МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных для программного средства поиска фильмов и сериалов с применением технологии продвинутой аналитики»

Выполнил студент Смолик В.А.

(Ф.И.О.)

Руководитель работы Нистюк О.А.

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

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

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

Курсовая работа защищена с оценкой

Минск 2023

**Содержание**

[Введение 5](#_Toc153376529)

[1 Постановка задачи 6](#_Toc153376530)

[2 Проектирование базы данных 7](#_Toc153376531)

[2.1 Схема базы данных 7](#_Toc153376532)

[2.2 Таблицы базы данных 8](#_Toc153376533)

[2.3 Вывод по разделу 12](#_Toc153376534)

[3 Разработка объектов базы данных 13](#_Toc153376535)

[3.1 Табличное пространство 13](#_Toc153376536)

[3.1 Таблицы 13](#_Toc153376537)

[3.2.1 Таблицы 13](#_Toc153376538)

[3.2.2 Вложенные таблицы 14](#_Toc153376539)

[3.3 Процедуры 14](#_Toc153376540)

[3.4 Функции 15](#_Toc153376541)

[3.5 Триггеры базы данных 15](#_Toc153376542)

[3.6 Вывод по разделу 16](#_Toc153376543)

[4 Описание процедур экспорта и импорта 17](#_Toc153376544)

[4.1 Описание процедуры экспорта 17](#_Toc153376545)

[4.2 Описание процедуры импорта 17](#_Toc153376546)

[4.3 Вывод по разделу 18](#_Toc153376547)

[5 Тестирование производительности 19](#_Toc153376548)

[5.1 Оптимизирование запросов 19](#_Toc153376549)

[5.2 Проверка выполнения хранимых процедур 21](#_Toc153376550)

[5.2 Вывод по разделу 22](#_Toc153376551)

[6 Описание и применение технологии в БД 23](#_Toc153376552)

[6.1 Описание взаимодействия с технологией 23](#_Toc153376553)

[6.2 Вывод по разделу 26](#_Toc153376554)

[7 Краткое описание приложения для демонстрации 27](#_Toc153376555)

[8 Руководство пользователя 30](#_Toc153376556)

[Заключение 31](#_Toc153376557)

[Список используемых источников 32](#_Toc153376558)

[Приложение A Создание таблиц 33](#_Toc153376559)

[Приложение Б Создание вложенных таблиц таблиц 35](#_Toc153376560)

[Приложение С Процедуры и функции 38](#_Toc153376561)

[Приложение D Создание триггера 48](#_Toc153376562)

# **Введение**

Базы данных являются неотъемлемой частью современных информационных систем, применяемых в различных сферах деятельности. Они обеспечивают эффективное хранение, организацию и обработку данных любого объема и сложности. Одной из наиболее мощных и широко используемых систем управления базами данных (СУБД) является Oracle Database, которая является стандартом для управления данными в крупных организациях и предприятиях. Она отличается высокой производительностью, масштабируемостью и надежностью, а также предлагает множество функций и возможностей для обработки, и анализа данных.

Дополнительные технологии, такие как Oracle Advanced Analytics, позволяют проводить анализ данных, обнаруживать скрытые закономерности, прогнозировать спрос и разрабатывать ценовые стратегии. Эти инструменты позволяют организациям получать ценную информацию из их данных и принимать более обоснованные и эффективные решения.

В сфере кинематографии важную роль играет база данных, которая служит ключевым компонентом управления информацией на веб-сайте о фильмах. База данных хранит разнообразную информацию о фильмах, включая данные об актерах, режиссерах, жанрах, рейтингах и других аспектах кинематографии. Корректное проектирование и реализация базы данных являются важными факторами для обеспечения эффективного функционирования сайта и удовлетворения потребностей администраторов и пользователей. Ошибки или недочеты в базе данных могут привести к проблемам, таким как неправильное отображение информации или замедление работы сайта.

Разработка базы данных для веб-сайта о фильмах или сериалах является сложным и трудоемким процессом, требующим внимательного анализа и глубоких знаний в данной области. Основная цель проекта заключается в создании реляционной базы данных, которая будет эффективно хранить информацию о фильмах, актерах, режиссерах и других связанных данных, а также обеспечивать быстрый доступ к этой информации. В рамках проекта также будет разработано веб-приложение, которое позволит пользователям легко и удобно получать информацию о фильмах, оставлять отзывы и оценки, а также делиться комментариями с другими пользователями. Веб-приложение было реализовано на Java, используя Web MVC фреймворк.

# **1 Постановка задачи**

В рамках данного проекта требуется спроектировать инфраструктуру базы данных веб-сайта о фильмах. Для этого необходимо провести анализ требований и определить следующие элементы и их содержимое в нашей базе данных: таблицы и связи между ними, ограничения целостности, профили безопасности, пользователи, триггеры, хранимые процедуры, функции и индексы. Затем необходимо разработать эти объекты в базе данных, используя СУБД Oracle 12c и написание SQL-скриптов.

Для заполнения таблиц данными предусмотрен импорт из JSON-файлов. Кроме того, будет использована технология Oracle Advanced Analytics для анализа данных в базе данных о количестве посещений пользователей в определенном временном интервале.

В рамках проекта также был разработан веб-сайт, который демонстрирует функционал и возможности, предоставляемые базой данных через процедуры и функции. Важными функциями приложения будут:

* определение ролей (администратор, пользователь);
* управление пользователями (регистрация, авторизация, добавление, изменение, удаление);
* управление информацией о фильмах или сериалах (добавление, изменение, удаление);
* управление отзывами о фильме или сериале (добавление, удаление);
* управление пользовательскими реакциями (добавление, удаление);
* управление возможностью просмотреть позже (добавление, удаление)

Реализация этих операций будет осуществляться с помощью хранимых процедур и функций.

Важным этапом проекта будет тестирование производительности базы данных на таблице, содержащей не менее 100 000 строк. В случае необходимости будут внесены изменения в структуру базы данных для оптимизации производительности.

Применить технологию Oracle Advanced Analytics для анализа данных в базе данных веб-сайта о фильмах и проанализировать преимущества (полезные функции), которые предоставляет данная технология.

2 Проектирование базы данных

2.1 Схема базы данных

Для начала создадим отдельную базу данных для курсового проекта с помощью утилиты Database Configuration Assistant. Далее определим, какие таблицы нам понадобятся для хранения информации о фильмах.

В базе данных о фильмах, а также сериалах, может быть множество таблиц, в зависимости от требований и функциональности проекта. Давайте ограничимся основными таблицами, необходимыми для хранения информации о пользователях, фильмах, актерах, режиссерах, жанрах и интересных фактах. Предварительная диаграмма таблиц и их отношения представлены на рисунке 2.1.

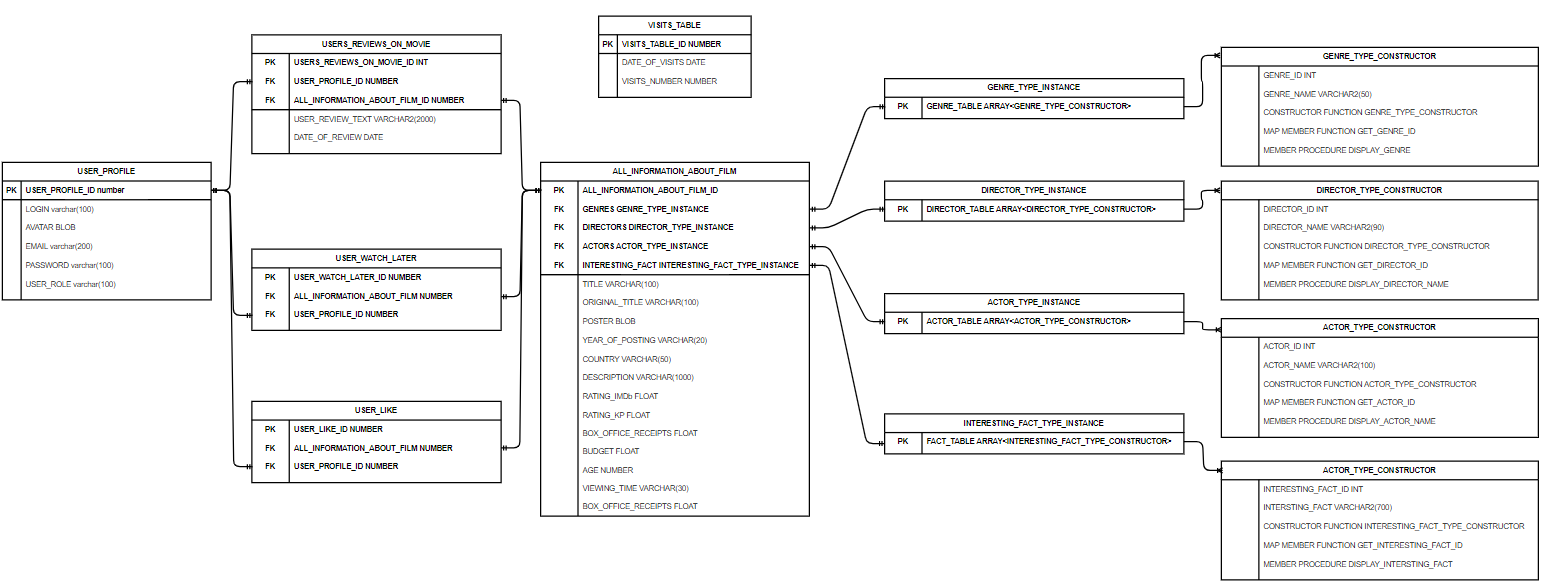


Рисунок 2.1 – Схема базы данных

Также в этих таблицах указаны предварительные типы данных для столбцов, которые могут измениться при необходимости.

Для эффективного использования системы важно определить роли пользователей и их сценарии использования. Сценарии описывают, как каждый пользователь будет взаимодействовать с системой в соответствии с их ролью. Это помогает определить доступные функции, доступные данные и организацию навигации в системе. Для визуализации взаимодействия между пользователями и системой используются диаграммы UML.

Роли пользователей определяют набор прав и привилегий, предоставляемых пользователям в системе. Каждая роль имеет свои уникальные возможности и доступ к функциям системы. В вашем проекте роли пользователей включают Admin (администратор) и User (пользователь). Это означает, что администратор будет иметь расширенные возможности и полный доступ ко всем функциям системы, в то время как пользователь будет иметь ограниченный доступ только к определенным функциям и данным, соответствующим его роли.

Далее определим, как будет происходить взаимодействие с системой. Наглядно это продемонстрирует Use-case диаграмма – рисунок 2.2.

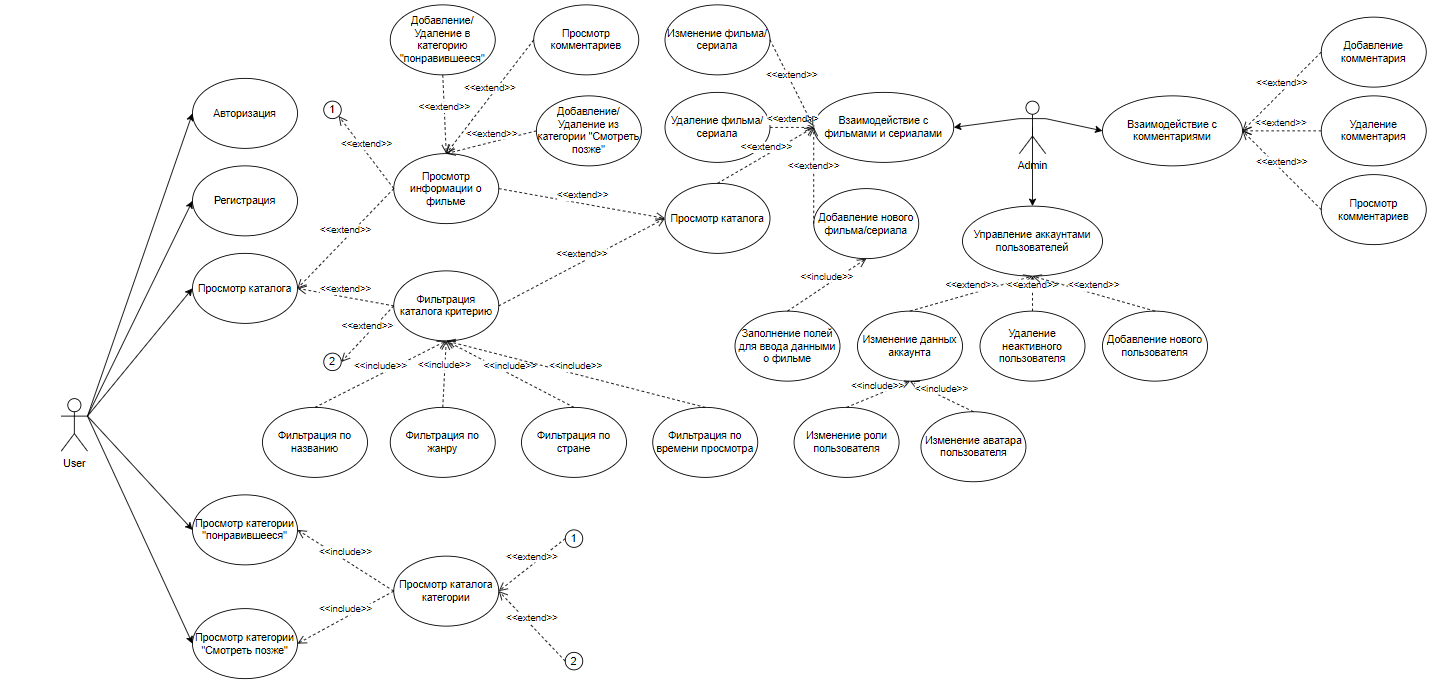


Рисунок 2.2 – Use Case диаграмма

Как мы установили ранее, нам понадобится две группы пользователей – Администраторы (пользователи, которые выполняют управление системой), а также обычные пользователи. Все действия данных пользователей будут выполняться через хранимые процедуры и функции.

2.2 Таблицы базы данных

Теперь опишем таблицы базы данных: названия, названия столбцов, типы данных столбцов, описание содержания столбцов. Описание представлено в таблицах 2.1 – 2.10.

Таблица 2.1 – Содержание таблицы USER\_PROFILE (Профиль пользователя)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| USER\_PROFILE\_ID | NUMBER | Уникальный идентификатор пользователя |
| LOGIN | varchar(100) | Логин пользователя |
| AVATAR | BLOB | Аватар пользователя |
| EMAIL | varchar(200) | Email пользователя |
| PASSWORD | varchar(100) | Пароль пользователя(передается в бд в закодированном виде) |
| USER\_ROLE | varchar(100) | Роль пользователя(Admin или User) |

Эта таблица хранит в себе данные о пользовательских профилях.

Таблица 2.2 – Содержание таблицы USERS\_REVIEWS\_ON\_MOVIE (Пользовательские отзывы на фильм)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| USERS\_REVIEWS\_ON\_MOVIE\_ID | INT | Уникальный идентификатор отзыва |
| USER\_REVIEW\_TEXT | VARCHAR2(2000) | Текст отзыва |
| DATE\_OF\_REVIEW | DATE | Дата написания пользователем отзыва |

Продолжение таблицы 2.2

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| USER\_PROFILE\_ID | NUMBER | ID пользователя, который написал отзыв |
| ALL\_INFORMATION\_ABOUT\_FILM\_ID | NUMBER | ID фильма, на который написан отзыв |

В этой таблице содержатся данные о отзывах, который были оставлены определенным пользователем на фильм.

Таблица 2.3 – Содержание таблицы VISITS\_TABLE (таблица активности)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| VISITS\_TABLE\_ID | NUMBER | Уникальный идентификатор дня |
| DATE\_OF\_VISITS | DATE | Дата дня |
| VISITS\_NUMBER | NUMBER | Количество авторизаций за день |

Эта таблица хранит в себе данные о количестве авторизаций в определенный день.

Таблица 2.4 – Содержание таблицы ALL\_INFORMATION\_ABOUT\_FILM (Информация о фильме)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ALL\_INFORMATION\_ABOUT\_FILM\_ID | NUMBER | Уникальный идентификатор фильма |
| TITLE | VARCHAR(100) | Название фильма |
| ORIGINAL\_TITLE | VARCHAR(100) | Название фильма (в оригинале) |
| POSTER | BLOB | Постер фильма |
| YEAR\_OF\_POSTING | VARCHAR(20) | Дата премьеры фильма |
| COUNTRY | VARCHAR(50) | Страна производитель |
| DESCRIPTION | VARCHAR(1000) | Описание фильма |
| RATING\_IMDb | FLOAT | Рейтинг IMDb |
| RATING\_KP | FLOAT | Рейтинг kinopoisk |
| BOX\_OFFICE\_RECEIPTS | FLOAT | Суммарные кассовые сборы |
| BUDGET | FLOAT | Бюджет фильма |
| AGE | NUMBER | Возрастное ограничение |
| VIEWING\_TIME | VARCHAR(30) | Время просмотра фильма |
| GENRES | GENRE\_TYPE\_INSTANCE | Жанры фильма (Вложенная пользовательская таблица) |
| DIRECTORS | DIRECTOR\_TYPE\_INSTANCE | Режиссеры фильма (Вложенная пользовательская таблица) |
| ACTORS | ACTOR\_TYPE\_INSTANCE | Актеры, снявшиеся в фильме (Вложенная пользовательская таблица) |
| INTERESTING\_FACT | INTERESTING\_FACT\_TYPE\_INSTANCE | Интересные факты о фильме (Вложенная пользовательская таблица) |

В данной таблице содержатся данные о Фильме, которые будут затем преобразованы и отображены на странице с фильмом.

Таблица 2.5 – Содержание таблицы GENRE\_TYPE\_INSTANCE (Пользовательская вложенная таблица, состоящая из массива объектов GENRE\_TYPE\_CONSTRUCTOR, содержание GENRE\_TYPE\_CONSTRUCTOR представлено ниже)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| GENRE\_ID | INT | Уникальный идентификатор жанра, генерируется через SEQUENCE в конструкторе. |
| GENRE\_NAME | VARCHAR2(50) | Название жанра |
| GENRE\_TYPE\_CONSTRUCTOR | CONSTRUCTOR FUNCTION | Конструктор для создания объекта типа GENRE\_TYPE\_CONSTRUCTOR,  возвращает объект. |
| GET\_GENRE\_ID | MAP MEMBER FUNCTION | Функция получения ID жанра, возвращает INT. |
| DISPLAY\_GENRE | MEMBER PROCEDURE | Процедура отображения данных о объекте(вывод ID и имени жанра) |

В данной таблице содержится массив GENRE\_TYPE\_CONSTRUCTOR, который содержит внутри себя 2 поля, конструктор, процедуру и функцию.

Таблица 2.6 – Содержание таблицы DIRECTOR\_TYPE\_INSTANCE (Пользовательская вложенная таблица, состоящая из массива объектов DIRECTOR\_TYPE\_CONSTRUCTOR, содержание DIRECTOR\_TYPE\_CONSTRUCTOR представлено ниже)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| DIRECTOR\_ID | INT | Уникальный идентификатор режиссера, генерируется через SEQUENCE в конструкторе. |
| DIRECTOR\_NAME | VARCHAR2(90) | Имя режиссера |
| DIRECTOR\_TYPE\_CONSTRUCTOR | CONSTRUCTOR FUNCTION | Конструктор для создания объекта типа DIRECTOR\_TYPE\_CONSTRUCTOR,  возвращает объект. |
| GET\_DIRECTOR\_ID | MAP MEMBER FUNCTION | Функция получения ID режиссера, возвращает INT. |
| DISPLAY\_DIRECTOR\_NAME | MEMBER PROCEDURE | Процедура отображения данных о объекте (вывод ID и имени режиссера) |

В данной таблице содержится массив DIRECTOR\_TYPE\_CONSTRUCTOR, который содержит внутри себя 2 поля, конструктор, процедуру и функцию. Поля используется для непосредственного хранения данных.

Таблица 2.7 – Содержание таблицы ACTOR\_TYPE\_INSTANCE (Пользовательская вложенная таблица, состоящая из массива объектов ACTOR\_TYPE\_CONSTRUCTOR, содержание ACTOR\_TYPE\_CONSTRUCTOR представлено ниже)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ACTOR\_ID | INT | Уникальный идентификатор актера, генерируется через SEQUENCE в конструкторе. |

Продолжение таблицы 2.7

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ACTOR\_NAME | VARCHAR2(100) | Имя актера |
| ACTOR\_TYPE\_CONSTRUCTOR | CONSTRUCTOR FUNCTION | Конструктор для создания объекта типа ACTOR\_TYPE\_CONSTRUCTOR, возвращает объект. |
| GET\_ACTOR\_ID | MAP MEMBER FUNCTION | Функция получения ID актера, возвращает INT. |
| DISPLAY\_ACTOR\_NAME | MEMBER PROCEDURE | Процедура отображения данных о объекте(вывод ID и имени актера) |

В данной таблице содержится массив ACTOR\_TYPE\_CONSTRUCTOR, который содержит внутри себя 2 поля, конструктор, процедуру и функцию.

Таблица 2.8 – Содержание таблицы INTERESTING\_FACT\_INSTANCE (Пользовательская вложенная таблица)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| INTERESTING\_FACT\_ID | INT | Уникальный идентификатор факта, генерируется через SEQUENCE в конструкторе. |
| INTERSTING\_FACT | VARCHAR2(700) | Текст факта |
| INTERESTING\_FACT\_TYPE\_CONSTRUCTOR | CONSTRUCTOR FUNCTION | Конструктор для создания объекта типа INTERESTING\_FACT\_TYPE\_CONSTRUCTOR,  возвращает объект. |
| GET\_INTERESTING\_FACT\_ID | MAP MEMBER FUNCTION | Функция получения ID факта, возвращает INT. |
| DISPLAY\_INTERSTING\_FACT | MEMBER PROCEDURE | Процедура отображения данных о объекте (вывод ID и текст факта) |

В данной таблице содержится массив INTERESTING\_FACT\_TYPE\_CONSTRUCTOR, который содержит внутри себя 2 поля, конструктор, процедуру и функцию.

Таблица 2.9 – Содержание таблицы USER\_WATCH\_LATER (таблица для категории «Просмотреть позже»)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| USER\_WATCH\_LATER\_ID | INT | Уникальный идентификатор добавления в «Просмотреть позже» |
| ALL\_INFORMATION\_ABOUT\_FILM\_ID | NUMBER | ID фильма, который нужно добавить в категорию «Просмотреть позже» |
| USER\_PROFILE\_ID | NUMBER | ID пользователя, который добавляет фильм в категорию «Просмотреть позже» |

В предпоследней таблице хранит данные о фильмах, которые пользователь добавил в категорию «Просмотреть позже».

Таблица 2.10 – Содержание таблицы USER\_LIKE (таблица для категории «Понравившееся»)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| USER\_LIKE\_ID | INT | Уникальный идентификатор добавления в «Понравившееся» |
| ALL\_INFORMATION\_ABOUT\_FILM\_ID | NUMBER | ID фильма, который нужно добавить в категорию «Понравившееся» |
| USER\_PROFILE\_ID | NUMBER | ID пользователя, который добавляет фильм в категорию «Понравившееся» |

Последняя таблица содержит данные о фильмах, которые пользователь добавил в категорию «Понравившееся».

Основные таблицы, которые обеспечат минимальную функциональность – USER\_PROFILE, ALL\_INFORMATION\_ABOUT\_FILM, GENRE\_TYPE\_INSTANCE, DIRECTOR\_TYPE\_INSTANCE, ACTOR\_TYPE\_INSTANCE и INTERESTING\_FACT\_TYPE\_INSTANCE. Таблицы USERS\_REVIEWS\_ON\_MOVIE, VISITS\_TABLE, USER\_WATCH\_LATER, USER\_LIKE– для дополнительного функционала. Первая – для реализации системы комментариев. Вторая таблица используется для трекинга количества посещений веб-сайта за определенный день. Идея данной таблица заключается в следующем: при авторизации с клиентской части приложения посылается PATCH запрос на сервер, который вызывает сервис, который в свою очередь вызывает репозиторий. Через репозиторий вызывается процедура, которая при помощи insert вставляет данные в таблицу VISITS\_TABLE, при помощи before триггера проверяется первое ли это посещение в дне, если да, то данные вставляются без изменений, если нет – то прошлое количество посещений в таблице увеличивается на единицу. Третья таблица из дополнительных используется для того, чтобы пользователь мог добавить фильм в категорию «Просмотреть позже», четвертая, в свою очередь, для добавления в категорию «Понравившееся».

2.3 Вывод по разделу

В данном разделе мы начали работу над созданием базы данных веб-сайта фильмов и сериалов: создана подключаемая база данных, спроектированы таблицы базы данных, столбцы и типы данных для них, определена модель взаимодействия с системой, а также каким образом будет реализован необходимый функционал при помощи объектов базы данных.

3 Разработка объектов базы данных

**3.1 Табличное пространство**

После создания подключаемой базы данных для нашего проекта и до создания таблиц и заполнения её данными нам необходимо создать табличное пространство – листинг 3.1

CREATE TABLESPACE CP\_TS

DATAFILE 'CP\_TS.DBF'

SIZE 50M

AUTOEXTEND ON NEXT 1M

MAXSIZE 150M

EXTENT MANAGEMENT LOCAL;

Листинг 3.1 – Создание табличного пространства

После создания пользователей необходимо сделать это табличное пространство для них табличным пространством по умолчанию и выделить квоту.

3.1 Таблицы

3.2.1 Таблицы

Таблица – это структурированное представление данных в базе данных, организованное в виде матрицы из столбцов и строк. Каждый столбец таблицы представляет отдельное поле, имеющее уникальное имя и тип данных, который определяет характер информации, которую оно хранит.

Ключевое поле в таблице играет важную роль, так как оно служит уникальным идентификатором для каждой записи. Это поле обеспечивает однозначное определение каждой строки таблицы и гарантирует уникальность значений в этом поле для каждой записи. Ключевое поле позволяет установить ссылку на конкретные данные и обеспечивает возможность идентификации и доступа к ним.

Из общего можно отметить, что в каждой таблице есть поле, представляющее ID, которое является первичным ключом и заполняется с помощью Identity с шагом 1. Числовые типы данных представлены типами NUMBER, FLOAT, INT. В таблицах USER\_PROFILE и ALL\_INFORMATION\_ABOUT\_FILM присутствует тип BLOB для реализации хранения изображений внутри бд, также присутствует тип DATE для хранения даты, например в таблице USERS\_REVIEWS\_ON\_MOVIE. Символьные данные представлены типами varchar2 и varchar. На столбцах стоит ограничение целостности «not null».

После создания таблиц они были заполнены небольшим количеством тестовых данных для проверки работоспособности и разработки объектов следующих пунктов.

В данном проекте базы данных содержится десять таблиц, каждая из которых описана в разделе 2. Соответствующий SQL-код для создания этих таблиц приведен в приложении A.

3.2.2 Вложенные таблицы

Вложенная таблица (Nested Table) - это коллекция или массив, которая может быть вложена в столбец таблицы базы данных. Она представляет собой структуру данных, которая может содержать несколько значений внутри каждой ячейки столбца.

Связывается же он с основной таблицей по отношению «один ко многим». Каждая запись в основной таблице может иметь связанный набор значений вложенной таблицы. Таким образом, вложенная таблица позволяет хранить и обрабатывать повторяющиеся данные в контексте одной строки основной таблицы.

Если необходимо гибкое решение для хранения и обработки коллекций данных в базе данных, то можно использовать их. Они могут использоваться для реализации списков, массивов или других структур данных, которые требуют хранения нескольких значений внутри каждой ячейки столбца. Вложенные таблицы обеспечивают удобный доступ к элементам коллекции и позволяют выполнять операции добавления, удаления, обновления и поиска данных внутри коллекции.

Их использование может улучшить организацию данных и сделать код более понятным и эффективным. Они часто применяются для реализации связей «многие ко многим» между таблицами или для хранения повторяющихся данных, которые не требуют отдельной таблицы и связей.

Код создания вложенных таблиц представлен в приложении B.

3.3 Процедуры

Хранимая процедура - это именованный фрагмент кода, написанный на языке T-SQL, который может быть создан, изменен и удален с использованием соответствующих операторов (CREATE, ALTER, DROP). Она представляет собой набор инструкций, которые могут быть вызваны и выполнены в базе данных.

Хранимые процедуры могут иметь входные параметры, которые позволяют передавать значения в процедуру при ее вызове, и могут возвращать результаты выполнения. Результаты могут быть в виде целочисленного значения с использованием оператора RETURN, одного или нескольких результирующих наборов с операторами SELECT, или вывода в стандартный поток с помощью оператора PRINT.

Также они могут содержать различные операторы T-SQL, такие как операторы для создания, изменения и удаления данных (DDL, DML и TCL). Они также могут использовать конструкции TRY/CATCH для обработки ошибок, работать с курсорами для обхода результатов запросов, использовать временные таблицы и другие средства, доступные в языке T-SQL.

Вызов хранимой процедуры осуществляется с помощью оператора EXECUTE (или EXEC), за которым следует имя процедуры и необходимые аргументы, если они есть.

Процедуры широко применяются для упрощения и стандартизации выполнения сложных и повторяющихся операций в базе данных, а также для обеспечения безопасности и повышения производительности.

В приложении C представлен листинг SQL-кода для создания процедур, использующихся в курсовом проекте.

3.4 Функции

Функция в PL/SQL - это модуль, который возвращает значение с помощью оператора RETURN. В отличие от процедур, функции могут использовать только входные параметры или не иметь их вовсе, а не аргументы OUT или IN OUT. Вызов функции всегда встроен в исполняемый оператор, может быть частью выражения или использоваться в качестве значения по умолчанию для переменных при их объявлении.

Возвращаемое функцией значение принадлежит определенному типу данных и может быть использовано в выражениях с аналогичным типом данных. Функции способствуют модульному подходу в разработке, позволяя выносить общую логику в отдельные модули. Это упрощает отладку, оптимизацию и обслуживание кода, так как функции могут быть повторно использованы в различных частях программы.

Одним из преимуществ использования функций является возможность разделения кода на более мелкие и самодостаточные модули, что облегчает понимание и поддержку программы. Функции также могут быть оптимизированы независимо от остального кода, что может привести к повышению производительности системы.

В приложении С представлен листинг SQL-кода для создания функций, осуществляющих различные операции.

**3.5 Триггеры базы данных**

Триггер - это особый тип хранимой процедуры, которая не вызывается пользователем напрямую, а автоматически срабатывает при определенных действиях по модификации данных. Он применяется для обеспечения целостности данных и реализации сложной бизнес-логики в базе данных, запускаются сервером базы данных автоматически при попытке изменения данных в таблице, с которой они связаны. Как только происходит действие, вызывающее срабатывание триггера, все модификации данных, выполненные им, рассматриваются как часть текущей транзакции. Это означает, что изменения, произведенные триггером, могут быть подвержены откату, если возникнет ошибка или нарушится целостность данных.

Также они могут выполнять различные операции, такие как вставка, обновление или удаление данных, выполнять сложные проверки, вычисления и взаимодействия с другими таблицами или процедурами в базе данных. Их можно использовать для автоматического обновления связанных данных, регистрации аудита, применения правил бизнес-логики или обеспечения согласованности данных.

Триггеры предоставляют мощный механизм для контроля и автоматизации операций в базе данных, но их использование также требует осторожности и тщательного планирования, чтобы избежать нежелательных побочных эффектов и обеспечить эффективность работы системы.

Для базы данных был создан триггер TR\_VISITS\_TABLE для таблицы VISITS\_TABLE. Триггер выполняется перед операцией вставки (BEFORE INSERT) для каждой строки (FOR EACH ROW). При помощи данного триггера проверяется первое ли это посещение в дне, если да, то данные вставляются без изменений, если нет – то прошлое количество посещений в таблице увеличивается на единицу. Листинг создания триггера представлен в приложении D.

* 1. Вывод по разделу

В данном разделе было описано и выполнено создание объектов для базы данных веб-сайта, таких как табличное пространство, роли и пользователи, таблицы, триггеры, хранимые процедуры. Также было определено взаимодействие всех этих элементов между собой.

# **4 Описание процедур экспорта и импорта**

4.1 Описание процедуры экспорта

База данных веб-сайта фильмов и сериалов может включать в себя большое количество информации. Здесь и возникает вопрос о том, каким образом можно эффективно экспортировать и импортировать данные. Для хранения данных был выбран JSON-формат. Теперь предстоит разработать соответствующие хранимые процедуры для импорта/экспорта данных. Разберём структуру процедур на примере таблицы VISITS\_TABLE – листинг 4.1.

|  |
| --- |
| CREATE OR REPLACE DIRECTORY MY\_DIRECTORY AS 'C:\JSON';  CREATE OR REPLACE PROCEDURE ExportVisitsTableToJson(  p\_directory VARCHAR2,  p\_file\_name VARCHAR2  ) AS  file\_handle UTL\_FILE.FILE\_TYPE;  cursor\_data SYS\_REFCURSOR;  json\_data CLOB;  BEGIN  file\_handle := UTL\_FILE.FOPEN(p\_directory, p\_file\_name, 'W');  OPEN cursor\_data FOR  SELECT JSON\_OBJECT(  'VISITS\_TABLE\_ID' VALUE VISITS\_TABLE\_ID,  'DATE\_OF\_VISITS' VALUE TO\_CHAR(DATE\_OF\_VISITS, 'YYYY-MM-DD'),  'VISITS\_NUMBER' VALUE VISITS\_NUMBER  ) AS json\_object  FROM VISITS\_TABLE;  LOOP  FETCH cursor\_data INTO json\_data;  EXIT WHEN cursor\_data%NOTFOUND;  UTL\_FILE.PUT\_LINE(file\_handle, json\_data);  END LOOP;  CLOSE cursor\_data;  UTL\_FILE.FCLOSE(file\_handle);  DBMS\_OUTPUT.PUT\_LINE('Data exported successfully.');  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  END; |

Листинг 4.1 – Процедура для экспорта данных из таблицы VISITS\_TABLE

Предварительно создаём объект Directory, где прописываем путь к папке для хранения данных.

4.2 Описание процедуры импорта

Процедуру импорта рассмотрим также на таблице VISITS\_TABLE. Процедура импорта представлена в листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE FUNCTION ImportDataFromJsonToVisitsTable(  p\_directory VARCHAR2,  p\_file\_name VARCHAR2  ) RETURN NUMBER IS  json\_data CLOB;  file\_handle UTL\_FILE.FILE\_TYPE;  total\_rows NUMBER := 0;  BEGIN  file\_handle := UTL\_FILE.FOPEN(p\_directory, p\_file\_name, 'R');  LOOP  BEGIN  UTL\_FILE.GET\_LINE(file\_handle, json\_data);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  EXIT;END;  INSERT INTO VISITS\_TABLE (  VISITS\_TABLE\_ID,  DATE\_OF\_VISITS,  VISITS\_NUMBER  )  SELECT  jt.VISITS\_TABLE\_ID,  TO\_DATE(jt.DATE\_OF\_VISITS, 'YYYY-MM-DD') AS DATE\_OF\_VISITS,  jt.VISITS\_NUMBER  FROM  JSON\_TABLE(json\_data, '$'  COLUMNS (  VISITS\_TABLE\_ID NUMBER PATH '$.VISITS\_TABLE\_ID',  DATE\_OF\_VISITS VARCHAR2(10) PATH '$.DATE\_OF\_VISITS',  VISITS\_NUMBER NUMBER PATH '$.VISITS\_NUMBER'  )  ) jt;  total\_rows := total\_rows + 1;  END LOOP;  UTL\_FILE.FCLOSE(file\_handle);  COMMIT;  RETURN total\_rows;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;RAISE; END; |

Листинг 4.2 – Процедура для импорта данных в таблицу VISITS\_TABLE

В результате импорта мы получим заполненную, ранее экспортированными, данными таблицу VISITS\_TABLE.

4.3 Вывод по разделу

В этом разделе была рассмотрена проблема заполнения таблиц данными (импорт) и экспорта данных из таблиц, а также предложено её решение с помощью хранимых процедур, реализующих экспорт и импорт данных в/из JSON файл.

# **5 Тестирование производительности**

5.1 Оптимизирование запросов

У нас уже созданы необходимые объекты базы данных, а также таблицы заполнены необходимым количеством данных – рисунок 5.1. Теперь необходимо протестировать, что всё работает, как нужно и как задумано.

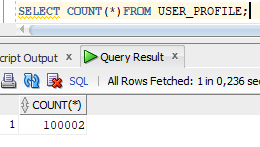


Рисунок 5.1 – Количество строк в таблицe USER\_PROFILE

Для начала проверим, какое количество времени занимает выполнение нашей функции для поиска пользователя по Email– Рисунок 5.2.

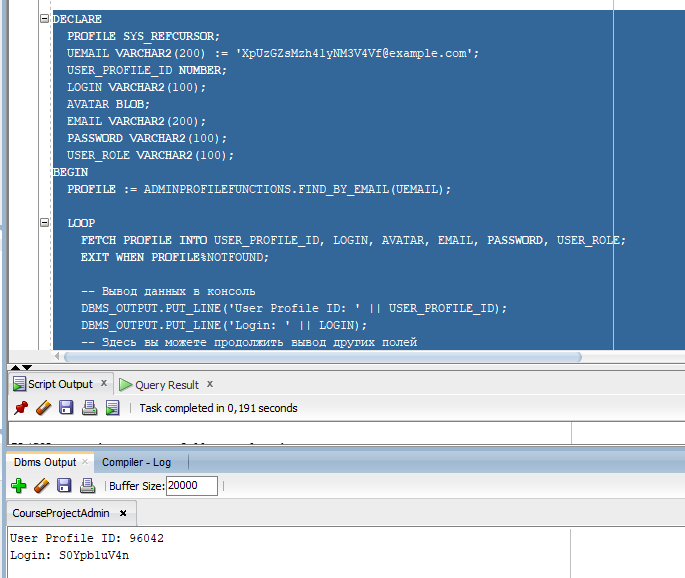


Рисунок 5.2 – Время поиска пользователя

Мы можем заметить, что время ожидания составило всего 0,191 секунды, что является приемлемым результатом. Однако в реальности, в базах данных крупных веб-сайтов хранятся миллионы записей пользователей, и поиск по ним может занимать значительное время. В данном случае мы просто смоделировали такую ситуацию. Теперь мы выполним простой запрос SELECT для поиска всех пользователей, как показано на рисунке 5.3.

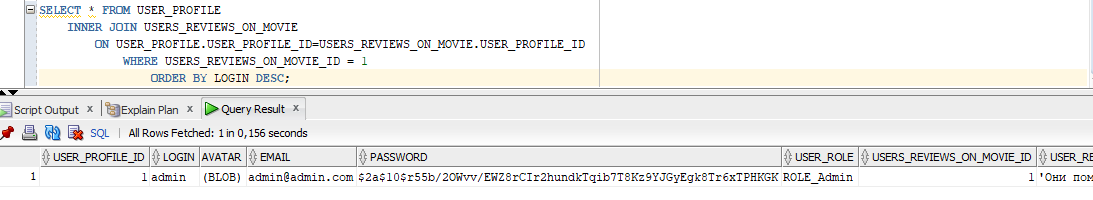


Рисунок 5.3 – Результат SELECT-запроса

Затем мы рассмотрим план выполнения этого запроса, как показано на рисунке 5.4.

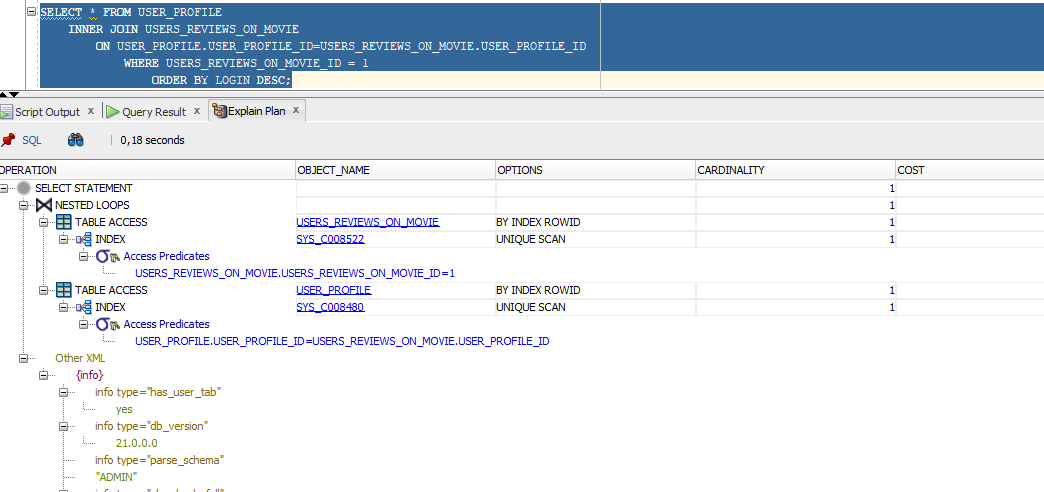


Рисунок 5.4 – План запроса

Теперь попробуем оптимизировать данный запрос с помощью индекса – Листинг 5.1.

CREATE INDEX IDX\_REVIEWS\_USER\_PROFILE\_ID ON USERS\_REVIEWS\_ON\_MOVIE (USER\_PROFILE\_ID);

Листинг 5.1 – Создание индекса

Создание такого индекса обуславливаем тем, что этот индекс поможет ускорить выполнение запроса, так как он будет использоваться для объединения записей между таблицами USER\_PROFILE и USERS\_REVIEWS\_ON\_MOVIE по столбцу USER\_PROFILE\_ID. Индекс также может улучшить сортировку по столбцу LOGIN.

Теперь проверим результат – рисунок 5.5.

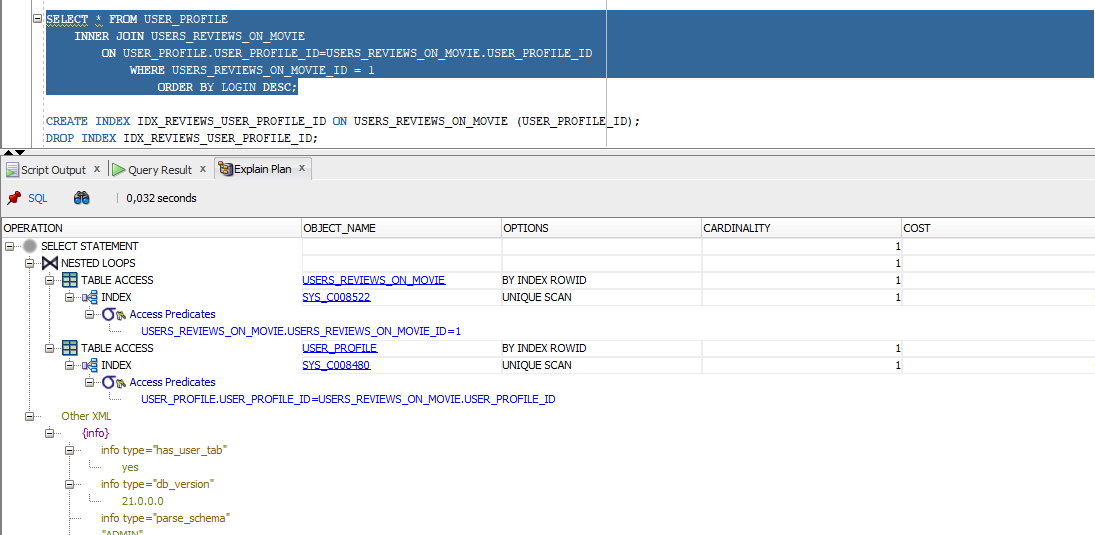


Рисунок 5.5 – Ситуация после создания индекса

После создания индекса время выполнения запроса уменьшилось почти в 6 раз (с 0,18 до 0,032), что свидетельствует о том, что индекс является эффективным.

5.2 Проверка выполнения хранимых процедур

Теперь протестируем основные хранимые процедуры таблицы PROFILE: ADD\_USER, FIND\_BY\_EMAIL, FIND\_BY\_ID, DELETE\_USER\_BY\_ID. При помощи процедуры ADD\_USER добавим нового пользователя – рисунок 5.6.

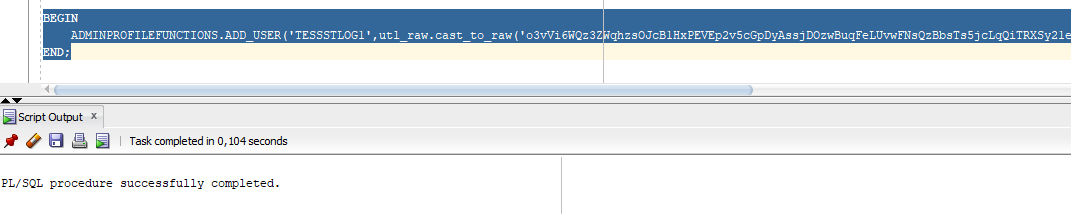


Рисунок 5.6 – Добавление нового пользователя

Далее найдем пользователя по Email. Рисунок 5.7.

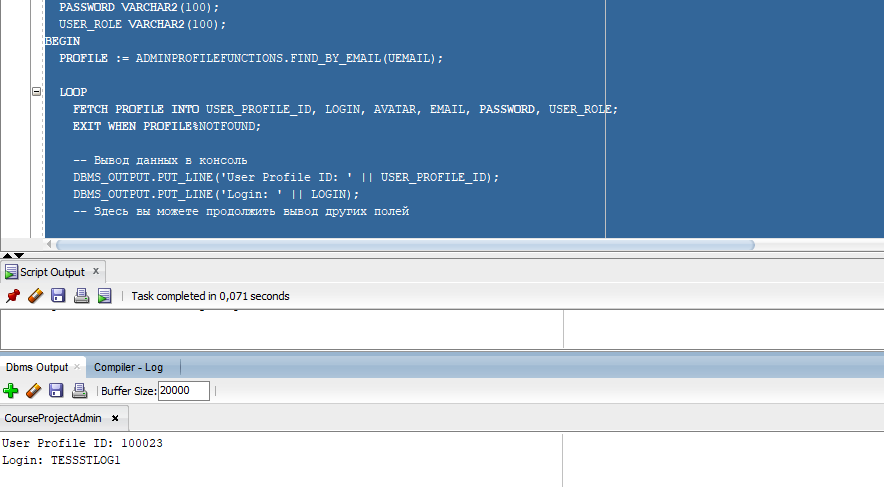


Рисунок 5.7 – Поиск пользователя по email

Найдем пользователя по id. Рисунок 5.8

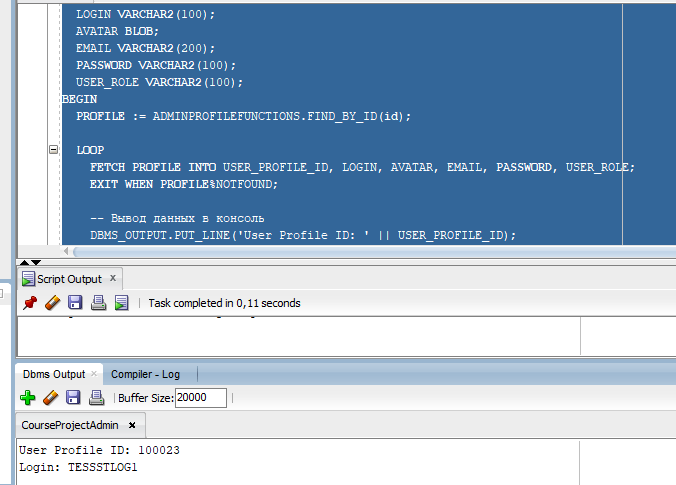


Рисунок 5.8 – Поиск пользователя по ID

Далее удалим пользователя. Рисунок 5.8.

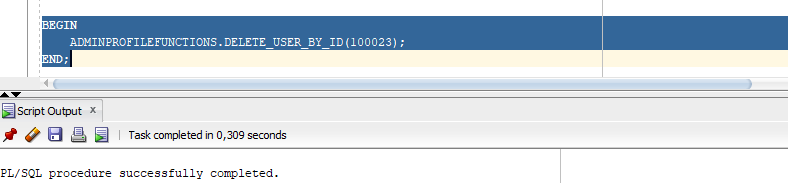


Рисунок 5.8 – Удаление пользователя по ID.

Далее проверим при помощи select запроса удалился ли он. Рисунок 5.9 соответственно.

|  |
| --- |
|  |

Рисунок 5.9 – Попытка доступа к удаленному пользователю

После select запроса ни одной записи не было возвращено, что свидетельствует о корректном выполнении процедуры DELETE\_USER\_BY\_ID.

5.2 Вывод по разделу

В данном разделе была протестирована таблица USER\_PROFILE на вставку и работу с большим количеством данных, создан индекс оптимизации, а также протестированы хранимые процедуры, использующие данную таблицу.

# **6 Описание и применение технологии в БД**

6.1 Описание взаимодействия с технологией

Oracle Analytics – это дополнительное приложение для визуализации данных, которое позволяет анализировать данные и строить графики и диаграммы без необходимости написания запросов. Оно предоставляет графический интерфейс, где пользователи могут выбирать нужные им данные для визуализации, используя функционал перетаскивания элементов.

Данное приложение имеет широкий спектр применения и может использоваться для анализа различных аспектов предприятия. Пользователи могут создавать собственные проекты и настраивать наборы данных - таблицы и столбцы, которые будут использоваться для визуализации. Приложение поддерживает русский язык, а также предлагает тестовый проект для практики.

После создания проекта в Oracle Analytics необходимо подключиться к базе данных, чтобы получить доступ к нужным данным. Пользователи могут использовать окно подключения для этой цели. Затем следует создать набор данных, определяя таблицы и столбцы, которые будут использоваться для визуализации. Также приложение предоставляет удобный интерфейс для работы с данными и позволяет строить различные графики и диаграммы. Это помогает пользователям анализировать данные и изучать различные аспекты своего предприятия.

После создания проекта необходимо создать набор данных – те таблицы и столбцы, по которые мы хотим визуализировать. Для этого нам необходимо выполнить подключение к своей базе данных через следующее окно – рисунок 6.1.

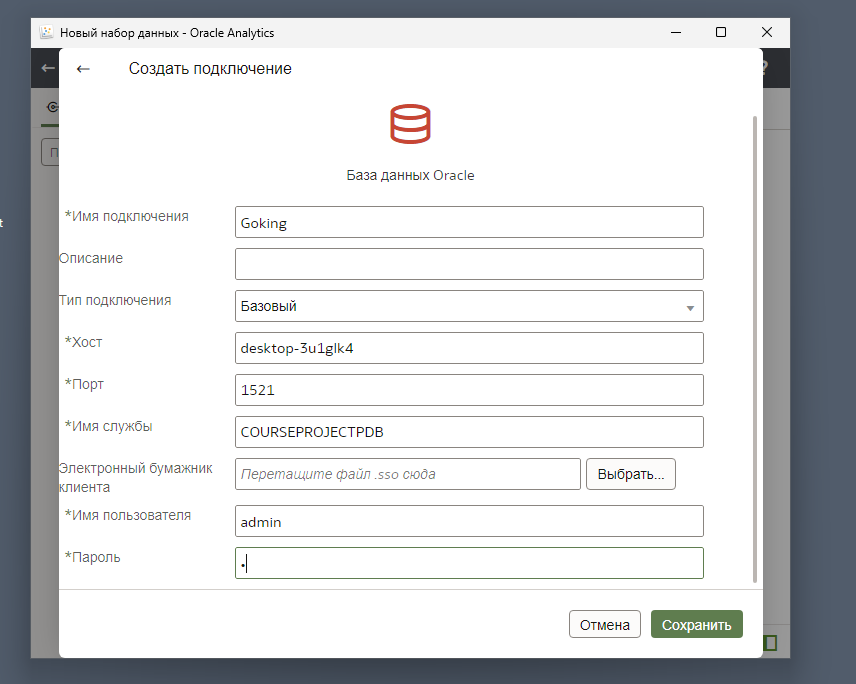


Рисунок 6.1 – Окно подключения

После подключения в основном окне выбираем исследование данных (рисунок 6.2).

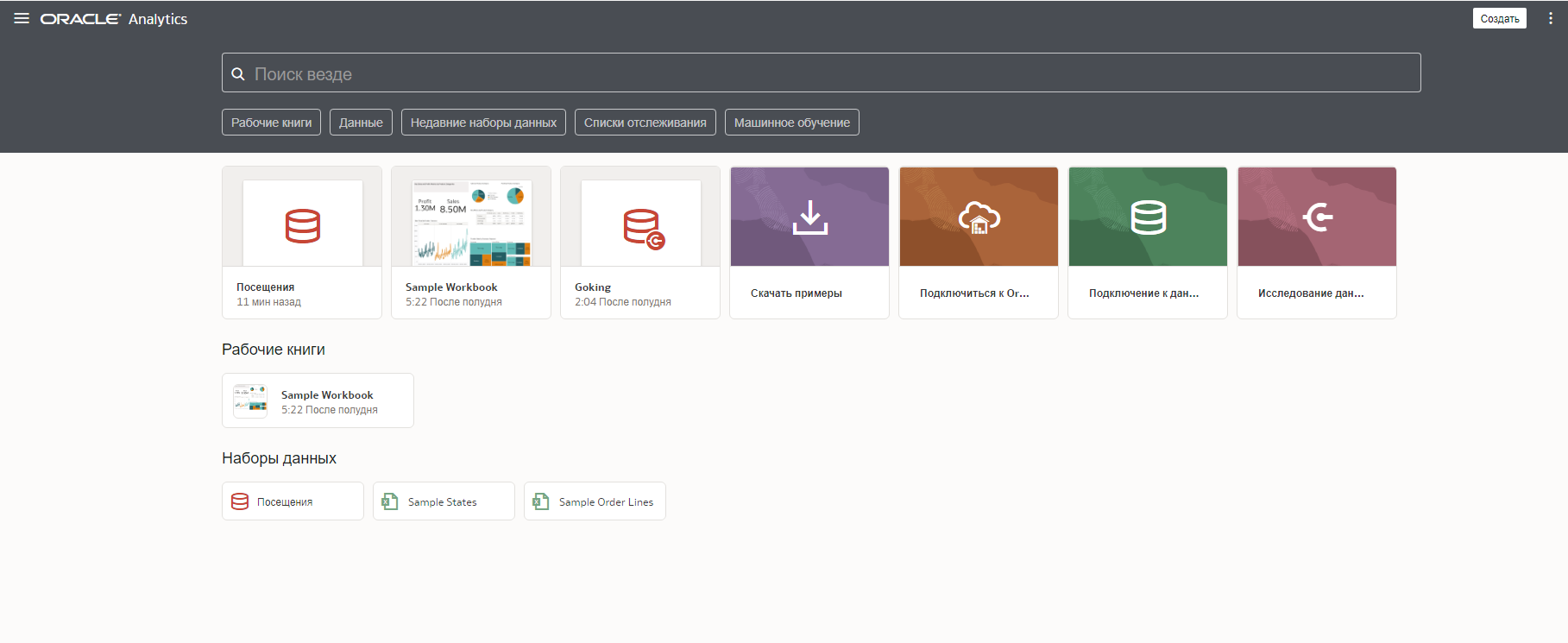


Рисунок 6.2 – Основное окно Oracle Analytics

После нажатия на блок исследования данных, откроется окно, которое предложит выбрать данные для анализа. Добавим новые данные, путем нажатия на кнопку создать набор данных, далее выбираем подключение, которое мы создавали в начале главы (рисунок 6.3).

|  |  |
| --- | --- |
|  |  |

Рисунок 6.3 – Создание набора данных для исследования

После выбираем данные для анализа. В моем случае будет анализироваться таблица VISITS\_TABLE (рисунок 6.4).

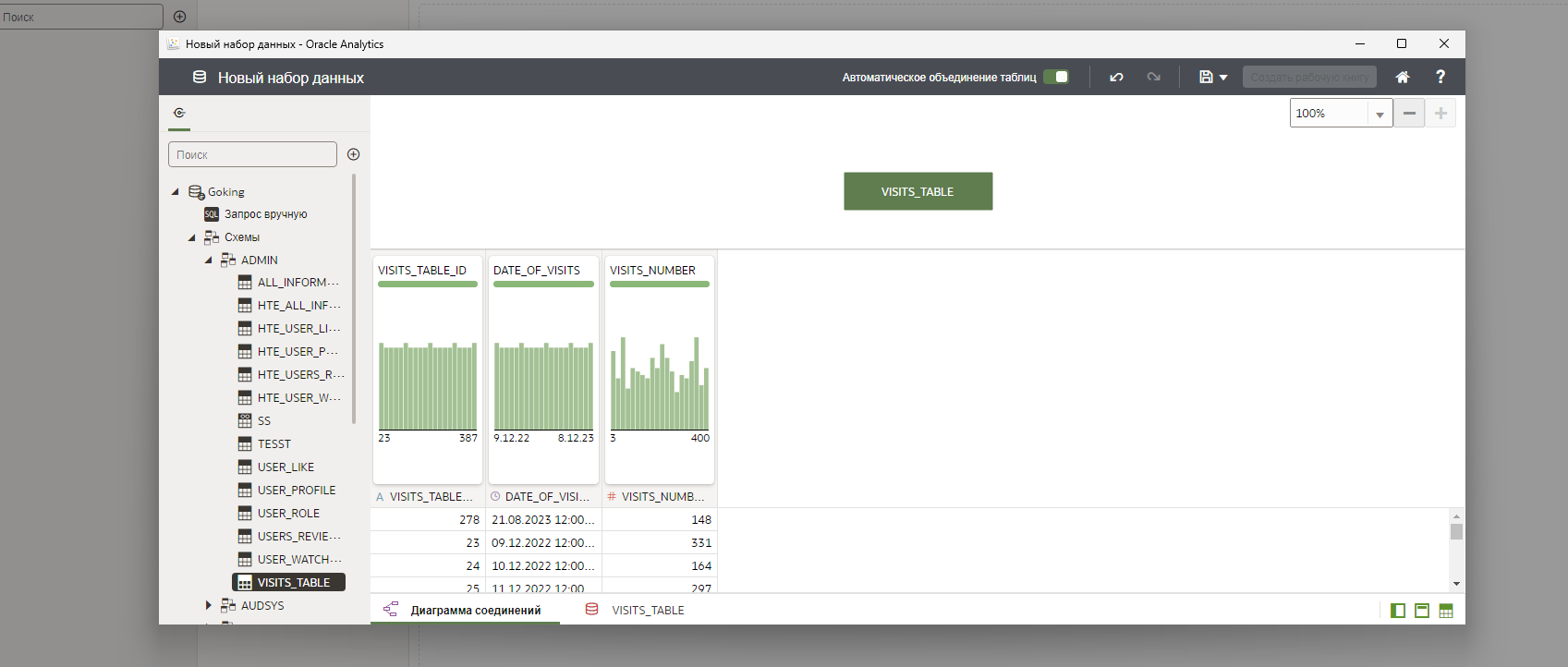


Рисунок 6.4 – Добавление таблицы в набор данных

Данная таблица содержит количество посещений веб-сайта пользователями за день. Внутри таблицы располагается 365 записей, что дает возможность осуществить приемлемый анализ данных, на основе которого будут приниматься бизнес решения.

Теперь на основе созданного набора данных создадим к примеру столбчатую диаграмму посещений за месяц, для этого перетащим столбцы DATE\_OF\_VISITS, а также столбец VISITS\_NUMBER в рабочую область. Выберем столбчатую диаграмму, а также нажмем по DATE\_OF\_VISITS в категория (Ось х), наведемся на показать по и выберем месяц. Результат действий представлен на рисунке 6.5.

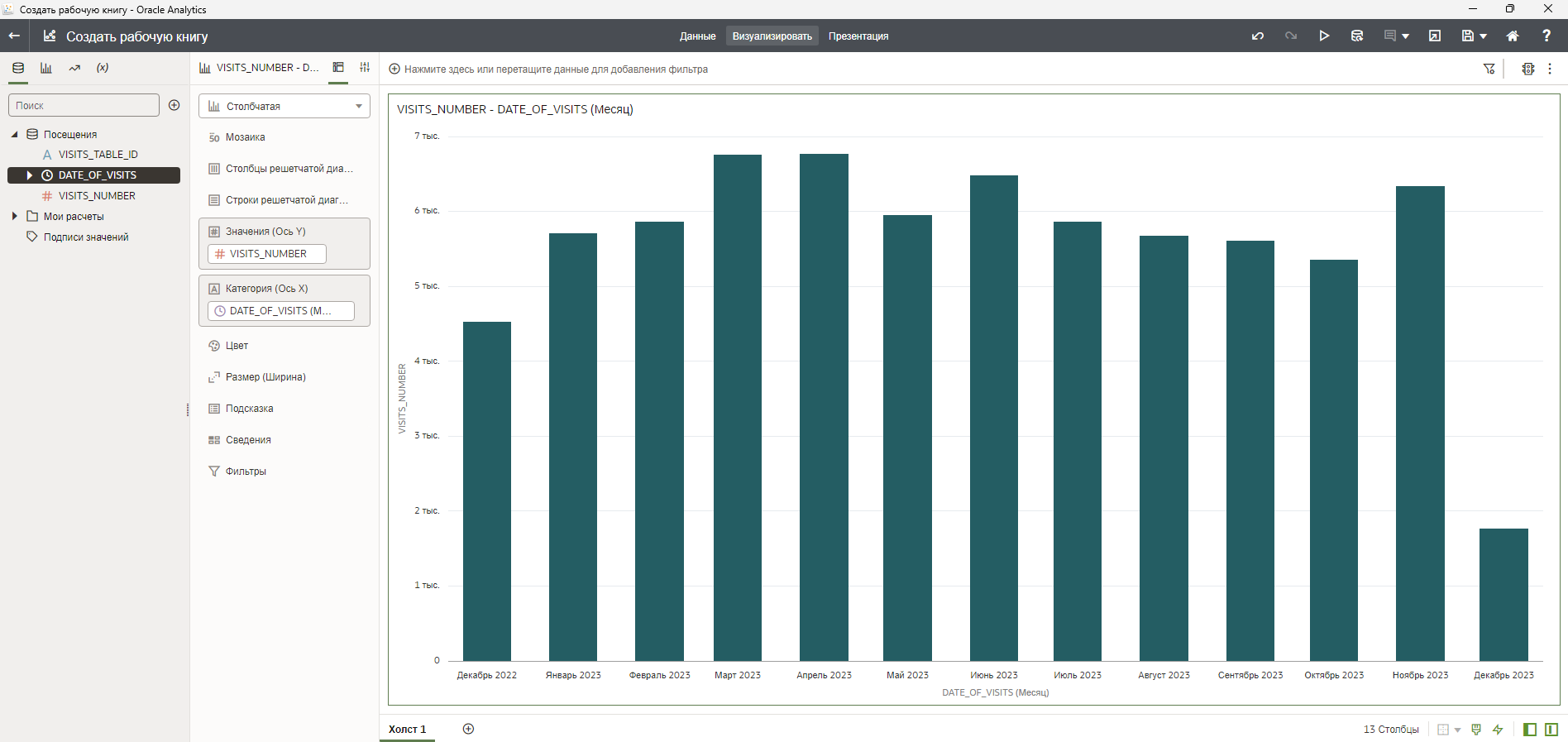


Рисунок 6.5 – Добавление таблицы в набор данных

Теперь посмотрим в процентном соотношении, для этого изменим тип отображения со столбчатой диаграммы на круговую. Результат представлен на рисунке 6.6.

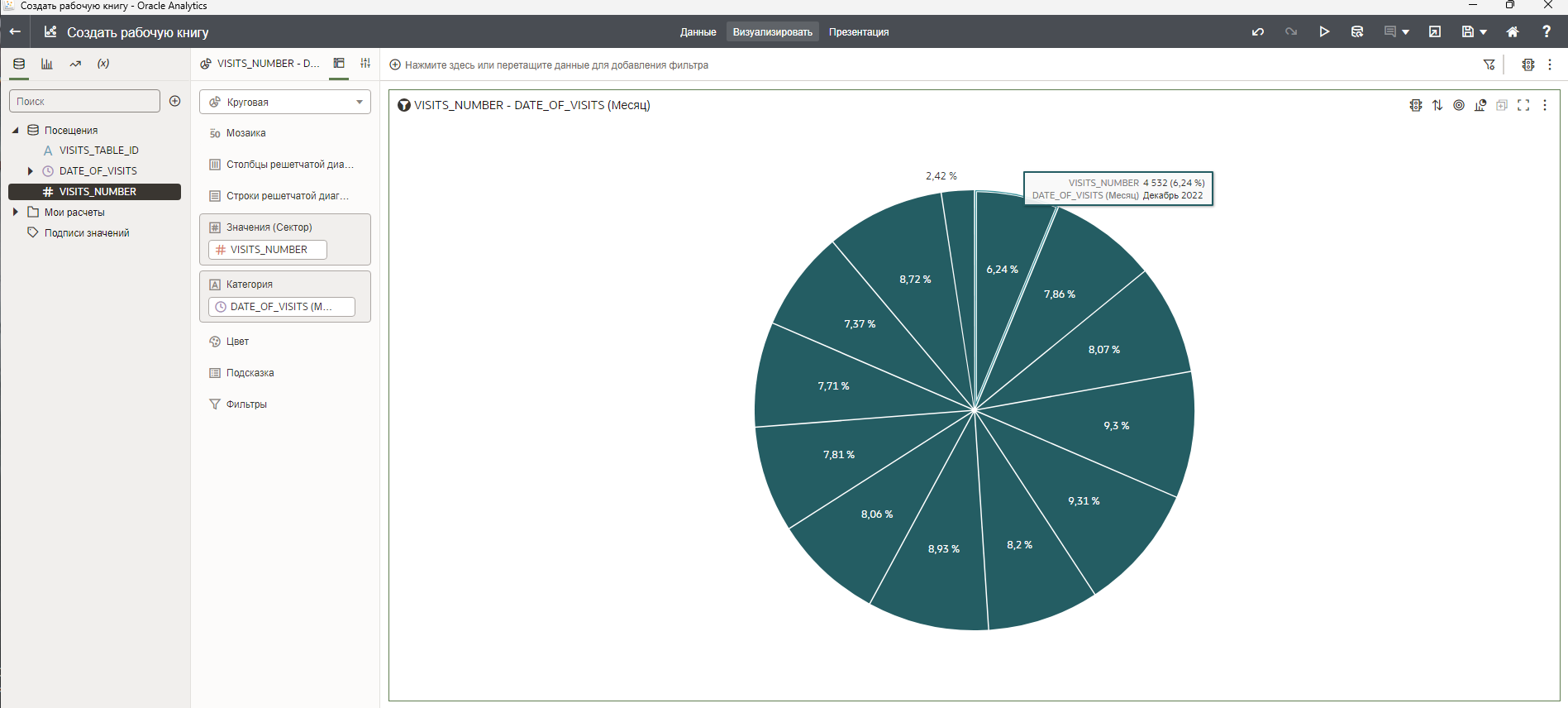


Рисунок 6.6 – Добавление таблицы в набор данных

В результате анализа данной таблицы были выявлены месяцы, во время которых посещение максимально. Это можно использовать в различных целях, например уменьшить количество рекламы (количество пользователей компенсирует уменьшение рекламы). В то же время пользователи, которым не будет надоедать навязчивая реклама, будут охотнее делиться ссылкой с друзьями, что может увеличить потенциальное количество новых пользователей.

6.2 Вывод по разделу

Oracle Analytics представляет собой мощный инструмент для анализа данных и создания визуализаций. С его помощью можно проводить глубокий и всесторонний анализ данных, выявлять тренды, паттерны и важные инсайды, которые могут помочь в принятии стратегических решений.

С помощью интуитивно понятного графического интерфейса приложение упрощает процесс работы с данными, позволяя пользователям без специализированных навыков программирования или аналитики создавать сложные запросы и визуализации. Это делает его доступным и полезным инструментом для компаний любого масштаба.

Благодаря возможностям Oracle Analytics, компании могут получить глубокое понимание своих данных, выявить тренды и паттерны, анализировать производительность и эффективность бизнес-процессов, и принимать взвешенные решения на основе фактов. Это может привести к оптимизации операций, снижению издержек, улучшению клиентского опыта и, в конечном итоге, к повышению конкурентоспособности компании.

Таким образом, Oracle Analytics является важным инструментом для бизнес-анализа, который может помочь компаниям эффективно использовать свои данные и принимать обоснованные стратегические решения.

7 Краткое описание приложения для демонстрации

В приложении есть две роли: пользователь и админ. Для пользователя были разработаны следующие страницы – авторизации и регистрации, всего каталога, понравившихся фильмов, фильмов, которые добавлены в категорию «Посмотреть позже», а также представление фильма. Страницы авторизации и регистрации представлены на рисунке 7.1.

|  |  |
| --- | --- |
|  |  |

Рисунок 7.1 – Страницы авторизации и регистрации

Для этих страниц разработана валидация: проверка на уникальность логина и email, выполненные внутри бд и если они не уникальны, то генерируются соответствующие исключения и на экран выводится обработанное сообщение об ошибке. На стороне клиента через JS выполнена валидация введенных значений на основе regex (к примеру пароль должен содержать минимум 1 заглавную букву, 1 цифру и содержать 8 символов).

Страница главной пользовательской страницы представлено на рисунке 7.2.

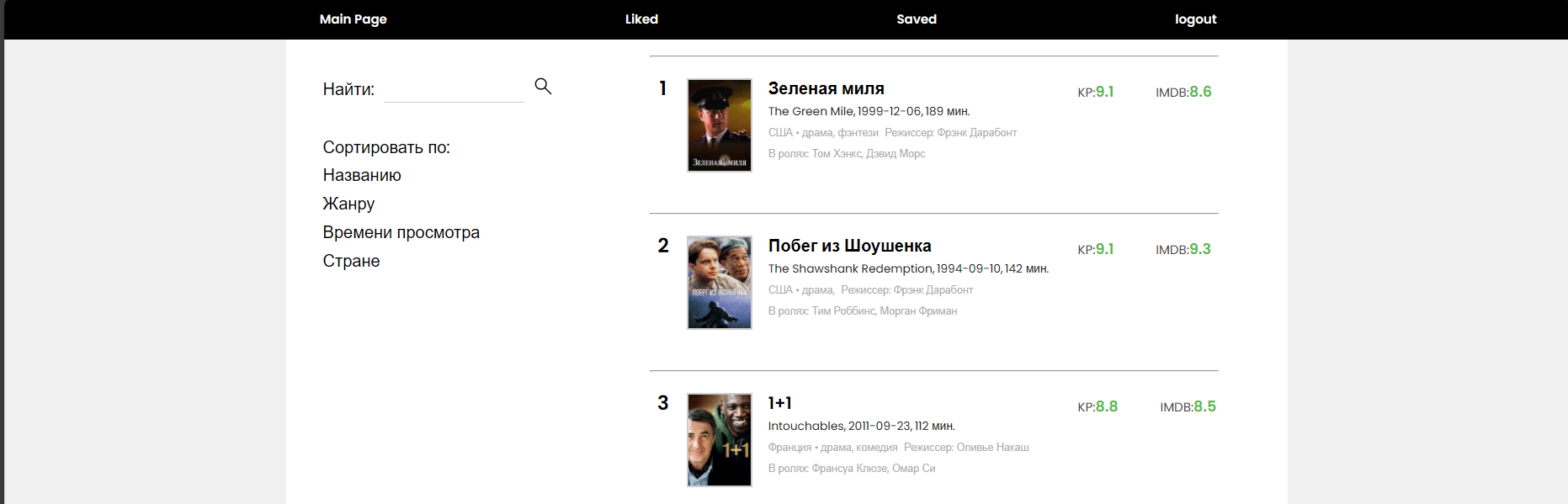


Рисунок 7.2 – главная страница пользователя

Пользователь имеет возможность сортировать фильмы по заданным критериям.

Окна «Liked» и «Saved»выглядят также, различие лишь в логике, написанной на JS.

Страница фильма представлена на рисунке 7.3.

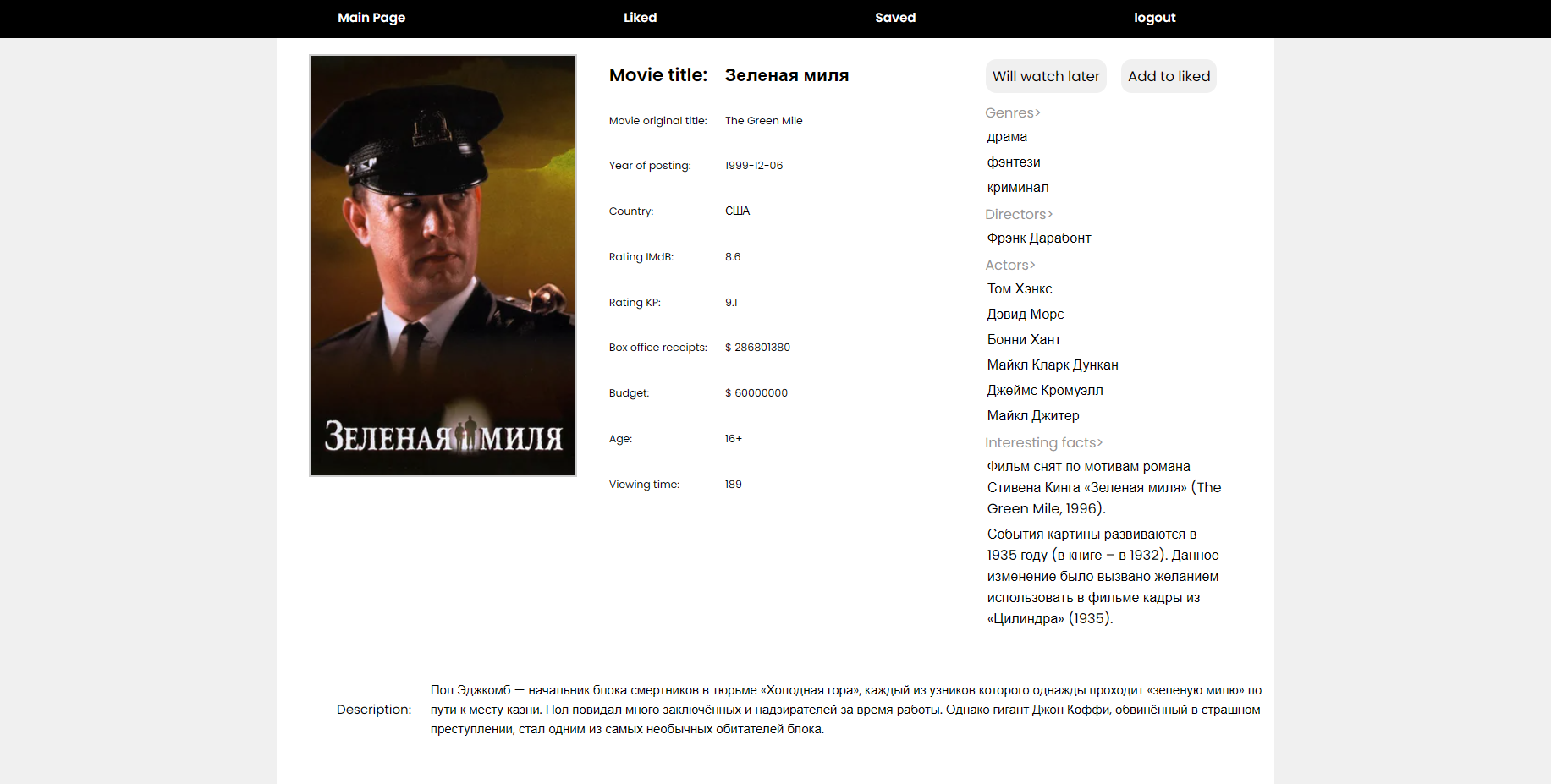


Рисунок 7.3 – Страница фильма

Есть возможность оставить отзыв под фильмом.

Теперь рассмотрим страницы администратора. Он содержит расширенные, по сравнение с пользователем окна, например, есть возможность добавлять, редактировать и удалять неактивных пользователей. Для этих целей разработана страница «Все пользователи», которая имеет попап меню для добавления и изменения пользовательских данных, есть валидация, идентичная валидации регистрации и авторизации, вид представлен на рисунке 7.4.

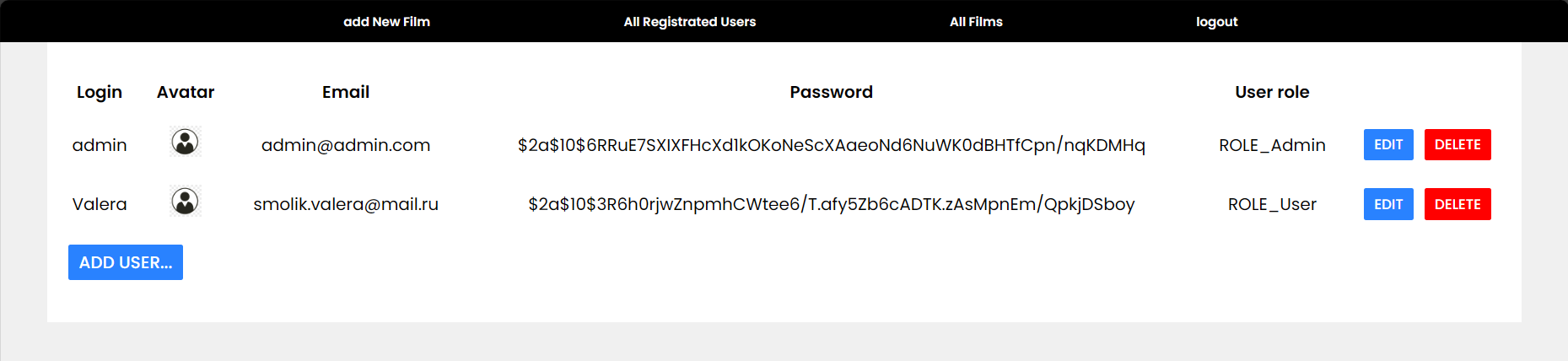


Рисунок 7.4 – Страница все пользователи

Вид страницы при добавлении и изменении пользователя представлен на рисунке 7.5.

|  |  |
| --- | --- |
|  |  |

Рисунок 7.5 – Страница все пользователи при изменении/добавлении пользователя

Также была разработана страница добавление нового фильма. Вид страницы представлен на рисунке 7.6.

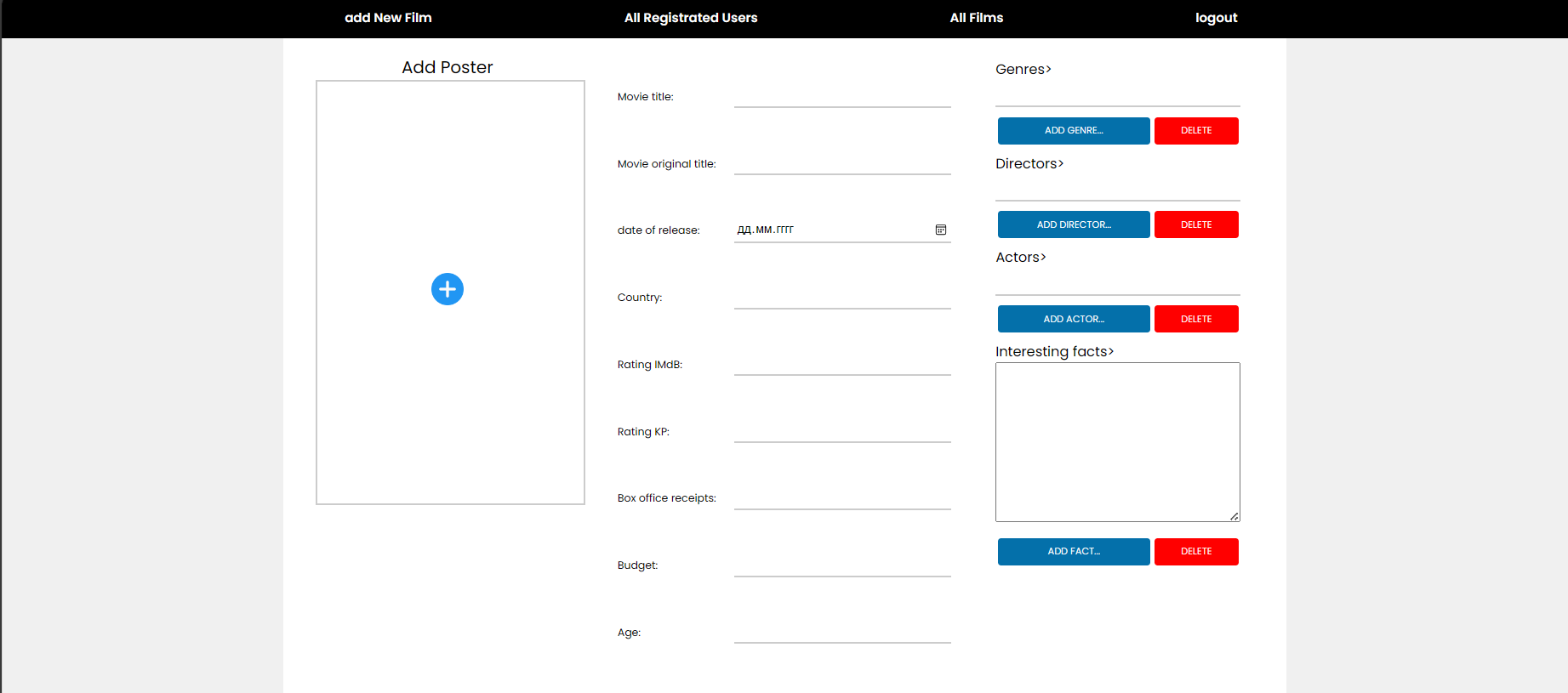


Рисунок 7.6 – Страница добавления нового фильма

Для каждого поле присутствует валидация, которая позволяет избежать ввода некорректных значений. Выполнена она на стороне клиента через JS. Для изменения данных о фильме используется эта же страница, предварительно заполненная данными.

Главная страница, содержащая каталог, имеет расширенный функционал, по сравнению с пользовательской версией рисунок 7.7.

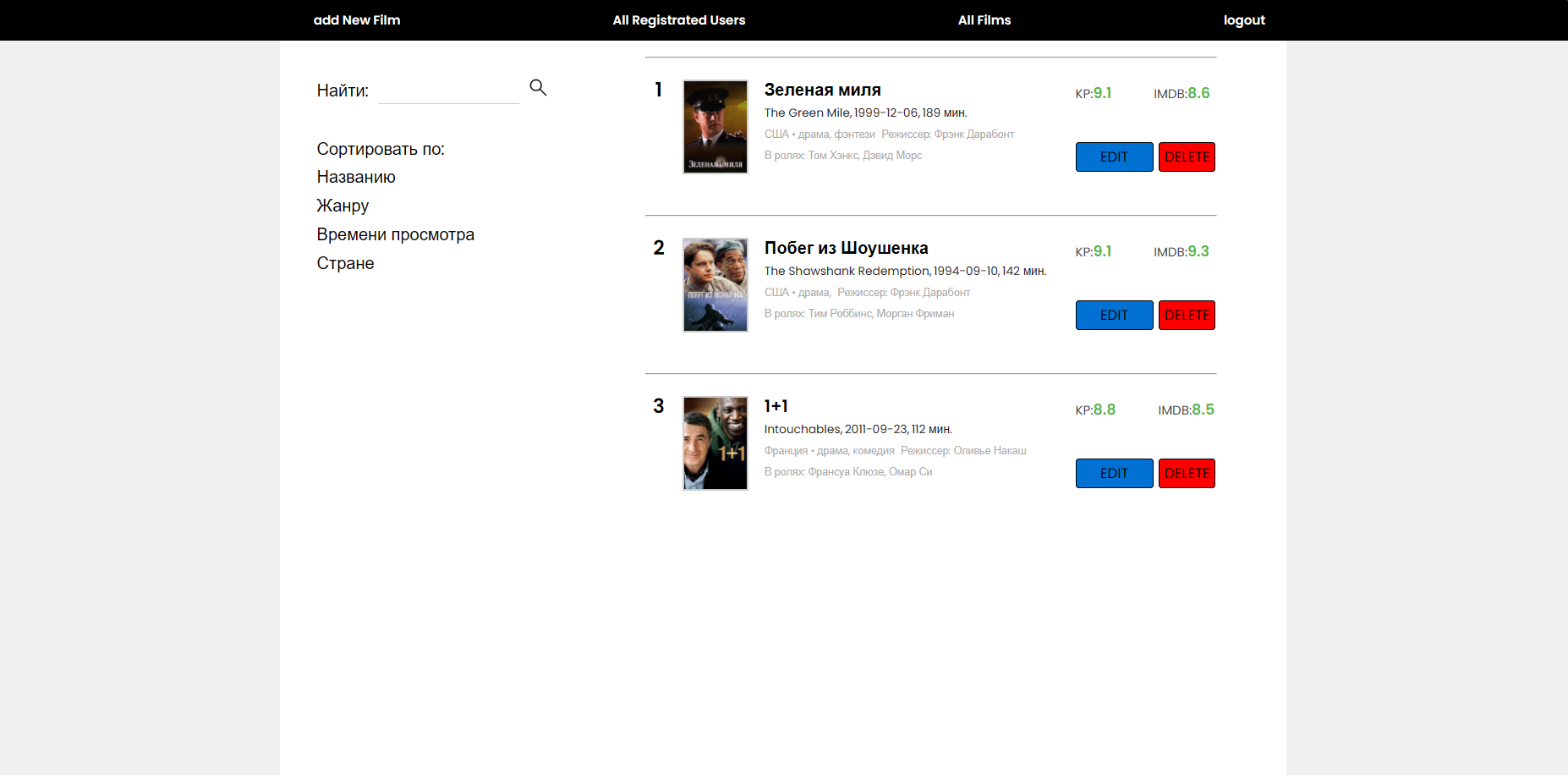


Рисунок 7.7 – Страница каталога

В завершении, на странице фильма есть возможность удалить комментарий пользователя или другого админа.

# **8 Руководство пользователя**

При входе на веб-сайт пользователь попадает на страницу регистрации, если у него уже есть аккаунт, то он может перейти на страницу авторизации (страницы авторизации и регистрации представлены на рисунке 7.1).

После авторизации пользователь попадает на страницу каталога со всеми фильмами, добавленными в базу данных, или на страницу со всеми пользователями, если зашел админ (рисунки 7.2 и 7.4 соответственно).

Далее пользователь может воспользоваться сортировками сайта или найти фильм по названию. После того как пользователь нашел интересующий его фильм, он может перейти на полную информацию о данном фильме рисунок 7.3, путем нажатия на блок с краткой информацией о фильме. При желании пользователь может оценить фильм, путем добавления его в «Понравившееся», также пользователь может сохранить фильм в категорию «Посмотреть позже» или оставить комментарий. Выполнить это можно нажимая на соответствующий кнопки на странице фильма.

Также есть возможность просмотра всех фильмов, которые были добавлены в категории «Понравившееся» и «Посмотреть позже», используя меню навигации хедера.

При авторизации администратора и попадании на страницу пользователей, он имеет возможность изменить или удалить неактивный пользовательский аккаунт.

Используя меню хедера, администратор может перейти на страницу добавления фильма рисунок 7.6.

При переходе на главную страницу, администратор может посмотреть, как выглядит новый добавленный фильм в каталоге рисунок 7.7, если какие-то данные нужно изменить, то можно воспользоваться кнопкой «EDIT», при удалении – кнопкой «Delete».

При переходе на страницу с полной информацией о фильме, администратор может добавить или удалить комментарий.

# **Заключение**

В ходе выполнения курсового проекта была спроектирована и реализована инфраструктура базы данных для веб-сайта, который содержит информацию о пользователях и фильмах или сериалах, с использованием СУБД Oracle 12c. Были определены и разработаны необходимые объекты базы данных, такие как табличные пространства, таблицы, ограничения целостности, пользователи, триггеры, хранимые процедуры. Также был проведен импорт данных из JSON файлов для заполнения таблиц.

Важным аспектом проекта было применение технологии Oracle Advanced Analytics для анализа данных в базе данных веб-сайта. Были изучены и применены полезные функции, предоставляемые данной технологией, для обеспечения анализа количества посещений за определенный промежуток времени. Это позволит получить ценные показатели, которые показывают активность пользователей в тот или иной день и в перспективе улучшить принятие управленческих решений.

В рамках проекта было разработан веб-сайт, использующий Spring Security на основе JWT, который позволяет запретить неавторизированный доступ пользователя к веб-ресурсам. Он демонстрирует функционал и возможности, предоставляемые базой данных курсового проекта. Реализованы функционал управления фильмами и пользователями.

Также внимание было уделено производительности базы данных. Проведено тестирование на таблице, содержащей более 100 000 строк, и проведён анализ необходимых изменений в структуре базы данных для обеспечения оптимальной производительности.

В результате успешной реализации курсового проекта была создана база данных для веб-сайта фильмов с широким функционалом, позволяющим эффективно управлять фильмами, пользователями, комментариями, анализировать активность. Проект демонстрирует преимущества использования СУБД Oracle 12c и технологии Oracle Advanced Analytics для создания мощной и гибкой системы управления веб-сайтом.

# **Список используемых источников**

1. Нистюк О. А. Курс лекций по базам данных, [Электронный ресурс]. – Режим доступа: https://diskstation.belstu.by:5001/. – Дата доступа: 01.09.2023.
2. Гончар Е. А. Курс лекций по Java Spring, [Электронный ресурс]. – Режим доступа: https://diskstation.belstu.by:5001/. – Дата доступа: 01.09.2023.
3. Работа с файлами в Oracle, [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/cd/F49540\_01/DOC/server.815/a68001/utl\_file.htm. – Дата доступа: 01.05.2023.
4. Пользовательское руководство по Oracle Analytics. Создание подключения, [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/middleware/bi/analytics-desktop/bidvd/connect-data-sources.html – Дата доступа: 02.12.2023.

5 Пользовательское руководство по Java Spring. Создание подключения, [Электронный ресурс]. – Режим доступа: https://docs.spring.io/spring-framework/reference/index.html – Дата доступа 01.11.2023.

Приложение A Создание таблиц

|  |
| --- |
| CREATE TABLE USER\_PROFILE(  USER\_PROFILE\_ID number GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  LOGIN varchar(100) NOT NULL,  AVATAR BLOB NOT NULL,  EMAIL varchar(200) NOT NULL,  PASSWORD varchar(100) NOT NULL,  USER\_ROLE varchar(100) NOT NULL,  CONSTRAINT CONSTRAIN\_USER\_ROLE CHECK(USER\_ROLE IN('ROLE\_User','ROLE\_Admin'))  )TABLESPACE CP\_TS;  DROP TABLE USER\_PROFILE PURGE;  ------------------------------------------------  CREATE TABLE USERS\_REVIEWS\_ON\_MOVIE(  USERS\_REVIEWS\_ON\_MOVIE\_ID INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  USER\_REVIEW\_TEXT VARCHAR2(2000) NOT NULL,  DATE\_OF\_REVIEW DATE NOT NULL,  USER\_PROFILE\_ID NUMBER NOT NULL,  ALL\_INFORMATION\_ABOUT\_FILM\_ID NUMBER NOT NULL,  CONSTRAINT USER\_PROFILE\_ID\_CONSTRAINT FOREIGN KEY (USER\_PROFILE\_ID) REFERENCES USER\_PROFILE(USER\_PROFILE\_ID),  CONSTRAINT ALL\_INFORMATION\_ABOUT\_FILM\_ID\_CONSTRAINT FOREIGN KEY (ALL\_INFORMATION\_ABOUT\_FILM\_ID) REFERENCES ALL\_INFORMATION\_ABOUT\_FILM(ALL\_INFORMATION\_ABOUT\_FILM\_ID)  )TABLESPACE CP\_TS;  DROP TABLE USERS\_REVIEWS\_ON\_MOVIE PURGE;  ------------------------------------------------  CREATE TABLE VISITS\_TABLE(  VISITS\_TABLE\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  DATE\_OF\_VISITS DATE NOT NULL,  VISITS\_NUMBER NUMBER NOT NULL  )TABLESPACE CP\_TS;  DROP TABLE VISITS\_TABLE PURGE;  ------------------------------------------------  CREATE TABLE ALL\_INFORMATION\_ABOUT\_FILM(  ALL\_INFORMATION\_ABOUT\_FILM\_ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  TITLE VARCHAR(100) NOT NULL,  ORIGINAL\_TITLE VARCHAR(100) NOT NULL,  POSTER BLOB NOT NULL,  YEAR\_OF\_POSTING VARCHAR(20) NOT NULL,  COUNTRY VARCHAR(50) NOT NULL,  DESCRIPTION VARCHAR(1000) NOT NULL,  RATING\_IMDb FLOAT NOT NULL,  RATING\_KP FLOAT NOT NULL,  BOX\_OFFICE\_RECEIPTS FLOAT NOT NULL,  BUDGET FLOAT NOT NULL,  AGE NUMBER NOT NULL,  VIEWING\_TIME VARCHAR(30) NOT NULL,  GENRES GENRE\_TYPE\_INSTANCE,  DIRECTORS DIRECTOR\_TYPE\_INSTANCE,  ACTORS ACTOR\_TYPE\_INSTANCE,  INTERESTING\_FACT INTERESTING\_FACT\_TYPE\_INSTANCE  )NESTED TABLE GENRES STORE AS GENRES\_NT  NESTED TABLE DIRECTORS STORE AS DIRECTORS\_NT  NESTED TABLE ACTORS STORE AS ACTORS\_NT  NESTED TABLE INTERESTING\_FACT STORE AS INTERESTING\_FACT\_NT (TABLESPACE CP\_TS);  DROP TABLE ALL\_INFORMATION\_ABOUT\_FILM PURGE;  ------------------------------------------------  CREATE TABLE USER\_WATCH\_LATER(  USER\_WATCH\_LATER\_ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  ALL\_INFORMATION\_ABOUT\_FILM NUMBER NOT NULL,  USER\_PROFILE\_ID NUMBER NOT NULL,  CONSTRAINT ALL\_INFORMATION\_ABOUT\_FILM\_USER\_WATCH\_LATER\_CONSTRAINT FOREIGN KEY (ALL\_INFORMATION\_ABOUT\_FILM) REFERENCES ALL\_INFORMATION\_ABOUT\_FILM(ALL\_INFORMATION\_ABOUT\_FILM\_ID),  CONSTRAINT USER\_PROFILE\_ID\_USER\_WATCH\_LATER\_CONSTRAINT FOREIGN KEY(USER\_PROFILE\_ID)REFERENCES USER\_PROFILE(USER\_PROFILE\_ID)  )TABLESPACE CP\_TS;  DROP TABLE USER\_WATCH\_LATER PURGE;  ------------------------------------------------  CREATE TABLE USER\_LIKE(  USER\_LIKE\_ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  ALL\_INFORMATION\_ABOUT\_FILM NUMBER NOT NULL,  USER\_PROFILE\_ID NUMBER NOT NULL,  CONSTRAINT ALL\_INFORMATION\_ABOUT\_FILM\_USER\_LIKE\_CONSTRAINT FOREIGN KEY (ALL\_INFORMATION\_ABOUT\_FILM) REFERENCES ALL\_INFORMATION\_ABOUT\_FILM(ALL\_INFORMATION\_ABOUT\_FILM\_ID),  CONSTRAINT USER\_PROFILE\_ID\_USER\_LIKE\_CONSTRAINT FOREIGN KEY(USER\_PROFILE\_ID)REFERENCES USER\_PROFILE(USER\_PROFILE\_ID)  )TABLESPACE CP\_TS;  DROP TABLE USER\_LIKE PURGE;  ------------------------------------------------ |

Приложение Б Создание вложенных таблиц таблиц

|  |
| --- |
| --TYPE FOR GENRE COLUMN OF ALL\_INFORMATION\_ABOUT\_FILM  --MAP MEMBER используется для определения методов, которые выполняют операции, не изменяющие состояние объекта.  --MEMBER используется для определения методов, которые могут изменять состояние объекта.  --SELF IN OUT NOCOPY(Режим передачи параметров NOCOPY требует, чтобы ядро PL/SQL передавало аргументы IN OUT по ссылке, а не по значению.)  CREATE SEQUENCE SEQ\_GENRE  START WITH 1  INCREMENT BY 1  NOCACHE  NOCYCLE;  --ПРОТОТИП  CREATE TYPE GENRE\_TYPE\_CONSTRUCTOR AS OBJECT(  GENRE\_ID INT, --атрибут  GENRE\_NAME VARCHAR2(50), --атрибут  CONSTRUCTOR FUNCTION GENRE\_TYPE\_CONSTRUCTOR(P\_GENRE\_NAME VARCHAR2)RETURN SELF AS RESULT, --конструктор для создания объекта  MAP MEMBER FUNCTION GET\_GENRE\_ID RETURN INT, --функция получения ID  MEMBER PROCEDURE DISPLAY\_GENRE(SELF IN OUT NOCOPY GENRE\_TYPE\_CONSTRUCTOR) --процедура печати  );  --РЕАЛИЗАЦИЯ  CREATE TYPE BODY GENRE\_TYPE\_CONSTRUCTOR AS  CONSTRUCTOR FUNCTION GENRE\_TYPE\_CONSTRUCTOR(P\_GENRE\_NAME VARCHAR2)RETURN SELF AS RESULT IS  BEGIN  SELF.GENRE\_ID := SEQ\_GENRE.NEXTVAL;  SELF.GENRE\_NAME := P\_GENRE\_NAME;  RETURN;  END;  MAP MEMBER FUNCTION GET\_GENRE\_ID RETURN INT IS  BEGIN  RETURN GENRE\_ID;  END;  MEMBER PROCEDURE DISPLAY\_GENRE(SELF IN OUT NOCOPY GENRE\_TYPE\_CONSTRUCTOR)IS  BEGIN  DBMS\_OUTPUT.put\_line(TO\_CHAR(GENRE\_ID) || ' - ' || 'GENRE: ' || ' - ' || GENRE\_NAME);  END;  END;  CREATE TYPE GENRE\_TYPE\_INSTANCE AS TABLE OF GENRE\_TYPE\_CONSTRUCTOR;  COMMIT;  DROP TYPE GENRE\_TYPE\_CONSTRUCTOR;  DROP TYPE GENRE\_TYPE\_INSTANCE;  DROP SEQUENCE SEQ\_GENRE;  COMMIT;  ------------------------------------------------  CREATE SEQUENCE SEQ\_DIRECTOR  START WITH 1  INCREMENT BY 1  NOCACHE  NOCYCLE;  CREATE TYPE DIRECTOR\_TYPE\_CONSTRUCTOR AS OBJECT(  DIRECTOR\_ID INT,  DIRECTOR\_NAME VARCHAR2(90),  CONSTRUCTOR FUNCTION DIRECTOR\_TYPE\_CONSTRUCTOR(P\_DIRECTOR\_NAME VARCHAR2)RETURN SELF AS RESULT,  MAP MEMBER FUNCTION GET\_DIRECTOR\_ID RETURN INT,  MEMBER PROCEDURE DISPLAY\_DIRECTOR\_NAME(SELF IN OUT NOCOPY DIRECTOR\_TYPE\_CONSTRUCTOR)  )  CREATE TYPE BODY DIRECTOR\_TYPE\_CONSTRUCTOR AS  CONSTRUCTOR FUNCTION DIRECTOR\_TYPE\_CONSTRUCTOR(P\_DIRECTOR\_NAME VARCHAR2)RETURN SELF AS RESULT IS  BEGIN  SELF.DIRECTOR\_ID := SEQ\_DIRECTOR.NEXTVAL;  SELF.DIRECTOR\_NAME := P\_DIRECTOR\_NAME;  RETURN;  END;  MAP MEMBER FUNCTION GET\_DIRECTOR\_ID RETURN INT IS  BEGIN  RETURN DIRECTOR\_ID;  END;  MEMBER PROCEDURE DISPLAY\_DIRECTOR\_NAME(SELF IN OUT NOCOPY DIRECTOR\_TYPE\_CONSTRUCTOR)IS  BEGIN  DBMS\_OUTPUT.put\_line(TO\_CHAR(DIRECTOR\_ID) || ' - ' || 'GENRE: ' || ' - ' || DIRECTOR\_NAME);  END;  END;  CREATE TYPE DIRECTOR\_TYPE\_INSTANCE AS TABLE OF DIRECTOR\_TYPE\_CONSTRUCTOR;  COMMIT;  DROP TYPE DIRECTOR\_TYPE\_CONSTRUCTOR;  DROP TYPE DIRECTOR\_TYPE\_INSTANCE;  DROP SEQUENCE SEQ\_DIRECTOR;  COMMIT;  ------------------------------------------------  CREATE SEQUENCE SEQ\_ACTOR  START WITH 1  INCREMENT BY 1  NOCACHE  NOCYCLE;  CREATE TYPE ACTOR\_TYPE\_CONSTRUCTOR AS OBJECT(  ACTOR\_ID INT,  ACTOR\_NAME VARCHAR2(100),  CONSTRUCTOR FUNCTION ACTOR\_TYPE\_CONSTRUCTOR(P\_ACTOR\_NAME VARCHAR2)RETURN SELF AS RESULT,  MAP MEMBER FUNCTION GET\_ACTOR\_ID RETURN INT,  MEMBER PROCEDURE DISPLAY\_ACTOR\_NAME(SELF IN OUT NOCOPY ACTOR\_TYPE\_CONSTRUCTOR)  )  CREATE TYPE BODY ACTOR\_TYPE\_CONSTRUCTOR AS  CONSTRUCTOR FUNCTION ACTOR\_TYPE\_CONSTRUCTOR(P\_ACTOR\_NAME VARCHAR2)RETURN SELF AS RESULT IS  BEGIN  SELF.ACTOR\_ID := SEQ\_ACTOR.NEXTVAL;  SELF.ACTOR\_NAME := P\_ACTOR\_NAME;  RETURN;  END;  MAP MEMBER FUNCTION GET\_ACTOR\_ID RETURN INT AS  BEGIN  RETURN ACTOR\_ID;  END;  MEMBER PROCEDURE DISPLAY\_ACTOR\_NAME(SELF IN OUT NOCOPY ACTOR\_TYPE\_CONSTRUCTOR)IS  BEGIN  DBMS\_OUTPUT.put\_line(TO\_CHAR(ACTOR\_ID) || ' - ' || 'GENRE: ' || ' - ' || ACTOR\_NAME);  END;  END;  CREATE TYPE ACTOR\_TYPE\_INSTANCE AS TABLE OF ACTOR\_TYPE\_CONSTRUCTOR;  COMMIT;  DROP TYPE ACTOR\_TYPE\_CONSTRUCTOR;  DROP TYPE ACTOR\_TYPE\_INSTANCE;  DROP SEQUENCE SEQ\_ACTOR;  COMMIT;  ------------------------------------------------  CREATE SEQUENCE SEQ\_FACTS  START WITH 1  INCREMENT BY 1  NOCACHE  NOCYCLE;    CREATE TYPE INTERESTING\_FACT\_TYPE\_CONSTRUCTOR AS OBJECT(  INTERESTING\_FACT\_ID INT,  INTERSTING\_FACT VARCHAR2(700),  CONSTRUCTOR FUNCTION INTERESTING\_FACT\_TYPE\_CONSTRUCTOR(P\_INTERSTING\_FACT VARCHAR2)RETURN SELF AS RESULT,  MAP MEMBER FUNCTION GET\_INTERESTING\_FACT\_ID RETURN INT,  MEMBER PROCEDURE DISPLAY\_INTERSTING\_FACT(SELF IN OUT NOCOPY INTERESTING\_FACT\_TYPE\_CONSTRUCTOR)  )  /  CREATE TYPE BODY INTERESTING\_FACT\_TYPE\_CONSTRUCTOR AS  CONSTRUCTOR FUNCTION INTERESTING\_FACT\_TYPE\_CONSTRUCTOR(P\_INTERSTING\_FACT VARCHAR2)RETURN SELF AS RESULT IS  BEGIN  SELF.INTERESTING\_FACT\_ID := SEQ\_FACTS.NEXTVAL;  SELF.INTERSTING\_FACT := P\_INTERSTING\_FACT;  RETURN;  END;  MAP MEMBER FUNCTION GET\_INTERESTING\_FACT\_ID RETURN INT AS  BEGIN  RETURN INTERESTING\_FACT\_ID;  END;  MEMBER PROCEDURE DISPLAY\_INTERSTING\_FACT(SELF IN OUT NOCOPY INTERESTING\_FACT\_TYPE\_CONSTRUCTOR)IS  BEGIN  DBMS\_OUTPUT.put\_line(TO\_CHAR(INTERESTING\_FACT\_ID) || ' - ' || 'GENRE: ' || ' - ' || INTERSTING\_FACT);  END;  END;  CREATE TYPE INTERESTING\_FACT\_TYPE\_INSTANCE AS TABLE OF INTERESTING\_FACT\_TYPE\_CONSTRUCTOR;  COMMIT;  DROP TYPE INTERESTING\_FACT\_TYPE\_CONSTRUCTOR;  DROP TYPE INTERESTING\_FACT\_TYPE\_INSTANCE;  DROP SEQUENCE SEQ\_FACTS;  COMMIT; |

Приложение С Процедуры и функции

|  |
| --- |
| CREATE OR REPLACE PACKAGE ADMINFILMFUNCTIONS AS  PROCEDURE ADD\_COMMENT(  USER\_REVIEW\_TEXT IN USERS\_REVIEWS\_ON\_MOVIE.USER\_REVIEW\_TEXT%TYPE,  DATE\_OF\_REVIEW IN USERS\_REVIEWS\_ON\_MOVIE.DATE\_OF\_REVIEW%TYPE,  USER\_PROFILE\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USER\_PROFILE\_ID%TYPE,  ALL\_INFORMATION\_ABOUT\_FILM\_ID IN USERS\_REVIEWS\_ON\_MOVIE.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  );  PROCEDURE DELETE\_COMMENT(  COMMENT\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USERS\_REVIEWS\_ON\_MOVIE\_ID%TYPE  );  PROCEDURE ADD\_FILM(  TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.TITLE%TYPE,  ORIGINAL\_TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.ORIGINAL\_TITLE%TYPE,  POSTER IN ALL\_INFORMATION\_ABOUT\_FILM.POSTER%TYPE,  YEAR\_OF\_POSTING IN ALL\_INFORMATION\_ABOUT\_FILM.YEAR\_OF\_POSTING%TYPE,  COUNTRY IN ALL\_INFORMATION\_ABOUT\_FILM.COUNTRY%TYPE,  DESCRIPTION IN ALL\_INFORMATION\_ABOUT\_FILM.DESCRIPTION%TYPE,  RATING\_IMDb IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_IMDb%TYPE,  RATING\_KP IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_KP%TYPE,  BOX\_OFFICE\_RECEIPTS IN ALL\_INFORMATION\_ABOUT\_FILM.BOX\_OFFICE\_RECEIPTS%TYPE,  BUDGET IN ALL\_INFORMATION\_ABOUT\_FILM.BUDGET%TYPE,  AGE IN ALL\_INFORMATION\_ABOUT\_FILM.AGE%TYPE,  VIEWING\_TIME IN ALL\_INFORMATION\_ABOUT\_FILM.VIEWING\_TIME%TYPE,  GENRES IN ALL\_INFORMATION\_ABOUT\_FILM.GENRES%TYPE,  DIRECTORS IN ALL\_INFORMATION\_ABOUT\_FILM.DIRECTORS%TYPE,  ACTORS IN ALL\_INFORMATION\_ABOUT\_FILM.ACTORS%TYPE,  INTERESTING\_FACT IN ALL\_INFORMATION\_ABOUT\_FILM.INTERESTING\_FACT%TYPE  );    PROCEDURE DELETE\_FILM\_BY\_ID(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  );    FUNCTION FIND\_BY\_ID(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) RETURN SYS\_REFCURSOR;    FUNCTION GET\_ALL\_BY\_ID(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) RETURN SYS\_REFCURSOR;    PROCEDURE UPDATE\_FILM(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE,  TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.TITLE%TYPE,  ORIGINAL\_TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.ORIGINAL\_TITLE%TYPE,  POSTER IN ALL\_INFORMATION\_ABOUT\_FILM.POSTER%TYPE,  YEAR\_OF\_POSTING IN ALL\_INFORMATION\_ABOUT\_FILM.YEAR\_OF\_POSTING%TYPE,  COUNTRY IN ALL\_INFORMATION\_ABOUT\_FILM.COUNTRY%TYPE,  DESCRIPTION IN ALL\_INFORMATION\_ABOUT\_FILM.DESCRIPTION%TYPE,  RATING\_IMDb IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_IMDb%TYPE,  RATING\_KP IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_KP%TYPE,  BOX\_OFFICE\_RECEIPTS IN ALL\_INFORMATION\_ABOUT\_FILM.BOX\_OFFICE\_RECEIPTS%TYPE,  BUDGET IN ALL\_INFORMATION\_ABOUT\_FILM.BUDGET%TYPE,  AGE IN ALL\_INFORMATION\_ABOUT\_FILM.AGE%TYPE,  VIEWING\_TIME IN ALL\_INFORMATION\_ABOUT\_FILM.VIEWING\_TIME%TYPE,  GENRES IN ALL\_INFORMATION\_ABOUT\_FILM.GENRES%TYPE,  DIRECTORS IN ALL\_INFORMATION\_ABOUT\_FILM.DIRECTORS%TYPE,  ACTORS IN ALL\_INFORMATION\_ABOUT\_FILM.ACTORS%TYPE,  INTERESTING\_FACT IN ALL\_INFORMATION\_ABOUT\_FILM.INTERESTING\_FACT%TYPE  );    END ADMINFILMFUNCTIONS;  /  CREATE OR REPLACE PACKAGE BODY ADMINFILMFUNCTIONS AS    PROCEDURE DELETE\_COMMENT(  COMMENT\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USERS\_REVIEWS\_ON\_MOVIE\_ID%TYPE  )IS  BEGIN  DELETE USERS\_REVIEWS\_ON\_MOVIE  WHERE USERS\_REVIEWS\_ON\_MOVIE.USERS\_REVIEWS\_ON\_MOVIE\_ID = COMMENT\_ID;  COMMIT;  END DELETE\_COMMENT;  PROCEDURE ADD\_COMMENT(  USER\_REVIEW\_TEXT IN USERS\_REVIEWS\_ON\_MOVIE.USER\_REVIEW\_TEXT%TYPE,  DATE\_OF\_REVIEW IN USERS\_REVIEWS\_ON\_MOVIE.DATE\_OF\_REVIEW%TYPE,  USER\_PROFILE\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USER\_PROFILE\_ID%TYPE,  ALL\_INFORMATION\_ABOUT\_FILM\_ID IN USERS\_REVIEWS\_ON\_MOVIE.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) IS  BEGIN  INSERT INTO USERS\_REVIEWS\_ON\_MOVIE VALUES(  DEFAULT,USER\_REVIEW\_TEXT,DATE\_OF\_REVIEW,USER\_PROFILE\_ID,ALL\_INFORMATION\_ABOUT\_FILM\_ID  );  COMMIT;  END ADD\_COMMENT;  PROCEDURE ADD\_FILM(  TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.TITLE%TYPE,  ORIGINAL\_TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.ORIGINAL\_TITLE%TYPE,  POSTER IN ALL\_INFORMATION\_ABOUT\_FILM.POSTER%TYPE,  YEAR\_OF\_POSTING IN ALL\_INFORMATION\_ABOUT\_FILM.YEAR\_OF\_POSTING%TYPE,  COUNTRY IN ALL\_INFORMATION\_ABOUT\_FILM.COUNTRY%TYPE,  DESCRIPTION IN ALL\_INFORMATION\_ABOUT\_FILM.DESCRIPTION%TYPE,  RATING\_IMDb IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_IMDb%TYPE,  RATING\_KP IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_KP%TYPE,  BOX\_OFFICE\_RECEIPTS IN ALL\_INFORMATION\_ABOUT\_FILM.BOX\_OFFICE\_RECEIPTS%TYPE,  BUDGET IN ALL\_INFORMATION\_ABOUT\_FILM.BUDGET%TYPE,  AGE IN ALL\_INFORMATION\_ABOUT\_FILM.AGE%TYPE,  VIEWING\_TIME IN ALL\_INFORMATION\_ABOUT\_FILM.VIEWING\_TIME%TYPE,  GENRES IN ALL\_INFORMATION\_ABOUT\_FILM.GENRES%TYPE,  DIRECTORS IN ALL\_INFORMATION\_ABOUT\_FILM.DIRECTORS%TYPE,  ACTORS IN ALL\_INFORMATION\_ABOUT\_FILM.ACTORS%TYPE,  INTERESTING\_FACT IN ALL\_INFORMATION\_ABOUT\_FILM.INTERESTING\_FACT%TYPE  ) IS  BEGIN  --  INSERT INTO ALL\_INFORMATION\_ABOUT\_FILM VALUES(  DEFAULT,TITLE,ORIGINAL\_TITLE,POSTER,YEAR\_OF\_POSTING,COUNTRY,DESCRIPTION,RATING\_IMDb,RATING\_KP,BOX\_OFFICE\_RECEIPTS,BUDGET,AGE,VIEWING\_TIME,GENRES,DIRECTORS,ACTORS,INTERESTING\_FACT  );  COMMIT;  END ADD\_FILM;    PROCEDURE DELETE\_FILM\_BY\_ID(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) IS  BEGIN  DELETE ALL\_INFORMATION\_ABOUT\_FILM  WHERE ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID=FILM\_ID;  COMMIT;  END DELETE\_FILM\_BY\_ID;    FUNCTION FIND\_BY\_ID(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) RETURN SYS\_REFCURSOR  IS  V\_FILM\_CURSOR SYS\_REFCURSOR;  BEGIN  OPEN V\_FILM\_CURSOR FOR  SELECT \*  FROM ALL\_INFORMATION\_ABOUT\_FILM  WHERE ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID = FILM\_ID;  RETURN V\_FILM\_CURSOR;  END FIND\_BY\_ID;    FUNCTION GET\_ALL\_BY\_ID(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) RETURN SYS\_REFCURSOR  IS  V\_FILM\_CURSOR SYS\_REFCURSOR;  BEGIN  OPEN V\_FILM\_CURSOR FOR  SELECT ALL\_INFORMATION\_ABOUT\_FILM.\*,USERS\_REVIEWS\_ON\_MOVIE.USER\_REVIEW\_TEXT,USERS\_REVIEWS\_ON\_MOVIE.DATE\_OF\_REVIEW,USER\_PROFILE.LOGIN,USER\_PROFILE.AVATAR,USER\_PROFILE.USER\_ROLE,USERS\_REVIEWS\_ON\_MOVIE.USERS\_REVIEWS\_ON\_MOVIE\_ID  FROM ALL\_INFORMATION\_ABOUT\_FILM  FULL OUTER JOIN USERS\_REVIEWS\_ON\_MOVIE  ON USERS\_REVIEWS\_ON\_MOVIE.ALL\_INFORMATION\_ABOUT\_FILM\_ID=ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID  FULL OUTER JOIN USER\_PROFILE  ON USER\_PROFILE.USER\_PROFILE\_ID = USERS\_REVIEWS\_ON\_MOVIE.USER\_PROFILE\_ID  WHERE ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID = FILM\_ID;  RETURN V\_FILM\_CURSOR;  END GET\_ALL\_BY\_ID;      PROCEDURE UPDATE\_FILM(  FILM\_ID IN ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE,  TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.TITLE%TYPE,  ORIGINAL\_TITLE IN ALL\_INFORMATION\_ABOUT\_FILM.ORIGINAL\_TITLE%TYPE,  POSTER IN ALL\_INFORMATION\_ABOUT\_FILM.POSTER%TYPE,  YEAR\_OF\_POSTING IN ALL\_INFORMATION\_ABOUT\_FILM.YEAR\_OF\_POSTING%TYPE,  COUNTRY IN ALL\_INFORMATION\_ABOUT\_FILM.COUNTRY%TYPE,  DESCRIPTION IN ALL\_INFORMATION\_ABOUT\_FILM.DESCRIPTION%TYPE,  RATING\_IMDb IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_IMDb%TYPE,  RATING\_KP IN ALL\_INFORMATION\_ABOUT\_FILM.RATING\_KP%TYPE,  BOX\_OFFICE\_RECEIPTS IN ALL\_INFORMATION\_ABOUT\_FILM.BOX\_OFFICE\_RECEIPTS%TYPE,  BUDGET IN ALL\_INFORMATION\_ABOUT\_FILM.BUDGET%TYPE,  AGE IN ALL\_INFORMATION\_ABOUT\_FILM.AGE%TYPE,  VIEWING\_TIME IN ALL\_INFORMATION\_ABOUT\_FILM.VIEWING\_TIME%TYPE,  GENRES IN ALL\_INFORMATION\_ABOUT\_FILM.GENRES%TYPE,  DIRECTORS IN ALL\_INFORMATION\_ABOUT\_FILM.DIRECTORS%TYPE,  ACTORS IN ALL\_INFORMATION\_ABOUT\_FILM.ACTORS%TYPE,  INTERESTING\_FACT IN ALL\_INFORMATION\_ABOUT\_FILM.INTERESTING\_FACT%TYPE  ) IS  BEGIN  UPDATE ALL\_INFORMATION\_ABOUT\_FILM  SET  ALL\_INFORMATION\_ABOUT\_FILM.TITLE = UPDATE\_FILM.TITLE,  ALL\_INFORMATION\_ABOUT\_FILM.ORIGINAL\_TITLE = UPDATE\_FILM.ORIGINAL\_TITLE,  ALL\_INFORMATION\_ABOUT\_FILM.POSTER = UPDATE\_FILM.POSTER,  ALL\_INFORMATION\_ABOUT\_FILM.YEAR\_OF\_POSTING = UPDATE\_FILM.YEAR\_OF\_POSTING,  ALL\_INFORMATION\_ABOUT\_FILM.COUNTRY = UPDATE\_FILM.COUNTRY,  ALL\_INFORMATION\_ABOUT\_FILM.DESCRIPTION = UPDATE\_FILM.DESCRIPTION,  ALL\_INFORMATION\_ABOUT\_FILM.RATING\_IMDb = UPDATE\_FILM.RATING\_IMDb,  ALL\_INFORMATION\_ABOUT\_FILM.RATING\_KP = UPDATE\_FILM.RATING\_KP,  ALL\_INFORMATION\_ABOUT\_FILM.BOX\_OFFICE\_RECEIPTS = UPDATE\_FILM.BOX\_OFFICE\_RECEIPTS,  ALL\_INFORMATION\_ABOUT\_FILM.BUDGET = UPDATE\_FILM.BUDGET,  ALL\_INFORMATION\_ABOUT\_FILM.AGE = UPDATE\_FILM.AGE,  ALL\_INFORMATION\_ABOUT\_FILM.VIEWING\_TIME = UPDATE\_FILM.VIEWING\_TIME,  ALL\_INFORMATION\_ABOUT\_FILM.GENRES = UPDATE\_FILM.GENRES,  ALL\_INFORMATION\_ABOUT\_FILM.DIRECTORS = UPDATE\_FILM.DIRECTORS,  ALL\_INFORMATION\_ABOUT\_FILM.ACTORS = UPDATE\_FILM.ACTORS,  ALL\_INFORMATION\_ABOUT\_FILM.INTERESTING\_FACT = UPDATE\_FILM.INTERESTING\_FACT  WHERE ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID = UPDATE\_FILM.FILM\_ID;  COMMIT;  END UPDATE\_FILM;    END ADMINFILMFUNCTIONS;  /  CREATE OR REPLACE PACKAGE ADMINPROFILEFUNCTIONS AS  PROCEDURE ADD\_USER(  LOGIN IN USER\_PROFILE.LOGIN%TYPE,  AVATAR IN USER\_PROFILE.AVATAR%TYPE,  EMAIL IN USER\_PROFILE.EMAIL%TYPE,  PASSWORD IN USER\_PROFILE.PASSWORD%TYPE,  USER\_ROLE IN USER\_PROFILE.USER\_ROLE%TYPE  );    PROCEDURE ADD\_USER\_GET\_ID(  LOGIN IN USER\_PROFILE.LOGIN%TYPE,  AVATAR IN USER\_PROFILE.AVATAR%TYPE,  EMAIL IN USER\_PROFILE.EMAIL%TYPE,  PASSWORD IN USER\_PROFILE.PASSWORD%TYPE,  USER\_ROLE IN USER\_PROFILE.USER\_ROLE%TYPE,  USER\_ID OUT NUMBER  );    FUNCTION FIND\_BY\_EMAIL(  UEMAIL varchar  ) RETURN SYS\_REFCURSOR;    FUNCTION FIND\_BY\_ID(  USER\_ID IN USER\_PROFILE.user\_profile\_id%TYPE  ) RETURN SYS\_REFCURSOR;    FUNCTION FIND\_BY\_EMAIL\_AND\_PASSWORD(  UEMAIL USER\_PROFILE.EMAIL%TYPE,  UPASSWORD USER\_PROFILE.PASSWORD%TYPE  ) RETURN SYS\_REFCURSOR;    PROCEDURE DELETE\_USER\_BY\_ID(  USER\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE  );    PROCEDURE UPDATE\_USER(  USER\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE,  LOGIN IN USER\_PROFILE.LOGIN%TYPE,  AVATAR IN USER\_PROFILE.AVATAR%TYPE,  EMAIL IN USER\_PROFILE.EMAIL%TYPE,  PASSWORD IN USER\_PROFILE.PASSWORD%TYPE,  USER\_ROLE IN USER\_PROFILE.USER\_ROLE%TYPE  );    END ADMINPROFILEFUNCTIONS;  /  CREATE OR REPLACE PACKAGE BODY ADMINPROFILEFUNCTIONS AS  PROCEDURE ADD\_USER(  LOGIN IN USER\_PROFILE.LOGIN%TYPE,  AVATAR IN USER\_PROFILE.AVATAR%TYPE,  EMAIL IN USER\_PROFILE.EMAIL%TYPE,  PASSWORD IN USER\_PROFILE.PASSWORD%TYPE,  USER\_ROLE IN USER\_PROFILE.USER\_ROLE%TYPE  ) IS  UNIQUE\_EMAIL NUMBER:=0;  UNIQUE\_LOGIN NUMBER:=0;  BEGIN  SELECT COUNT(\*) INTO UNIQUE\_EMAIL FROM USER\_PROFILE WHERE USER\_PROFILE.email=ADD\_USER.EMAIL;  IF UNIQUE\_EMAIL = 0  THEN  SELECT COUNT(\*) INTO UNIQUE\_LOGIN FROM USER\_PROFILE WHERE USER\_PROFILE.LOGIN=ADD\_USER.LOGIN;  IF UNIQUE\_LOGIN = 0  THEN  INSERT INTO USER\_PROFILE VALUES(DEFAULT, LOGIN, AVATAR, EMAIL, PASSWORD, USER\_ROLE);  COMMIT;  ELSE  RAISE\_APPLICATION\_ERROR(-20003,'LOGIN ALREDY USED');  END IF;  ELSE  RAISE\_APPLICATION\_ERROR(-20000,'EMAIL ALREDY USED');  END IF;  END ADD\_USER;  PROCEDURE ADD\_USER\_GET\_ID(  LOGIN IN USER\_PROFILE.LOGIN%TYPE,  AVATAR IN USER\_PROFILE.AVATAR%TYPE,  EMAIL IN USER\_PROFILE.EMAIL%TYPE,  PASSWORD IN USER\_PROFILE.PASSWORD%TYPE,  USER\_ROLE IN USER\_PROFILE.USER\_ROLE%TYPE,  USER\_ID OUT NUMBER  )IS  UNIQUE\_EMAIL NUMBER:=0;  UNIQUE\_LOGIN NUMBER:=0;  BEGIN  SELECT COUNT(\*) INTO UNIQUE\_EMAIL FROM USER\_PROFILE WHERE USER\_PROFILE.EMAIL=ADD\_USER\_GET\_ID.EMAIL;  IF UNIQUE\_EMAIL = 0  THEN  SELECT COUNT(\*) INTO UNIQUE\_LOGIN FROM USER\_PROFILE WHERE USER\_PROFILE.LOGIN=ADD\_USER\_GET\_ID.LOGIN;  IF UNIQUE\_LOGIN = 0  THEN  INSERT INTO USER\_PROFILE VALUES(DEFAULT, LOGIN, AVATAR, EMAIL, PASSWORD, USER\_ROLE);  SELECT MAX(user\_profile\_id) INTO USER\_ID FROM USER\_PROFILE;  COMMIT;  ELSE  RAISE\_APPLICATION\_ERROR(-20003,'LOGIN ALREDY USED');  END IF;  ELSE  RAISE\_APPLICATION\_ERROR(-20000,'EMAIL ALREDY USED');  END IF;  END ADD\_USER\_GET\_ID;    PROCEDURE DELETE\_USER\_BY\_ID(  USER\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE  )  IS  BEGIN  DELETE FROM USER\_PROFILE WHERE USER\_PROFILE.USER\_PROFILE\_ID= USER\_ID;  COMMIT;  END DELETE\_USER\_BY\_ID;    PROCEDURE UPDATE\_USER(  USER\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE,  LOGIN IN USER\_PROFILE.LOGIN%TYPE,  AVATAR IN USER\_PROFILE.AVATAR%TYPE,  EMAIL IN USER\_PROFILE.EMAIL%TYPE,  PASSWORD IN USER\_PROFILE.PASSWORD%TYPE,  USER\_ROLE IN USER\_PROFILE.USER\_ROLE%TYPE  )IS  UNIQUE\_EMAIL NUMBER:=0;  UNIQUE\_LOGIN NUMBER:=0;  BEGIN  SELECT COUNT(\*) INTO UNIQUE\_EMAIL FROM USER\_PROFILE  WHERE USER\_PROFILE.email=UPDATE\_USER.EMAIL  AND USER\_PROFILE.USER\_PROFILE\_ID<>UPDATE\_USER.USER\_ID;  IF UNIQUE\_EMAIL = 0  THEN  SELECT COUNT(\*) INTO UNIQUE\_LOGIN FROM USER\_PROFILE  WHERE USER\_PROFILE.LOGIN=UPDATE\_USER.LOGIN  AND USER\_PROFILE.USER\_PROFILE\_ID<>UPDATE\_USER.USER\_ID;  IF UNIQUE\_LOGIN = 0  THEN  UPDATE USER\_PROFILE SET  USER\_PROFILE.LOGIN = UPDATE\_USER.LOGIN,  USER\_PROFILE.AVATAR = UPDATE\_USER.AVATAR,  USER\_PROFILE.EMAIL = UPDATE\_USER.EMAIL,  USER\_PROFILE.PASSWORD = UPDATE\_USER.PASSWORD,  USER\_PROFILE.USER\_ROLE = UPDATE\_USER.USER\_ROLE  WHERE USER\_PROFILE.USER\_PROFILE\_ID = UPDATE\_USER.USER\_ID;  COMMIT;  ELSE  RAISE\_APPLICATION\_ERROR(-20003,'LOGIN ALREDY USED');  END IF;  ELSE  RAISE\_APPLICATION\_ERROR(-20000,'EMAIL ALREDY USED');  END IF;    END UPDATE\_USER;    FUNCTION FIND\_BY\_EMAIL(  UEMAIL varchar  ) RETURN SYS\_REFCURSOR  IS  V\_USER\_CURSOR SYS\_REFCURSOR;  FIND NUMBER:=0;  BEGIN  SELECT COUNT(\*) INTO FIND FROM USER\_PROFILE WHERE USER\_PROFILE.EMAIL = UEMAIL;  IF FIND=1 THEN  OPEN V\_USER\_CURSOR FOR  SELECT \*  FROM USER\_PROFILE  WHERE USER\_PROFILE.EMAIL = UEMAIL;  RETURN V\_USER\_CURSOR;  ELSE  RAISE\_APPLICATION\_ERROR(-20002,'USER DO NOT FIND,CHECK CREDENTIALS');  END IF;  END FIND\_BY\_EMAIL;    FUNCTION FIND\_BY\_ID(  USER\_ID IN USER\_PROFILE.user\_profile\_id%TYPE  ) RETURN SYS\_REFCURSOR  IS  V\_USER\_CURSOR SYS\_REFCURSOR;  BEGIN  OPEN V\_USER\_CURSOR FOR  SELECT \*  FROM USER\_PROFILE  WHERE user\_profile.user\_profile\_id = USER\_ID;  RETURN V\_USER\_CURSOR;  END FIND\_BY\_ID;    FUNCTION FIND\_BY\_EMAIL\_AND\_PASSWORD(  UEMAIL USER\_PROFILE.EMAIL%TYPE,  UPASSWORD USER\_PROFILE.PASSWORD%TYPE  ) RETURN SYS\_REFCURSOR  IS  V\_USER\_CURSOR SYS\_REFCURSOR;  BEGIN  OPEN V\_USER\_CURSOR FOR  SELECT \*  FROM USER\_PROFILE  WHERE USER\_PROFILE.EMAIL='UEMAIL'  AND  USER\_PROFILE.PASSWORD='UPASSWORD';  BEGIN  IF V\_USER\_CURSOR%NOTFOUND THEN  CLOSE V\_USER\_CURSOR;  RAISE\_APPLICATION\_ERROR(-20001, 'No user found with the provided email');  END IF;  END;  RETURN V\_USER\_CURSOR;  END FIND\_BY\_EMAIL\_AND\_PASSWORD;    END ADMINPROFILEFUNCTIONS;  /  CREATE OR REPLACE PACKAGE SHAREDFUNCTIONS AS  FUNCTION GET\_ALL\_DATA\_FROM\_TABLE(table\_name IN VARCHAR2)  RETURN SYS\_REFCURSOR;    FUNCTION GET\_ALL\_LIKED\_FILMS(USER\_PROFILE\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE)  RETURN SYS\_REFCURSOR;    FUNCTION GET\_ALL\_SAVED\_FILMS(USER\_PROFILE\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE)  RETURN SYS\_REFCURSOR;    PROCEDURE ADD\_TO\_SAVED(  USER\_PROFILE\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USER\_PROFILE\_ID%TYPE,  ALL\_INFORMATION\_ABOUT\_FILM\_ID IN USERS\_REVIEWS\_ON\_MOVIE.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  );    PROCEDURE ADD\_TO\_LIKED(  USER\_PROFILE\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USER\_PROFILE\_ID%TYPE,  ALL\_INFORMATION\_ABOUT\_FILM\_ID IN USERS\_REVIEWS\_ON\_MOVIE.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  );  PROCEDURE ADD\_NEW\_VISIT(  DATE\_OF\_VISIT IN VISITS\_TABLE.DATE\_OF\_VISITS%TYPE  );  --INSERT INTO VISITS\_TABLE VALUES (DEFAULT,TO\_DATE('2003-12-12', 'YYYY-MM-DD'), 0);  END SharedFunctions;  /  CREATE OR REPLACE PACKAGE BODY ADMIN.SHAREDFUNCTIONS AS  FUNCTION GET\_ALL\_DATA\_FROM\_TABLE(table\_name IN VARCHAR2)  RETURN SYS\_REFCURSOR  IS  RESULT SYS\_REFCURSOR;  BEGIN  OPEN RESULT FOR  'SELECT \* FROM '||table\_name;  RETURN RESULT;  END;    FUNCTION GET\_ALL\_LIKED\_FILMS(  USER\_PROFILE\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE  ) RETURN SYS\_REFCURSOR IS  RESULT SYS\_REFCURSOR;  BEGIN  OPEN RESULT FOR  SELECT \* FROM USER\_LIKE  INNER JOIN ALL\_INFORMATION\_ABOUT\_FILM  ON USER\_LIKE.ALL\_INFORMATION\_ABOUT\_FILM = ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID  WHERE USER\_LIKE.USER\_PROFILE\_ID = GET\_ALL\_LIKED\_FILMS.USER\_PROFILE\_ID;  RETURN RESULT;  END GET\_ALL\_LIKED\_FILMS;  FUNCTION GET\_ALL\_SAVED\_FILMS(  USER\_PROFILE\_ID IN USER\_PROFILE.USER\_PROFILE\_ID%TYPE  ) RETURN SYS\_REFCURSOR IS  RESULT SYS\_REFCURSOR;  BEGIN  OPEN RESULT FOR  SELECT \* FROM USER\_WATCH\_LATER  INNER JOIN ALL\_INFORMATION\_ABOUT\_FILM  ON USER\_WATCH\_LATER.ALL\_INFORMATION\_ABOUT\_FILM = ALL\_INFORMATION\_ABOUT\_FILM.ALL\_INFORMATION\_ABOUT\_FILM\_ID  WHERE USER\_WATCH\_LATER.USER\_PROFILE\_ID = GET\_ALL\_SAVED\_FILMS.USER\_PROFILE\_ID;  RETURN RESULT;  END GET\_ALL\_SAVED\_FILMS;    PROCEDURE ADD\_NEW\_VISIT(  DATE\_OF\_VISIT IN VISITS\_TABLE.DATE\_OF\_VISITS%TYPE  ) IS  BEGIN  INSERT INTO VISITS\_TABLE VALUES (DEFAULT,TO\_DATE(DATE\_OF\_VISIT, 'YYYY-MM-DD'), 1);  COMMIT;  END;  PROCEDURE ADD\_TO\_SAVED(  USER\_PROFILE\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USER\_PROFILE\_ID%TYPE,  ALL\_INFORMATION\_ABOUT\_FILM\_ID IN USERS\_REVIEWS\_ON\_MOVIE.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) IS  HERE NUMBER:=0;  BEGIN  SELECT COUNT(\*) INTO HERE FROM USER\_WATCH\_LATER  WHERE USER\_WATCH\_LATER.USER\_PROFILE\_ID =USER\_PROFILE\_ID  AND  USER\_WATCH\_LATER.ALL\_INFORMATION\_ABOUT\_FILM =ALL\_INFORMATION\_ABOUT\_FILM\_ID;    IF(HERE=1)THEN  DELETE USER\_WATCH\_LATER  WHERE USER\_WATCH\_LATER.USER\_PROFILE\_ID =USER\_PROFILE\_ID AND USER\_WATCH\_LATER.ALL\_INFORMATION\_ABOUT\_FILM =ALL\_INFORMATION\_ABOUT\_FILM\_ID;  COMMIT;  ELSE  INSERT INTO USER\_WATCH\_LATER VALUES(  DEFAULT,ALL\_INFORMATION\_ABOUT\_FILM\_ID,USER\_PROFILE\_ID  );  COMMIT;  END IF;  END ADD\_TO\_SAVED;    PROCEDURE ADD\_TO\_LIKED(  USER\_PROFILE\_ID IN USERS\_REVIEWS\_ON\_MOVIE.USER\_PROFILE\_ID%TYPE,  ALL\_INFORMATION\_ABOUT\_FILM\_ID IN USERS\_REVIEWS\_ON\_MOVIE.ALL\_INFORMATION\_ABOUT\_FILM\_ID%TYPE  ) IS  HERE NUMBER:=0;  BEGIN  SELECT COUNT(\*) INTO HERE FROM USER\_LIKE  WHERE USER\_LIKE.USER\_PROFILE\_ID =USER\_PROFILE\_ID  AND  USER\_LIKE.ALL\_INFORMATION\_ABOUT\_FILM =ALL\_INFORMATION\_ABOUT\_FILM\_ID;    IF(HERE=1)THEN  DELETE USER\_LIKE  WHERE USER\_LIKE.USER\_PROFILE\_ID =ADD\_TO\_LIKED.USER\_PROFILE\_ID AND USER\_LIKE.ALL\_INFORMATION\_ABOUT\_FILM =ADD\_TO\_LIKED.ALL\_INFORMATION\_ABOUT\_FILM\_ID;  COMMIT;  ELSE  INSERT INTO USER\_LIKE VALUES(  DEFAULT,ALL\_INFORMATION\_ABOUT\_FILM\_ID,USER\_PROFILE\_ID  );  COMMIT;  END IF;  END ADD\_TO\_LIKED;  END SharedFunctions;  / |

Приложение D Создание триггера

|  |
| --- |
| CREATE OR REPLACE TRIGGER TR\_VISITS\_TABLE  BEFORE INSERT ON VISITS\_TABLE  FOR EACH ROW  DECLARE  visits\_count NUMBER;  PREV\_VALUE NUMBER;  PREV\_ID NUMBER;  BEGIN  SELECT COUNT(\*) INTO visits\_count  FROM VISITS\_TABLE  WHERE DATE\_OF\_VISITS = :new.DATE\_OF\_VISITS;  IF visits\_count > 0 THEN  dbms\_output.put\_line('allgood');  SELECT VISITS\_TABLE.VISITS\_NUMBER,VISITS\_TABLE.VISITS\_TABLE\_ID INTO PREV\_VALUE,PREV\_ID FROM VISITS\_TABLE  WHERE DATE\_OF\_VISITS = :new.DATE\_OF\_VISITS;  PREV\_VALUE:= PREV\_VALUE+1;  DELETE VISITS\_TABLE WHERE VISITS\_TABLE\_ID = PREV\_ID;  :new.VISITS\_TABLE\_ID := PREV\_ID;  :new.VISITS\_NUMBER := PREV\_VALUE;  END IF;  END;  / |