

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий  
Кафедра информационных систем и технологий  
Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

**Тема: «Реализация базы данных для NFT маркетплейса с использованием технологий применения мультимедийных типов данных и системы email уведомлений о событиях базы данных»**

**Исполнитель**

студент 2 курса 1 группы

\_\_\_\_\_  
подпись, дата

Велютич Д. И.

**Руководитель**

преподаватель-стажер  
должность, учен. степень, ученое звание

\_\_\_\_\_  
подпись, дата

Уласевич Н.И.

Допущен(а) к защите \_\_\_\_\_  
\_\_\_\_\_  
дата, подпись

Курсовой проект защищен с оценкой \_\_\_\_\_

Руководитель \_\_\_\_\_  
подпись

\_\_\_\_\_  
дата

Уласевич Н.И.  
инициалы и фамилия

Минск 2024

## Содержание

Введение .....	2
Постановка задачи .....	3
1. Анализ требований .....	4
1.1. Обзор аналогов.....	4
1.2. Разработка функциональных требований .....	10
1.3. Вывод по разделу.....	10
2. Разработка архитектуры проекта .....	11
2.1. Структура управлением приложения .....	11
2.2. Диаграммы UML, взаимосвязь компонентов .....	11
2.3. Описание объектов и ограничение целостности.....	12
2.4. Вывод по разделу.....	15
3. Разработка модели базы данных. ....	16
3.1 Создание необходимых объектов. ....	16
3.1.1 Таблицы .....	16
3.1.2 Процедуры.....	16
3.1.3 Функции.....	16
3.1.4 Триггеры .....	17
3.1.5 Индексы .....	17
3.1.6 Представления.....	18
3.2. Описание используемой технологии. ....	19
3.3 Описание технологии импорта и экспорта. ....	20
Тестирование производительности.....	22
Краткое описание приложения для демонстрации .....	25
Руководство пользователя .....	32
Заключение .....	35
Список использованных источников.....	36
Приложение А - Таблицы .....	37
Приложение Б – Процедуры.....	39
Приложение В - Функции .....	55
Приложение Г - Триггеры.....	58
Приложение Д – Представления .....	60

## **Введение**

С развитием технологий блокчейн и криптовалют появился новый вид цифровых активов - невзаимозаменяемые токены (NFT), которые представляют собой уникальные цифровые объекты, такие как искусство, коллекционные предметы и виртуальные артефакты. Рост интереса к NFT привел к созданию специализированных онлайн-платформ, где пользователи могут обмениваться, продавать и приобретать эти уникальные активы.

Целью данного проекта является создание базы данных для «NFT Marketplace» - онлайн-платформы, предназначенной для торговли и обмена NFT. В рамках проекта планируется разработать функциональную базу данных, обеспечивающую управление NFT и связанными данными, а также предоставляющую удобный интерфейс для пользователей и администраторов.

### **Постановка задачи**

Целью проекта является создание базы данных для «NFT Marketplace» с выполнением следующих задач:

- Определение ролей пользователей (администратор, пользователь) и разработка соответствующей системы доступа к данным.
- Разработка функционала по добавлению и удалению NFT на площадке для пользователей.
- Реализация возможности просмотра информации о пользователях для администраторов.
- Создание механизма поиска NFT по различным категориям и критериям для пользователей.
- Разработка функционала оценки NFT пользователем.
- Создание системы для формирования коллекций NFT пользователями.

Для реализации поставленных задач база данных должна быть создана с использованием Oracle Database, а доступ к данным должен осуществляться только через соответствующие процедуры. Также необходимо провести импорт и экспорт данных из/в XML формат, протестировать производительность базы данных на большом объеме данных и применить соответствующую технологию базы данных для обеспечения эффективной работы системы.

## 1. Анализ требований

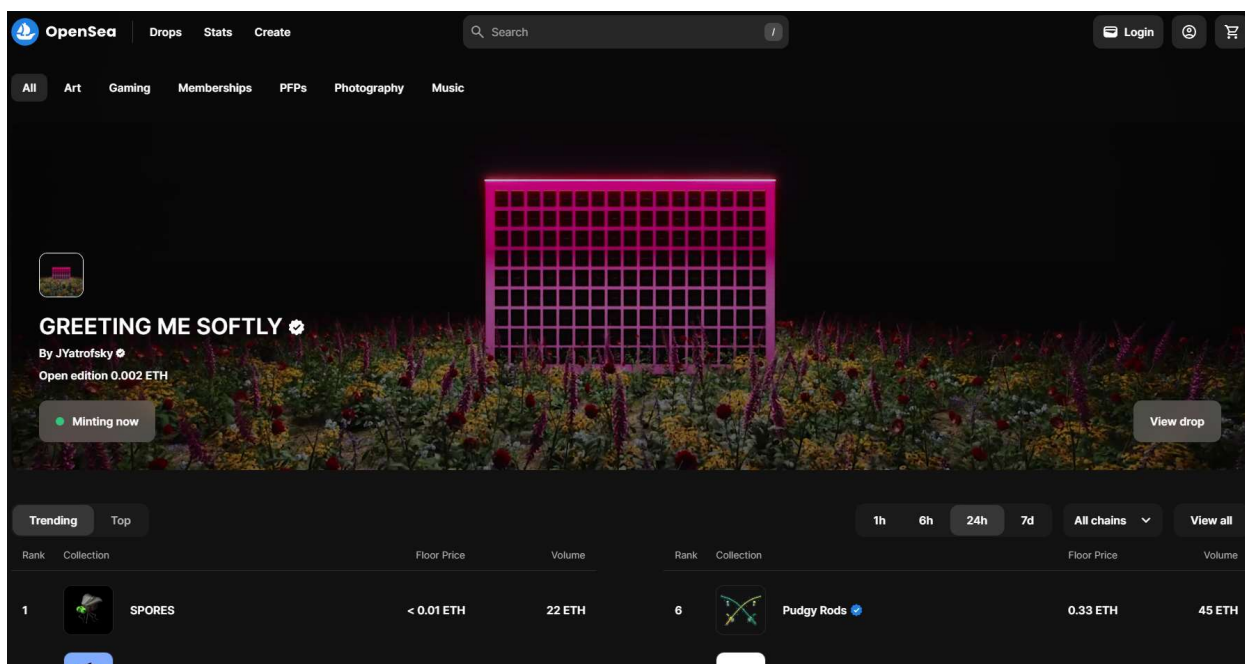
### 1.1. Обзор аналогов

В рамках курсового проекта по разработке базы данных для магазина планируется провести аналитический обзор аналогов платформ для продажи NFT-контента онлайн. Этот обзор поможет лучше понять, какие функции и возможности могут быть реализованы в платформе, и какие инструменты и подходы могут быть применены для разработки базы данных, которая будет соответствовать требованиям и особенностям «NFT Marketplace».

#### 1.1.1. Аналог – OpenSea

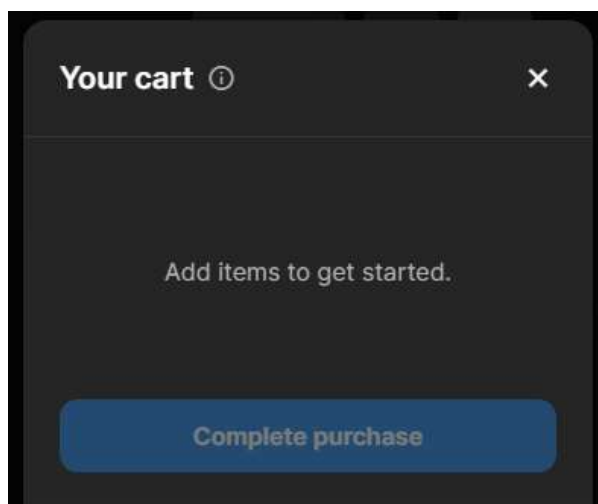
Внешний вид: OpenSea представляет собой платформу с современным и привлекательным дизайном, который отражает тематику NFT-контента. Интерфейс сайта обеспечивает удобство использования и хорошую читаемость. Цветовая палитра подобрана гармонично, что способствует приятному визуальному восприятию пользователей.

Оформление главной страницы: На главной странице OpenSea ясно выделены разделы, что упрощает поиск интересующего контента. Пользователи могут легко найти актуальные NFT-коллекции.



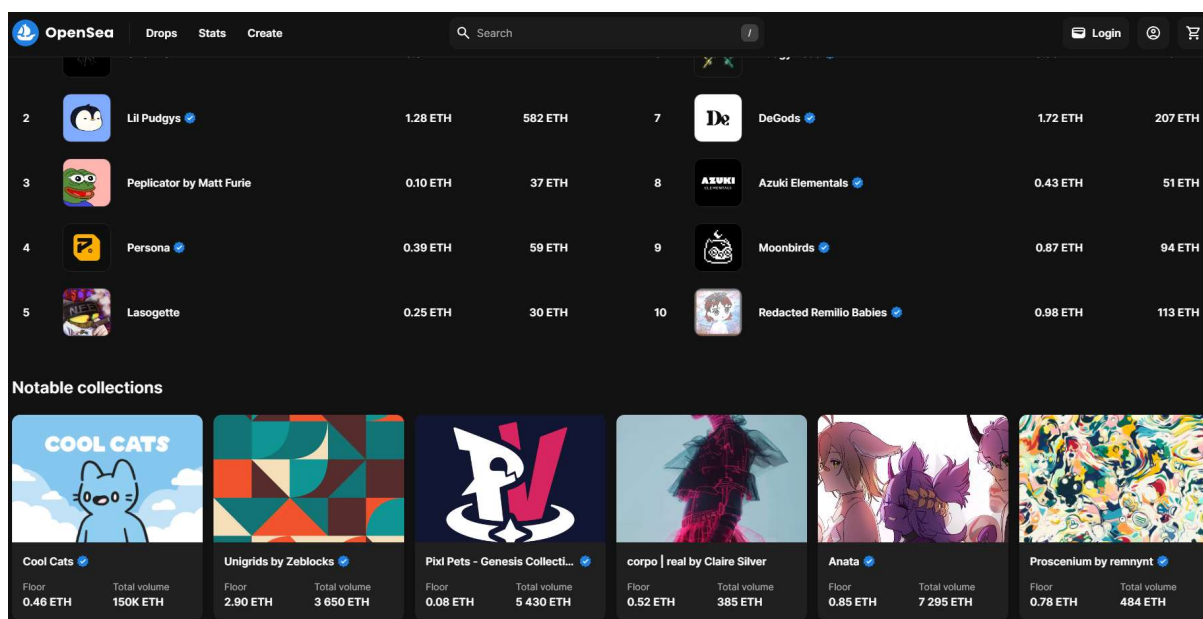
Изображение 1.1 — Главная страница OpenSea

Реализация корзины: Платформа предоставляет удобную функцию добавления NFT-работ в корзину с минимальным числом шагов. Пользователи имеют возможность просматривать и редактировать содержимое корзины перед окончательным оформлением заказа. Процесс оформления заказа четко структурирован и интуитивно понятен.



Изображение 1.2 — Корзина OpenSea

Анализ выявил, что OpenSea обладает современным и привлекательным дизайном, который способствует удобству использования платформы. Функционал корзины и раздела «Избранное» обеспечивает пользователей легкостью и интуитивной понятностью, что создает позитивный опыт покупателей и способствует успешному завершению сделок.

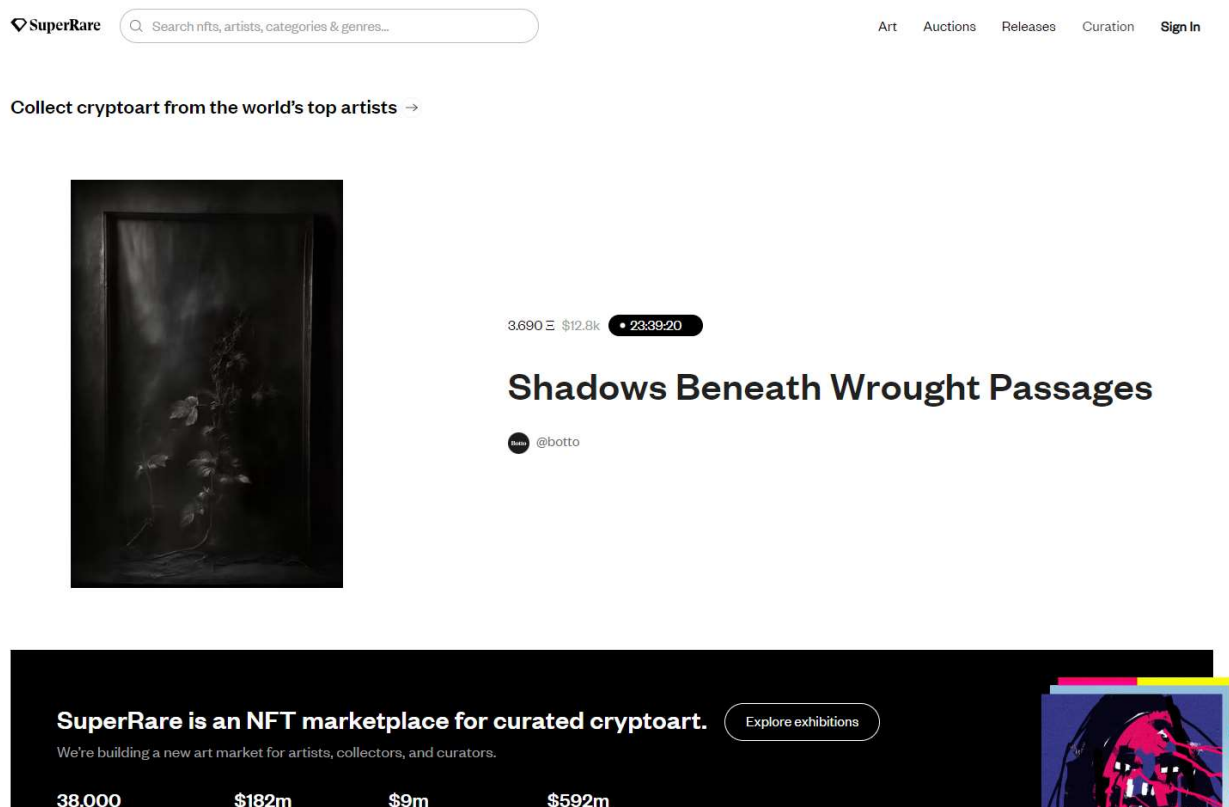


Изображение 1.3 — Коллекции OpenSea

### 1.1.2. Аналог - SuperRare

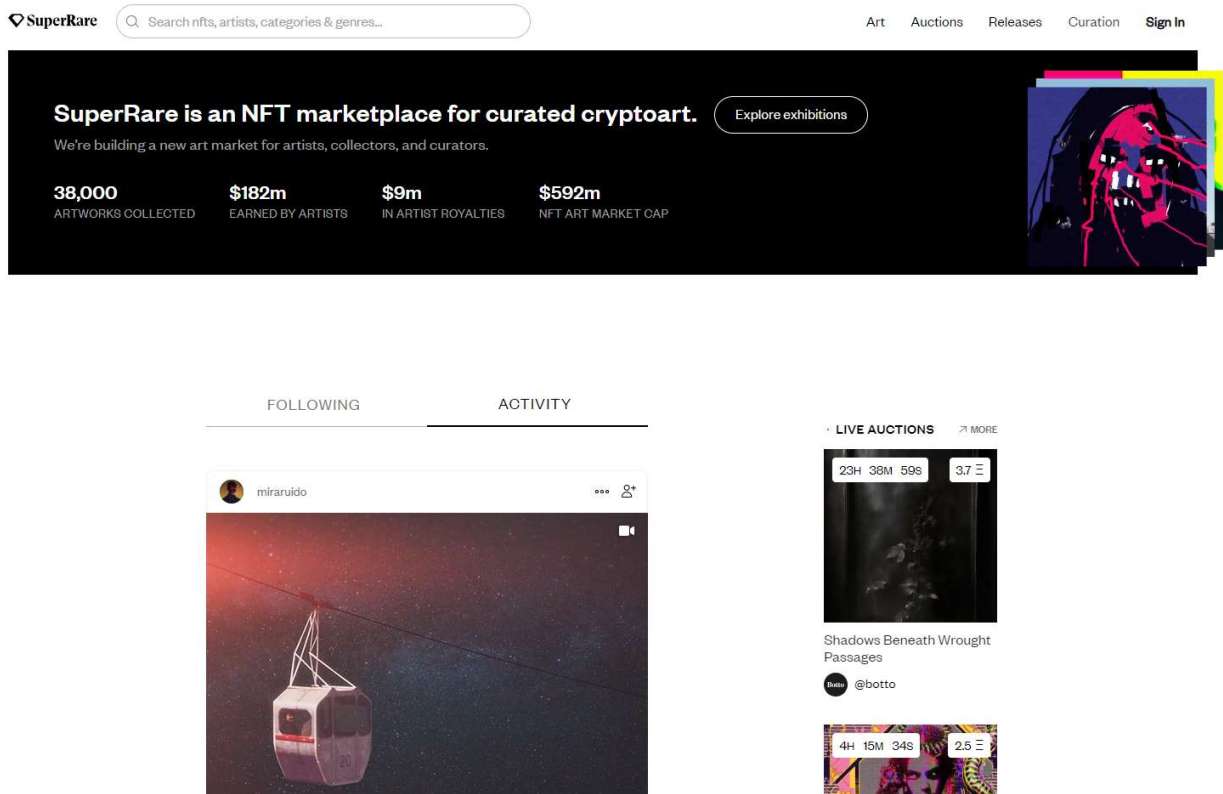
**Уникальные особенности:** SuperRare предоставляет пользователю возможность общаться с художниками и коллекционерами напрямую, что создает уникальное сообщество ценителей искусства NFT. Кроме того, платформа активно поддерживает новых и перспективных художников, предоставляя им шанс на успешное развитие карьеры в сфере цифрового искусства.

**Инновационный подход:** SuperRare внедряет инновационные технологии для обеспечения безопасности и прозрачности торговли NFT. Использование блокчейна Ethereum позволяет гарантировать уникальность и подлинность каждой работы, а также обеспечивает прозрачность всех сделок на платформе.



Изображение 1.4 — Главная страница SuperRare

**Широкий выбор контента:** На платформе SuperRare представлен широкий спектр NFT-работ различных стилей и направлений. Это позволяет пользователям находить работы, соответствующие их вкусам и предпочтениям, а также расширять свой кругозор в области цифрового искусства.



Изображение 1.5 — Актуальное SuperRare

Поддержка художников: SuperRare активно поддерживает художников, предоставляя им справедливую долю от продаж своих работ и создавая условия для развития их творческого потенциала. Это способствует росту сообщества художников и взаимодействию с поклонниками искусства.

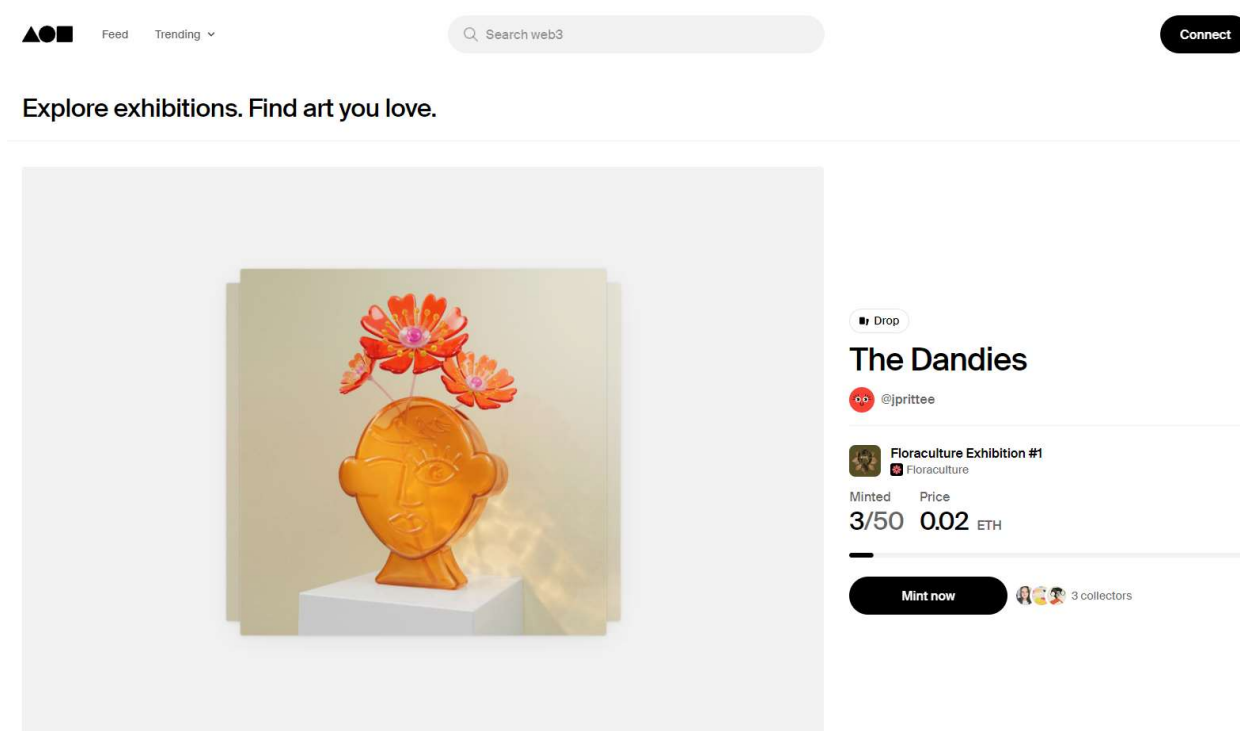
Таким образом, SuperRare выделяется своим уникальным сообществом ценителей искусства NFT, инновационным подходом к безопасности и прозрачности торговли, широким выбором контента и поддержкой художников.



### 1.1.3. Аналог - Foundation

Уникальные особенности: Foundation выделяется своим акцентом на кураторстве и качестве представленных на платформе работ. Здесь пользователи могут найти уникальные и высококачественные NFT-произведения искусства, отобранные профессиональными кураторами. Этот подход способствует созданию качественной коллекции искусства для покупателей и инвесторов.

Специализация на художественных ценностях: Foundation ориентирована на поддержку и продвижение художников и творческих работ, которые обладают художественной ценностью и оригинальным подходом. Платформа предоставляет художникам возможность проявить себя и представить свои уникальные произведения широкой аудитории.



Изображение 1.6 — Главная страница Foundation

Эксклюзивный контент: На платформе Foundation пользователи могут найти эксклюзивные NFT-работы, которые недоступны на других платформах. Это делает Foundation привлекательным местом для коллекционеров, желающих приобрести уникальные и ограниченные в выпуске произведения искусства.

Коммуникация с художниками: Foundation обеспечивает прямое взаимодействие пользователей с художниками, что способствует созданию сообщества искусства и поддерживает тесные связи между художниками и их поклонниками. Это позволяет пользователям получить уникальную перспективу на творческий процесс и взаимодействовать с людьми, создающими искусство.

## Featured exhibitions



Floraculture Exhibition #1

Jack Prettyman and 15 others

[Open](#)

Presented by Floraculture

[Save](#)

West Coast Hip Hop History

Barry Sutton Studio

[Open](#)

Presented by Hip Hop: The Lost Archive by Barry Sutton

[Save](#)

Изображение 1.7 — Главная страница Foundation

Таким образом, Foundation выделяется своим акцентом на кураторстве и качестве работ, специализацией на художественных ценностях, предоставлением эксклюзивного контента и поддержкой коммуникации между художниками и поклонниками искусства.

## **1.2. Разработка функциональных требований для «NFT Marketplace»**

Функциональные требования - это описание всех функций, которые выполняет платформа в рамках определенного задания.

В первую очередь, пользователь должен иметь возможность осуществлять поиск NFT по различным критериям, включая категорию, художника и название произведения. Также важно, чтобы пользователь мог резервировать NFT на определенный срок и добавлять их в свою корзину для последующей покупки.

Во-вторых, вся предоставляемая информация должна быть представлена в привлекательном и удобном формате. Это включает в себя не только эстетически приятный дизайн, но и удобное расположение информации. Также следует обеспечить возможность просмотра NFT в различных форматах для удовлетворения потребностей пользователей.

Реализация корзины должна быть удобной и интуитивно понятной для пользователя, а оформление покупки — минимальным и безлимитным. Наличие раздела «Избранное» или «Любимое» дает пользователям возможность сохранять и отслеживать понравившиеся NFT.

## **1.3. Вывод по разделу**

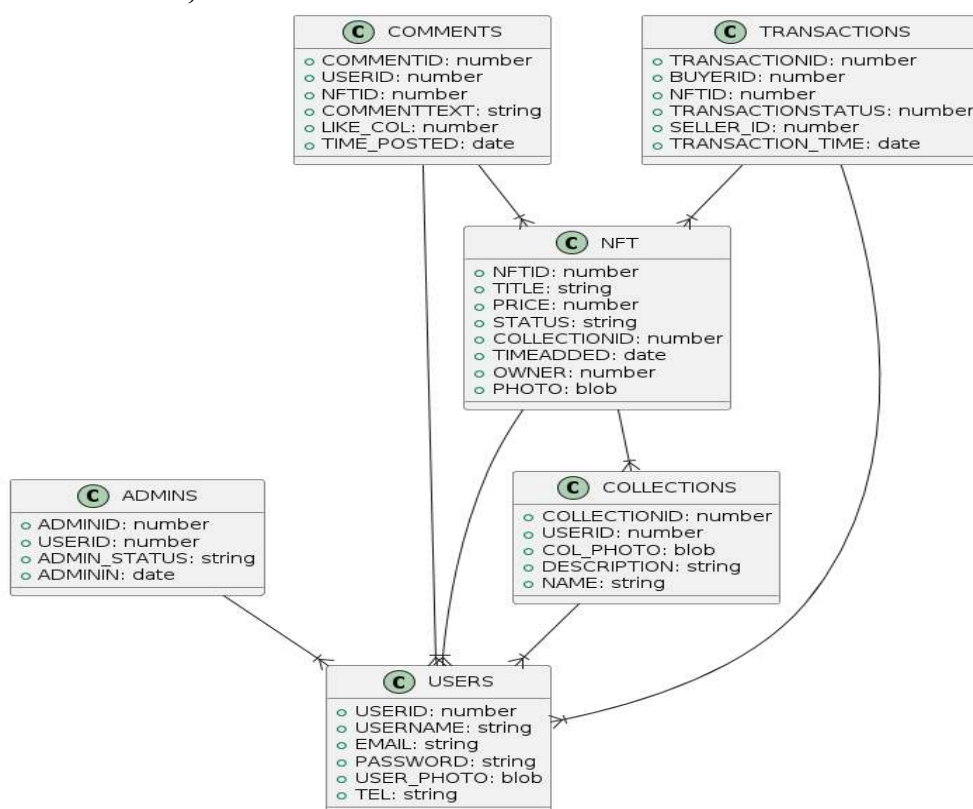
Проект предполагает разработку базы данных для платформы веб-сайта на «NFT Marketplace». Для обеспечения удовлетворения потребностей клиентов и администраторов платформы, таких как резервирование NFT, покупка NFT, поиск по различным критериям, добавление, удаление, поиск и анализ проданных NFT.

## 2. Разработка архитектуры проекта

### 2.1. Структура управлением приложения

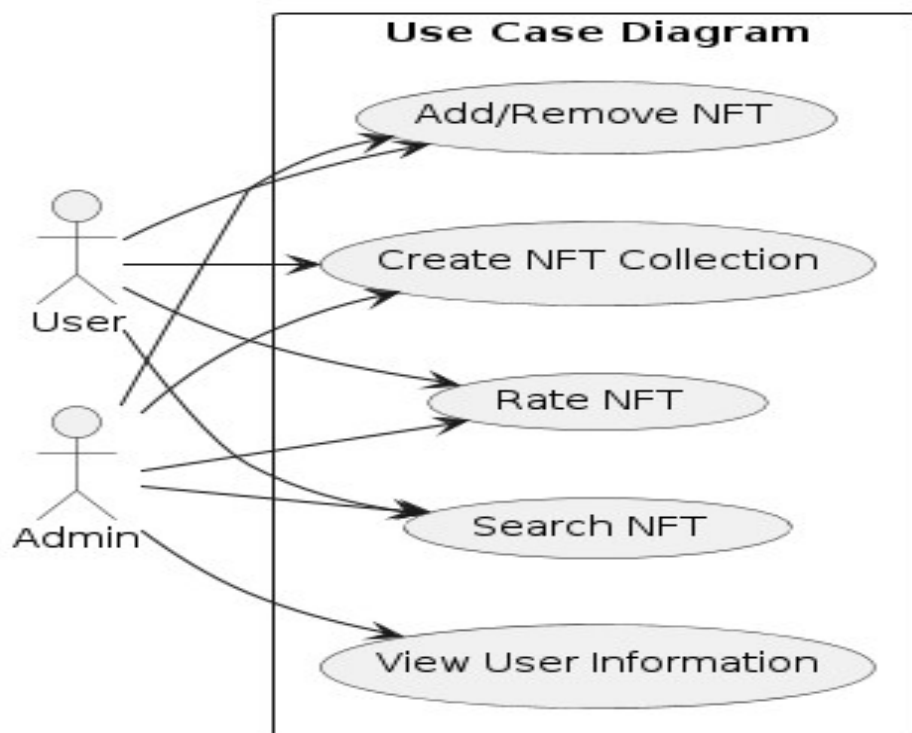
При проектировании базы данных для «NFT Marketplace» с использованием технологий применения мультимедийных типов данных и системы email уведомлений о событиях базы данных важно учитывать обобщенную структуру управления приложением. Эта структура включает в себя несколько компонентов, которые выполняют различные функции и взаимодействуют друг с другом для обеспечения полноценной работы приложения. Oracle 21XE поддерживает работу с мультимедиа, используя Oracle Multimedia, а также работу с Email уведомлениями используя Email Notification and Health Check Report Overview. Таким образом создание специализированных методов для работы с мультимедиа и email уведомлениями не должны стать проблемой при создании базы данных на базе Oracle 21XE.

### 2.2. Диаграммы UML, взаимосвязь компонентов



Изображение 1.8 — UML-диаграмма структуры БД

Диаграммы UML (Unified Modeling Language) широко используются для моделирования баз «NFT Marketplace» и визуализации взаимосвязи ее компонентов. Они помогают разработчикам и заинтересованным сторонам понять, как связаны между собой данные и как они хранятся. UML-диаграммы, особенно диаграммы таблиц, создаются для удобства визуализации структуры базы данных. Они помогают понять, как элементы взаимодействуют друг с другом, а также обмениваются данными.



Изображение 1.9 — UML-диаграмма вариантов использования

Каждый «NFT Marketplace» должен иметь все необходимые таблицы, чтобы хранить информацию о NFT, NFT-коллекциях, резервированных NFT, пользователях и авторах. Диаграмма должна показывать всю структуру базы данных, включая процессы и взаимодействие всех основных сущностей. Она также должна показывать первичные и внешние ключи, которые накладывают ограничения на хранение данных.

### 2.3. Описание объектов и ограничение целостности

Таблица 1 -- Таблицы базы данных

Имя таблицы	Назначение таблицы
Пользователи (Users)	Содержит информацию о зарегистрированных пользователях, их учетные записи и личные данные.
NFT	Хранит информацию о каждом NFT, включая его характеристики, цену и статус (продан, доступен для продажи и т. д.).
Транзакции (Transactions)	Содержит данные о сделках между пользователями, включая информацию о купленных NFT и статус транзакции.
Коллекции (Collections)	Связывает пользователей с NFT, находящимися в их коллекции.
Комментарии (Comments)	Хранит комментарии пользователей к NFT и обсуждения в сообществе.
Администраторы (Admins)	Содержит информацию об администраторах платформы.

Описание всех таблиц находящихся в базе данных.

Таблица 2 -- Связи между таблицами

Таблица PK	Таблица FK	Описание связи
Users.UserID	Collections.UserID	Связь между пользователями и их коллекциями.
NFT.CollectionID	Collections.CollectionID	Связь между NFT и коллекциями, к которым они принадлежат.
Transactions.BuyerID	Users.UserID	Связь между транзакциями и пользователями, являющимися покупателями.
Transactions.seller_id	Users.UserID	Связь между транзакциями и пользователями, являющимися продавцами.
Transactions.NFTID	NFT.NFTID	Связь между транзакциями и NFT, которые были куплены.
Comments.UserID	Users.UserID	Связь между комментариями и пользователями, оставившими их.
Comments.NFTID	NFT.NFTID	Связь между комментариями и соответствующими NFT.
Admins.AdminID	Users.UserID	Связь между администраторами и пользователями (администраторы также являются пользователями).
NFT.Owner	Users.UserID	Связь между NFT и пользователями, являющимися владельцами.

Описание связей между таблицами базы данных.

Таблица 3 -- Таблица Коллекции (Collections)

Поле таблицы	Назначение поля
CollectionID	Уникальный идентификатор коллекции.
UserID	Идентификатор пользователя, владеющего коллекцией.
col_photo	Фото (аватар) коллекции.
description	Описание коллекции.
name	Имя коллекции.

Описание полей в таблице Collections.

Таблица 4 -- Таблица Пользователи (Users)

Поле таблицы	Назначение поля
UserID	Уникальный идентификатор пользователя.
UserName	Имя пользователя.
Email	Электронная почта пользователя.
Password	Пароль пользователя.
User_photo	Фото профиля пользователя.
tel	Телефон пользователя.

Описание полей в таблице Users.

Таблица 5 -- Таблица Транзакции (Transactions)

Поле таблицы	Назначение поля
TransactionID	Уникальный идентификатор транзакции.
BuyerID	Идентификатор пользователя-покупателя.
NFTID	Идентификатор NFT, связанного с транзакцией.
TransactionStatus	Статус транзакции (например, «в процессе», «завершена»).
Seller_id	Идентификатор пользователя-продавца.
Transaction_time	Время, когда транзакция была совершена.

Описание полей в таблице Transactions.

Таблица 6 -- Таблица Комментарии (Comments)

Поле таблицы	Назначение поля
CommentID	Уникальный идентификатор комментария.
UserID	Идентификатор пользователя, оставившего комментарий.
NFTID	Идентификатор NFT, к которому относится комментарий.
CommentText	Текст комментария.
Time_posted	Время публикации комментария.
Like_col	Количество лайков.

Описание полей в таблице Comments.

Таблица 7 -- Таблица Администраторы (Admins)

Поле таблицы	Назначение поля
AdminID	Уникальный идентификатор администратора.
UserID	Идентификатор пользователя, являющегося администратором.
Admin_status	Статус администратора (онлайн/оффлайн)

Описание полей в таблице Admins.

Таблица 8 -- Таблица **NFT**

<b>Поле таблицы</b>	<b>Назначение поля</b>
NFTID	Уникальный идентификатор NFT.
Title	Заголовок или название NFT.
Price	Цена NFT.
Status	Статус NFT (продан, доступен для продажи и т. д.).
CollectionID	Идентификатор коллекции, к которой относится NFT.
Time_added	Время публикации NFT
Description	Описание NFT
Owner	Актуальный владелец NFT

Описание полей в таблице NFT.

## 2.4. Вывод по разделу

Users, NFT, Transactions, Collections, Comments, Admins будут представлены в базе данных NFT магазина во время разработки проекта. Внешние и первичные ключи связывают данные таблиц.



### 3. Разработка модели базы данных.

#### 3.1 Создание необходимых объектов.

##### 3.1.1 Таблицы

Таблица — это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Она состоит из столбцов и строк.

База данных данного курсового проекта содержит 6 таблиц, которые описаны в главе 2.

##### 3.1.2 Процедуры

Процедуры — это именованные блоки кода на языке PL/SQL, которые хранятся в базе данных и могут быть вызваны при необходимости. Они используются для группировки и переиспользования кода, который должен выполняться несколько раз, а также для уменьшения нагрузки на сеть при обращении к базе данных. Процедуры могут содержать в себе SQL-запросы, циклы, условные операторы, переменные и многое другое.

Процедуры, разработанные в рамках курсового проекта:

- add\_nft. Процедура создания NFT;
- create\_collection. Процедура создания коллекции;
- define\_roles. Процедура проверки роли пользователя;
- delete\_nft. Процедура удаления NFT;
- dequeue\_and\_send\_emails. Процедура для реализации технологии отправки Email уведомлений;
- enqueue\_collection\_notification. Процедура создания Email письма для последующей отправки;
- Insert\_NFT\_Photo. Процедура для реализации технологии хранения мультимедиа данных в бд Oracle;
- rate\_nft. Процедура оценивания NFT;
- search\_nfts. Процедура поиска NFT;
- view\_user\_info. Процедура просмотра информации о пользователе;
- export\_NFT\_to\_XML. Процедура экспорта данных из таблицы NFT в файл формата XML;
- IMPORT\_NFT\_FROM\_XML. Процедура импорта данных из файла XML в таблицу NFT;
- like\_comment. Процедура для лайка существующего комментария;
- add\_comment. Процедура для добавления комментария пользователя.

##### 3.1.3 Функции

Функции являются объектами базы данных, которые позволяют выполнять определенные действия на стороне сервера базы данных. Функции могут использоваться для выполнения различных задач, таких как обработка данных, преобразование данных, агрегация данных и т. д.

Функции, разработанные в рамках курсового проекта:

- `create_user_with_privileges`. Функция для создания привилегированного пользователя;
- `blob_to_png`. Функция для преобразования мультимедиа данных для их последующего просмотра.

### 3.1.4 Триггеры

Триггеры в Oracle — это функции, которые автоматически вызываются при определенных событиях, происходящих в базе данных, таких как вставка, обновление или удаление строк в таблицах. Триггеры могут выполнять различные действия, например, проверять данные перед вставкой или изменением, обновлять связанные данные или записывать изменения в другие таблицы.

Триггер, разработанный в рамках курсового проекта:

- `trg_after_collection_insert`. Триггер на добавление строк в таблицу `collections`;
- `check_transaction_time_and_comments`. Триггер на проверку времени транзакций и комментариев.

### 3.1.5 Индексы

Индексы являются объектами базы данных, которые ускоряют выполнение запросов на поиск, сортировку или группировку данных в таблицах. Индексы представляют собой специальные структуры данных, которые позволяют быстро находить записи в таблице на основе значений столбцов, входящих в индекс.

Индексы, разработанные в рамках курсового проекта:

- `IDX_COLLECTIONS_USERID`. Индекс для поиска коллекций по `id` пользователя;
- `IDX_COMMENTS_NFTID`. Индекс для поиска комментариев по `id` NFT.
- `IDX_COMMENTS_USERID`. Индекс для поиска комментариев по `id` пользователя.
- `IDX_NFT_COLLECTIONID`. Индекс для поиска NFT по `id` коллекции.
- `IDX_NFT_OWNER`. Индекс для поиска владельца NFT.
- `IDX_NFT_STATUS`. Индекс для поиска текущего статуса NFT.
- `IDX_TRANSACTIONS_BUYERID`. Индекс для поиска транзакций покупателя.
- `IDX_TRANSACTIONS_NFTID`. Индекс для поиска всех транзакций по `id` NFT.
- `IDX_TRANSACTIONS_SELLERID`. Индекс для поиска транзакций продавца.

### 3.1.6 Представления

Представления (view) - это виртуальные таблицы, создаваемые на основе запроса к одной или нескольким таблицам в базе данных. Представления представляют собой логические структуры данных, которые позволяют пользователю или приложению видеть данные из базы данных в определенном формате или с определенными фильтрами, без необходимости изменения фактической структуры таблиц.

Представления, разработанные в рамках курсового проекта:

- **ActiveNFTs**. Представление активных NFT;
- **CollectionsWithNFTCount**. Представление информации о коллекции с количеством NFT;
- **TransactionDetails**. Представление полной информации о транзакциях.

### 3.2. Описание используемой технологии.

Были выбраны технологии применения мультимедийных типов данных и системы Email уведомлений о событиях базы данных. Разберем работу каждой из технологий по порядку:

#### Работа с медиа данными

Было решено хранить медиаданные (изображения) в формате данных BLOB (Binary Large Object). Данные в данном формате хранятся в виде бинарной строки, позволяя хранить в себе одновременно до 4 294 967 296 байт (4 ГБ).

Просто так медиаданные не будут вставлены в ячейку BLOB, именно из-за этого были разработаны соответствующие технологии:

Процедура **Insert\_NFT\_Photo**, которая принимает NFTID и путь до медиафайла. Далее функция конвертирует медиафайл в бинарную строку и затем заносит данные в соответствующую ячейку таблицы.

Функция **blob\_to\_png**, которая принимает BLOB значение из ячейки, конвертирует его обратно в PNG (Portable Network Graphics) и передаёт дальше для возможности последующего просмотра извне.

#### Отправка Email уведомлений о состоянии БД

Было решено отправлять уведомления о создании новой коллекции.

Когда в таблицу Collections добавляется новая строка, срабатывает триггер **trg\_after\_collection\_insert**, который в свою очередь запускает процедуры **dequeue\_and\_send\_emails** и **enqueue\_collection\_notification**. Одна создает шаблон письма, другая отправляет письмо через собственный SMTP (Simple Mail Transfer Protocol) сервер, на базе smtp.gmail.com.

### 3.3 Описание технологии импорта и экспорта.

Были разработаны соответствующие процедуры **export\_NFT\_to\_XML** и **import\_NFT\_from\_XML**, которые соответственно производят экспорт и импорт данных из/в файл формата XML.

```

create or replace NONEDITIONABLE PROCEDURE ex-
port_NFT_to_XML(filename IN VARCHAR2) AS
  v_file UTL_FILE.FILE_TYPE;
  v_xml XMLTYPE;
BEGIN
  -- Создание XMLTYPE объекта с данными из таблицы NFT
  SELECT XMLELEMENT(«NFTs»,
    XMLAGG(XMLELEMENT(«NFT»,
      XMLFOREST(NFTID AS «NFTID»,
        Title AS «Title»,
        Price AS «Price»,
        Status AS «Status»,
        CollectionID AS «CollectionID»,
        TimeAdded AS «TimeAdded»,
        Owner AS «Owner»)))) INTO v_xml
  FROM NFT;

  -- Создание файла и запись в него XML данных
  v_file := UTL_FILE.FOPEN('XML_DIR', filename, 'W');
  UTL_FILE.PUT_LINE(v_file, v_xml.getClobVal());
  UTL_FILE.FCLOSE(v_file);
EXCEPTION
WHEN OTHERS THEN
  IF UTL_FILE.IS_OPEN(v_file) THEN
    UTL_FILE.FCLOSE(v_file);
  END IF;
  RAISE;
END;
```

Листинг 2.0 – процедура export\_NFT\_to\_XML

```

create or replace NONEDITIONABLE PROCEDURE IM-
PORT_NFT_FROM_XML(filename IN VARCHAR2) AS
  v_file UTL_FILE.FILE_TYPE;
  v_line VARCHAR2(32767); -- Line buffer
  v_xml XMLTYPE;
BEGIN
  -- Open the XML file for reading
  v_file := UTL_FILE.FOPEN('XML_DIR', filename, 'R');

  -- Read the XML data from the file
  UTL_FILE.GET_LINE(v_file, v_line);
  v_xml := XMLTYPE(v_line);

  -- Insert the data into the NFT table
```

```

        INSERT INTO NFT (NFTID, Title, Price, Status, CollectionID,
TimeAdded, Owner)
        SELECT x.NFTID,
        x.Title,
        x.Price,
        x.Status,
        x.CollectionID,
        TO_TIMESTAMP(x.TimeAdded, 'YYYY-MM-DD HH24:MI:SS'),
        x.Owner
        FROM XMLTABLE('/NFTs/NFT'
        PASSING v_xml
        COLUMNS
        NFTID INT PATH 'NFTID',
        Title VARCHAR2(100) PATH 'Title',
        Price DECIMAL(18,2) PATH 'Price',
        Status VARCHAR2(20) PATH 'Status',
        CollectionID INT PATH 'CollectionID',
        TimeAdded VARCHAR2(20) PATH 'TimeAdded',
        Owner INT PATH 'Owner') AS x;

-- Close the file
UTL_FILE.FCLOSE(v_file);
EXCEPTION
WHEN OTHERS THEN
IF UTL_FILE.IS_OPEN(v_file) THEN
UTL_FILE.FCLOSE(v_file);
END IF;
RAISE;
END;
```

Листинг 2.1 – процедура IMPORT\_NFT\_FROM\_XML

## Тестирование производительности

Согласно задаче, тестировать производительность БД мы будем путём вставки в неё 100,000 строк, в нашем случае это будут 100,000 пользователей. Выполнять мы это будем через скрипт, написанный на ЯП Python

```
import cx_Oracle

# Connect to the database
connection = cx_Oracle.connect(«system», «Dim123as», «localhost»)

try:
    # Создание курсора
    cursor = connection.cursor()

    # Начало вставки с UserID, начиная с 2
    start_userid = 2

    # Вставка строк с UserID, начиная с start_userid и заканчивая
    start_userid + 99999
    for i in range(start_userid, start_userid + 100000):
        # Формирование уникального значения для каждой ячейки путем
        # добавления номера текущей итерации
        unique_userid = str(i)
        unique_email = «test» + str(i) + «@gmail.com»

        cursor.execute(«««
        INSERT INTO Users (UserID, UserName, Email, Password)
        VALUES (:UserID, :UserName, :Email, 'Test123')
        «««, {'UserID': unique_userid, 'UserName': unique_userid,
        'Email': unique_email})

    # Подтверждение транзакции
    connection.commit()
    print(«Данные успешно вставлены.»)

except cx_Oracle.Error as error:
    print(«Ошибка:», error)
    # Откат транзакции в случае ошибки
    connection.rollback()

finally:
    # Закрытие курсора и соединения
    cursor.close()
    connection.close()
```

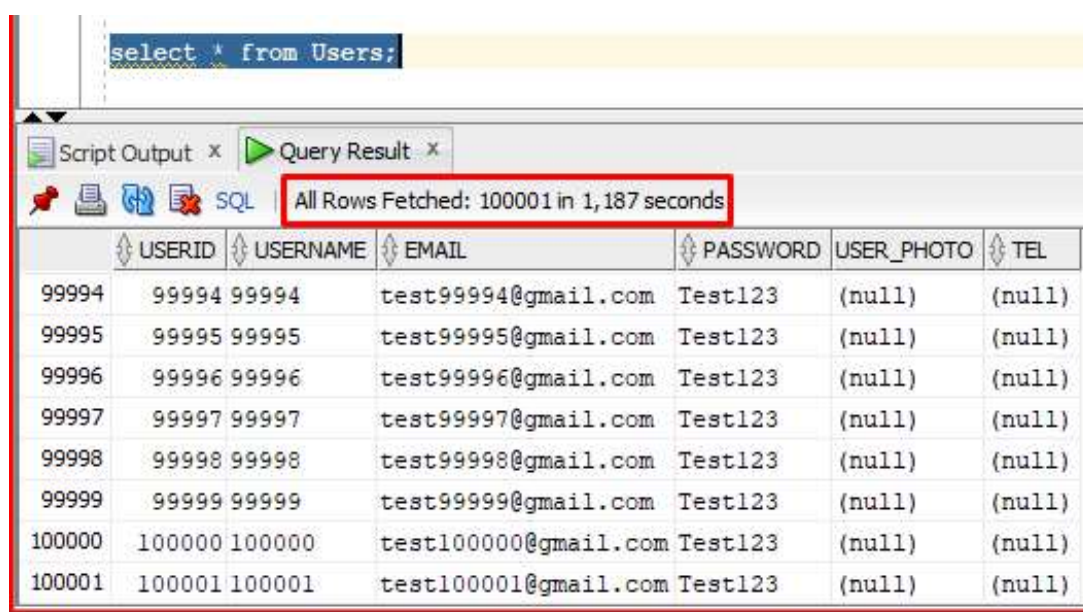
Листинг 2.2 – Python код для вставки 100,000 строк

В результате получаем сообщение о том, что все данные вставлены успешно.

```
PS C:\Users\KROU4\YandexDisk\Уник\Курсач\1 сем БД\App> python .\100krows.py
Данные успешно вставлены.
```

Изображение 2.3 – Уведомление о успешной вставке всех данных

Проверим это через соответствующий select запрос и видим, что действительно все 100,000 строк были успешно занесены в таблицу. Select запрос на обработку выполняется за 1,187 секунды, что для такого количества данных является очень неплохим результатом.



select \* from Users;

Script Output x Query Result x

All Rows Fetched: 100001 in 1,187 seconds

	USERID	USERNAME	EMAIL	PASSWORD	USER_PHOTO	TEL
99994	99994 99994		test99994@gmail.com	Test123	(null)	(null)
99995	99995 99995		test99995@gmail.com	Test123	(null)	(null)
99996	99996 99996		test99996@gmail.com	Test123	(null)	(null)
99997	99997 99997		test99997@gmail.com	Test123	(null)	(null)
99998	99998 99998		test99998@gmail.com	Test123	(null)	(null)
99999	99999 99999		test99999@gmail.com	Test123	(null)	(null)
100000	100000 100000		test100000@gmail.com	Test123	(null)	(null)
100001	100001 100001		test100001@gmail.com	Test123	(null)	(null)

Изображение 2.4 – Демонстрация 100,000 строк (после индексирования)



select \* from Users;

Script Output x Query Result x

All Rows Fetched: 100001 in 21,349 seconds

	USERID	USERNAME	EMAIL	PASSWORD	USER_PHOTO	TEL
99992	99992	99992	test99992@gmail.com	Test123	(null)	(null)
99993	99993	99993	test99993@gmail.com	Test123	(null)	(null)
99994	99994	99994	test99994@gmail.com	Test123	(null)	(null)
99995	99995	99995	test99995@gmail.com	Test123	(null)	(null)
99996	99996	99996	test99996@gmail.com	Test123	(null)	(null)
99997	99997	99997	test99997@gmail.com	Test123	(null)	(null)
99998	99998	99998	test99998@gmail.com	Test123	(null)	(null)
99999	99999	99999	test99999@gmail.com	Test123	(null)	(null)
100000	100000	100000	test100000@gmail.com	Test123	(null)	(null)
100001	100001	100001	test100001@gmail.com	Test123	(null)	(null)

Изображение 2.5 – Демонстрация 100,000 строк (до индексирования)

Благодаря индексам БД прекрасно ориентируется в своих данных в кратчайшие сроки.

select \* from Users where userid = 31627;

Script Output x Query Result x

All Rows Fetched: 1 in 0,001 seconds

	USERID	USERNAME	EMAIL	PASSWORD	USER_PHOTO	TEL
1	31627	31627	test31627@gmail.com	Test123	(null)	(null)

True Random Number Generator

Min:

Max:

Result: **31627**

Min: 1, Max: 100000  
2024-05-06 22:08:29 UTC

Powered by [RANDOM.ORG](https://RANDOM.ORG)

Изображение 2.6 – Поиск случайной (случайной) строки

## Краткое описание приложения для демонстрации

Приложение представляет из себя Python приложение, которое подключается к базе данных и отправляет ей соответствующие запросы используя курсоры в Python. Вся логика происходит непосредственно в БД через соответствующие процедуры и функции.

```
import tkinter as tk
from tkinter import filedialog
import cx_Oracle
from PIL import Image, ImageTk
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import io

# Детали подключения к базе данных
db_user = 'system'
db_password = '*****'
db_dsn = 'localhost'

# Функция для подключения к базе данных с обработкой ошибок
def connect_to_db():
    try:
        connection = cx_Oracle.connect(db_user, db_password,
db_dsn)
        print("Успешное подключение к базе данных!")
        return connection
    except cx_Oracle.Error as error:
        print("Ошибка подключения к базе данных:", error)
        return None

# Функция для выполнения SQL-запроса и получения данных BLOB
def fetch_blob_data(nft_id):
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        # Получение данных BLOB на основе nft_id
        cursor.execute("SELECT photo FROM NFT WHERE NFTID =
:id", id=nft_id)
        result = cursor.fetchone()
        if result:
            blob_data = result[0].read() # Чтение данных
BLOB
            return blob_data
        else:
            print("Ошибка: NFT не найден с ID:", nft_id)
            conn.close()
    else:
        print("Ошибка: Не удалось подключиться к базе дан-
ных.")
    return None
```

```

# Функция для преобразования BLOB в PNG и отображения
def process_and_display_image(nft_id, root, nft_label):
    blob_data = fetch_blob_data(nft_id)
    if blob_data:
        # Преобразование данных BLOB в изображение
        image = Image.open(io.BytesIO(blob_data))
        # Изменение размера изображения до 256x256 пикселей
        image.thumbnail((256, 256))
        photo = ImageTk.PhotoImage(image)
        # Отображение изображения и текста в метке
        label = tk.Label(root, image=photo, text=nft_label,
compound=tk.TOP)
        label.image = photo # Сохранение ссылки для предот-
вращения сборки мусора
        label.pack(side=tk.LEFT, padx=10, pady=10)
    else:
        nft_label.config(text=f"Ошибка при получении NFT с
ID: {nft_id}")

# Функция для получения данных NFT из базы данных
def fetch_nfts():
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        cursor.execute("SELECT NFTID, Title, Price FROM
NFT") # Получение nft_id, title и price
        nfts = cursor.fetchall()
        conn.close()
        return nfts
    else:
        return None

# Функция для вставки данных коллекции в базу данных
def insert_collection_data(user_id, collection_name, collec-
tion_description, photo_path):
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        try:
            # Чтение файла изображения и преобразование его
в бинарные данные
            with open(photo_path, 'rb') as f:
                photo_data = f.read()
            # Вставка данных коллекции в базу данных
            cursor.execute(
                "INSERT INTO Collections (CollectionID,
UserID, col_photo, description, name) "
                "VALUES (COLLECTION_ID_SEQ.NEXTVAL,
:user_id, :photo, :description, :name)",
                user_id=user_id, photo=photo_data, descrip-
tion=collection_description, name=collection_name
            )
            conn.commit()

```

```

        print("Данные коллекции успешно вставлены!")
        return True
    except cx_Oracle.Error as error:
        print("Ошибка вставки данных коллекции:", error)
        conn.rollback()
    finally:
        conn.close()
    return False

# Функция для обработки события нажатия кнопки для выбора
фото
def select_photo():
    # Запрос пользователю выбрать файл фото
    photo_path = filedialog.askopenfilename()
    photo_path_entry.delete(0, tk.END)
    photo_path_entry.insert(0, photo_path)

# Функция для обработки события нажатия кнопки для добавле-
ния коллекции
def add_collection():
    # Получение данных из полей ввода
    user_id = user_id_entry.get()
    collection_name = collection_name_entry.get()
    collection_description = collection_description_en-
try.get()
    photo_path = photo_path_entry.get()
    # Получение email пользователя
    user_email = fetch_user_email(user_id)
    if user_email:
        # Вставка данных коллекции в базу данных
        if insert_collection_data(user_id, collection_name,
collection_description, photo_path):
            # Отправка email-уведомления
            send_email_notification(user_email, collec-
tion_name, collection_description, photo_path)
            # Очистка полей ввода
            user_id_entry.delete(0, tk.END)
            collection_name_entry.delete(0, tk.END)
            collection_description_entry.delete(0, tk.END)
            photo_path_entry.delete(0, tk.END)
            status_label.config(text="Коллекция успешно до-
бавлена и уведомление по электронной почте отправлено.")
            return
        status_label.config(text="Ошибка добавления коллекции:
Пользователь не найден или неверный UserID.")

# Функция для получения email пользователя из базы данных
def fetch_user_email(user_id):
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        cursor.execute("SELECT email FROM Users WHERE UserID
= :id", id=user_id)
        result = cursor.fetchone()

```

```

        if result:
            user_email = result[0]
            conn.close()
            return user_email
        else:
            print("Ошибка: Пользователь не найден с ID:",
user_id)
            conn.close()
        else:
            print("Ошибка: Не удалось подключиться к базе дан-
ных.")
            return None

# Функция для отправки email-уведомления
def send_email_notification(user_email, collection_name,
collection_description, photo_path):
    # Данные сервера электронной почты
    smtp_server = 'smtp.gmail.com'
    smtp_port = 587
    smtp_username = '*****@gmail.com'
    smtp_password = '*****'
    # Содержимое email
    sender_email = '*****@gmail.com'
    receiver_email = user_email
    subject = 'Добавлена новая коллекция'
    body = f'Здравствуйте,\n\nВаша коллекция «{collec-
tion_name}» успешно добавлена.\n'
    body += f'Описание коллекции: {collection_descrip-
tion}\n'
    body += f'Путь к фото: {photo_path}'
    # Создание многочастного сообщения и установка заголов-
ков
    message = MIMEMultipart()
    message['From'] = sender_email
    message['To'] = receiver_email
    message['Subject'] = subject
    # Добавление текста сообщения
    message.attach(MIMEText(body, 'plain'))
    # Подключение к SMTP-серверу и отправка email
    try:
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(smtp_username, smtp_password)
        text = message.as_string()
        server.sendmail(sender_email, receiver_email, text)
        print("Email-уведомление успешно отправлено!")
        server.quit()
        return True
    except Exception as e:
        print("Ошибка отправки email-уведомления:", e)
        return False

# Создание главного окна приложения
root = tk.Tk()

```

```

root.title("NFT Marketplace и Менеджер Коллекций")

# Фрейм для всех NFT
nft_frame = tk.Frame(root)
nft_frame.pack()

# Функция для получения и отображения NFT
def display_nfts():
    nfts = fetch_nfts()
    if nfts:
        for nft_id, title, price in nfts:
            nft_data = f"{nft_id}) {title} - {price}$"
            process_and_display_image(nft_id, nft_frame,
nft_data)
    else:
        nft_label = tk.Label(nft_frame, text="Ошибка при
получении NFT из базы данных")
        nft_label.pack()

# Кнопка для получения и отображения NFT
fetch_button = tk.Button(root, text="Получить NFT", com-
mand=display_nfts)
fetch_button.pack()

# Метка и поле ввода для UserID
user_id_label = tk.Label(root, text="UserID:")
user_id_label.pack()
user_id_entry = tk.Entry(root)
user_id_entry.pack()

# Метка и поле ввода для имени коллекции
collection_name_label = tk.Label(root, text="Имя
коллекции:")
collection_name_label.pack()
collection_name_entry = tk.Entry(root)
collection_name_entry.pack()

# Метка и поле ввода для описания коллекции
collection_description_label = tk.Label(root, text="Описание
коллекции:")
collection_description_label.pack()
collection_description_entry = tk.Entry(root)
collection_description_entry.pack()

# Метка и поле ввода для пути к фото
photo_path_label = tk.Label(root, text="Путь к фото:")
photo_path_label.pack()
photo_path_entry = tk.Entry(root)
photo_path_entry.pack()

# Кнопка для выбора файла фото
browse_button = tk.Button(root, text="Выбрать", command=se-
lect_photo)
browse_button.pack()

```

```

# Кнопка для добавления коллекции
add_button = tk.Button(root, text="Добавить коллекцию", com-
mand=add_collection)
add_button.pack()

# Метка для статуса сообщения
status_label = tk.Label(root, text="")
status_label.pack()

# Метка для отображения данных NFT
nft_label = tk.Label(root, text="")
nft_label.pack()

# Запуск цикла обработки событий Tkinter
root.mainloop()

```

Листинг 2.7 – Мини-приложение демонстрирующее работу

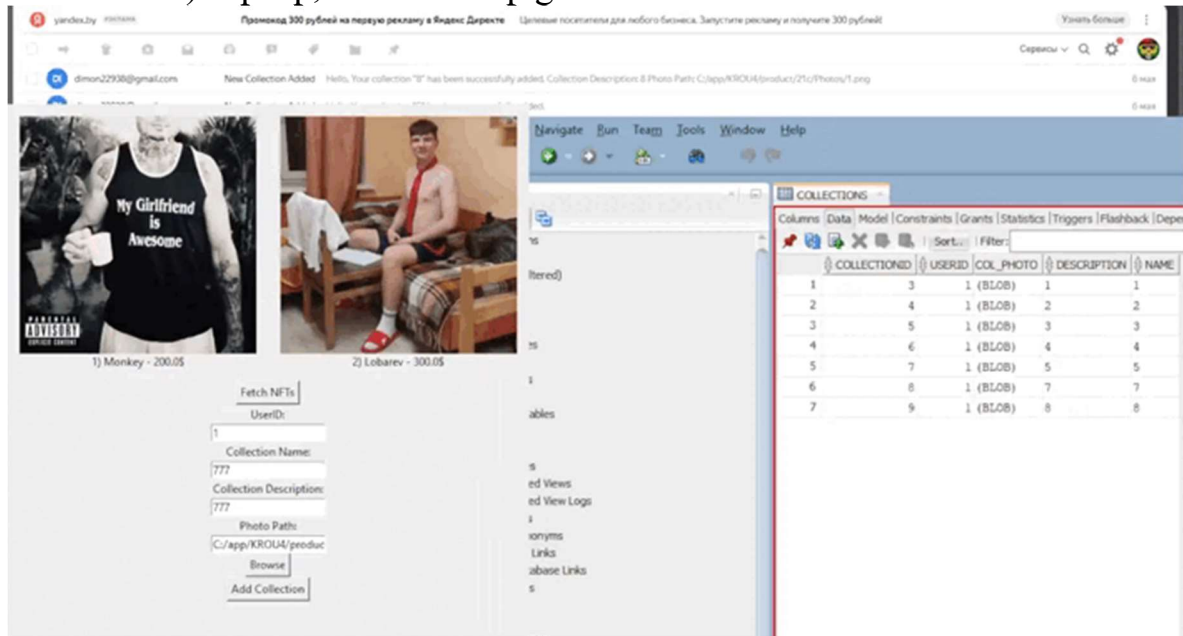
Приложение выглядит следующим образом:

Gif-изображение 2.8 – Демонстрация технологии работы с мультимедиа

При нажатии кнопки Fetch NFT программа посылает запрос в БД, извлекая данные, попутно преобразуя BLOB изображение в PNG через функцию blob\_to\_png внутри Oracle DB.

Далее рассмотрим функционал кнопки Add Collection. Для начала вводим всю необходимую информацию, а также добавляем фото для коллекции. Фото преобразуется в BLOB через процедуру Insert\_NFT\_Photo внутри Oracle DB.

Как только в таблицу Collections добавится новая строка, то сработает триггер **trg\_after\_collection\_insert**, который в свою очередь запускает процедуры **dequeue\_and\_send\_emails** и **enqueue\_collection\_notification**. Одна создает шаблон письма, другая отправляет письмо через собственный SMTP (Simple Mail Transfer Protocol) сервер, на базе smtp.gmail.com.



Gif-изображение 2.9 – Демонстрация технологии отправки Email уведомлений о состоянии БД.



## Руководство пользователя: База данных «NFT Marketplace»

Функция: blob\_to\_png

Описание: Конвертирует BLOB изображение в файл PNG.

```
DECLARE
    l_result VARCHAR2(255);
BEGIN
    l_result := blob_to_png(
        p_blob => (SELECT user_photo FROM Users WHERE UserID
= 1),
        p_file_name => 'user_photo.png'
    );
    DBMS_OUTPUT.PUT_LINE(l_result);
END;
```

Функция: create\_user\_with\_privileges

Описание: Создает нового пользователя с заданными правами.

```
DECLARE
    l_result VARCHAR2(255);
BEGIN
    l_result := create_user_with_privileges(
        p_username => 'new_user',
        p_password => 'password123',
        p_email => 'new_user@example.com',
        p_tel => '+1234567890'
    );
    DBMS_OUTPUT.PUT_LINE(l_result);
END;
```

Процедура: add\_nft

Описание: Добавляет новый NFT в базу данных.

```
BEGIN
    add_nft(
        p_title => 'NFT Title',
        p_price => 100.00,
        p_collection_id => 1,
        p_owner_id => 1,
        p_photo => (SELECT user_photo FROM Users WHERE
UserID = 1)
    );
END;
```

Процедура: create\_collection

Описание: Создает новую коллекцию NFT

```
BEGIN
    create_collection(
        p_user_id => 1,
        p_name => 'Collection Name',
        p_description => 'Collection description',
```

```

        p_col_photo => (SELECT user_photo FROM Users WHERE
UserID = 1)
    );
END;
```

#### Процедура: define\_roles

Описание: Определяет роль пользователя (администратор или обычный пользователь).

```

BEGIN
    define_roles(user_id => 1);
END;
```

#### Процедура: delete\_nft

Описание: Удаляет NFT из базы данных.

```

BEGIN
    delete_nft(p_nft_id => 1, p_owner_id => 1);
END;
```

#### Процедура: enqueue\_collection\_notification

Описание: Добавляет сообщение о создании новой коллекции в очередь «COLLECTION\_QUEUE».

```

BEGIN
    enqueue_collection_notification(
        p_collection_id => 1,
        p_user_id => 1,
        p_collection_name => 'Collection Name'
    );
END;
```

#### Процедура: export\_NFT\_to\_XML

Описание: Экспортирует данные NFT в XML файл.

```

BEGIN
    export_NFT_to_XML(filename => 'nfts.xml');
END;
```

#### Процедура: IMPORT\_NFT\_FROM\_XML

Описание: Импортирует данные NFT из XML файла.

```

BEGIN
    IMPORT_NFT_FROM_XML(filename => 'nfts.xml');
END;
```

#### Процедура: Insert\_NFT\_Photo

Описание: Добавляет фотографию NFT в базу данных.

```

BEGIN
    Insert_NFT_Photo(
```

```

        p_nft_id => 1,
        p_photo_path => 'nft_photo.png'
    );
END;
```

#### Процедура: rate\_nft

Описание: Добавляет или обновляет рейтинг NFT.

```

BEGIN
    rate_nft(
        p_user_id => 1,
        p_nft_id => 1,
        p_rating => 5
    );
END;
```

#### Процедура: search\_nfts

Описание: Ищет NFT по заданным параметрам.

```

BEGIN
    search_nfts(
        p_search_term => 'NFT',
        p_collection_id => 1,
        p_status => 'Available',
        p_min_price => 50.00,
        p_max_price => 200.00
    );
END;
```

#### Процедура: view\_user\_info

Описание: Показывает информацию о пользователях (только для администраторов).

```

BEGIN
    view_user_info(p_admin_id => 1);
END;
```

#### Процедура: add\_comment

Описание: Добавляет комментарий к NFT.

```

BEGIN
    add_comment(
        p_UserID => 1,
        p_NFTID => 1,
        p_CommentText => 'This is a great NFT!'
    );
END;
```

## Заключение

В процессе реализации базы данных для «NFT Marketplace» было проделано значительное количество работы, что позволило создать функциональную и надежную систему, способную удовлетворить потребности как пользователей, так и администраторов. Разработка базы данных в Oracle Database и использование хранимых процедур, триггеров и представлений позволило обеспечить эффективное управление NFT и связанными данными.

Проект успешно реализовал все функциональные требования, включая определение ролей, добавление и удаление NFT, просмотр информации о пользователях, поиск NFT по различным критериям, оценку NFT и создание коллекций. Эти возможности сделали платформу привлекательной и удобной для пользователей, обеспечивая им широкий спектр функциональности для работы с цифровыми активами.

Кроме того, проведенные тесты производительности позволили выявить потенциальные узкие места и оптимизировать работу базы данных для обеспечения стабильной и эффективной работы даже при больших объемах данных.

В целом, разработанная база данных представляет собой важное достижение в развитии платформы «NFT Marketplace», обеспечивая ее надежность, функциональность и готовность к масштабированию в будущем.

**Список использованных источников**

1. OpenSea [Электронный ресурс] / Режим доступа: URL <https://www.opensea.com/> - Дата доступа: 22.04.2024
2. SuperRare [Электронный ресурс] / Режим доступа: URL <https://www.superrare.com/> - Дата доступа: 22.04.2024
3. Foundation [Электронный ресурс] / Режим доступа: URL <https://www.foundation.com/> - Дата доступа: 22.04.2024
4. SQL и реляционные базы данных: учебное пособие / М. П. Баранов, С. М. Бабенко, Ю. В. Михайленко. – М.: Издательский центр «Академия», 2019. - 200 с.

## Приложение А - Таблицы

```
-- Создание таблицы Users
CREATE TABLE Users (
  UserID INT PRIMARY KEY,
  UserName VARCHAR2(50) NOT NULL,
  Email VARCHAR2(100) UNIQUE NOT NULL,
  Password VARCHAR2(64) NOT NULL,
  user_photo BLOB,
  tel VARCHAR2(20),
  CONSTRAINT check_email CHECK (REGEXP_LIKE(Email, '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'))
);

-- Создание таблицы Admins
CREATE TABLE Admins (
  AdminID INT PRIMARY KEY,
  UserID INT UNIQUE NOT NULL,
  admin_status VARCHAR2(20),
  AdminIn datetime,
  CONSTRAINT fk_admin_user FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- Создание таблицы NFT
CREATE TABLE NFT (
  NFTID INT PRIMARY KEY,
  Title VARCHAR2(100) NOT NULL,
  Price DECIMAL(18,2),
  Status VARCHAR2(20),
  CollectionID INT,
  TimeAdded datetime,
  Owner INT,
  CONSTRAINT fk_nft_collection FOREIGN KEY (CollectionID) REFERENCES Collections(CollectionID),
  CONSTRAINT fk_nft_user FOREIGN KEY (Owner) REFERENCES Users(UserID)
);

-- Создание таблицы Collections
CREATE TABLE Collections (
  CollectionID INT PRIMARY KEY,
  UserID INT NOT NULL,
  col_photo BLOB,
  description VARCHAR2(1000),
  name VARCHAR2(50),
  CONSTRAINT fk_collection_user FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- Создание таблицы Transactions
CREATE TABLE Transactions (
  TransactionID INT PRIMARY KEY,
```

```

    BuyerID INT NOT NULL,
    NFTID INT NOT NULL,
    TransactionStatus BIT NOT NULL,
    seller_id INT NOT NULL,
    transaction_time datetime,
    CONSTRAINT fk_transaction_buyer FOREIGN KEY (BuyerID) REFER-
ENCES Users(UserID),
    CONSTRAINT fk_transaction_nft FOREIGN KEY (NFTID) REFERENCES
NFT(NFTID),
    CONSTRAINT fk_transaction_seller FOREIGN KEY (seller_id) REFER-
ENCES Users(UserID)
);

-- Создание таблицы Comments
CREATE TABLE Comments (
    CommentID INT PRIMARY KEY,
    UserID INT NOT NULL,
    NFTID INT NOT NULL,
    CommentText VARCHAR2(4000),
    like_col INT DEFAULT 0,
    time_posted datetime,
    CONSTRAINT fk_comment_user FOREIGN KEY (UserID) REFERENCES Us-
ers(UserID),
    CONSTRAINT fk_comment_nft FOREIGN KEY (NFTID) REFERENCES
NFT(NFTID)
);

```

## Приложение Б – Процедуры

```

create or replace NONEDITIONABLE PROCEDURE add_nft (
  p_title IN NFT.Title%TYPE,
  p_price IN NFT.Price%TYPE,
  p_collection_id
  IN NFT.CollectionID%TYPE,
  p_owner_id IN NFT.Owner%TYPE,
  p_photo IN NFT.Photo%TYPE
)
AS
BEGIN
  INSERT INTO NFT
    (Title, Price, Status, CollectionID, TimeAdded, Owner, Photo)
  VALUES (p_title, p_price, 'Available', p_collection_id,
    SYSDATE, p_owner_id, p_photo);

  DBMS_OUTPUT.PUT_LINE('NFT
  added successfully!');

  EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error
    adding NFT: ' ||
    SQLERRM);
END;

```

### Приложение Б1 – SQL код для процедуры add\_nft

```

create or replace NONEDITIONABLE PROCEDURE create_collection (
  p_user_id IN Collections.UserID%TYPE,
  p_name IN Collections.name%TYPE,
  p_description
  IN Collections.description%TYPE,

```



```

p_col_photo IN Collections.col_photo%TYPE DEFAULT NULL
)
AS
BEGIN
INSERT INTO Collections
(UserID, name, description,
col_photo)
VALUES (p_user_id, p_name, p_description,
p_col_photo);

DBMS_OUTPUT.PUT_LINE('Collection
created successfully!');
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error
creating collection: ' ||
SQLERRM);
END;

```

## Приложение Б2 – SQL код для процедуры create\_collection

```

create or replace NONEDITIONABLE PROCEDURE define_roles (
user_id IN INT
)
AS
user_role VARCHAR2(20);
BEGIN
SELECT admin_status
INTO user_role
FROM Admins
WHERE UserID = user_id;

IF user_role IS NOT NULL THEN
DBMS_OUTPUT.PUT_LINE('Role:
Administrator');

```

```

ELSE
DBMS_OUTPUT.PUT_LINE('Role:
User');
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Role:
User');
--
Assuming users without an admin record are regular users
END;
```

### Приложение Б3 – SQL код для процедуры define\_roles

```

create or replace NONEDITIONABLE PROCEDURE delete_nft (
p_nft_id IN NFT.NFTID%TYPE,
p_owner_id IN NFT.Owner%TYPE
)
AS
nft_owner_id
NFT.Owner%TYPE;
BEGIN
--
Check if the NFT exists and is owned by the user
SELECT Owner
INTO nft_owner_id
FROM NFT
WHERE NFTID = p_nft_id;

IF
nft_owner_id = p_owner_id THEN
DELETE FROM NFT
WHERE NFTID = p_nft_id;
DBMS_OUTPUT.PUT_LINE('NFT
deleted successfully!');
```

```

ELSE
DBMS_OUTPUT.PUT_LINE('You
are not authorized to delete this NFT. ');
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NFT
not found. ');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error
deleting NFT: ' ||
SQLERRM);
END;
```

#### Приложение Б4 – SQL код для процедуры delete\_nft

```

create or replace NONEDITIONABLE PROCEDURE
dequeue_and_send_emails AS
    l_dequeue_options
DBMS_AQ.dequeue_options_t;
    l_message_properties
DBMS_AQ.message_properties_t;
    l_message_id          RAW(16);
    l_payload
SYS.AQ$_JMS_TEXT_MESSAGE;
    l_message
VARCHAR2(4000);
    l_user_id
NUMBER;
    l_user_email
VARCHAR2(100);
    l_mail_conn
UTL_SMTP.connection;
BEGIN
    DBMS_AQ.dequeue (
```

```

queue_name
=> 'COLLECTION_QUEUE',
dequeue_options
=> l_dequeue_options,
message_properties
=> l_message_properties,
payload
=> l_payload,
msgid
=> l_message_id
);

l_message :=
l_payload.text_vc;

--

Extract User ID from the message
l_user_id :=
REGEXP_SUBSTR(l_message, 'User ID: (\d+)',
1, 1, NULL,
1);

--

Get user email based on User ID
SELECT Email INTO l_user_email FROM Users WHERE UserID =
l_user_id;

-- Configure SMTP settings
l_mail_conn
:= UTL_SMTP.open_connection('smtp.gmail.com', 587);
--

UTL_SMTP.helo(l_mail_conn,
'dimon22938@gmail.com');

```

```

UTL_SMTP.mail(l_mail_conn,
'dimon22938@gmail.com');
UTL_SMTP.rcpt(l_mail_conn,
l_user_email);

-- Send the email
UTL_SMTP.data(l_mail_conn,

'From:
«NFT Marketplace» < dimon22938@gmail.com >' ||
UTL_TCP.crlf ||
'To:
' || l_user_email || UTL_TCP.crlf ||
'Subject:
New Collection Created!' ||
UTL_TCP.crlf ||
l_message || UTL_TCP.crlf
);

UTL_SMTP.quit(l_mail_conn);

COMMIT;

EXCEPTION

WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('User
ID not found in message: ' ||
l_message);
WHEN OTHERS THEN
ROLLBACK;

-- Handle other exceptions and logging
DBMS_OUTPUT.PUT_LINE('Error
sending email: ' ||
SQLERRM);

```

END;

## Приложение Б5 – SQL код для процедуры dequeue\_and\_send\_emails

```

create or replace NONEDITIONABLE PROCEDURE enqueue_collection_no-
tification (
    p_collection_id
    IN NUMBER,
    p_user_id IN NUMBER,
    p_collection_name IN VARCHAR2
) AS
    l_enqueue_options
    DBMS_AQ.enqueue_options_t;
    l_message_properties
    DBMS_AQ.message_properties_t;
    l_message_id      RAW(16);
    l_payload
    SYS.AQ$_JMS_TEXT_MESSAGE;
BEGIN
    l_payload :=
    SYS.AQ$_JMS_TEXT_MESSAGE.construct();
    l_payload.set_text('New
collection created: ' ||
p_collection_name || ' (ID: ' ||
p_collection_id || ') by User ID: ' || p_user_id);

    DBMS_AQ.enqueue (
        queue_name
        => 'COLLECTION_QUEUE',
        enqueue_options
        => l_enqueue_options,
        message_properties
        => l_message_properties,
        payload
        => l_payload,

```

```

msgid
=> l_message_id
);
COMMIT;
END;

```

## Приложение Б6 – SQL код для процедуры enqueue\_collection\_notification

```

create or replace NONEDITIONABLE PROCEDURE ex-
port_NFT_to_XML(filename IN VARCHAR2) AS
v_file
UTL_FILE.FILE_TYPE;
v_xml XMLTYPE;
BEGIN
-- Создание XMLTYPE объекта с данными из таблицы NFT
SELECT XMLELEMENT («NFTs»,
XMLAGG (XMLELEMENT («NFT»,
XMLFOREST (NFTID
AS «NFTID»,
Title
AS «Title»,
Price
AS «Price»,
Status
AS «Status»,
CollectionID
AS «CollectionID»,
TimeAdded
AS «TimeAdded»,
Owner
AS «Owner»)))
INTO v_xml
FROM NFT;

-- Создание файла и запись в него XML данных

```

```

v_file :=
UTL_FILE.FOPEN('XML_DIR', filename, 'W');
UTL_FILE.PUT_LINE(v_file,
v_xml.getClobVal());
UTL_FILE.FCLOSE(v_file);
EXCEPTION
WHEN OTHERS THEN
IF
UTL_FILE.IS_OPEN(v_file) THEN
UTL_FILE.FCLOSE(v_file);
END IF;
RAISE;
END;

```

### Приложение Б7 – SQL код для процедуры export\_NFT\_to\_XML

```

create or replace NONEDITIONABLE PROCEDURE IM-
PORT_NFT_FROM_XML(filename IN VARCHAR2) AS
v_file
UTL_FILE.FILE_TYPE;
v_line VARCHAR2(32767); -- Line buffer
v_xml XMLTYPE;
BEGIN
--
Open the XML file for reading
v_file
:= UTL_FILE.FOPEN('XML_DIR', filename, 'R');

--
Read the XML data from the file
UTL_FILE.GET_LINE(v_file,
v_line);
v_xml
:= XMLTYPE(v_line);

```



```

--
Insert the data into the NFT table
INSERT INTO NFT
(NFTID, Title, Price, Status, CollectionID, TimeAdded, Owner)
SELECT x.NFTID,
x.Title,
x.Price,
x.Status,
x.CollectionID,
TO_TIMESTAMP(x.TimeAdded,
'YYYY-MM-DD
HH24:MI:SS'),
x.Owner
FROM XMLTABLE('/NFTs/NFT'
PASSING
v_xml
COLUMNS
NFTID
INT PATH 'NFTID',
Title
VARCHAR2(100) PATH 'Title',
Price
DECIMAL(18,2) PATH 'Price',
Status
VARCHAR2(20) PATH 'Status',
CollectionID
INT PATH 'CollectionID',
TimeAdded
VARCHAR2(20) PATH 'TimeAdded',
Owner
INT PATH 'Owner') AS x;

-- Close the file
UTL_FILE.FCLOSE(v_file);

```

```

EXCEPTION
WHEN OTHERS THEN
IF
UTL_FILE.IS_OPEN(v_file) THEN
UTL_FILE.FCLOSE(v_file);
END IF;
RAISE;
END;

```

### Приложение Б8 – SQL код для процедуры IMPORT\_NFT\_FROM\_XML

```

create or replace NONEDITIONABLE PROCEDURE Insert_NFT_Photo(
p_nft_id INT,
p_photo_path VARCHAR2
) AS
v_blob BLOB;
v_blob_length INTEGER;
v_file
UTL_FILE.FILE_TYPE;
BEGIN
-- Чтение файла PNG в BLOB переменную
DBMS_LOB.CREATETEMPORARY(v_blob,
TRUE);
v_file :=
UTL_FILE.FOPEN('PHOTO_DIR', p_photo_path, 'r');
UTL_FILE.GET_RAW(v_file,
v_blob);
v_blob_length
:= DBMS_LOB.GETLENGTH(v_blob);
UTL_FILE.FCLOSE(v_file);

-- Обновление фотографии в таблице NFT
UPDATE NFT
SET PHOTO = v_blob
WHERE NFTID = p_nft_id;

```

```

COMMIT;

DBMS_OUTPUT.PUT_LINE('Photo
added successfully for NFTID: ' ||
p_nft_id);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NFT
not found with ID: ' ||
p_nft_id);
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error:
' || SQLERRM);
END Insert_NFT_Photo;

```

### Приложение Б9 – SQL код для процедуры Insert\_NFT\_Photo

```

create or replace NONEDITIONABLE PROCEDURE rate_nft (
p_user_id
IN Comments.UserID%TYPE,
p_nft_id
IN Comments.NFTID%TYPE,
p_rating
IN Comments.like_col%TYPE
)
AS
existing_rating
Comments.like_col%TYPE;
BEGIN
--
Check if the user has already rated the NFT
SELECT like_col
INTO existing_rating
FROM Comments

```

```

WHERE UserID = p_user_id AND NFTID = p_nft_id;

IF
existing_rating IS NULL THEN
-- Insert new rating
INSERT INTO Comments
(UserID, NFTID, like_col, time_posted)
VALUES (p_user_id, p_nft_id, p_rating,
SYSDATE);
DBMS_OUTPUT.PUT_LINE('NFT
rated successfully!');
ELSE
-- Update existing rating
UPDATE Comments
SET like_col = p_rating, time_posted =
SYSDATE
WHERE UserID = p_user_id AND NFTID = p_nft_id;
DBMS_OUTPUT.PUT_LINE('NFT
rating updated successfully!');
END IF;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error
rating NFT: ' ||
SQLERRM);
END;
```

### Приложение Б10 – SQL код для процедуры rate\_nft

```

create or replace NONEDITIONABLE PROCEDURE search_nfts (
p_search_term
IN VARCHAR2 DEFAULT NULL,
p_collection_id
IN NFT.CollectionID%TYPE DEFAULT NULL,
p_status IN NFT.Status%TYPE DEFAULT NULL,
p_min_price IN NFT.Price%TYPE DEFAULT NULL,
```

```

p_max_price IN NFT.Price%TYPE DEFAULT NULL
)
AS
BEGIN
--
Build dynamic WHERE clause based on provided parameters
FOR nft_rec IN (
SELECT NFTID, Title, Price, Status,
CollectionID
FROM NFT
WHERE (p_search_term IS NULL OR Title LIKE '%' ||
p_search_term || '%')
AND (p_collection_id IS NULL OR CollectionID = p_collection_id)
AND (p_status IS NULL OR Status = p_status)
AND (p_min_price IS NULL OR Price >= p_min_price)
AND (p_max_price IS NULL OR Price <= p_max_price)
) LOOP
DBMS_OUTPUT.PUT_LINE('NFTID:
' || nft_rec.NFTID);
DBMS_OUTPUT.PUT_LINE('Title:
' || nft_rec.Title);
DBMS_OUTPUT.PUT_LINE('Price:
' || nft_rec.Price);
DBMS_OUTPUT.PUT_LINE('Status:
' || nft_rec.Status);
DBMS_OUTPUT.PUT_LINE('CollectionID:
' || nft_rec.CollectionID);
DBMS_OUTPUT.PUT_LINE('-----');
END LOOP;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error
searching NFTs: ' ||
SQLERRM);

```

END;

## Приложение Б11 – SQL код для процедуры search\_nfts

```

create or replace NONEDITIONABLE PROCEDURE view_user_info (
  p_admin_id IN Admins.AdminID%TYPE
)
AS
  user_role VARCHAR2(20);
BEGIN
  --
  Verify if the user is an admin
  SELECT admin_status
  INTO user_role
  FROM Admins
  WHERE AdminID = p_admin_id;

  IF user_role IS NOT NULL THEN
    -- Display user information
    FOR user_rec IN (SELECT UserID, UserName, Email, tel FROM Users)
    LOOP
      DBMS_OUTPUT.PUT_LINE('UserID:
        ' || user_rec.UserID);
      DBMS_OUTPUT.PUT_LINE('Username:
        ' || user_rec.UserName);
      DBMS_OUTPUT.PUT_LINE('Email:
        ' || user_rec.Email);
      DBMS_OUTPUT.PUT_LINE('Phone:
        ' || user_rec.tel);
      DBMS_OUTPUT.PUT_LINE('-----');
    END LOOP;
  ELSE
    DBMS_OUTPUT.PUT_LINE('Access
      denied. You must be an administrator to view user information.');
```

END IF;

EXCEPTION

```

WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No
users found. ');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Error:
' || SQLERRM);
END;
```

### Приложение Б12 – SQL код для процедуры view\_user\_info

```

CREATE OR REPLACE PROCEDURE add_comment (
p_UserID IN Comments.UserID%TYPE,
p_NFTID IN Comments.NFTID%TYPE,
p_CommentText IN Comments.CommentText%TYPE
)
AS
BEGIN
INSERT INTO Comments (UserID, NFTID, CommentText, time_posted)
VALUES (p_UserID, p_NFTID, p_CommentText, SYSDATE);

COMMIT;

END;
```

### Приложение Б13 – SQL код для процедуры add\_comment

```

CREATE OR REPLACE PROCEDURE like_comment (
p_CommentID IN Comments.CommentID%TYPE
)
AS
BEGIN
UPDATE Comments
SET like_col = like_col + 1
WHERE CommentID = p_CommentID;

COMMIT;

END;
```

### Приложение Б15 – SQL код для процедуры like\_comment

## Приложение В - Функции

```
create or replace NONEDITIONABLE FUNCTION blob_to_png (  
  p_blob IN BLOB,  
  p_file_name IN VARCHAR2  
) RETURN VARCHAR2  
AS  
  l_output_file  
  UTL_FILE.FILE_TYPE;  
  l_chunk_size  
  BINARY_INTEGER := 32767;  
  l_buffer RAW(32767);  
  l_blob_len INTEGER;  
BEGIN  
  -- Открытие файла для записи  
  l_output_file  
  := UTL_FILE.FOPEN('IMAGE_DIR', p_file_name, 'wb');  
  
  -- Получение длины BLOB  
  l_blob_len :=  
  DBMS_LOB.getlength(p_blob);  
  
  -- Чтение BLOB и запись в файл порциями  
  FOR i IN 1..CEIL(l_blob_len / l_chunk_size)  
  LOOP  
    DBMS_LOB.read(p_blob,  
      l_chunk_size, (i - 1) * l_chunk_size + 1, l_buffer);  
    UTL_FILE.PUT_RAW(l_output_file,  
      l_buffer, TRUE);  
  END LOOP;  
  
  -- Закрытие файла  
  UTL_FILE.FCLOSE(l_output_file);
```



```

-- Проверка успешности операции
IF
UTL_FILE.IS_OPEN(l_output_file) THEN
RETURN 'File overwritten successfully!';
ELSE
RETURN 'File created successfully!';
END IF;
EXCEPTION
WHEN OTHERS THEN
IF
UTL_FILE.IS_OPEN(l_output_file) THEN
UTL_FILE.FCLOSE(l_output_file);
END IF;
RETURN 'Error creating file: ' || SQLERRM;
END;

```

## Приложение B1 – SQL код для функции blob\_to\_png

```

create or replace NONEDITIONABLE FUNCTION create_user_with_privi-
leges (
p_username IN VARCHAR2,
p_password IN VARCHAR2,
p_email IN VARCHAR2,
p_tel IN VARCHAR2
) RETURN VARCHAR2
AS
l_user_id NUMBER;
BEGIN
-- 1. Create User
EXECUTE IMMEDIATE 'CREATE USER ' || p_username || '
IDENTIFIED BY ' ||
p_password;

-- 2. Get User ID
SELECT user_id

```

```

    INTO l_user_id
  FROM all_users
  WHERE username = p_username;

--

3. Insert User Data into Users Table
INSERT INTO Users
  (UserID, UserName, Email, Password, tel)
VALUES (l_user_id, p_username, p_email,
  p_password, p_tel);

-- 4. Grant Basic Privileges
EXECUTE IMMEDIATE 'GRANT CONNECT,
  RESOURCE TO ' ||
  p_username;

--

5. Grant Object Privileges (Customize as needed)
EXECUTE IMMEDIATE 'GRANT SELECT,
  INSERT, UPDATE, DELETE ON NFT TO ' ||
  p_username;

--

Add more grants for other tables or procedures as required

RETURN 'User created successfully!';
EXCEPTION
  WHEN OTHERS THEN
    RETURN 'Error creating user: ' || SQLERRM;
END;
```

## Приложение B2 – SQL код для функции create\_user\_with\_privileges

## Приложение Г - Триггеры

```

create or replace NONEDITIONABLE TRIGGER trg_after_collection_in-
sert
AFTER INSERT ON Collections
FOR EACH ROW
BEGIN
enqueue_collection_notification(:NEW.CollectionID,
:NEW.UserID, :NEW.name);
END;
```

### Приложение Г1 – SQL код для триггера trg\_after\_collection\_insert

```

CREATE OR REPLACE TRIGGER check_transaction_time_and_comments
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
nft_status NFT.Status%TYPE;
comment_count INTEGER;
BEGIN
-- Проверка статуса NFT
SELECT Status INTO nft_status
FROM NFT
WHERE NFTID = :NEW.NFTID;

IF nft_status <> 'Available' THEN
RAISE_APPLICATION_ERROR(-20001, 'NFT is not available for pur-
chase.');
```

```

END IF;

-- Проверка количества комментариев
SELECT COUNT(*) INTO comment_count
FROM Comments
WHERE NFTID = :NEW.NFTID;
```

```
IF comment_count < 5 THEN  
    RAISE_APPLICATION_ERROR(-20002, 'NFT must have at least 5 com-  
ments before it can be purchased.');
```

---

```
END IF;  
END;
```

Приложение Г2 – SQL код для триггера check\_transaction\_time\_and\_comments

## Приложение Д – Представления

```
CREATE OR REPLACE VIEW ActiveNFTs AS
SELECT NFTID, Title, Price, CollectionID, Owner
FROM NFT
WHERE Status = 'Available';
```

### Приложение Д1 – SQL код для представления ActiveNFTs

```
CREATE OR REPLACE VIEW CollectionsWithNFTCount AS
SELECT c.CollectionID, c.name, c.description, c.UserID,
COUNT(n.NFTID) AS NFTCount
FROM Collections c
LEFT JOIN NFT n ON c.CollectionID = n.CollectionID
GROUP BY c.CollectionID, c.name, c.description, c.UserID;
```

### Приложение Д2 – SQL код для представления CollectionsWithNFTCount

```
CREATE OR REPLACE VIEW TransactionDetails AS
SELECT
t.TransactionID,
t.NFTID,
n.Title AS NFTTitle,
t.BuyerID,
b.UserName AS BuyerName,
t.seller_id,
s.UserName AS SellerName,
t.TransactionStatus,
t.transaction_time
FROM Transactions t
JOIN NFT n ON t.NFTID = n.NFTID
JOIN Users b ON t.BuyerID = b.UserID
JOIN Users s ON t.seller_id = s.UserID;
```

### Приложение Д3 – SQL код для представления TransactionDetails