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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Велютич Д. И.

подпись, дата

**Руководитель**

преподаватель-стажер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласевич Н.И.

должность, учен. степень, ученое звание подпись, дата

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласевич Н.И.

подпись дата инициалы и фамилия

Минск 2024

**Содержание**

[Введение 2](#_Toc166786809)

[Постановка задачи 3](#_Toc166786810)

[1.Анализ требований 4](#_Toc166786811)

[1.1. Обзор аналогов 4](#_Toc166786812)

[1.2. Разработка функциональных требований 10](#_Toc166786813)

[1.3. Вывод по разделу 10](#_Toc166786814)

[2. Разработка архитектуры проекта 11](#_Toc166786815)

[2.1. Структура управлением приложения](#_Toc166786816) ………...………………..……..……11

[2.2. Диаграммы UML, взаимосвязь компонентов 11](#_Toc166786817)

[2.3. Описание объектов и ограничение целостности 12](#_Toc166786818)

[2.4. Вывод по разделу 15](#_Toc166786819)

[3. Разработка модели базы данных. 16](#_Toc166786820)

[3.1 Создание необходимых объектов. 16](#_Toc166786821)

[3.1.1 Таблицы 16](#_Toc166786822)

[3.1.2 Процедуры 16](#_Toc166786823)

[3.1.3 Функции 16](#_Toc166786824)

[3.1.4 Триггеры 17](#_Toc166786825)

[3.1.5 Индексы 17](#_Toc166786826)

[3.1.6 Представления 18](#_Toc166786827)

[3.2. Описание используемой технологии. 19](#_Toc166786828)

[3.3 Описание технологии импорта и экспорта. 20](#_Toc166786829)

[Тестирование производительности 22](#_Toc166786830)

[Краткое описание приложения для демонстрации 25](#_Toc166786831)

[Руководство пользователя 32](#_Toc166786832)

[Заключение 35](#_Toc166786833)

[Список использованных источников 36](#_Toc166786834)

[Приложение А - Таблицы 37](#_Toc166786835)

[Приложение Б – Процедуры 39](#_Toc166786836)

[Приложение В - Функции 55](#_Toc166786837)

[Приложение Г - Триггеры 58](#_Toc166786838)

[Приложение Д – Представления 60](#_Toc166786839)

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 соответству:ющей 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.

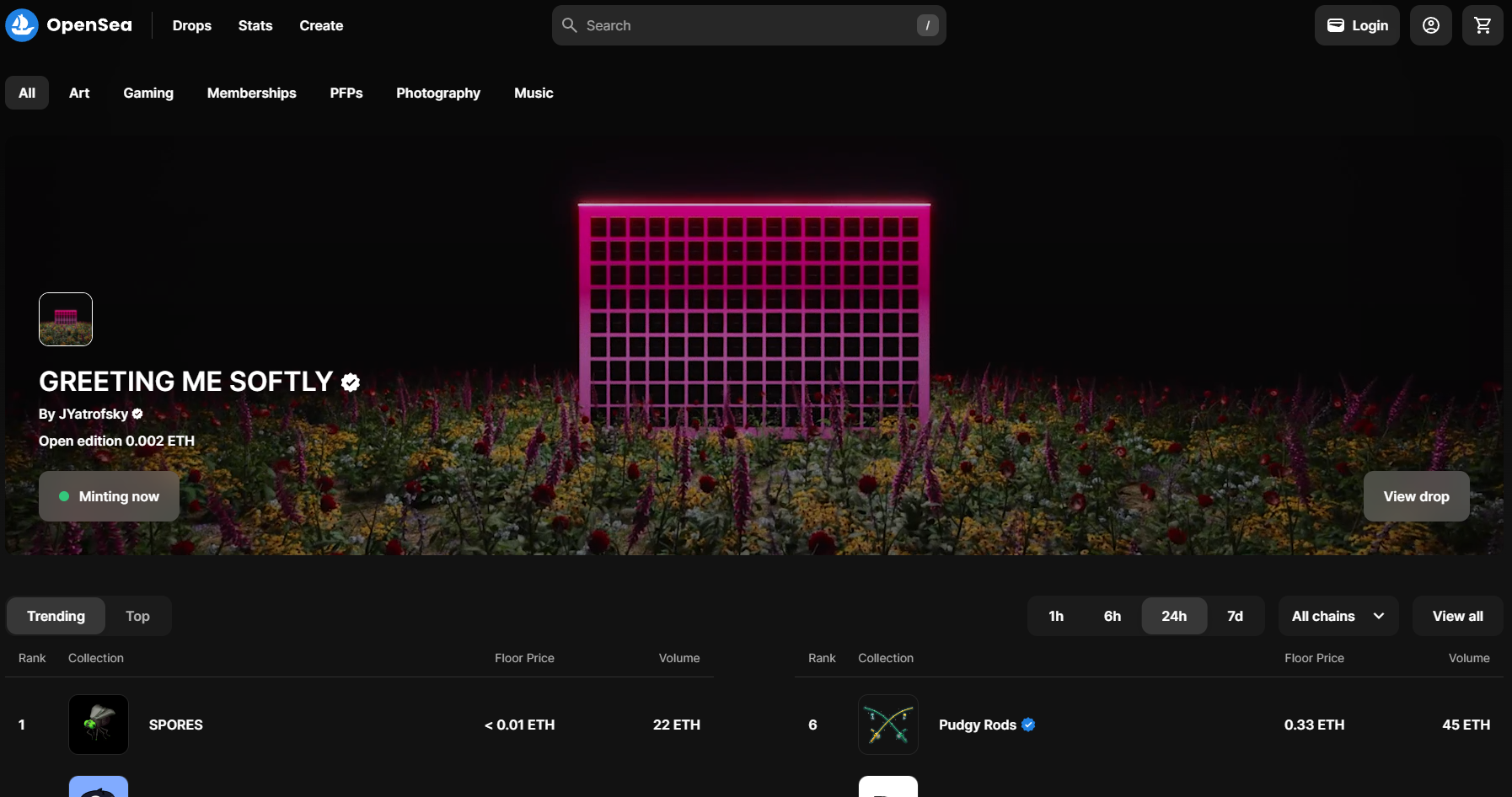
1.Анализ требований

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.



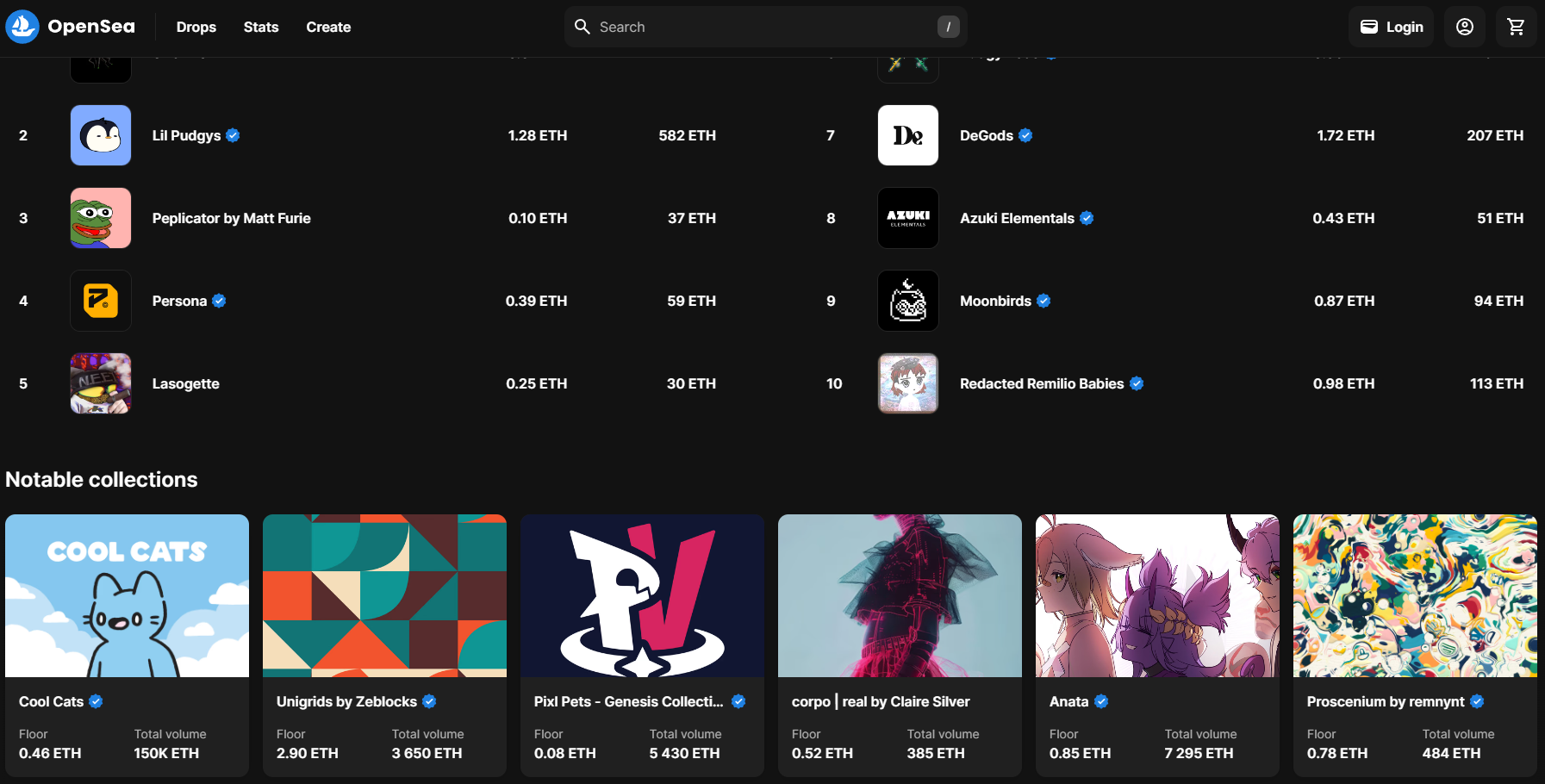
Изображение 1.1 — Главная страница OpenSea

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.

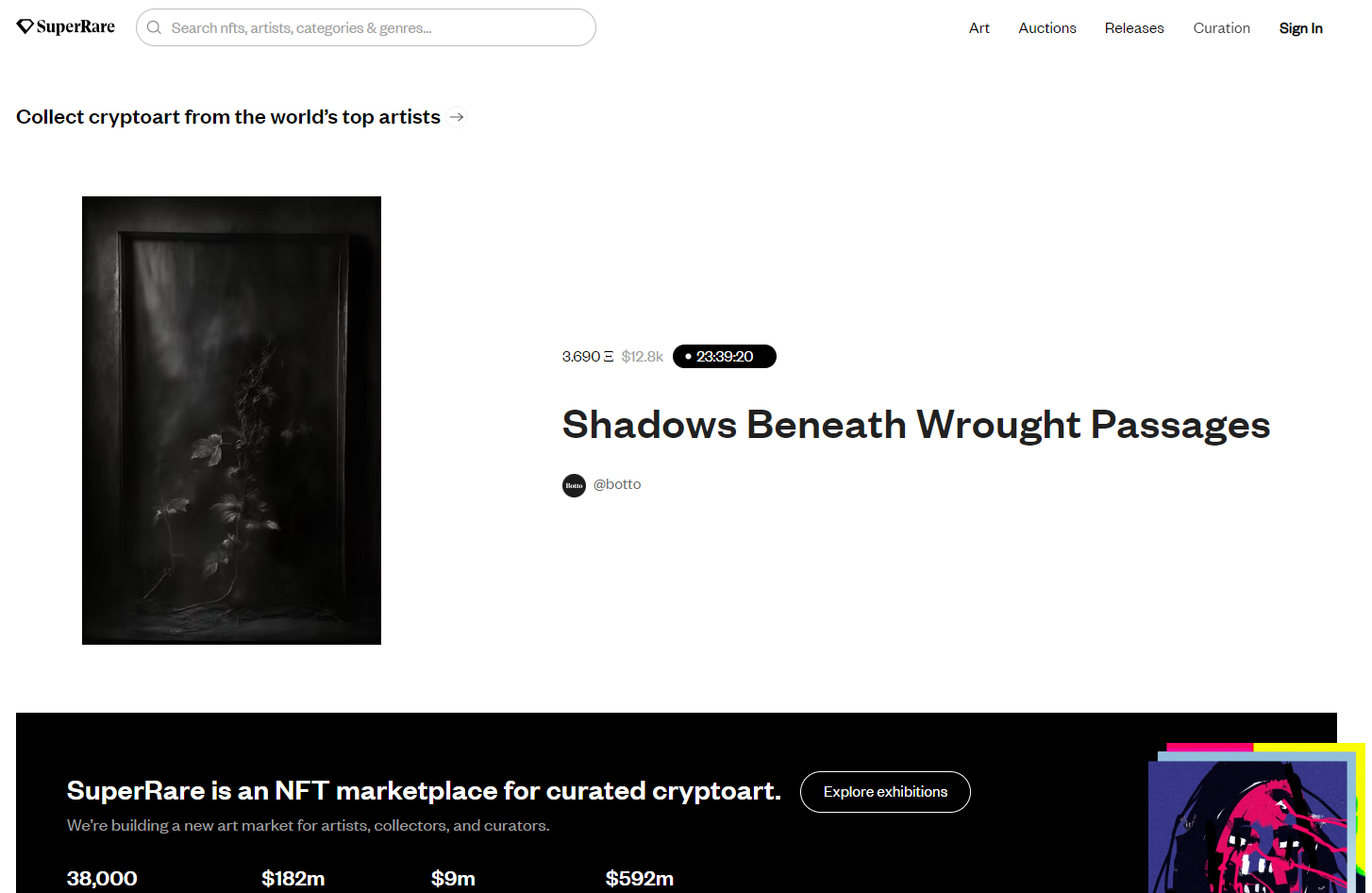


Изображение 1.3 — Коллекции OpenSea

**1.1.2. Аналог - SuperRare**

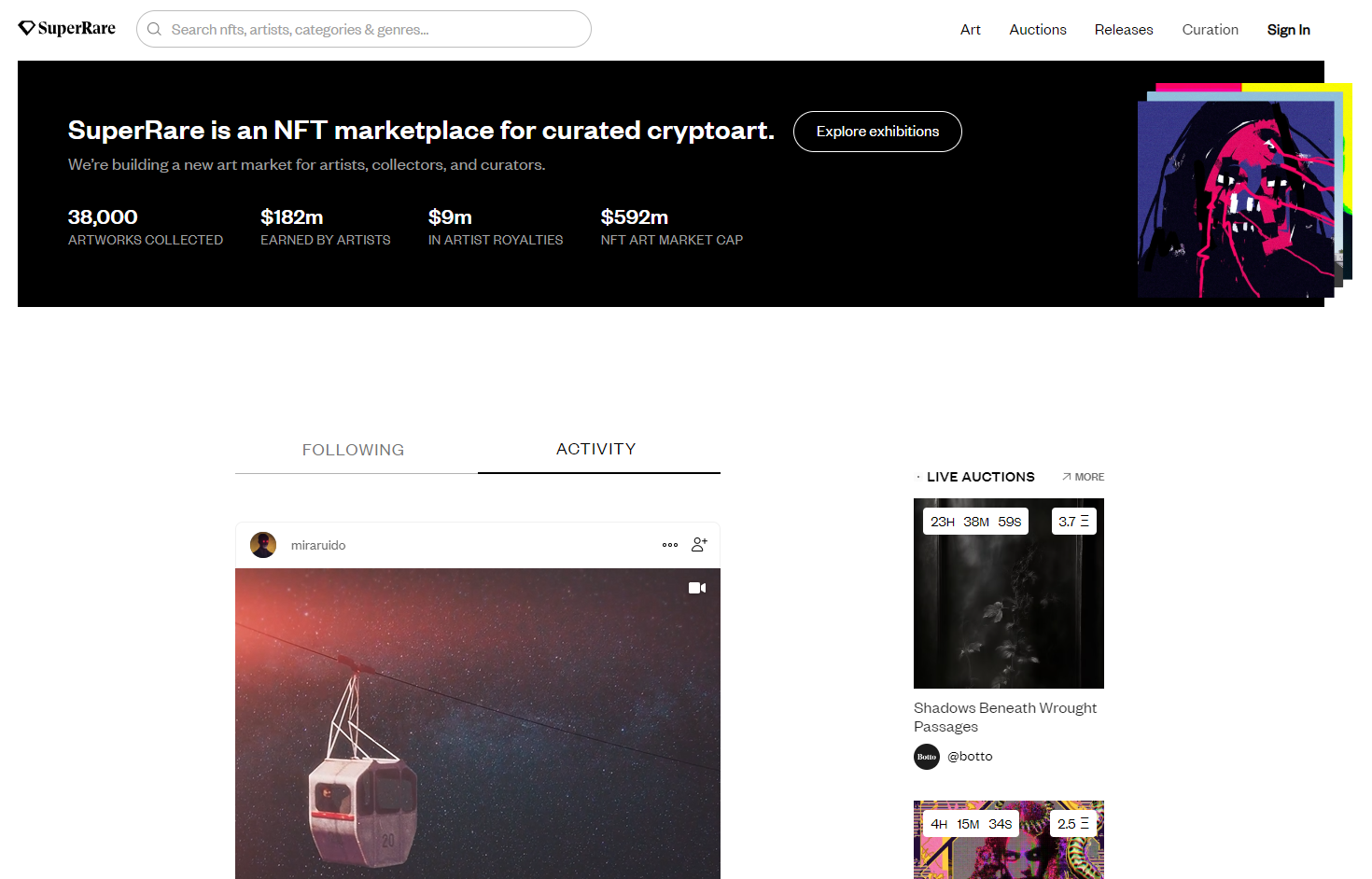
Уникальные особенности: SuperRare предоставляет пользователю возможность общаться с художниками и коллекционерами напрямую, что создает уникальное сообщество ценителей искусства NFT. Кроме того, платформа активно поддерживает новых и перспективных художников, предоставляя им шанс на успешное развитие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

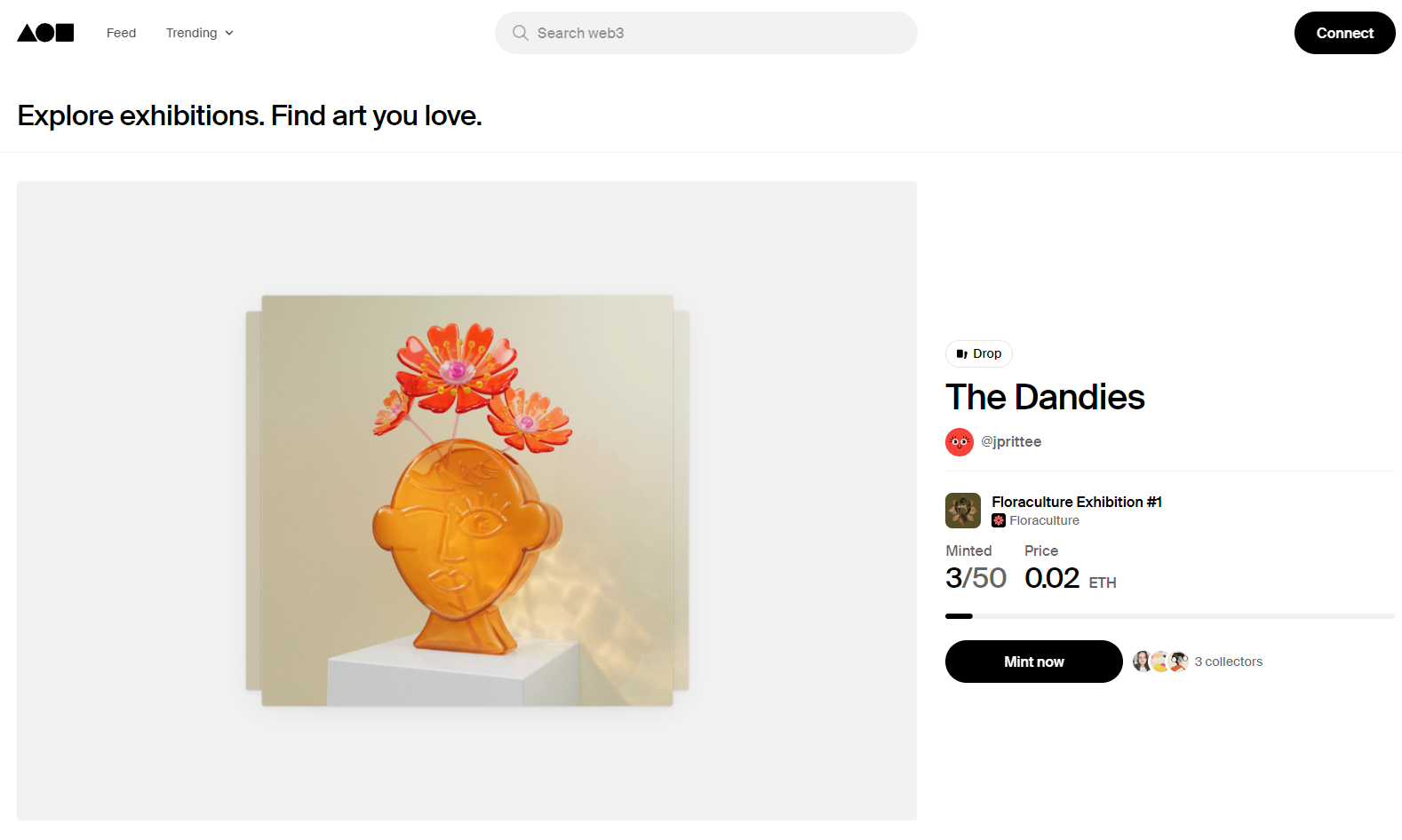
Поддержка художников: SuperRare активно поддерживает художников, предоставляя им справедливую долю от продаж своих работ и создавая условия для развития их творческого потенциала. Это способствует росту сообщества художников и взаимодействию с поклонниками искусства.

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.

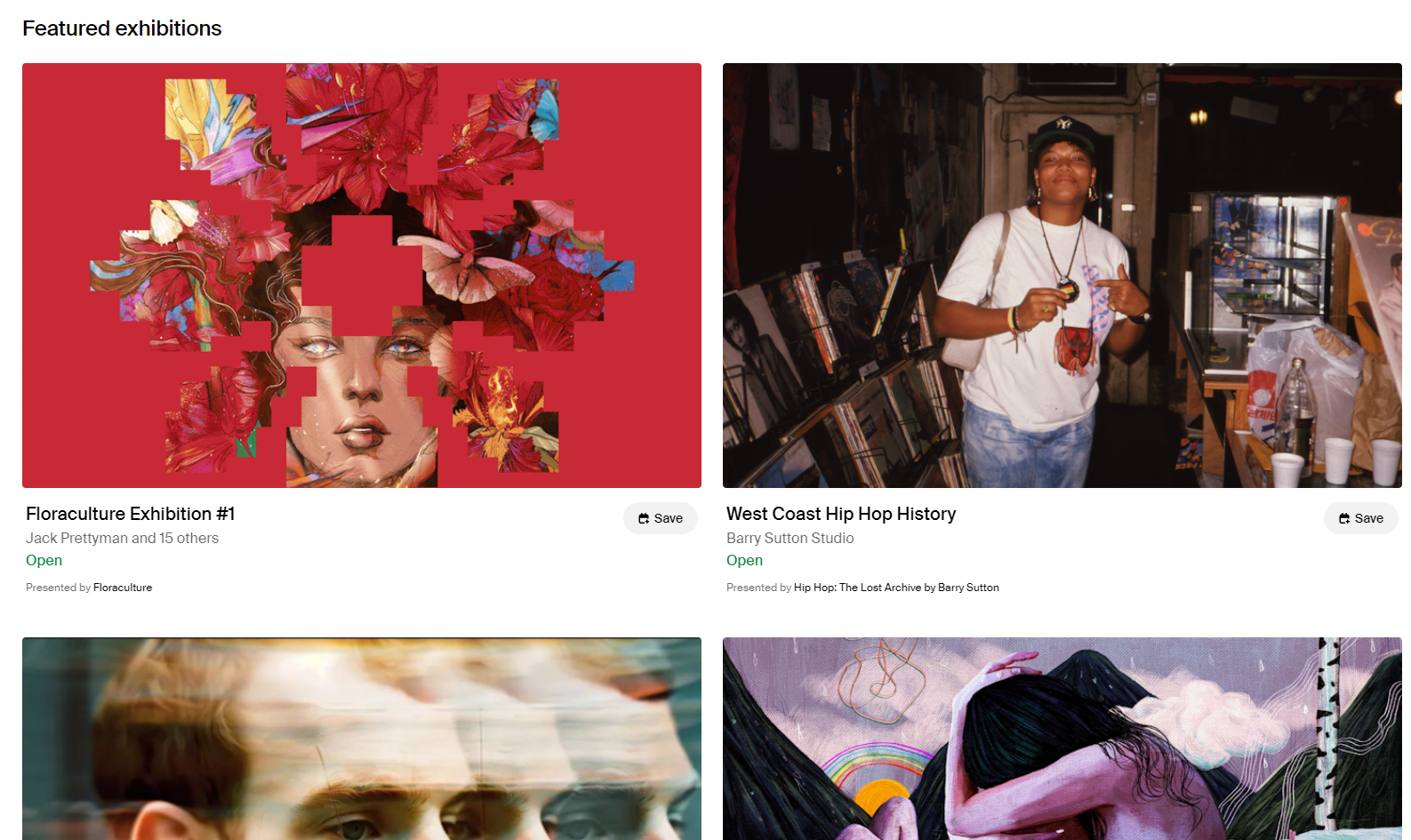
Специализация на художественных ценностях: Foundation ориентирована на поддержку и продвижение художников и творческих работ, которые обладают художественной ценностью и оригинальным подходом. Платформа предоставляет художникам возможность проявить себя и представить свои уникальные произведения широкой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



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.

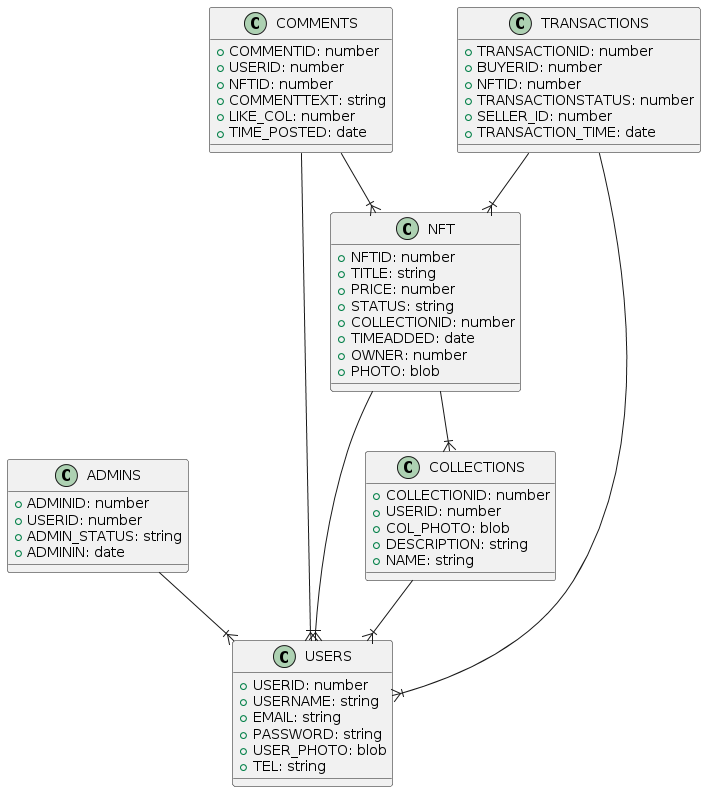
1.3. Вывод по разделу

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

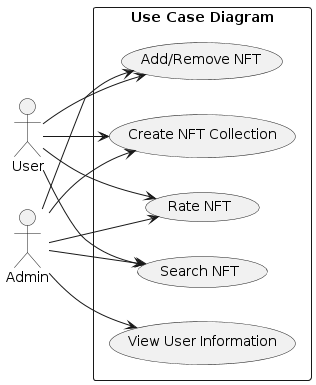
2.1. Структура управлением приложения  
 При проектировании базы данных для «NFT Marketplace» с использованием технологий применения мультимедийных типов данных и системы email уведомлений о событиях базы данных важно учитывать обобщенную структуру управления приложением. Эта структура включает в себя 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**

|  |  |
| --- | --- |
| **Поле таблицы** | **Назначение поля** |
| 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 строчки, позволяя хранить в себе единовременно 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 строчку 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 строчка, 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 соответсвенно производят экспорт и импорт данных из/в файл формата XML.

create or replace NONEDITIONABLE PROCEDURE export\_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 IMPORT\_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()

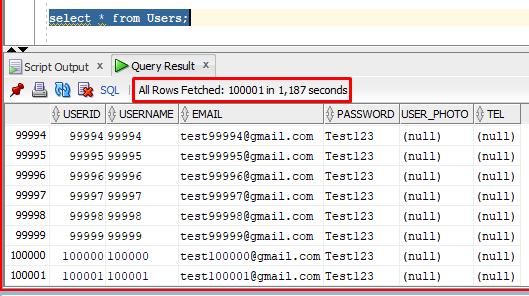
Листинг 2.2 – Python код для вставки 100,000 строк

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

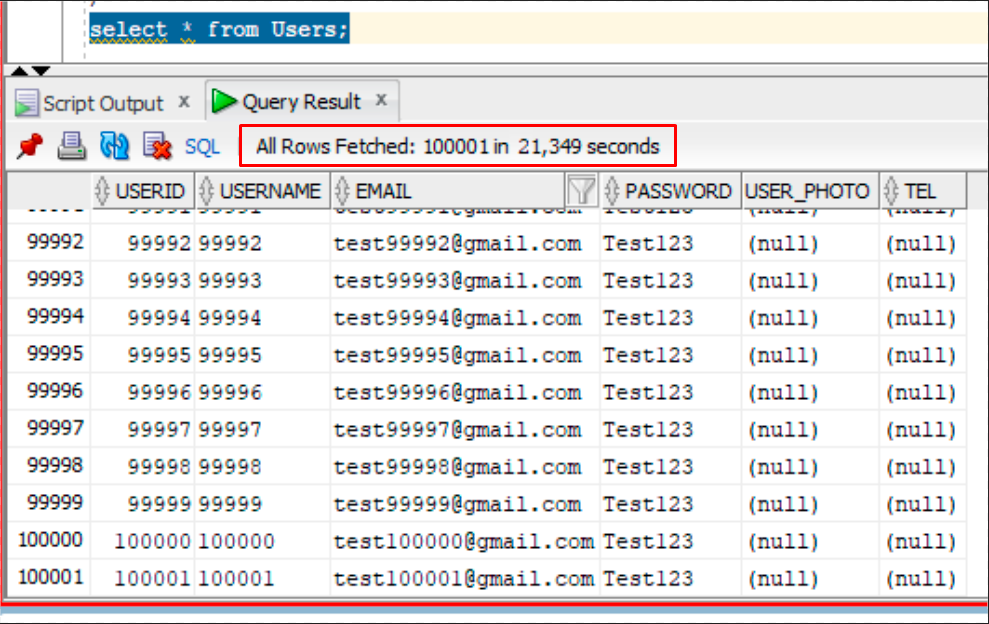


Изображение 2.3 – Уведомление о успешной вставке всех данных

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.

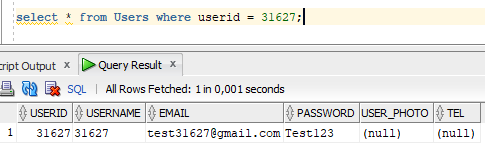
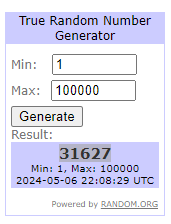


Изображение 2.4 – Демонстрация 100,000 строк (после индексирования)



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

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

Изображение 2.6 – Поиск рандомной (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 данных.")

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 предотвращения 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, collection\_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, description=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 добавления 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\_entry.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, collection\_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 добавлена 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 данных.")

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 «{collection\_name}» успешно добавлена.\n'

body += f'Описание коллекции: {collection\_description}\n'

body += f'Путь к фото: {photo\_path}'

# Создание многочастного сообщения и установка заголовков

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", command=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=select\_photo)

browse\_button.pack()

# Кнопка для добавления коллекции

add\_button = tk.Button(root, text="Добавить коллекцию", command=add\_collection)

add\_button.pack()

# Метка для статуса сообщения

status\_label = tk.Label(root, text="")

status\_label.pack()

# Метка для отображения данных NFT

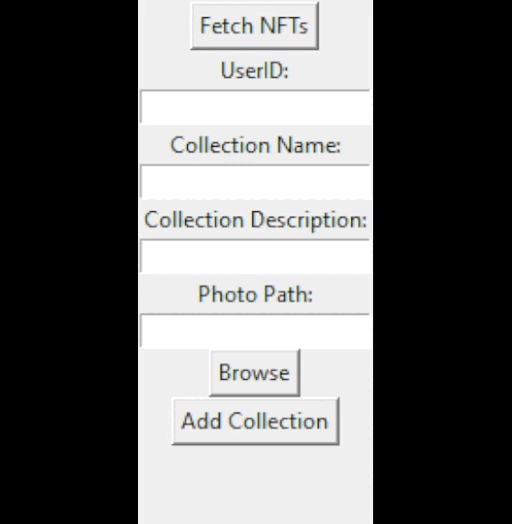
nft\_label = tk.Label(root, text="")

nft\_label.pack()

# Запуск цикла обработки событий Tkinter

root.mainloop()Листинг 2.7 – Мини-приложение демонстрирующее работу

Приложение выглядит следующим образом:

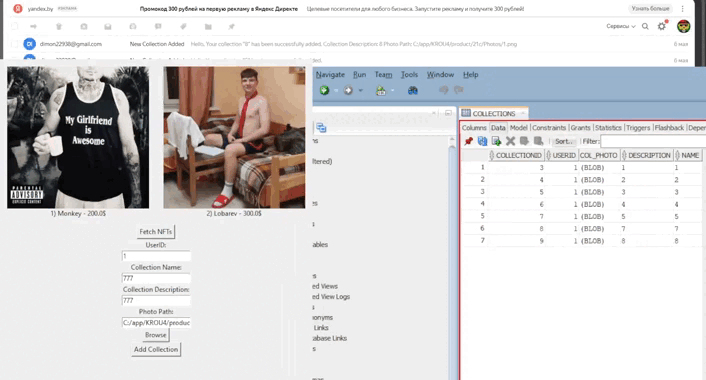


Gif-изображение 2.8 – Демонстрация технологии работы с мультимедиа

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) REFERENCES Users(UserID),

CONSTRAINT fk\_transaction\_nft FOREIGN KEY (NFTID) REFERENCES NFT(NFTID),

CONSTRAINT fk\_transaction\_seller FOREIGN KEY (seller\_id) REFERENCES 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 Users(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\_notification (

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 export\_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 IMPORT\_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 В1 – SQL код для функции blob\_to\_png

create or replace NONEDITIONABLE FUNCTION create\_user\_with\_privileges (

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 В2 – SQL код для функции create\_user\_with\_privileges

Приложение Г - Триггеры

create or replace NONEDITIONABLE TRIGGER trg\_after\_collection\_insert

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 purchase.');

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 comments 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