МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 6-05-0612-01 Программная инженерия (профилизация Программное обеспечение информационных технологий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных системы награждений музыкантов с применением технологии репликации данных между серверами СУБД»

Выполнил студент Бебра хиххихиххихихиихихихих

(Ф.И.О.)

Руководитель работы асс. Бернацкий П.В.

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

Зав. кафедрой ст. преп. Блинова Е.А.

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

Курсовая работа защищена с оценкой

Минск 2025

Оглавление

[Введение 4](#_Toc216876935)

[1 Постановка задачи 5](#_Toc216876936)

[1.1 Обзор аналогичных решений 5](#_Toc216876937)

[1.2 Требования к проекту 8](#_Toc216876938)

[1.3. Выводы к разделу 9](#_Toc216876939)

[2. Проектирование базы данных 9](#_Toc216876940)

[2.1. Диаграмма и таблицы базы данных 9](#_Toc216876941)

[2.2 Описание таблиц базы данных 10](#_Toc216876942)

[2.3 Описание других объектов базы данных 19](#_Toc216876943)

[2.4 Вывод 20](#_Toc216876944)

[3 Разработка объектов базы данных 21](#_Toc216876945)

[3.1 Разработка таблиц базы данных 21](#_Toc216876946)

[3.2 Разработка процедур и пакетов базы данных 21](#_Toc216876947)

[3.3 Разработка триггеров базы данных 23](#_Toc216876948)

[3.4 Разработка функций базы данных 23](#_Toc216876949)

[3.5 Разработка пользователей 24](#_Toc216876950)

[3.6 Разработка синонимов базы данных 25](#_Toc216876951)

[3.7 Вывод 25](#_Toc216876952)

[4. Описание процедур импорта и экспорта 27](#_Toc216876953)

[5. Тестирование производительности 29](#_Toc216876954)

[6. Описание технологии и ее применения в базе данных 31](#_Toc216876955)

[6.1 Настройка репликации базы данных 31](#_Toc216876956)

[6.2 Проверка работы репликации базы данных 34](#_Toc216876957)

[6.3 Вывод 35](#_Toc216876958)

[7. Краткое описание приложения для демонстрации 36](#_Toc216876959)

[8. Руководство пользователя 37](#_Toc216876960)

[Заключение 39](#_Toc216876961)

[Список используемых источников 40](#_Toc216876962)

[Приложение А 41](#_Toc216876963)

[Приложение Б 41](#_Toc216876964)

[Приложение В 42](#_Toc216876965)

[Приложение Г 44](#_Toc216876966)

[Приложение Д 61](#_Toc216876967)

# Введение

В музыкальной индустрии ежегодно присуждаются тысячи наград. Данные о награждениях используются одновременно несколькими сторонами: организаторами премий, стриминговыми сервисами, медиа и аналитическими платформами. Отсутствие единой актуальной базы приводит к дублированию, рассинхронизации и ошибкам. Решением является реляционная база данных с поддержкой репликации между серверами СУБД.

Цель курсовой работы – спроектировать и реализовать реляционную базу данных для системы учёта наград музыкантов на СУБД Oracle с использованием технологий репликации.

Для достижения цели будет разработана реляционная нормализованная модель и физическая структура базы данных для хранения информации о музыкантах, музыкальных релизах вроде альбомов и синглов, типах наград, номинациях и победителях.

Для репликации применяется технология Oracle GoldenGate в конфигурации двусторонней синхронизации между двумя экземплярами базы данных, что гарантирует актуальность данных в реальном времени и отказоустойчивость. Весь проект реализуется на Docker Desktop с использованием Docker Compose для удобного развертывания и тестирования окружения.

Также, производительность базы данных будет протестирована на таблице, содержащей около 100000 строк. Будет обеспечен импорт и экспорт данных в формате JSON.

# 1 Постановка задачи

## 1.1 Обзор аналогичных решений

Перед началом проектирования базы данных для системы награждений музыкантов крайне важно провести детальный анализ существующих аналогичных решений: это позволит изучить проверенные подходы к организации хранения данных, выявить основные сущности с их атрибутами, определить оптимальные связи между таблицами, сформулировать бизнес-правила, которые следует реализовать на уровне СУБД с помощью ограничений, триггеров и проверок, а также заранее спланировать необходимые таблицы, внешние ключи, хранимые процедуры и методы достижения безопасности данных.

* + 1. Аналог «billboardmusicawards.com»

Первый аналог – интернет-ресурс «billboardmusicawards.com» [1].

«Billboard Music Awards» – это сайт, ежегодной американской музыкальной награды, учреждённой журналом Billboard. Основной целью сайта является информирование пользователей о премии: объявление номинантов, победителей, исполнителей, дат проведения церемонии, а также публикация связанных новостей и материалов.

Главная страница сайта представлена на рисунке 1.1.1.



Рисунок 1.1.1 – Главная страница сайта «Billboard Music Awards»

Анализ содержимого официального сайта премии Billboard Music Awards показывает, что основная информация о наградах представлена в разделе новостей и на отдельных страницах, посвящённых конкретным годам. На этих страницах указываются только победители в каждой категории.

Структура награждений включает основные категории и жанровые подразделы. Отдельная страница «Winners Database» предоставляет архив всех лауреатов премии с 1990 года в форме поиска по критериям: исполнитель, название произведения или альбом, год, категория. Результаты поиска выводятся в виде списка или таблицы, содержащей исключительно данные о победителях.

Страница «Winners Database» представлена на рисунке 1.1.2.

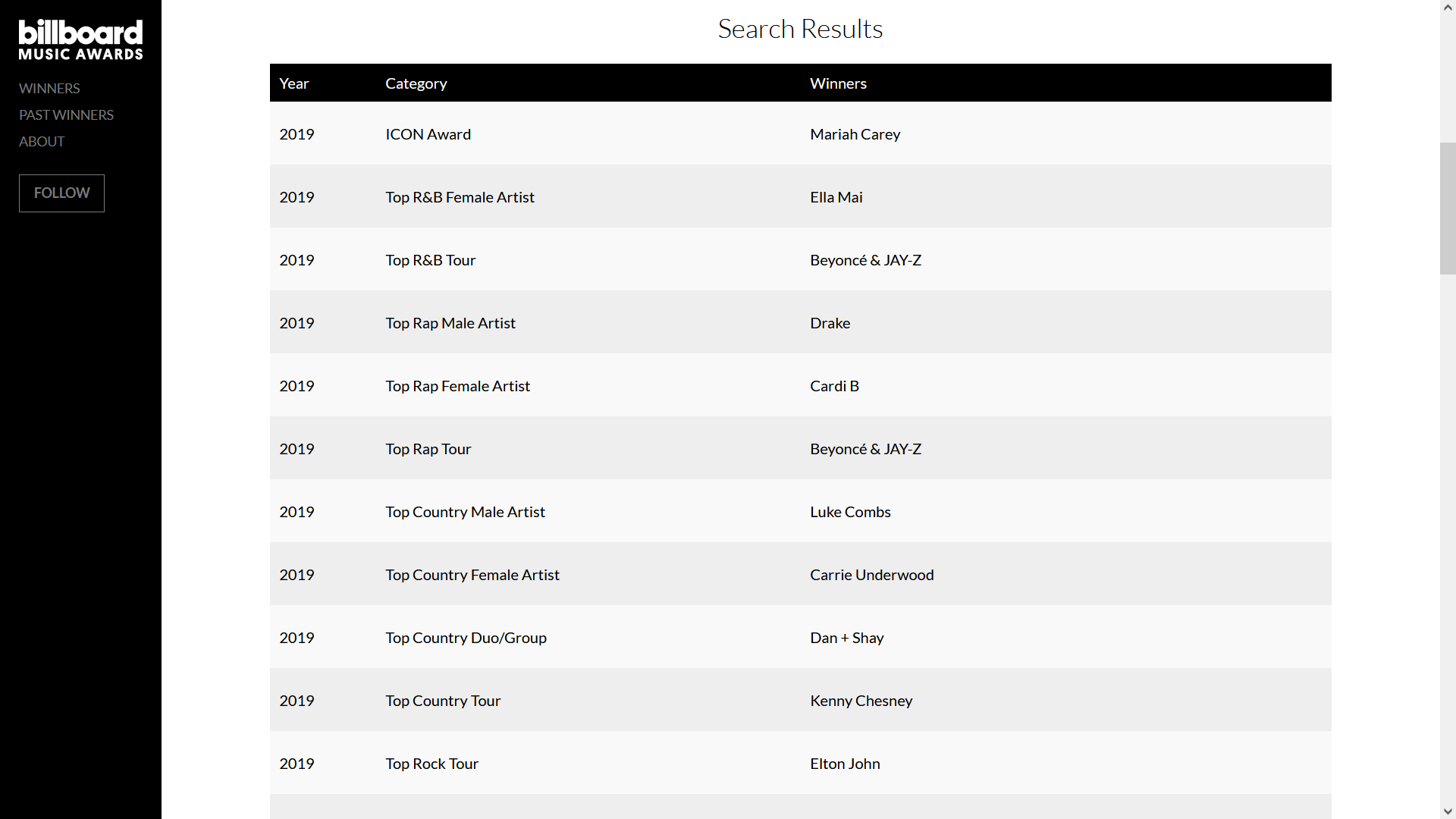


Рисунок 1.1.2 – Страница «Winners Database»

На основании анализа архива победителей за все годы можно предположить связи между основными сущностями. Между сущностями «Год» и «Категория» существует связь «один-ко-многим» с обеих сторон: один год включает множество категорий, а одна категория повторяется в множестве лет. Комбинация атрибутов «год» и «категория» является уникальной и определяет ровно одну победу. Лауреат может включать одного или нескольких исполнителей. Таким образом, связь между сущностью «победа» и «исполнитель» имеет связь «многие-ко-многим» и требует реализации через промежуточную таблицу. В категориях, связанных с конкретным треком или альбомом (например, «Лучшая песня Hot 100» или «Лучший альбом Billboard 200»), победа привязана ровно к одному музыкальному объекту. Это соответствует связи «один-к-одному».

Центральной сущностью является «Победа в номинации в конкретный год». Эта сущность выступает в качестве ассоциативной, связывая год, категорию, исполнителей и, при необходимости, музыкальное произведение. Составной первичный ключ – комбинация «год» и «категория» – обеспечивает уникальность записей и соответствует требованиям третьей нормальной формы.

Такой подход позволяет избежать избыточности данных, обеспечить целостность и упростить выполнение запросов.

* + 1. **Аналог «musicawardsjapan.com»**

Второй аналог – интернет-ресурс «musicawardsjapan.com» [2].

Music Awards Japan (MAJ) – это международная музыкальная награда, учреждённая пятью японскими отраслевыми ассоциациями. Премия 2025 года уже завершена.

Главная страница сайта представлена на рисунке 1.3.

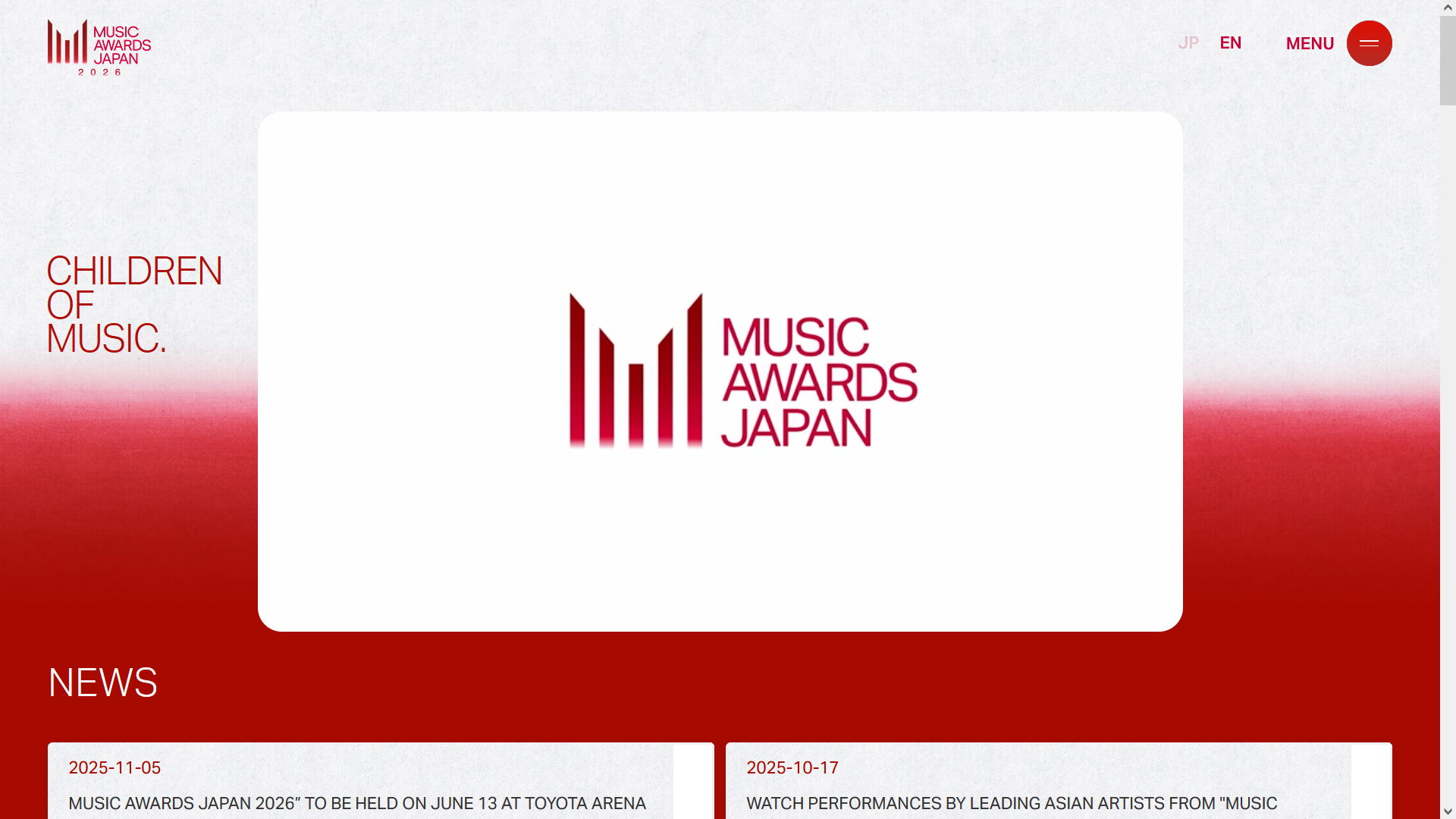


Рисунок 1.1.3 – Главная страница сайта Music Awards Japan

На странице категорий премии Music Awards Japan 2025 информация о более чем 100 номинациях, разделённых на тематические группы, представлена в текстовом формате в виде заголовков и подзаголовков без табличного отображения номинантов или победителей.

Номинанты и лауреаты доступны в отдельных разделах сайта. Страница Nominees – в списочном формате по категориям без указания победителей, страница Awards – с перечнем победивших треков или артистов по категориям.

На основе полной структуры премии можно выделить основные сущности и связи: сущность «Год» имеет отношение «один-ко-многим» к победам, сущность «Категория» обладает уникальным идентификатором, названием. Центральной сущностью выступает «Победа в номинации в конкретный год», дополненная сущностью «Номинация» для хранения финалистов. Связь между сущностью «Исполнитель» и «номинацией» имеет отношение «многие-ко-многим». Вероятно, реализуется через промежуточную таблицу, поскольку один исполнитель может быть номинирован или победить в нескольких категориях, а категория включает нескольких исполнителей.

Страница «Nominees» представлена на рисунке 1.1.4.

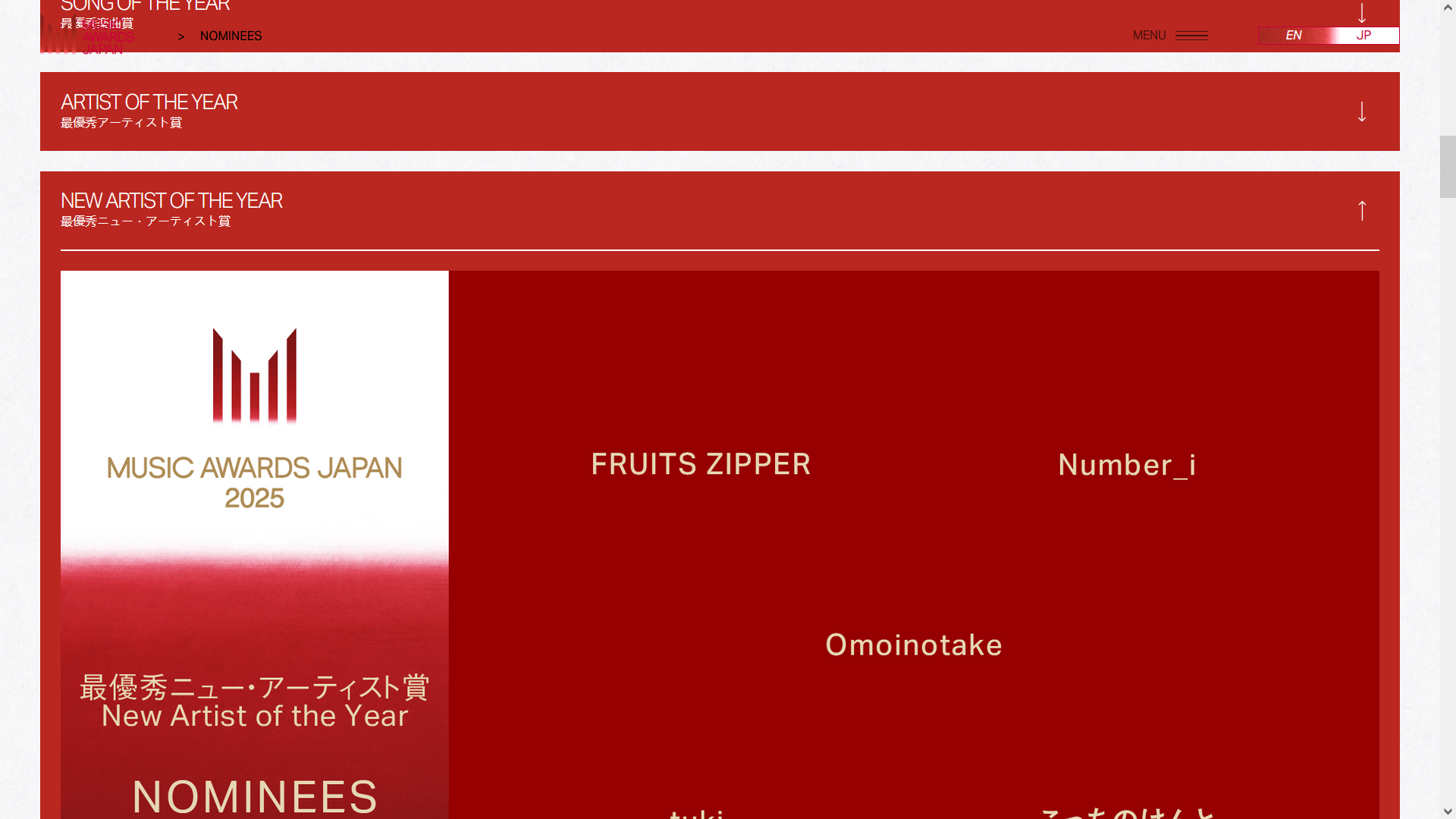


Рисунок 1.1.4 – Страница «Nominees»

Для проектирования собственной базы данных из него можно выделить полезные элементы: наличие фиксированного числа номинантов в каждой категории, чёткое разделение номинантов и лауреатов, а также использование сущности «Номинация» отдельно от «Победа».

## 1.2 Требования к проекту

Проведя анализ результатов, полученных при обзоре аналогов, а также учитывая особенности предметной области музыкальных наград, можно выделить ряд задач и требований, которые должны быть выполнены по окончанию работы над курсовым проектом.

В системе должно быть реализовано полноценное управление ролями пользователей: администратор, менеджер и пользователь с чётким разграничением прав доступа. Администратор осуществляет управление менеджерами, включая добавление, удаление и изменение их учётных записей. Менеджер выполняет управление данными о музыкантах с возможностью добавления, удаления, изменения и дисквалификации, управление альбомами с операциями добавления, удаления и изменения, управление списком музыкальных треков с аналогичными операциями, а также управление отзывами с правом удаления. Пользователь получает возможность оставлять отзывы на альбомы. Система обеспечивает автоматический подсчёт оценок на основе отзывов.

1.3. Выводы к разделу

Анализ аналогичных решений позволил сформулировать основные требования к программному обеспечению. Оно должно быть удобным, информативным и интуитивно понятным как для пользователей при оставлении отзывов и получении информации о наградах, так и для администраторов и менеджеров при управлении сущностями и поддержании целостности связей.

Выводы позволили определить структуру будущей базы данных и состав функциональных задач проекта, что создаёт основу для эффективной автоматизации работы системы награждений и повышения удобства взаимодействия пользователей. Кроме того, выявленные требования обусловливают целесообразность внедрения механизма репликации данных.

# 2. Проектирование базы данных

2.1. Диаграмма и таблицы базы данных

Функциональные требования к базе данных в первую очередь определяют методы обработки информации и обеспечивают необходимую функциональность для пользовательского интерфейса. Важно четко описать, как данные будут храниться и структурироваться, каким образом будет осуществляться их выборка, каким образом выполняются обновления, а также как обеспечивается безопасность информации. Кроме того, следует учитывать взаимодействие базы данных с внешними системами и приложениями.

Помимо функциональных требований, важно определить роли пользователей и сценарии их взаимодействия с системой. Такие сценарии показывают, как пользователи будут работать с системой в зависимости от своих ролей, что помогает определить доступные функции и данные для каждой роли, а также организовать навигацию внутри системы. Обычно сценарии оформляются в виде диаграмм вариантов использования (Use Case), наглядно демонстрирующих взаимодействие пользователей с системой. Пример use-case диаграммы приведен в приложении А.

Для проверки архитектуры базы данных была составлена диаграмма, на которой отображены все сущности и их взаимосвязи. Это позволяет быстро оценить способы связи между таблицами, ключевые поля и применяемые ограничения. Диаграмма структуры базы данных приведена на рисунке 2.1.

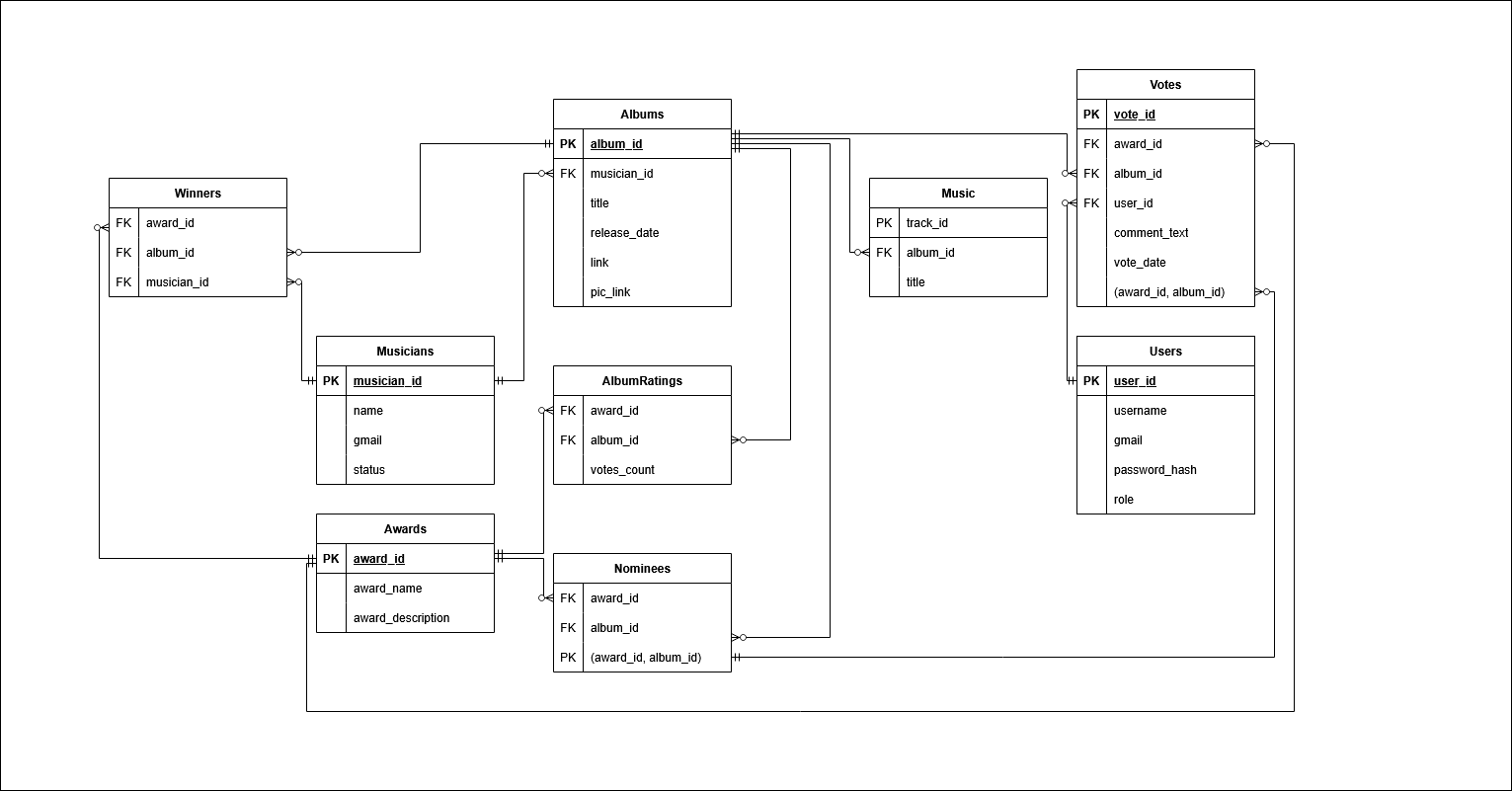


Рисунок 2.1 – Диаграмма базы данных

В проектируемой базе данных ключевым типом связей является «один – ко – многим». Он применяется во многих основных сценариях. Такой способ организации отношений точно обеспечивает корректное представление взаимосвязей между сущностями.

## 2.2 Описание таблиц базы данных

Проанализировав требования к системе награждения музыкантов и имеющиеся данные, было решено выделить 9 таблиц: Musicians, Albums, Music, Users, Awards, Nominants, Winners, Votes, AlbumRatings.

Таблица Musicians предназначена для хранения данных музыкантах, которые принимают участии в премиях. В ней представлена вся ключевая информация, включая имя или псевдоним музыканта, краткая информация о нем и статус участия. Статус принимает два значения: участвует и дисквалифицирован.

Это обеспечивает удобный учёт всех участников премий и позволяет быстро фильтровать музыкантов по их статусу для дальнейшей работы с данными.

Структура данной таблицы представлена в таблице 2.1.

Таблица 2.1 – Структура таблицы Musicians

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| musician\_id | NUMBER | GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY | Уникальный идентификатор музыканта. Автоматически создается при добавлении новой записи. Обеспечивает отсутствие дубликатов. |
| name | VARCHAR2(100) | NOT NULL | Хранит имя или псевдоним музыканта; обязательное поле, не допускающее пустых значений. |
| status | VARCHAR2(20) | CHECK (status IN ('participating','disqualified')) NOT NULL | Указывает текущий статус музыканта в премии: участвует или дисквалифицирован. Обязательное поле с ограничением допустимых значений. |

Таблица Albums содержит в себе данные о каждом альбоме, включая название, дату выпуска, ссылку на прослушиваниеи ссылку на изображения для корректного отображения в будущей реализации.

Структура данной таблицы представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы Albums

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| album\_id | NUMBER | GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY | Уникальный идентификатор альбома. Автоматически создаётся при добавлении записи, исключая дублирование. |

Продолжение таблицы 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| musician\_id | NUMBER | NOT NULL | Ссылка на музыканта, которому принадлежит альбом. Обязательное поле для установления связи с таблицей музыкантов. |
| title | VARCHAR(150) | NOT NULL | Название альбома. Поле, не допускающее пустых значений. |
| release\_date | DATE |  | Дата выхода альбома, необязательное поле для хранения информации о релизе. |
| link | VARCHAR(200) | NOT NULL | Ссылка на альбом или источник его прослушивания, обязательное поле, не допускающее пустых значений. |
| pic\_link | VARCHAR(200) |  | Ссылка на изображение обложки альбома, необязательное поле для хранения визуального представления. |

Таблица Music содержит в себе уникальный идентификатор каждого произведения (track\_id как первичный ключ), внешний ключ на альбом (album\_id), к которому относится это произведение, и название трека (title). Связь с таблицей Albums реализована через внешний ключ с опцией каскадного удаления, что обеспечивает автоматическое удаление всех треков при удалении соответствующего альбома и поддерживает ссылочную целостность данных.

Структура данной таблицы представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы Music

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| track\_id | NUMBER | GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY | Уникальный идентификатор трека, автоматически создаётся при добавлении записи, исключая дублирование. |
| album\_id | NUMBER | NOT NULL | Ссылка на альбом, к которому относится произведение. Обязательное поле для установления связи с таблицей альбомов. |
| title | VARCHAR2(150) | NOT NULL | Название произведения, обязательное поле, не допускающее пустых значений. |

Таблица Users предназначена для хранения информации о каждом пользователе, используя уникальный генерируемый идентификатор, а также учитывая его роль, уникальный адрес электронной почты и уникальное имя.

Роль проверяется в соответствии с уже тремя существующими: admin\_user, manager\_user, guest\_user.

Структура таблицы Users представлена в таблице 2.4.

Таблица 2.4 – Структура таблицы Users

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| user\_id | NUMBER | GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY | Уникальный идентификатор пользователя, автоматически создаётся при добавлении записи, исключая дублирование. |
| username | VARCHAR2(50) | NOT NULL UNIQUE | Имя пользователя для входа в систему, обязательное и уникальное, предотвращает дублирование аккаунтов. |
| gmail | VARCHAR2(100) | NOT NULL UNIQUE | Требует обязательного заполнения |
| password\_hash | VARCHAR2(200) | NOT NULL UNIQUE | Хэш пароля пользователя, обязательное поле для безопасного хранения учетных данных. |
| role | VARCHAR(20) | DEFAULT ‘guest\_user’ CHECK (role IN (‘admin\_user’, ‘manager\_user’, ‘guest\_user’)) | Роль пользователя в системе, со значениями «admin\_user», «manager\_user» или «guest\_user». По умолчанию Определяет уровень доступа. |

Таблица Awards предназначена для хранения премий, в которых в будущем будут участвовать альбомы музыкантов.

Она содержит генерируемый уникальный идентификатор премии, название премии и краткое описание, что позволяет гибко управлять разнообразными номинациями. Таблица послужит центральным справочником для связей с номинантами, победителями и голосами.

Структура таблицы Awards представлена в таблице 2.5

Таблица 2.5 Структура таблицы Awards

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| award\_id | NUMBER | GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY | Обеспечивает генерацию уникального идентификатора для каждой записи и гарантирует невозможность появления дублирующихся записей |
| award\_name | VARCHAR2(150) | NOT NULL | Название премии, обязательное поле для идентификации и отображения награды в системе. |
| award\_description | VARCHAR2(500) |  | Описание премии, необязательное поле для хранения дополнительной информации о премии |

Таблица Nominants содержит список всех альбомов, выдвинутых в качестве номинантов в каждой категории премии, и служит основой для последующего голосования и определения победителей.

Использование отдельной таблицы Nominants обусловлено необходимостью отражать реальную практику музыкальных премий, где один альбом музыканта может претендовать на награды в нескольких номинациях одновременно. Без такой промежуточной таблицы пришлось бы дублировать данные об альбомах в таблице Awards или вводить сложные структуры атрибутов, что привело бы к избыточности и нарушению нормальных форм.

Структура таблицы Nominants представлена в таблице 2.6.

Таблица 2.6 – Структура таблицы Nominanys

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| award\_id | NUMBER | NOT NULL | Ссылка на премию, в рамках которой выдвинут номинант, обязательное поле для связи таблицы nominants с таблицей awards. |
| album\_id | INTEGER | NOT NULL | Ссылка на альбом, выдвинутый на премию, обязательное поле для установления связи таблицы nominants с таблицей albums. |

Таблица Winners, предназначенная для хранения информации об участниках и победивших альбомах после завершения голосования и подсчета результатов.

Хранит в себе уникальный идентификатор премии, в которой принимал участие альбом и уникальный идентификатор альбома. Предполагается, что таблица будет оставаться пустой до окончания голосования и будет ограничиваться до одного победителя в одной премии.

Структура таблицы Winners представлена в таблице 2.7

Таблица 2.7 – Структура таблицы Winners

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| award\_id | NUMBER | NOT NULL | Ссылка на премию, в которой был определён победитель, обязательное поле для связи. |

Продолжение таблицы 2.7

|  |  |  |  |
| --- | --- | --- | --- |
| album\_id | NUMBER | NOT NULL | Ссылка на альбом, получивший награду, обязательное поле для установления связи с таблицей albums. |
| musician\_id | NUMBER | NOT NULL | Ссылка на музыканта-победителя, обязательное поле для установления связи с таблицей musicians. |

Таблица AlbumRatings, предназначенная для хранения голосов. Предполагается, что с помощью этой таблицы будут подсчитываться данные для таблицы Winners.

Структура таблицы AlbumRatings представлена в таблице 2.8.

Таблица 2.8 – Структура таблицы Winners

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| award\_id | NUMBER | NOT NULL | Ссылка на премию, в которой был определён победитель, обязательное поле для связи с таблицей awards. |
| album\_id | NUMBER | NOT NULL | Ссылка на альбом, получивший награду, обязательное поле для установления связи с таблицей albums. |

Продолжение таблицы 2.8

|  |  |  |  |
| --- | --- | --- | --- |
| votes\_count | NUMBER | DEFAULT 0 | Количество набранных голосов, по умолчанию равно нулю и используется для подсчёта результатов. |

Таблица Votes, предназначенная для хранения комментариев, которые будут оставляться при голосовании.

Хранит в себе уникальный идентификатор премии, в которой принимал участие альбом, уникальный идентификатор альбома, уникальный идентификатор пользователя, оставившего голос, и комментарий. Предполагается, что записи будут ограничены. На одного пользователя один голос к одному альбому в премии.

Структура таблицы Votes представлена в таблице 2.9

Таблица 2.9 – Структура таблицы Winners

|  |  |  |  |
| --- | --- | --- | --- |
| Имя столбца | Тип данных | Ограничение целостности | Назначение |
| vote\_id | NUMBER | GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY | Ссылка на альбом, который стал победителем, обязательное поле. |
| award\_id | NUMBER | GENERATED NOT NULL | Ссылка на премию, в которой был определён победитель, обязательное поле для связи с таблицей awards. |
| album\_id | NUMBER | NIT NULL | Ссылка на альбом, получивший награду, обязательное поле для установления связи с таблицей albums. |

Продолжение таблицы 2.9

|  |  |  |  |
| --- | --- | --- | --- |
| user\_id | NUMBER | NOT NULL | Ссылка на пользователя, отдавшего голос, обязательное поле для связи с таблицей users. |
| comment\_text | VARCHAR2(4000) |  | Текстовый комментарий пользователя к голосу, необязательное поле для хранения отзывов. |

## 2.3 Описание других объектов базы данных

Для эффективного функционирования базы данных необходимо создать правильные роли и пользователей с чётким разграничением прав доступа в соответствии с функциональными требованиями системы.

В таблице 2.8 представлен обзор ролей, создаваемых в базе данных.

Таблица 2.10 – Роли пользователей

|  |  |
| --- | --- |
| Название роли | Описание |
| admin\_role | Роль для администратора, который будет назначать менеджеров. |
| manager\_role | Роль для менеджера, который будет управлять данными в таблицах, обеспечивая удаление, обновление и добавления новой информации. |
| guest\_role | Роль для гостя, пользователя, посетившего сайт. Роль применяется по умолчанию для каждого нового пользователя. |

Каждой из ролей соответствует пользователь, с помощью которого будут выполняться операции в базе данных в зависимости от уровня доступа.

## 2.4 Вывод

В данном разделе был рассмотрен процесс проектирования базы данных для системы награждения музыкантов, который стал основой для дальнейшей разработки и реализации всей информационной системы. В ходе работы были проанализированы и описаны основные варианты взаимодействия пользователей с базой данных, используя роли – гость, менеджер и администратор – что позволило разграничить функциональные возможности и права доступа для каждого типа пользователя.

Также были описаны таблицы, включая первичные и внешние ключи, ограничения целостности. предназначенные для правильного хранения различных данных о системе награждений музыкантов.

Всё это создаёт прочную основу для разработки правильной, логичной и масштабируемой архитектуры реляционной базы данных и функциональных задач системы награждения музыкантов.

# 3 Разработка объектов базы данных

## 3.1 Разработка таблиц базы данных

Данный раздел содержит SQL-код, необходимый для создания структуры базы данных награждений музыкантов. Представленные операторы CREATE TABLE формируют логически связанную схему, предназначенную для хранения необходимых данных.

Пример создания таблицы приведён на листинге 3.1. Полный скрипт создания таблиц находится в приложении Б.

|  |
| --- |
| CREATE TABLE Albums (  album\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  musician\_id NUMBER NOT NULL,  title VARCHAR2(150) NOT NULL,  release\_date DATE,  link VARCHAR2(200) NOT NULL,  pic\_link VARCHAR2(100),  CONSTRAINT fk\_album\_musician FOREIGN KEY (musician\_id)  REFERENCES Musicians(musician\_id)  ON DELETE CASCADE  ) TABLESPACE music\_ts; |

Листинг 3.1 – Скрипт для создания таблицы

Каждая таблица в проектируемой базе данных размещается в специально созданном табличном пространстве music\_ts.

Табличное пространство позволяет физически отделить данные проектируемой информационной системы от системных объектов Oracle.

Создание табличного пространства представлено в листинге 3.2.

|  |
| --- |
| CREATE TABLESPACE music\_ts  DATAFILE '/opt/oracle/oradata/FREE/music\_ts01.dbf'  SIZE 100M  AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED; |

Листинг 3.2 – Скрипт для создания табличного пространства

## 3.2 Разработка процедур и пакетов базы данных

Процедуры в Oracle PL/SQL — это именованные блоки кода, предназначенные для выполнения конкретных действий или последовательности операций. В отличие от функций, процедуры не возвращают значение через оператор RETURN, но могут передавать результаты с помощью выходных параметров (режим OUT).

В теле процедуры допускается использование любых DML-операторов (INSERT, UPDATE, DELETE), что позволяет напрямую модифицировать данные в таблицах. Кроме того, процедуры поддерживают реализацию сложной бизнес-логики: условные конструкции, циклы, обработку исключений, а также вызовы других процедур и функций.

Применение процедур обеспечивает централизацию логики обработки данных, сокращение дублирования кода, повышение стабильности и безопасности системы, поскольку все операции с данными выполняются в контролируемом и стандартизированном виде.

В рамках проекта были созданы процедуры для выполнения всей бизнес-логики.

Содержимое одной из процедур приведено в листинге 3.3.

|  |
| --- |
| PROCEDURE get\_all\_managers(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT user\_id, username, gmail  FROM Users  WHERE role = 'manager\_user';  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20027, 'Error fetching managers: ' || SQLERRM);  END; |

Листинг 3.3 – Процедура получения пользователей

Процедура get\_all\_managers формирует и возвращает результирующий набор данных со списком пользователей, имеющих роль «manager\_user». Скрипты создания остальных процедур пакета приведены в приложении В.

Все процедуры объединены в специальные блоки – пакеты PL/SQL, которые представляют собой объекты базы данных Oracle, группирующие логически связанные процедуры, функции, переменные, константы и типы в единый именованный блок. Такое объединение обеспечивает лучшую организацию кода, инкапсуляцию данных и удобство сопровождения системы.

В базе данных разработаны такие пакеты, как pkg\_admin, pkg\_manager и pkg\_guest. Пример скрипта для создания пакета представлен в листинге 3.4.

|  |
| --- |
| CREATE OR REPLACE PACKAGE pkg\_admin AS  FUNCTION hash\_password(p\_password VARCHAR2) RETURN VARCHAR2;  PROCEDURE add\_admin(p\_username VARCHAR2, p\_gmail VARCHAR2, p\_password VARCHAR2);  PROCEDURE add\_manager(p\_username VARCHAR2, p\_gmail VARCHAR2, p\_password VARCHAR2);  PROCEDURE update\_manager(p\_user\_id NUMBER, p\_username VARCHAR2, p\_gmail VARCHAR2, p\_password VARCHAR2);  PROCEDURE delete\_manager(p\_user\_id NUMBER);  PROCEDURE get\_all\_managers(p\_cursor OUT SYS\_REFCURSOR);  END pkg\_admin; |

Листинг 3.4 – Пакет процедур pkg\_admin

## 3.3 Разработка триггеров базы данных

Эффективная работа разработанной базы данных системы награждения музыкантов напрямую зависит от триггеров – специальных объектов Oracle Database, автоматически выполняющих заданные действия при возникновении определённых событий в таблицах.

Триггеры обеспечивают поддержание бизнес-логики и целостности данных на уровне СУБД без вмешательства приложения.

Например, триггер может автоматически обновлять счётчики голосов в таблице AlbumRatings при добавлении или удалении записи в Votes, предотвращают. Полный скрипт создания триггеров представлен в приложении Г.

|  |
| --- |
| CREATE OR REPLACE TRIGGER admin\_user.trg\_vote\_insert  AFTER INSERT ON admin\_user.Votes  FOR EACH ROW  DECLARE  BEGIN  MERGE INTO admin\_user.AlbumRatings ar  USING (SELECT :NEW.album\_id AS album\_id, :NEW.award\_id AS award\_id FROM dual) src  ON (ar.album\_id = src.album\_id AND ar.award\_id = src.award\_id)  WHEN MATCHED THEN  UPDATE SET ar.votes\_count = ar.votes\_count + 1  WHEN NOT MATCHED THEN  INSERT (album\_id, award\_id, votes\_count)  VALUES (src.album\_id, src.award\_id, 1);  END;  / |

Листинг 3.5 – Скрипт для создания индекса

## 3.4 Разработка функций базы данных

В проектируемой системе взаимодействие приложения с базой данных осуществляется через вызов функций Oracle PL/SQL.

Функция представляет собой именованный, однократно компилируемый блок кода, хранящийся на сервере и предназначенный для многократного использования. В отличие от процедур, функции не позволяют выполнять операторы модификации данных (DML) или изменения структуры (DDL), а только возвращают значение через оператор RETURN, что делает их подходящими для вычислений и получения данных без побочных эффектов.

В рамках данного проекта реализовано две такие функции, обеспечивающие необходимую логику обработки запросов со стороны приложения.

Пример создания функции приведён на листинге 3.6.

|  |
| --- |
| FUNCTION validate\_email(p\_email VARCHAR2) RETURN NUMBER IS  BEGIN  IF REGEXP\_LIKE(p\_email, '^[A-Za-z0-9.\_%+-]+@gmail\.com$') THEN  RETURN 1;  ELSE  RETURN 0;  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21001, 'Error validating email: ' || SQLERRM);  END; |

Листинг 3.6 – Функция проверки правильности электронной почты

Функция validate\_email предназначена для проверки соответствия входной строки формату адреса электронной почты. Она принимает параметр p\_email типа VARCHAR2 и возвращает 1 при успешном прохождении проверки с помощью регулярного выражения REGEXP\_LIKE, либо 0 в противном случае.

Блок обработки исключений обеспечивает контролируемую реакцию на возможные ошибки, генерируя пользовательское исключение с кодом -21001 и сообщением.

Реализация данной функции позволяет централизованно валидировать адреса электронной почты при регистрации или обновлении пользователей, повышая качество данных в таблице Users и предотвращая внесение некорректных записей.

## 3.5 Разработка пользователей

Пользователь базы данных представляет собой субъект, который взаимодействует с информационной системой для извлечения, обработки или модификации данных в соответствии с назначенными правами доступа.

На всех этапах жизненного цикла базы данных – от проектирования и реализации до эксплуатации и развития – задействованы различные категории пользователей с чётко определёнными обязанностями и привилегиями, что обеспечивает безопасность и эффективность работы системы.

В проектируемой базе данных реализована трёхуровневая модель разграничения доступа, включающая роли обычного пользователя (гость), менеджера и администратора, где каждая роль соответствует конкретным функциональным задачам: от просмотра и голосования для пользователей с ролью guest\_user, до управления содержимым и администрирования учётных записей пользователями с ролью manager\_user и admin\_user.

Соответствующий SQL-код создания пользователей администратора, менеджера и гостя представлен в листинге 3.7.

|  |
| --- |
| create user admin\_user identified by Kate2006  DEFAULT TABLESPACE music\_ts  TEMPORARY TABLESPACE temp  QUOTA UNLIMITED ON music\_ts;    CREATE USER manager\_user IDENTIFIED BY Kate2006  DEFAULT TABLESPACE music\_ts  TEMPORARY TABLESPACE temp  QUOTA UNLIMITED ON music\_ts;    CREATE USER guest\_user IDENTIFIED BY Kate2006  DEFAULT TABLESPACE music\_ts  TEMPORARY TABLESPACE temp  QUOTA UNLIMITED ON music\_ts; |

Листинг 3.7 – Создание пользователей

## 3.6 Разработка синонимов базы данных

Для обеспечения корректной работы механизма репликации базы данных в выбранной технологии, репликации базы данных, потребовалось использование специальных объектов – синонимов.

Синоним – это альтернативное имя для существующего объекта базы данных. Синоним не содержит собственных данных, а лишь указывает на целевой объект, позволяя обращаться к нему под другим именем.

Пример скрипта для создания синонимов для пользователя manager\_user представлен в листинге 3.8.

|  |
| --- |
| CREATE SYNONYM manager\_user.Musicians FOR admin\_user.Musicians;  CREATE SYNONYM manager\_user.Albums FOR admin\_user.Albums;  CREATE SYNONYM manager\_user.Music FOR admin\_user.Music;  CREATE SYNONYM manager\_user.Awards FOR admin\_user.Awards;  CREATE SYNONYM manager\_user.Winners FOR admin\_user.Winners;  CREATE SYNONYM manager\_user.Nominants FOR admin\_user.Nominants;  CREATE SYNONYM manager\_user.Votes FOR admin\_user.Votes; |

Листинг 3.6 – Синонимы для пользователя manager\_user

## 3.7 Вывод

В данном разделе была выполнена разработка объектов базы данных для полноценной системы награждения музыкантов. Все созданные объекты формируют структуру, обеспечивающую корректное функционирование и централизованное управление данными о музыкантах, альбомах, номинациях, голосованиях и победителях премии.

Были разработаны таблицы для хранения информации о всех использующихся сущностях, функции для валидации данных, процедуры с разграничением прав доступа, триггеры, а также синонимы для упрощения работы с объектами.

# 4. Описание процедур импорта и экспорта

Для таблицы Users реализована возможность экспорта и импорта данных в формате JSON с помощью специальных процедур в пакете PL/SQL. Она открывает файл с указанным именем, последовательно записывает записи пользователей в виде массива JSON-объектов, корректно обрабатывая запятые между элементами и обеспечивая валидный синтаксис JSON.

Данный функционал обеспечивает удобное резервное копирование информации, миграцию между серверами или обмен данными с внешними системами.

Данный функционал обеспечивает удобный обмен данными между системой и внешними приложениями, упрощает резервное копирование и восстановление учётных записей пользователей, включая роли и хэши паролей, а также позволяет мигрировать данные при изменении структуры или переносе на другую среду. Скрипт соответствующей процедуры приведён в листинге 4.1.

|  |
| --- |
| PROCEDURE export\_users\_to\_file(p\_filename VARCHAR2) IS  f UTL\_FILE.FILE\_TYPE;  CURSOR c\_users IS  SELECT user\_id, username, gmail, password\_hash, role  FROM Users;  l\_first BOOLEAN := TRUE;  BEGIN  f := UTL\_FILE.FOPEN('JSON\_DIR', p\_filename, 'w', 32767);  UTL\_FILE.PUT\_LINE(f, '[');  FOR r IN c\_users LOOP  IF NOT l\_first THEN  UTL\_FILE.PUT\_LINE(f, ',');  END IF;  UTL\_FILE.PUT\_LINE(f,  '{' ||  '"user\_id":' || r.user\_id || ',' ||  '"username":"' || r.username || '",' ||  '"gmail":"' || r.gmail || '",' ||  '"password\_hash":"' || r.password\_hash || '",' ||  '"role":"' || r.role || '"' ||  '}'  );  l\_first := FALSE;  END LOOP;  UTL\_FILE.PUT\_LINE(f, ']');  UTL\_FILE.FCLOSE(f);  DBMS\_OUTPUT.PUT\_LINE('Export completed: ' || p\_filename);  END; |

Листинг 4.1 – Скрипт процедуры процедуры экспорта

Процедура импорта данных в таблицу Users из JSON-файла предназначена для массового восстановления или добавления учётных записей пользователей в систему награждения музыкантов. Скрипт процедуры импорта представлен в листинге 4.2.

|  |
| --- |
| PROCEDURE import\_users\_from\_file(p\_filename VARCHAR2) IS  f UTL\_FILE.FILE\_TYPE;  l\_line VARCHAR2(32767);  l\_json CLOB;  BEGIN  DBMS\_LOB.CREATETEMPORARY(l\_json, TRUE);  f := UTL\_FILE.FOPEN('JSON\_DIR', p\_filename, 'r', 32767);  LOOP BEGIN  UTL\_FILE.GET\_LINE(f, l\_line);  DBMS\_LOB.APPEND(l\_json, l\_line);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  EXIT;  END;  END LOOP;  UTL\_FILE.FCLOSE(f);  BEGIN  INSERT INTO Users (username, gmail, password\_hash, role)  SELECT jt.username, jt.gmail, jt.password\_hash, jt.role  FROM JSON\_TABLE(l\_json,'$[\*]'  COLUMNS (  username VARCHAR2(50) PATH '$.username',  gmail VARCHAR2(100) PATH '$.gmail',  password\_hash VARCHAR2(200) PATH '$.password\_hash',  role VARCHAR2(20) PATH '$.role'  )  ) jt;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  DBMS\_OUTPUT.PUT\_LINE('Skipping duplicate username or gmail');  END;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Import completed: ' || p\_filename);  DBMS\_LOB.FREETEMPORARY(l\_json);  EXCEPTION  WHEN OTHERS THEN  IF UTL\_FILE.IS\_OPEN(f) THEN  UTL\_FILE.FCLOSE(f);  END IF;  RAISE\_APPLICATION\_ERROR(-20050, 'Error importing users: ' || SQLERRM);  END import\_users\_from\_file; |

Листинг 4.1 – Скрипт процедуры процедуры импорта

В сочетании с аналогичной процедурой импорта она обеспечивает полный цикл обмена данными в удобном и универсальном формате JSON, повышая гибкость администрирования и сопровождения системы награждения музыкантов.

# 5. Тестирование производительности

Тестирование базы данных представляет собой процесс проверки корректности, надёжности, производительности и безопасности разработанной структуры и объектов базы данных на соответствие заданным требованиям и предметной области.

В процессе тестирования был выполнен запрос, генерирующий 100 000 записей в таблицу Users с автоматическим формированием разнообразных тестовых значений для ключевых атрибутов. Данный подход к заполнению данными позволяет провести измерение производительности запросов на выборку и фильтрацию записей в условиях, близких к реальной эксплуатационной нагрузке, а также оценить масштабируемость системы и её отзывчивость при обработке значительного объёма информации о пользователях.

Скрипт представлен в листинге 5.1.

|  |
| --- |
| BEGIN  FOR i IN 1..100000 LOOP  INSERT INTO Users (username, gmail, password\_hash, role)  VALUES (  'user\_' || i,  'user\_' || i || '@gmail.com',  pkg\_admin.hash\_password('password\_' || i),  'guest\_user');  IF MOD(i, 1000) = 0 THEN  COMMIT;  END IF;  END LOOP;  COMMIT;  END;  / |

Листинг 5.1 – Заполнение таблицы 100000 строк

Для анализа производительности был выполнен скрипт, содержащий типовой запрос на поиск пользователей с применением фильтров по роли. Анализ выполнялся с использованием режима AUTOTRACE, активированного командой SET AUTOTRACE ON, который позволяет получить план выполнения запроса и статистику его работы непосредственно при выполнении операции.

Содержимое скрипта представлено в листинге 5.2.

|  |
| --- |
| SELECT /\*+ gather\_plan\_statistics \*/ role  FROM Users  WHERE role = 'manager\_user'; |

Листинг 5.2 – Содержимое скрипта для анализа выполнения запроса

Результат анализа выполнения представлен на рисунке 4.1

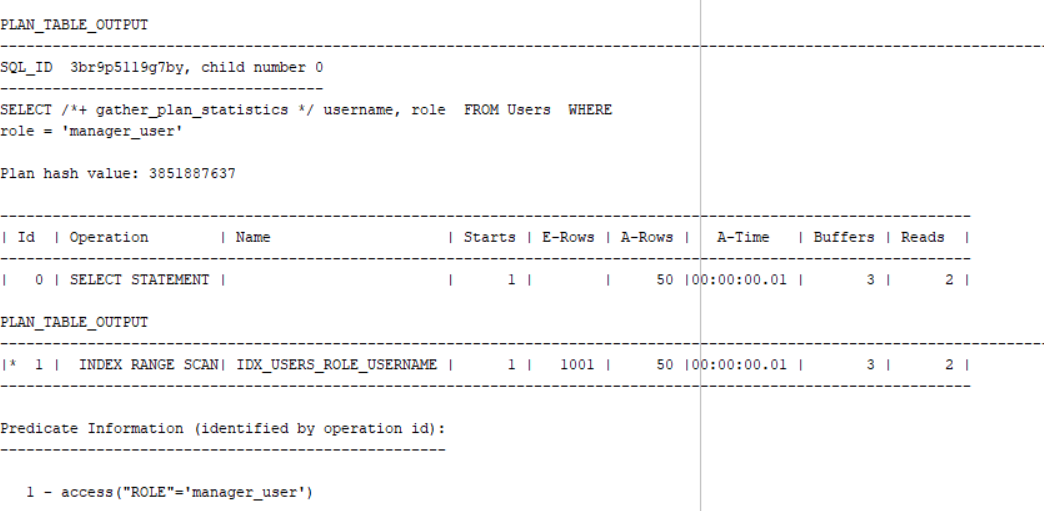


Рисунок 5.1 – Анализ результатов

Также можно проверить производительность, используя технологию Explain, которая покажет большее количество информации о запросе. На рисунке 5.2 представлены результаты для сравнения.

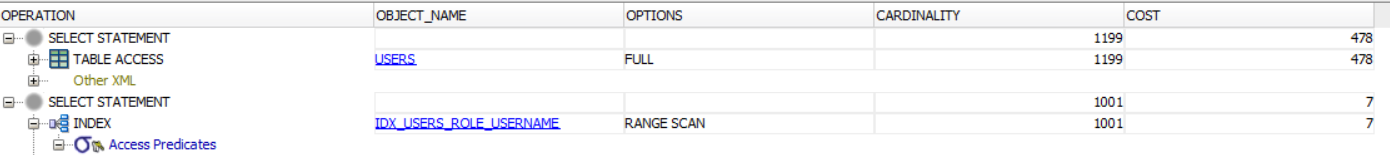


Рисунок 5.2 – Анализ результатов с помощью технологии Explain

Анализ плана выполнения запроса (EXPLAIN PLAN) показал, что благодаря созданному индексу IDX\_USERS\_ROLE\_USERNAME операция выборки осуществлена методом INDEX RANGE SCAN: система использовала диапазонное сканирование по индексу, затратив всего 3 буфера и 2 операции чтения с диска, а время выполнения составило менее 0,01 секунды даже при большом количестве записей.

# 6. Описание технологии и ее применения в базе данных

Репликация между серверами СУБД Oracle с помощью GoldenGate – это способ держать две базы данных на двух разных серверах в точной идентичности. Для демонстрации данной технологии был выбран Oracle GoldenGate.

Когда в главной базе-источнике добавляются, изменяются или удаляются записи, GoldenGate мгновенно просматривает это изменение в журнале Oracle и точно так же повторяет его во второй базе-приёмнике. Таким образом, главная база работает как обычно, а вторая база является всегда актуальной копией. Например, на случай отказа сервера, для отчётов или для разгрузки главной базы. Всё происходит практически без задержки и без нагрузки на главную базу.

## 6.1 Настройка репликации базы данных

Для демонстрации и настройки технологии репликации между серверами систем управления баз данных была создана вторая, абсолютно идентичная база данных.

Настройка включает этапы установки GoldenGate, создания менеджеров процессов, конфигурирования экстракторов и репликаторов, а также тестирования корректности передачи изменений (INSERT, UPDATE, DELETE) из исходной базы данных в реплицируемую.

Первым этапом является создание общего пользователя GoldenGate, к которому в будущем будет осуществляться подключение, а также настройка системы на готовность к репликации.

Скрипты для настройки системы представлены в листинге 6.1.

|  |
| --- |
| alter system set enable\_goldengate\_replication=true scope=both;  alter database force logging;  alter database add supplemental log data;  GRANT OGG\_CAPTURE TO GGADMIN;  GRANT CONNECT, RESOURCE TO GGADMIN;  ALTER DATABASE ARCHIVELOG;  ALTER DATABASE FORCE LOGGING; |

Листинг 6.1 – Настройка системы

Следующий этап – создание хранилища учётных данных, или credential store, в среде Oracle GoldenGate с помощью команды ADD CREDENTIALSTORE. В это хранилище добавляются учётные записи (алиасы) пользователей баз данных источника и приёмника с зашифрованным хранением паролей.

Далее в параметры конфигурационных файлов экстракторов или репликаторов указываются соответствующие алиасы вместо прямого написания логинов и паролей.

Такой подход снижает вероятность ошибок при конфигурировании процессов GoldenGate.

Команды для его настройки представлены в листинге 6.2.

|  |
| --- |
| --создание подключения к базе данных-ресурсу  ALTER CREDENTIALSTORE ADD USER ggadmin@//oracle-source:1521/FREEPDB1 ALIAS srcdb DOMAIN OracleGoldenGate PASSWORD Kate2006  --создание подключения к базе данных-цели  ALTER CREDENTIALSTORE ADD USER ggadmin@//oracle-target:1521/FREEPDB1 ALIAS tgtdb DOMAIN OracleGoldenGate PASSWORD Kate2006 |

Листинг 6.2 – Настройка хранилища учетных данных

После этого следует подключиться к самому Oracle GoldenGate с базы данных-источника, чтобы продолжить настройку уже самой репликации, в частности, добавления пути извлечения и его регистрация.

Скрипт для добавления и регистрации пути извлечения представлен в листинге 6.3.

|  |
| --- |
| ogg-microservices /u01/ogg/bin/adminclient  connect https://localhost:443 deployment Demo as ggadmin password Kate2006 !  DBLOGIN USERIDALIAS srcdb DOMAIN OracleGoldenGate  ADD EXTRACT ext1 INTEGRATED TRANLOG BEGIN NOW  ADD EXTTRAIL am EXTRACT ext1  REGISTER EXTRACT ext1 DATABASE |

Листинг 6.3 – Регистрация пути извлечения

При создании пути открывается файл параметра в доступном для среды текстовом редакторе, в который нужно прописать таблицы, используемые для репликации. Важно отметить, что файл параметра чувствителен к регистру.

Скрипт для заполнения файла параметров представлен в листинге 6.4.

|  |
| --- |
| EXTRACT ext1  USERIDALIAS srcdb DOMAIN OracleGoldenGate  EXTTRAIL aa  TABLE ADMIN\_USER.MUSICIANS;  TABLE ADMIN\_USER.ALBUMS;  TABLE ADMIN\_USER.MUSIC;  TABLE ADMIN\_USER.USERS;  TABLE ADMIN\_USER.AWARDS;  TABLE ADMIN\_USER.NOMINANTS;  TABLE ADMIN\_USER.WINNERS;  TABLE ADMIN\_USER.VOTES;  TABLE ADMIN\_USER.ALBUMRATINGS; |

Листинг 6.4 – Файл параметров для пути извлечения

Чтобы добавить путь для репликации, следует подключиться к Oracle GoldenGate с целевой базы данных.

Скрипт для добавления пути репликации представлен в листинге 6.5.

|  |
| --- |
| ogg-microservices /u01/ogg/bin/adminclient  connect https://localhost:443 deployment Demo as ggadmin password Kate2006 !  DBLOGIN USERIDALIAS tgtdb DOMAIN OracleGoldenGate  ADD REPLICAT rep1 INTEGRATED EXTTRAIL am CHECKPOINTTABLE ggadmin.chkpt |

Листинг 6.5 – Добавление пути репликации

Для репликации также открывается файл параметра, в котором нужно вписать, какие таблицы обмениваются информацией с целевой базой данных. Список таблиц из файла параметров пути извлечения должен быть идентичен списку таблиц в файле параметров для репликации.

Файл параметров для репликации представлен в листинге 6.6.

|  |
| --- |
| REPLICAT rep1  USERIDALIAS tgtdb DOMAIN OracleGoldenGate  HANDLECOLLISIONS  MAP admin\_user.Musicians, TARGET admin\_user.Musicians;  MAP admin\_user.Albums, TARGET admin\_user.Albums;  MAP admin\_user.Music, TARGET admin\_user.Music;  MAP admin\_user.Users, TARGET admin\_user.Users;  MAP admin\_user.Awards, TARGET admin\_user.Awards;  MAP admin\_user.Nominants, TARGET admin\_user.Nominants;  MAP admin\_user.Winners, TARGET admin\_user.Winners;  MAP admin\_user.Votes, TARGET admin\_user.Votes;  MAP admin\_user.AlbumRatings, TARGET admin\_user.AlbumRatings; |

Листинг 6.6 – Файл параметров репликации

Затем нужно начать репликацию. Для проверки настройки репликации можно использовать команды, представленные в листинге 6.7.

|  |
| --- |
| VIEW REPORT ext1  VIEW REPORT rep1  INFO EXTRACT ext1  INFO REPLICAT rep1 |

Листинг 6.7 – Команды, используемые для проверки репликации

Использование первых двух команд демонстрирует логирование соответствующего процесса в файле событий GoldenGate и в отчётном файле конкретного экстрактора или репликатора. Эти команды следует применять для оперативной локализации ошибок, анализа предупреждений и отслеживания хода репликации, поскольку они позволяют быстро выявить причины остановки процессов или несоответствия данных.

Последние две команды выводят на экран текущее состояние соответствующего процесса, включая позицию в журнале транзакций, количество обработанных операций и время последней активности. При успешной настройке и запуске репликации оба процесса должны иметь состояние «RUNNING» (со статусом ABENDED только в случае ошибки), что подтверждает стабильную синхронизацию изменений между исходной и целевой базами данных системы награждения музыкантов. Постоянный мониторинг этих состояний обеспечивает своевременное реагирование на возможные сбои и гарантирует высокую доступность данных.

## 6.2 Проверка работы репликации базы данных

В рамках тестирования функциональности репликации базы данных будут созданы подключения к целевой базе данных: target\_guest, target\_manager и target\_admin, соответствующие ролям системы с аналогичными привилегиями, что и в исходной базе.

Для проверки корректности репликации будет выполнена вставка новой информации в одну из таблиц с использованием подключения к базе данных-источнику.

Информация, содержащаяся в таблице до изменения данных, представлена на рисунке 6.1.

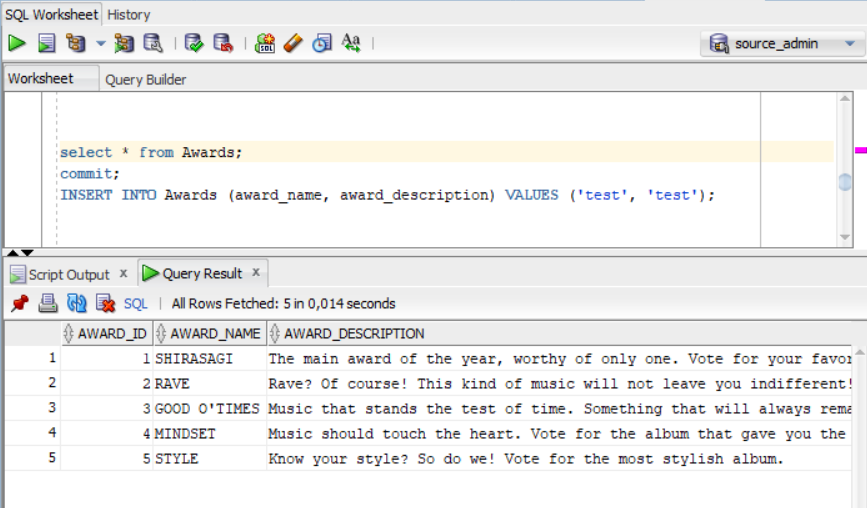


Рисунок 6.1 – Содержание таблицы до изменения данных

После выполнения скрипта, подключившись к целевой базе данных через одно из созданных подключений, будет выполнен запрос на выборку вставленных данных. При успешной репликации добавленные записи должны появиться в целевой базе с минимальной задержкой, что подтвердит работоспособность механизма Oracle GoldenGate и обеспечит синхронизацию изменений в реальном времени.

Демонстрация успешной репликации представлена на рисунке 6.2.

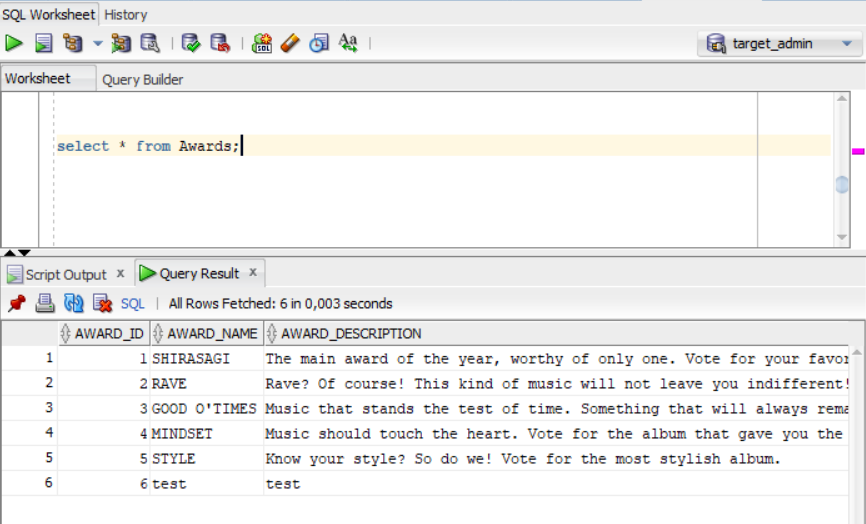


Рисунок 6.2 – Демонстрация успешной репликации

Это подтверждает, что система обеспечивает надёжную отказоустойчивость и высокую доступность данных благодаря успешной настройке репликации с использованием Oracle GoldenGate. Изменения, внесённые в исходную базу данных (вставка, обновление или удаление записей), оперативно и точно воспроизводятся в целевой базе с минимальной задержкой.

## 6.3 Вывод

В ходе разработки и тестирования базы данных была успешно реализована технология репликации с использованием Oracle GoldenGate, обеспечивающая асинхронную синхронизацию данных в реальном времени между основной и резервной базами данных системы награждения музыкантов. Создание идентичной целевой базы, настройка процессов экстракции и репликации, применение credential store для безопасного хранения учётных данных, использование синонимов для прозрачного доступа к объектам, а также мониторинг состояний процессов и тестирование операций вставки.

# 7. Краткое описание приложения для демонстрации

Для наглядной демонстрации возможностей разработанной базы данных системы награждения музыкантов было создано клиентское приложение на платформе Node.js с использованием языка JavaScript. Приложение служит практическим интерфейсом для взаимодействия пользователей с системой и визуализации работы всех реализованных объектов базы данных, включая процедуры, функции и триггеры.

Приложение построено по принципу клиент-серверной архитектуры с прямым подключением к Oracle Database через модуль oracledb. Основной особенностью приложения является адаптивный интерфейс, который динамически изменяется в зависимости от роли вошедшего пользователя. После успешной аутентификации система автоматически определяет роль (гость, менеджер или администратор) и предоставляет соответствующий набор функций через интуитивно понятную систему навигации.

Для каждой роли реализован специализированный интерфейс. Гости (обычные пользователи) получают доступ к главной странице со списком всех премий. При клике на премию осуществляется переход на страницу голосования, где можно однократно проголосовать за номинированный альбом. При выборе конкретного альбома открывается детальная страница с информацией о списке музыки в альбоме и списком комментариев. Менеджеры и администраторы имеют доступ к отдельной административной странице, представляющей собой набор интерактивных таблиц для просмотра и изменения данных о музыкантах, альбомах, музыке, номинациях, победителях и пользователях.

Приложение демонстрирует все ключевые функции системы через вызов соответствующих процедур и функций Oracle. Особое внимание уделено работе с пользовательским контентом – приложение отображает отзывы, обеспечивает ограничение на однократное голосование в каждой премии и реализует механизмы безопасной обработки данных между клиентом и сервером БД.

Приложение полностью покрывает функциональные требования к системе и служит эффективным инструментом для демонстрации возможностей разработанной базы данных в реальных сценариях использования.

# 8. Руководство пользователя

При запуске приложения открывается окно авторизации. Существует три роли: клиент, менеджер и администратор. Регистрация нового пользователя выполняется процедурой register\_user с предварительной валидацией Gmail через validate\_email и хэшированием пароля функцией hash\_password, а аутентификация – процедурой login\_user.

Окно авторизации представлено на рисунке 8.1.

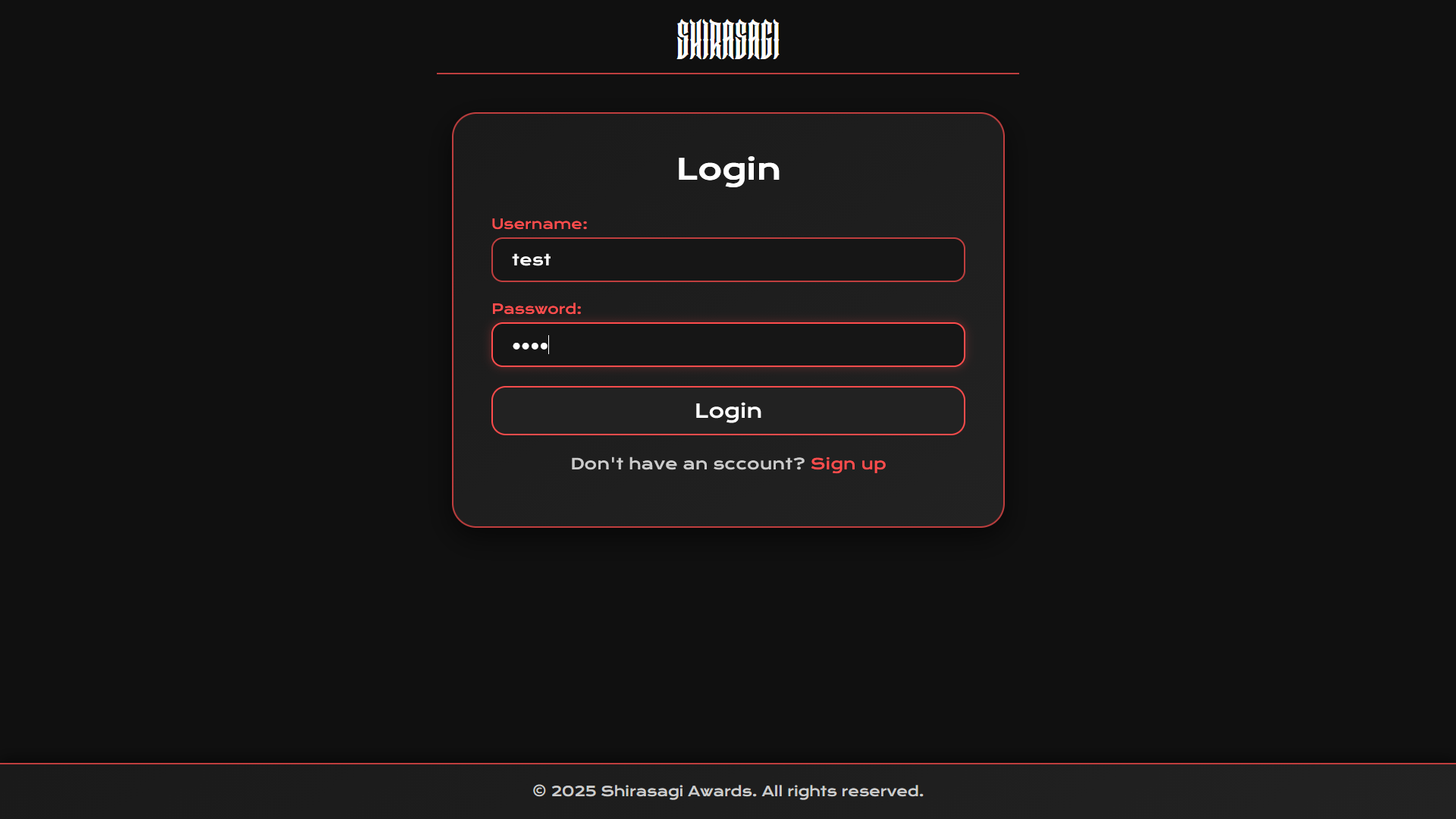


Рисунок 8.1 – Окно авторизации

После авторизации гость получает доступ к базовому функционалу системы. Он может просматривать список всех премий через get\_awards, получать список номинированных альбомов в конкретной премии через get\_albums\_by\_award, а также проверять свой голос в данной премии с помощью get\_user\_vote.

Страница для голосования представлена на рисунке 6.2.

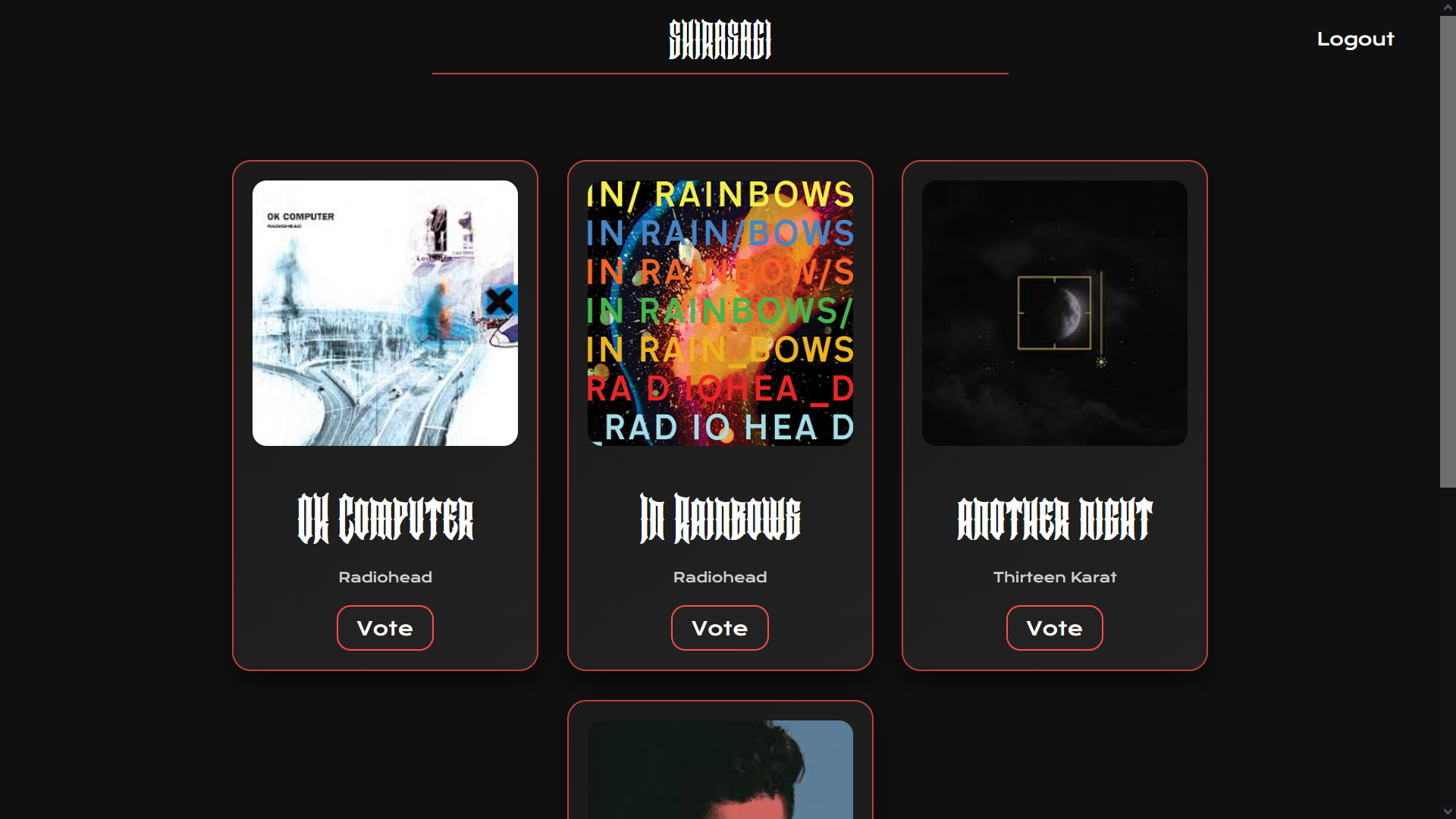


Рисунок 6.2 – Страница голосования

Голосование за альбом осуществляется процедурой vote\_for\_album с возможностью добавления комментария. Для детального просмотра альбома доступны процедуры get\_album\_description (основная информация об альбоме и музыканте), get\_tracks\_by\_album (список треков альбома) и get\_comments\_by\_album (комментарии и отзывы других пользователей). Кроме того, гость может просматривать список всех победителей премий через get\_winners, когда голосование завершается.

После авторизации администратор получает доступ к инструментам управления менеджерами. Он может добавлять добавлять менеджеров через add\_manager, обновлять данные менеджеров с помощью update\_manager, удалять менеджеров процедурой delete\_manager, а также просматривать список всех менеджеров через get\_all\_managers.

Страница администратора представлена на рисунке 6.3.

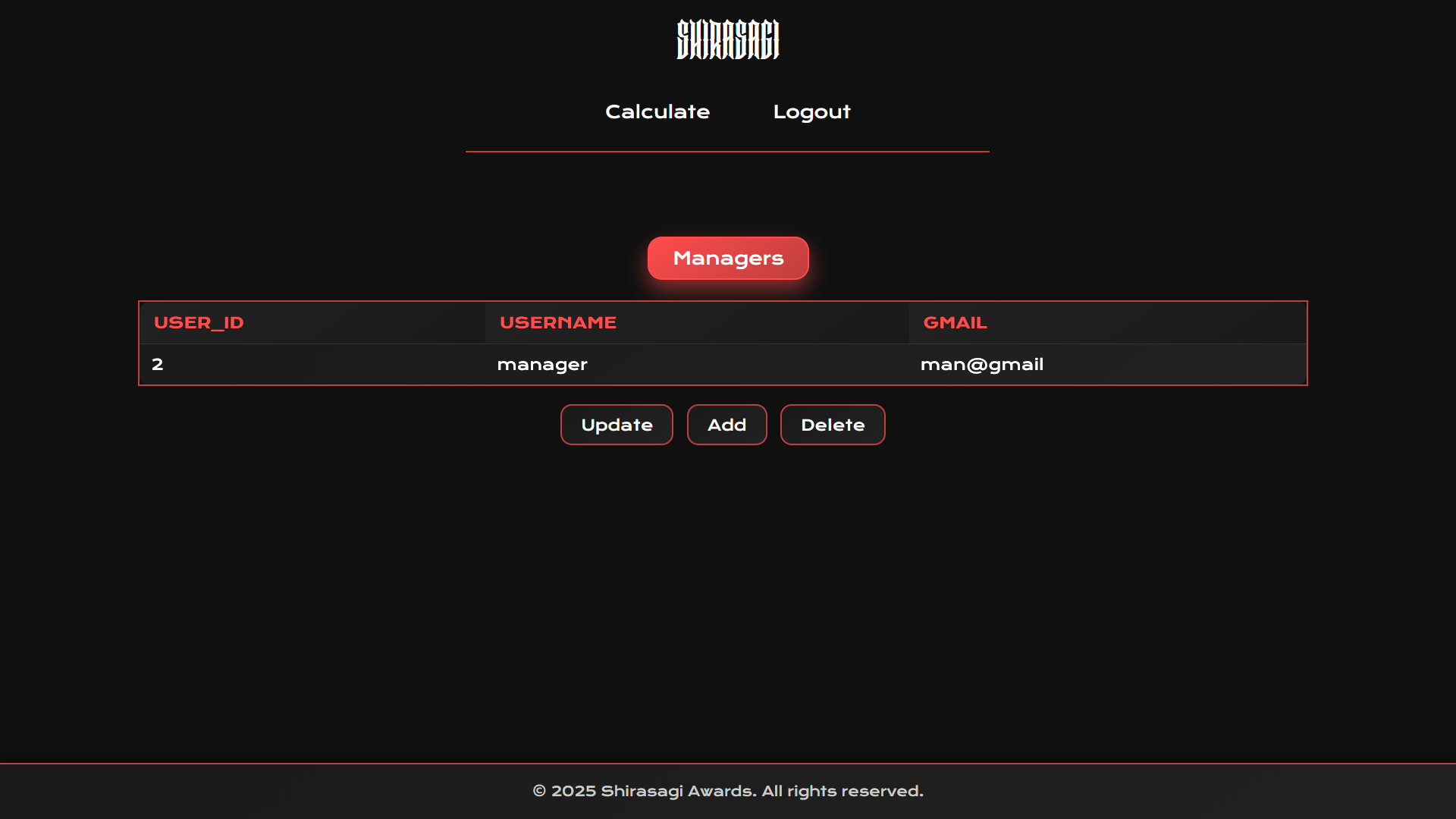


Рисунок 6.3 – Страница администратора

Менеджер после авторизации работает с основным содержимым системы. Управление музыкантами включает добавление (add\_musician), обновление (update\_musician, включая изменение статуса на disqualified), удаление (delete\_musician) и просмотр списка (get\_musicians). Для альбомов предусмотрены процедуры добавления (add\_album), обновления (update\_album), удаления (delete\_album) и получения списка (get\_albums). Работа с треками осуществляется через add\_track, update\_track, delete\_track, get\_tracks (все треки) и get\_tracks\_by\_album (треки конкретного альбома). Менеджер может просматривать и удалять голоса пользователей (get\_votes, delete\_vote), управлять номинантами (get\_nominants, add\_nominant), просматривать и обновлять награды (get\_awards, update\_award), а также запускать пересчёт победителей по количеству голосов (calculate\_winners), получать топ музыкантов (get\_top\_musicians) и общее количество победителей (get\_winners\_count).

# Заключение

В рамках курсового проекта была успешно разработана и реализована комплексная система управления музыкальными наградами на основе реляционной базы данных Oracle Database, обеспечивающая полный цикл процессов номинирования, голосования и награждения музыкантов. Созданная архитектура включает нормализованные таблицы с ключевыми сущностями (музыканты, альбомы, треки, награды, номинанты, победители, голоса), пакеты PL/SQL для разграничения доступа по ролям (гость, менеджер, администратор), функции валидации данных и процедуры автоматизации бизнес-логики (подсчёт голосов, определение победителей).

Настройка репликации с использованием Oracle GoldenGate гарантирует отказоустойчивость и высокую доступность, а клиентское приложение на Node.js с JavaScript предоставляет интуитивный интерфейс для взаимодействия пользователей с системой.

Полученный результат полностью соответствует функциональным требованиям и демонстрирует практическую ценность автоматизированной системы награждения музыкантов для повышения прозрачности и удобства проведения музыкальных премий. Проведенное тестирование подтвердило стабильность работы всех компонентов системы.

Полученный результат полностью соответствует функциональным требованиям и демонстрирует практическую ценность автоматизированной системы награждения музыкантов для повышения прозрачности и удобства проведения премий.

# Список используемых источников

1. Онлайн‑ресурс «Music Awards Japan» [Электронный ресурс] – Режим доступа: https://www.musicawardsjapan.com/en/ – Режим доступа: 15.11.2025
2. Онлайн‑ресурс «Billboard Music Awards» [Электронный ресурс] – Режим доступа: [https://www.billboardmusicawards/](https://www.bmw.ru/) – Дата доступа: 01.12.2025

# Приложение А

# 

# Приложение Б

|  |
| --- |
| CREATE TABLE Musicians (  musician\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  name VARCHAR2(100) NOT NULL,  info VARCHAR2(500),  status VARCHAR2(20) CHECK (status IN ('participating','disqualified')) NOT NULL  ) TABLESPACE music\_ts;  CREATE TABLE Albums (  album\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  musician\_id NUMBER NOT NULL,  title VARCHAR2(150) NOT NULL,  release\_date DATE,  link VARCHAR2(200) NOT NULL,  pic\_link VARCHAR2(100),  CONSTRAINT fk\_album\_musician FOREIGN KEY (musician\_id) REFERENCES Musicians(musician\_id) ON DELETE CASCADE  ) TABLESPACE music\_ts;  CREATE TABLE Music (  track\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  album\_id NUMBER NOT NULL,  title VARCHAR2(150) NOT NULL,  CONSTRAINT fk\_music\_album FOREIGN KEY (album\_id) REFERENCES Albums(album\_id) ON DELETE CASCADE  ) TABLESPACE music\_ts;  CREATE TABLE Users (  user\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  username VARCHAR2(50) NOT NULL UNIQUE,  gmail VARCHAR2(100) NOT NULL UNIQUE,  password\_hash VARCHAR2(200) NOT NULL,  role VARCHAR2(20) DEFAULT 'guest\_user' CONSTRAINT chk\_role CHECK (role IN ('admin\_user','manager\_user','guest\_user'))  ) TABLESPACE music\_ts;  CREATE TABLE Awards (  award\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  award\_name VARCHAR2(150) NOT NULL,  award\_description VARCHAR2(500)  ) TABLESPACE music\_ts;  CREATE TABLE Nominants (  award\_id NUMBER NOT NULL,  album\_id NUMBER NOT NULL,  CONSTRAINT pk\_nominants PRIMARY KEY (award\_id,album\_id),  CONSTRAINT fk\_nominant\_award FOREIGN KEY (award\_id) REFERENCES Awards(award\_id) ON DELETE CASCADE,  CONSTRAINT fk\_nominant\_album FOREIGN KEY (album\_id) REFERENCES Albums(album\_id) ON DELETE CASCADE  ) TABLESPACE music\_ts;  CREATE TABLE Winners (  award\_id NUMBER NOT NULL,  album\_id NUMBER NOT NULL,  musician\_id NUMBER NOT NULL,  CONSTRAINT pk\_winners PRIMARY KEY (award\_id,album\_id),  CONSTRAINT fk\_winners\_award FOREIGN KEY (award\_id) REFERENCES Awards(award\_id) ON DELETE CASCADE,  CONSTRAINT fk\_winners\_album FOREIGN KEY (album\_id) REFERENCES Albums(album\_id) ON DELETE CASCADE,  CONSTRAINT fk\_winners\_musician FOREIGN KEY (musician\_id) REFERENCES Musicians(musician\_id) ON DELETE CASCADE  ) TABLESPACE music\_ts;  CREATE TABLE Votes (  vote\_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  award\_id NUMBER NOT NULL,  album\_id NUMBER NOT NULL,  user\_id NUMBER NOT NULL,  comment\_text VARCHAR2(4000),  CONSTRAINT uq\_vote\_user\_award UNIQUE (user\_id,award\_id),  CONSTRAINT fk\_vote\_award FOREIGN KEY (award\_id) REFERENCES Awards(award\_id) ON DELETE CASCADE,  CONSTRAINT fk\_vote\_album FOREIGN KEY (album\_id) REFERENCES Albums(album\_id) ON DELETE CASCADE,  CONSTRAINT fk\_vote\_user FOREIGN KEY (user\_id) REFERENCES Users(user\_id) ON DELETE CASCADE,  CONSTRAINT fk\_vote\_nomination FOREIGN KEY (award\_id,album\_id) REFERENCES Nominants(award\_id,album\_id) ON DELETE CASCADE  ) TABLESPACE music\_ts;  CREATE TABLE AlbumRatings (  award\_id NUMBER NOT NULL,  album\_id NUMBER NOT NULL,  votes\_count NUMBER DEFAULT 0,  PRIMARY KEY (award\_id,album\_id),  CONSTRAINT fk\_rating\_award FOREIGN KEY (award\_id) REFERENCES Awards(award\_id) ON DELETE CASCADE,  CONSTRAINT fk\_rating\_album FOREIGN KEY (album\_id) REFERENCES Albums(album\_id) ON DELETE CASCADE  ) TABLESPACE music\_ts; |

# Приложение В

|  |
| --- |
| CREATE OR REPLACE PACKAGE pkg\_admin AS  FUNCTION hash\_password(p\_password VARCHAR2) RETURN VARCHAR2;  FUNCTION validate\_email(p\_email VARCHAR2) RETURN NUMBER;  PROCEDURE add\_admin(p\_username VARCHAR2, p\_gmail VARCHAR2, p\_password VARCHAR2);  PROCEDURE add\_manager(p\_username VARCHAR2, p\_gmail VARCHAR2, p\_password VARCHAR2);  PROCEDURE update\_manager(p\_user\_id NUMBER, p\_username VARCHAR2, p\_gmail VARCHAR2, p\_password VARCHAR2);  PROCEDURE delete\_manager(p\_user\_id NUMBER);  PROCEDURE get\_all\_managers(p\_cursor OUT SYS\_REFCURSOR);  END pkg\_admin;  /  CREATE OR REPLACE PACKAGE BODY pkg\_admin AS  --хешировать пароль  FUNCTION hash\_password(p\_password VARCHAR2) RETURN VARCHAR2 IS  v\_raw RAW(256);  v\_hex VARCHAR2(256);  BEGIN  v\_raw := DBMS\_CRYPTO.HASH(  UTL\_RAW.CAST\_TO\_RAW(p\_password),  DBMS\_CRYPTO.HASH\_SH256  );  v\_hex := RAWTOHEX(v\_raw);  RETURN v\_hex;  END;  --проверить правильность написания емейла  FUNCTION validate\_email(p\_email VARCHAR2) RETURN NUMBER IS  BEGIN  IF REGEXP\_LIKE(p\_email, '^[A-Za-z0-9.\_%+-]+@gmail\.com$') THEN  RETURN 1;  ELSE  RETURN 0;  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(  -21001,  'Error validating email: ' || SQLERRM  );  END;  PROCEDURE add\_admin(  p\_username VARCHAR2,  p\_gmail VARCHAR2,  p\_password VARCHAR2  ) IS  v\_hash VARCHAR2(256);  BEGIN  v\_hash := pkg\_admin.hash\_password(p\_password);  INSERT INTO Users (username, gmail, password\_hash, role)  VALUES (p\_username, p\_gmail, v\_hash, 'admin\_user');  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(  -20010,  'Error adding admin: User with this username or email already exists.'  );  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(  -20011,  'Error adding admin: ' || SQLERRM  );  END;  -- добавить менеджера  PROCEDURE add\_manager(  p\_username VARCHAR2,  p\_gmail VARCHAR2,  p\_password VARCHAR2  ) IS  v\_hash VARCHAR2(256);  BEGIN  v\_hash := pkg\_admin.hash\_password(p\_password);  INSERT INTO Users (username, gmail, password\_hash, role)  VALUES (p\_username, p\_gmail, v\_hash, 'manager\_user');  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(  -20020,  'Error adding manager: User with this username or email already exists.'  );  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(  -20021,  'Error adding manager: ' || SQLERRM  );  END;  -- обновить менеджера  PROCEDURE update\_manager(  p\_user\_id NUMBER,  p\_username VARCHAR2,  p\_gmail VARCHAR2,  p\_password VARCHAR2  ) IS  password\_hashik VARCHAR2(256) := NULL;  BEGIN  IF p\_password IS NOT NULL THEN  password\_hashik := pkg\_admin.hash\_password(p\_password);  END IF;  UPDATE Users  SET username = NVL(p\_username, username),  gmail = NVL(p\_gmail, gmail),  password\_hash = NVL(password\_hashik, password\_hash)  WHERE user\_id = p\_user\_id  AND role = 'manager\_user';  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(  -20022,  'Manager not found or user is not a manager: ID ' || p\_user\_id  );  END IF;  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(  -20023,  'Error updating manager: New username or email already in use.'  );  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(  -20024,  'Error updating manager: ' || SQLERRM  );  END;    -- удалить менеджера  PROCEDURE delete\_manager(p\_user\_id NUMBER) IS  BEGIN  DELETE FROM Users  WHERE user\_id = p\_user\_id  AND role = 'manager\_user';  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(  -20025,  'Manager not found or user is not a manager: ID ' || p\_user\_id  );  END IF;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(  -20026,  'Error deleting manager: ' || SQLERRM  );  END;  -- получить всех менеджеров  PROCEDURE get\_all\_managers(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT user\_id, username, gmail, password\_hash  FROM Users  WHERE role = 'manager\_user';  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(  -20027,  'Error fetching managers: ' || SQLERRM  );  END;  END pkg\_admin;  /  --интерфейс для манеджера  CREATE OR REPLACE PACKAGE pkg\_manager AS  -- Musicians  PROCEDURE add\_musician(p\_name VARCHAR2, p\_info VARCHAR2);  PROCEDURE update\_musician(p\_id NUMBER, p\_name VARCHAR2, p\_info VARCHAR2, p\_status VARCHAR2 DEFAULT NULL);  PROCEDURE get\_musicians(p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE delete\_musician(p\_musician\_id NUMBER);  -- Albums  PROCEDURE add\_album(p\_musician\_id NUMBER, p\_title VARCHAR2, p\_release\_date DATE,  p\_link VARCHAR2, p\_pic\_link VARCHAR2);  PROCEDURE update\_album(p\_album\_id NUMBER, p\_musician\_id NUMBER, p\_title VARCHAR2, p\_release\_date DATE,  p\_link VARCHAR2, p\_pic\_link VARCHAR2);  PROCEDURE delete\_album(p\_album\_id NUMBER);  PROCEDURE get\_albums(p\_cursor OUT SYS\_REFCURSOR);  -- Tracks  PROCEDURE add\_track(p\_album\_id NUMBER, p\_title VARCHAR2);  PROCEDURE update\_track(p\_track\_id NUMBER, p\_title VARCHAR2);  PROCEDURE delete\_track(p\_track\_id NUMBER);  PROCEDURE get\_tracks(p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE get\_tracks\_by\_album(p\_album\_id NUMBER, p\_cursor OUT SYS\_REFCURSOR);  -- Votes  PROCEDURE delete\_vote(p\_vote\_id NUMBER);  PROCEDURE get\_votes(p\_cursor OUT SYS\_REFCURSOR);  --Nominants  PROCEDURE get\_nominants(p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE add\_nominant(p\_award\_id NUMBER, p\_album\_id NUMBER);  -- Awards  PROCEDURE get\_awards(p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE update\_award(p\_award\_id NUMBER, p\_name VARCHAR2, p\_description VARCHAR2);  PROCEDURE calculate\_winners;  PROCEDURE get\_top\_musicians(p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE get\_winners\_count(p\_count OUT NUMBER);  END pkg\_manager;  /  CREATE OR REPLACE PACKAGE BODY pkg\_manager AS  -- Возвращает общее количество победителей  PROCEDURE get\_winners\_count(p\_count OUT NUMBER) IS  BEGIN  SELECT COUNT(\*) INTO p\_count FROM Winners;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20090, 'Ошибка при подсчёте победителей: ' || SQLERRM);  END;  -- Добавляет нового музыканта  PROCEDURE add\_musician(p\_name VARCHAR2, p\_info VARCHAR2) IS  BEGIN  INSERT INTO Musicians(name, info, status)  VALUES (p\_name, p\_info, 'participating');  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20010, 'Error adding musician: ' || SQLERRM);  END;  -- Обновляет данные музыканта  PROCEDURE update\_musician(  p\_id NUMBER,  p\_name VARCHAR2,  p\_info VARCHAR2,  p\_status VARCHAR2  ) IS  BEGIN  UPDATE Musicians  SET name = p\_name, info = p\_info, status = NVL(p\_status, status)  WHERE musician\_id = p\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20011, 'Musician not found: ' || p\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20012, 'Error updating musician: ' || SQLERRM);  END;  -- Удаляет музыканта  PROCEDURE delete\_musician(p\_musician\_id NUMBER) IS  BEGIN  DELETE FROM Musicians WHERE musician\_id = p\_musician\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20060, 'Musician not found: ' || p\_musician\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20061, 'Error deleting musician: ' || SQLERRM);  END;  -- Возвращает список всех музыкантов  PROCEDURE get\_musicians(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT musician\_id, name, info, status FROM Musicians;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20015, 'Error fetching musicians: ' || SQLERRM);  END;  -- Добавляет новый альбом  PROCEDURE add\_album(  p\_musician\_id NUMBER,  p\_title VARCHAR2,  p\_release\_date DATE,  p\_link VARCHAR2,  p\_pic\_link VARCHAR2  ) IS  v\_link VARCHAR2(200);  v\_pic\_link VARCHAR2(100);  BEGIN  v\_link := NVL(p\_link, 'https://open.spotify.com/track/2NqCUYqbMgHsDmSxJMTjBa?si=1c31c957ff394ec9');  v\_pic\_link := NVL(p\_pic\_link, '/img/DBimg/test.jpg');  INSERT INTO Albums(musician\_id, title, release\_date, link, pic\_link)  VALUES (p\_musician\_id, p\_title, p\_release\_date, v\_link, v\_pic\_link);  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20020, 'Error adding album: ' || SQLERRM);  END;  -- Обновляет данные альбома  PROCEDURE update\_album(  p\_album\_id NUMBER,  p\_musician\_id NUMBER,  p\_title VARCHAR2,  p\_release\_date DATE,  p\_link VARCHAR2,  p\_pic\_link VARCHAR2  ) IS  BEGIN  UPDATE Albums  SET title = p\_title, release\_date = p\_release\_date, link = p\_link, pic\_link = p\_pic\_link  WHERE album\_id = p\_album\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20021, 'Album not found: ' || p\_album\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20022, 'Error updating album: ' || SQLERRM);  END;  -- Удаляет альбом  PROCEDURE delete\_album(p\_album\_id NUMBER) IS  BEGIN  DELETE FROM Albums WHERE album\_id = p\_album\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20023, 'Album not found: ' || p\_album\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20024, 'Error deleting album: ' || SQLERRM);  END;  -- Возвращает список всех альбомов  PROCEDURE get\_albums(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT album\_id, musician\_id, title,  TO\_CHAR(release\_date, 'YYYY-MM-DD') AS release\_date,  link, pic\_link  FROM Albums;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20025, 'Error fetching albums: ' || SQLERRM);  END;  -- Возвращает список всех треков с альбомами и музыкантами  PROCEDURE get\_tracks(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT t.track\_id, t.album\_id, t.title,  a.title AS album\_title,  m.name AS musician\_name  FROM Music t  JOIN Albums a ON t.album\_id = a.album\_id  JOIN Musicians m ON a.musician\_id = m.musician\_id  ORDER BY t.track\_id;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20036, 'Error fetching all tracks: ' || SQLERRM);  END;  -- Добавляет трек в альбом  PROCEDURE add\_track(p\_album\_id NUMBER, p\_title VARCHAR2) IS  BEGIN  INSERT INTO Music(album\_id, title)  VALUES (p\_album\_id, p\_title);  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20030, 'Error adding track: ' || SQLERRM);  END;  -- Обновляет название трека  PROCEDURE update\_track(p\_track\_id NUMBER, p\_title VARCHAR2) IS  BEGIN  UPDATE Music SET title = p\_title WHERE track\_id = p\_track\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20031, 'Track not found: ' || p\_track\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20032, 'Error updating track: ' || SQLERRM);  END;  -- Удаляет трек  PROCEDURE delete\_track(p\_track\_id NUMBER) IS  BEGIN  DELETE FROM Music WHERE track\_id = p\_track\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20033, 'Track not found: ' || p\_track\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20034, 'Error deleting track: ' || SQLERRM);  END;  -- Возвращает треки конкретного альбома  PROCEDURE get\_tracks\_by\_album(p\_album\_id NUMBER, p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT track\_id, title FROM Music WHERE album\_id = p\_album\_id;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20035, 'Error fetching tracks: ' || SQLERRM);  END;  -- Удаляет голос  PROCEDURE delete\_vote(p\_vote\_id NUMBER) IS  BEGIN  DELETE FROM Votes WHERE vote\_id = p\_vote\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20040, 'Vote not found: ' || p\_vote\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20041, 'Error deleting vote: ' || SQLERRM);  END;  -- Возвращает список всех голосов  PROCEDURE get\_votes(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT vote\_id, user\_id, award\_id, album\_id, comment\_text FROM Votes;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20042, 'Error fetching votes: ' || SQLERRM);  END;  -- Возвращает список номинантов  PROCEDURE get\_nominants(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT n.award\_id, a.award\_name, n.album\_id,  alb.title AS album\_title,  m.name AS musician\_name  FROM Nominants n  JOIN Awards a ON n.award\_id = a.award\_id  JOIN Albums alb ON n.album\_id = alb.album\_id  JOIN Musicians m ON alb.musician\_id = m.musician\_id  ORDER BY a.award\_name, m.name, alb.title;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20070, 'Error fetching nominants: ' || SQLERRM);  END;  -- Добавляет альбом в номинацию  PROCEDURE add\_nominant(p\_award\_id NUMBER, p\_album\_id NUMBER) IS  BEGIN  INSERT INTO Nominants(award\_id, album\_id)  VALUES (p\_award\_id, p\_album\_id);  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(-20076, 'Error adding nominant: This album is already nominated for this award.');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20077, 'Error adding nominant: ' || SQLERRM);  END;  -- Возвращает список наград  PROCEDURE get\_awards(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT award\_id, award\_name, award\_description FROM Awards;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20050, 'Error fetching awards: ' || SQLERRM);  END;  -- Обновляет данные награды  PROCEDURE update\_award(p\_award\_id NUMBER, p\_name VARCHAR2, p\_description VARCHAR2) IS  BEGIN  UPDATE Awards  SET award\_name = p\_name, award\_description = p\_description  WHERE award\_id = p\_award\_id;  IF SQL%ROWCOUNT = 0 THEN  RAISE\_APPLICATION\_ERROR(-20051, 'Award not found: ' || p\_award\_id);  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20052, 'Error updating award: ' || SQLERRM);  END;  -- Пересчитывает победителей по количеству голосов  PROCEDURE calculate\_winners IS  BEGIN  DELETE FROM Winners;  INSERT INTO Winners(award\_id, album\_id, musician\_id)  SELECT r.award\_id, r.album\_id, a.musician\_id  FROM AlbumRatings r  JOIN Albums a ON r.album\_id = a.album\_id  WHERE (r.award\_id, r.votes\_count) IN (  SELECT award\_id, MAX(votes\_count)  FROM AlbumRatings  GROUP BY award\_id  );  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE\_APPLICATION\_ERROR(-20080, 'Ошибка расчёта победителей: ' || SQLERRM);  END;  -- Возвращает топ музыкантов по количеству голосов  PROCEDURE get\_top\_musicians(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT w.album\_id, w.musician\_id, COUNT(v.vote\_id) AS vote\_count  FROM Winners w  LEFT JOIN Votes v ON w.album\_id = v.album\_id  GROUP BY w.album\_id, w.musician\_id  ORDER BY vote\_count DESC;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20054, 'Error fetching top musicians: ' || SQLERRM);  END;  END pkg\_manager;  /  --ИНТЕрфейс для геста  CREATE OR REPLACE PACKAGE pkg\_guest AS  --функции  FUNCTION hash\_password(p\_password VARCHAR2) RETURN VARCHAR2;  FUNCTION validate\_email(p\_email VARCHAR2) RETURN NUMBER;  --процедуры  PROCEDURE vote\_for\_album(p\_user\_id NUMBER, p\_award\_id NUMBER, p\_album\_id NUMBER, p\_comment\_text VARCHAR2);  PROCEDURE register\_user(p\_username IN VARCHAR2, p\_gmail IN VARCHAR2, p\_password\_hash IN VARCHAR2);  PROCEDURE login\_user(p\_username IN VARCHAR2, p\_password\_hash IN VARCHAR2, p\_user\_id OUT NUMBER, p\_role OUT VARCHAR2);  PROCEDURE get\_awards(p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE get\_albums\_by\_award(p\_award\_id NUMBER, p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE get\_user\_vote(p\_user\_id IN votes.user\_id%TYPE, p\_award\_id IN votes.award\_id%TYPE, p\_album\_id OUT votes.album\_id%TYPE);  PROCEDURE get\_comments\_by\_album(p\_album\_id IN votes.album\_id%TYPE, p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE get\_album\_description(p\_album\_id IN albums.album\_id%TYPE, p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE get\_tracks\_by\_album(p\_album\_id NUMBER, p\_cursor OUT SYS\_REFCURSOR);  PROCEDURE get\_winners(p\_cursor OUT SYS\_REFCURSOR);  END pkg\_guest;  /  CREATE OR REPLACE PACKAGE BODY pkg\_guest AS  -- Хэширует пароль  FUNCTION hash\_password(p\_password VARCHAR2) RETURN VARCHAR2 IS  v\_raw RAW(256);  v\_hex VARCHAR2(256);  BEGIN  v\_raw := DBMS\_CRYPTO.HASH(UTL\_RAW.CAST\_TO\_RAW(p\_password), DBMS\_CRYPTO.HASH\_SH256);  v\_hex := RAWTOHEX(v\_raw);  RETURN v\_hex;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21000, 'Error hashing password: ' || SQLERRM);  END;  -- Проверяет валидность Gmail  FUNCTION validate\_email(p\_email VARCHAR2) RETURN NUMBER IS  BEGIN  IF REGEXP\_LIKE(p\_email, '^[A-Za-z0-9.\_%+-]+@gmail\.com$') THEN  RETURN 1;  ELSE  RETURN 0;  END IF;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21001, 'Error validating email: ' || SQLERRM);  END;  -- Голосует за альбом  PROCEDURE vote\_for\_album(p\_user\_id NUMBER, p\_award\_id NUMBER, p\_album\_id NUMBER, p\_comment\_text VARCHAR2) IS  v\_dummy NUMBER;  BEGIN  SELECT 1 INTO v\_dummy FROM Nominants WHERE award\_id = p\_award\_id AND album\_id = p\_album\_id;  INSERT INTO Votes(award\_id, album\_id, user\_id, comment\_text)  VALUES(p\_award\_id, p\_album\_id, p\_user\_id, NVL(p\_comment\_text,''));  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20010, 'This album does not participate in this nomination.');  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(-20011, 'User has already voted for this award.');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20012, 'Voting is closed ' || SQLERRM);  END;  -- Регистрирует нового пользователя  PROCEDURE register\_user(p\_username IN VARCHAR2, p\_gmail IN VARCHAR2, p\_password\_hash IN VARCHAR2) IS  BEGIN  INSERT INTO Users(username, gmail, password\_hash, role) VALUES(p\_username, p\_gmail, p\_password\_hash, 'guest\_user');  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(-21020, 'Username or Gmail already exists.');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21021, 'Error registering user: ' || SQLERRM);  END;  -- Авторизация пользователя  PROCEDURE login\_user(p\_username IN VARCHAR2, p\_password\_hash IN VARCHAR2, p\_user\_id OUT NUMBER, p\_role OUT VARCHAR2) IS  BEGIN  SELECT user\_id, role INTO p\_user\_id, p\_role  FROM Users  WHERE username = p\_username AND password\_hash = p\_password\_hash;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-21030, 'Invalid username or password.');  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21031, 'Error during login: ' || SQLERRM);  END;  -- Получает список наград  PROCEDURE get\_awards(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR SELECT award\_id, award\_name, award\_description FROM Awards;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21040, 'Error fetching awards: ' || SQLERRM);  END;  -- Получает список альбомов по награде  PROCEDURE get\_albums\_by\_award(p\_award\_id NUMBER, p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT a.album\_id, a.musician\_id, m.name AS musician\_name,  a.title, TO\_CHAR(a.release\_date, 'YYYY-MM-DD') as release\_date,  a.link, a.pic\_link  FROM Albums a  JOIN Nominants n ON a.album\_id = n.album\_id  JOIN Musicians m ON a.musician\_id = m.musician\_id  WHERE n.award\_id = p\_award\_id AND m.status = 'participating';  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21041, 'Error fetching albums by award: ' || SQLERRM);  END;  -- Получает голос пользователя для конкретной награды  PROCEDURE get\_user\_vote(p\_user\_id IN votes.user\_id%TYPE, p\_award\_id IN votes.award\_id%TYPE, p\_album\_id OUT votes.album\_id%TYPE) IS  BEGIN  SELECT album\_id INTO p\_album\_id FROM Votes WHERE user\_id = p\_user\_id AND award\_id = p\_award\_id;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  p\_album\_id := NULL;  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21042, 'Error fetching user vote: ' || SQLERRM);  END;  -- Получает комментарии к альбому  PROCEDURE get\_comments\_by\_album(p\_album\_id IN votes.album\_id%TYPE, p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT u.username, v.comment\_text  FROM Votes v  JOIN Users u ON v.user\_id = u.user\_id  WHERE v.album\_id = p\_album\_id;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21043, 'Error fetching comments: ' || SQLERRM);  END;  -- Получает описание альбома  PROCEDURE get\_album\_description(p\_album\_id IN albums.album\_id%TYPE, p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT a.album\_id, a.musician\_id, m.name, m.info, a.title,  TO\_CHAR(a.release\_date, 'YYYY-MM-DD') as release\_date, a.link, a.pic\_link  FROM Albums a  JOIN Musicians m ON a.musician\_id = m.musician\_id  WHERE a.album\_id = p\_album\_id AND m.status = 'participating';  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21044, 'Error fetching album description: ' || SQLERRM);  END;  -- Получает треки альбома  PROCEDURE get\_tracks\_by\_album(p\_album\_id NUMBER, p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT t.track\_id, t.title  FROM Music t  JOIN Albums a ON t.album\_id = a.album\_id  JOIN Musicians m ON a.musician\_id = m.musician\_id  WHERE t.album\_id = p\_album\_id AND m.status = 'participating';  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-21045, 'Error fetching tracks: ' || SQLERRM);  END;  -- Получает список победителей  PROCEDURE get\_winners(p\_cursor OUT SYS\_REFCURSOR) IS  BEGIN  OPEN p\_cursor FOR  SELECT w.award\_id, aw.award\_name, w.album\_id, al.title AS album\_title, w.musician\_id, m.name AS musician\_name, al.pic\_link  FROM Winners w  JOIN Awards aw ON aw.award\_id = w.award\_id  JOIN Albums al ON al.album\_id = w.album\_id  JOIN Musicians m ON m.musician\_id = w.musician\_id  ORDER BY w.award\_id;  EXCEPTION  WHEN OTHERS THEN  RAISE\_APPLICATION\_ERROR(-20110, 'Error fetching winners: ' || SQLERRM);  END;  END pkg\_guest;  / |

# Приложение Г

|  |
| --- |
| --триггер, чтобы подсчитать рейтинг  CREATE OR REPLACE TRIGGER admin\_user.trg\_vote\_insert  AFTER INSERT ON admin\_user.Votes  FOR EACH ROW  DECLARE  BEGIN  MERGE INTO admin\_user.AlbumRatings ar  USING (SELECT :NEW.album\_id AS album\_id, :NEW.award\_id AS award\_id FROM dual) src  ON (ar.album\_id = src.album\_id AND ar.award\_id = src.award\_id)  WHEN MATCHED THEN  UPDATE SET ar.votes\_count = ar.votes\_count + 1  WHEN NOT MATCHED THEN  INSERT (album\_id, award\_id, votes\_count)  VALUES (src.album\_id, src.award\_id, 1);  END;  /  -- голосовать нельзя после подсчета победитеоей  CREATE OR REPLACE TRIGGER trg\_block\_votes\_after\_winners  BEFORE INSERT ON Votes  FOR EACH ROW  DECLARE  v\_count NUMBER;  BEGIN  SELECT COUNT(\*)  INTO v\_count  FROM Winners;  IF v\_count > 0 THEN  RAISE\_APPLICATION\_ERROR(  -20120,  'Voting is closed. Winners have already been determined.'  );  END IF;  END;  /  --один музыкант не может иметь более двух альбомов в сумме за все номинации  CREATE OR REPLACE TRIGGER trg\_album\_limit  BEFORE INSERT ON Albums  FOR EACH ROW  DECLARE  v\_count NUMBER;  BEGIN  SELECT COUNT(\*)  INTO v\_count  FROM Albums a  JOIN Nominants n ON a.album\_id = n.album\_id  WHERE a.musician\_id = :NEW.musician\_id;  IF v\_count >= 2 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'This musician already has 2 albums in nominations.');  END IF;  END;  /  -- при равенстве голосов выбирается рандомный победитель  CREATE OR REPLACE TRIGGER trg\_winner\_random  BEFORE INSERT ON Winners  FOR EACH ROW  DECLARE  v\_max\_votes NUMBER;  v\_count NUMBER;  v\_random\_album\_id NUMBER;  BEGIN  SELECT MAX(votes\_count) INTO v\_max\_votes  FROM AlbumRatings  WHERE award\_id = :NEW.award\_id;  SELECT COUNT(\*) INTO v\_count  FROM AlbumRatings  WHERE award\_id = :NEW.award\_id  AND votes\_count = v\_max\_votes;  IF v\_count > 1 THEN  SELECT album\_id INTO v\_random\_album\_id  FROM (SELECT album\_id FROM AlbumRatings WHERE award\_id = :NEW.award\_id AND votes\_count = v\_max\_votes ORDER BY DBMS\_RANDOM.VALUE)  WHERE ROWNUM = 1;  :NEW.album\_id := v\_random\_album\_id;  END IF;  END;  / |

# Приложение Д

|  |
| --- |
| -- для менеджера  CREATE SYNONYM manager\_user.Musicians FOR admin\_user.Musicians;  CREATE SYNONYM manager\_user.Albums FOR admin\_user.Albums;  CREATE SYNONYM manager\_user.Music FOR admin\_user.Music;  CREATE SYNONYM manager\_user.Awards FOR admin\_user.Awards;  CREATE SYNONYM manager\_user.Nominants FOR admin\_user.Nominants;  CREATE SYNONYM manager\_user.Winners FOR admin\_user.Winners;  CREATE SYNONYM manager\_user.Votes FOR admin\_user.Votes;  CREATE SYNONYM manager\_user.pkg\_manager FOR admin\_user.pkg\_manager;  --for guest  CREATE SYNONYM guest\_user.Musicians FOR admin\_user.Musicians;  CREATE SYNONYM guest\_user.Albums FOR admin\_user.Albums;  CREATE SYNONYM guest\_user.Music FOR admin\_user.Music;  CREATE SYNONYM guest\_user.Awards FOR admin\_user.Awards;  CREATE SYNONYM guest\_user.Winners FOR admin\_user.Winners;  CREATE SYNONYM guest\_user.Nominants FOR admin\_user.Nominants;  CREATE SYNONYM guest\_user.Votes FOR admin\_user.Votes;  CREATE SYNONYM guest\_user.pkg\_guest FOR admin\_user.pkg\_guest;  CREATE SYNONYM guest\_user.trg\_vote\_insert FOR admin\_user.trg\_vote\_insert; |