AS $$  BEGIN  IF new\_name IS NULL OR new\_name = '' OR new\_last\_name IS NULL OR new\_last\_name = '' OR new\_password IS NULL OR new\_password = '' OR new\_email IS NULL OR new\_email = '' OR new\_role IS NULL OR new\_role = '' THEN  RAISE EXCEPTION 'Все поля должны быть заполнены';  END IF;  IF new\_role NOT IN ('Администратор', 'Пользователь', 'Менеджер') THEN  RAISE EXCEPTION 'Роль должна быть либо Администратор, либо Пользователь, либо Менеджер';  END IF;  UPDATE Пользователи  SET  Имя = new\_name,  Фамилия = new\_last\_name,  Пароль = new\_password,  email = new\_email,  Роль = new\_role  WHERE  ID\_Пользователя = user\_id;  EXCEPTION WHEN others THEN  RAISE EXCEPTION 'Произошла ошибка при обновлении пользователя: %', SQLERRM;  END;  $$ LANGUAGE plpgsql; |

Листинг 2 – Процедура редактирования пользователя

# Приложение Г

|  |
| --- |
| CREATE OR REPLACE FUNCTION IMPORT\_LOTS\_FROM\_JSON\_FILE(FILE\_PATH TEXT)  RETURNS TABLE (  ID\_Лота INTEGER,  Наименование VARCHAR(255),  Описание VARCHAR,  Изображение VARCHAR(255)  )AS $$  DECLARE  FILE\_CONTENT TEXT;  JSON\_DATA JSON;  LOT\_DATA JSON;  BEGIN  CREATE TEMP TABLE IF NOT EXISTS temp\_lots (  ID\_Лота INTEGER,  Наименование VARCHAR(255),  Описание VARCHAR,  Изображение VARCHAR(255)  );  DELETE FROM temp\_lots;  BEGIN  FILE\_CONTENT := pg\_read\_file(FILE\_PATH, 0, 1000000000);  EXCEPTION WHEN OTHERS THEN  RAISE 'Файл не найден: %', FILE\_PATH;  END;  BEGIN  JSON\_DATA := FILE\_CONTENT::JSON;  EXCEPTION WHEN OTHERS THEN  RAISE 'Некорректный JSON: %', SQLERRM;  END;  FOR LOT\_DATA IN SELECT \* FROM json\_array\_elements(JSON\_DATA)  LOOP  IF NOT (LOT\_DATA::jsonb ? 'id\_Лота' AND LOT\_DATA::jsonb ? 'Наименование' AND LOT\_DATA::jsonb ? 'Описание' AND LOT\_DATA::jsonb ? 'Изображение') THEN  CONTINUE;  END IF;  INSERT INTO temp\_lots (ID\_Лота, Наименование, Описание, Изображение)  VALUES (  CAST(LOT\_DATA->>'id\_Лота' AS INTEGER),  LOT\_DATA->>'Наименование',  LOT\_DATA->>'Описание',  LOT\_DATA->>'Изображение'  );  END LOOP;  RETURN QUERY SELECT \* FROM temp\_lots;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 1 – Функция IMPORT\_LOTS\_FROM\_JSON\_FILE

# Приложение Д

|  |
| --- |
| GRANT CONNECT ON DATABASE tradex TO Гость;  GRANT SELECT ON TABLE "Аукционы", "Лоты", "Ставки" TO Гость;  GRANT SELECT, INSERT ON TABLE Пользователи TO Гость;  GRANT SELECT ON num\_lots\_on\_auction TO Гость;  GRANT SELECT ON last\_bid\_for\_lot TO Гость;  GRANT SELECT ON num\_bids\_per\_user TO Гость;  GRANT SELECT ON user\_role\_and\_num\_bids TO Гость;  GRANT EXECUTE ON FUNCTION authenticate\_user TO Гость;  GRANT EXECUTE ON FUNCTION hash\_password TO Гость;  GRANT EXECUTE ON FUNCTION hash\_password\_trigger TO Гость;  GRANT EXECUTE ON FUNCTION check\_duplicate\_email TO Гость;  GRANT EXECUTE ON FUNCTION get\_lots\_list TO Гость;  GRANT EXECUTE ON FUNCTION search\_lots\_by\_name TO Гость;  GRANT EXECUTE ON PROCEDURE Add\_user TO Гость; |

Листинг 1 – Роль гостя

|  |
| --- |
| GRANT CONNECT ON DATABASE tradex TO Пользователь;  GRANT ALL PRIVILEGES ON TABLESPACE Пользователи, Аукционы, Ставки TO Пользователь;  GRANT SELECT, INSERT, UPDATE ON TABLE Пользователи TO Пользователь;  GRANT SELECT, INSERT ON TABLE Аукционы TO Пользователь;  GRANT SELECT, INSERT ON TABLE Лоты TO Пользователь;  GRANT SELECT, INSERT ON TABLE Ставки TO Пользователь;  GRANT SELECT ON num\_lots\_on\_auction TO Пользователь;  GRANT SELECT ON last\_bid\_for\_lot TO Пользователь;  GRANT SELECT ON num\_bids\_per\_user TO Пользователь;  GRANT SELECT ON user\_role\_and\_num\_bids TO Пользователь;  GRANT EXECUTE ON FUNCTION authenticate\_user TO Пользователь;  GRANT EXECUTE ON FUNCTION create\_lot TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_lots\_list TO Пользователь;  GRANT EXECUTE ON FUNCTION search\_lots\_by\_name TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_all\_users TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_all\_auctions TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_all\_bets TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_active\_auctions TO Пользователь;  GRANT EXECUTE ON FUNCTION update\_auction\_status TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_max\_bid\_for\_auction TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_min\_bid\_for\_auction TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_bids\_for\_lot TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_auctions\_for\_user TO Пользователь;  GRANT EXECUTE ON FUNCTION increase\_bid TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_auctions\_for\_user TO Пользователь;  GRANT EXECUTE ON FUNCTION get\_users\_with\_won\_lots TO Пользователь;  GRANT EXECUTE ON FUNCTION hash\_password TO Пользователь;  GRANT EXECUTE ON FUNCTION hash\_password\_trigger TO Пользователь;  GRANT EXECUTE ON FUNCTION check\_auction\_dates TO Пользователь;  GRANT EXECUTE ON FUNCTION check\_bid TO Пользователь;  GRANT EXECUTE ON FUNCTION check\_auction\_status TO Пользователь;  GRANT EXECUTE ON FUNCTION check\_duplicate\_email TO Пользователь;  GRANT EXECUTE ON PROCEDURE Add\_user TO Пользователь;  GRANT EXECUTE ON PROCEDURE delete\_user TO Пользователь;  GRANT EXECUTE ON PROCEDURE update\_user TO Пользователь;  GRANT EXECUTE ON PROCEDURE create\_auction TO Пользователь;  GRANT EXECUTE ON PROCEDURE edit\_auction TO Пользователь;  GRANT EXECUTE ON PROCEDURE update\_lot TO Пользователь;  GRANT EXECUTE ON PROCEDURE delete\_lot TO Пользователь;  GRANT EXECUTE ON PROCEDURE create\_bid TO Пользователь;  GRANT EXECUTE ON PROCEDURE delete\_bid TO Пользователь; |

Листинг 2 – Роль пользователя

|  |
| --- |
| GRANT CONNECT ON DATABASE tradex TO Менеджер;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE "Аукционы", "Лоты" TO Менеджер;  GRANT SELECT, UPDATE, INSERT ON TABLE Пользователи TO Менеджер;  GRANT SELECT ON num\_lots\_on\_auction TO Менеджер;  GRANT SELECT ON last\_bid\_for\_lot TO Менеджер;  GRANT SELECT ON num\_bids\_per\_user TO Менеджер;  GRANT SELECT ON user\_role\_and\_num\_bids TO Менеджер;  GRANT EXECUTE ON FUNCTION authenticate\_user TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_lots\_list TO Менеджер;  GRANT EXECUTE ON FUNCTION search\_lots\_by\_name TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_all\_users TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_all\_auctions TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_all\_bets TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_active\_auctions TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_max\_bid\_for\_auction TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_min\_bid\_for\_auction TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_bids\_for\_lot TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_auctions\_for\_user TO Менеджер;  GRANT EXECUTE ON FUNCTION get\_users\_with\_won\_lots TO Менеджер;  GRANT EXECUTE ON FUNCTION hash\_password TO Менеджер;  GRANT EXECUTE ON FUNCTION hash\_password\_trigger TO Менеджер;  GRANT EXECUTE ON FUNCTION check\_auction\_dates TO Пользователь;  GRANT EXECUTE ON FUNCTION check\_bid TO Пользователь;  GRANT EXECUTE ON FUNCTION check\_auction\_status TO Пользователь;  GRANT EXECUTE ON FUNCTION check\_duplicate\_email TO Пользователь;  GRANT EXECUTE ON PROCEDURE update\_auction\_status TO Менеджер;  GRANT EXECUTE ON PROCEDURE Add\_user TO Менеджер;  GRANT EXECUTE ON PROCEDURE update\_user TO Менеджер;  GRANT EXECUTE ON PROCEDURE edit\_auction TO Менеджер;  GRANT EXECUTE ON PROCEDURE update\_lot TO Менеджер; |

Листинг 3 – Роль менеджера

|  |
| --- |
| GRANT ALL PRIVILEGES ON DATABASE tradex TO Администратор;  GRANT ALL PRIVILEGES ON TABLESPACE Пользователи, Аукционы, Ставки TO Администратор;  GRANT ALL PRIVILEGES ON Пользователи, Аукционы, Ставки, Лоты TO Администратор;  GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO Администратор;  GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO Администратор;  GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO Администратор; |

Листинг 4 – Роль администратора