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

Специализация 1–40 01 01 Программное обеспечение информационных технологий (программирование интернет – изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

База данных «Разработка базы данных “Сеть кинотеатров” c использованием технологии полнотекстового поиска в БД

»

Выполнил студент Симакин Алексей Алксандрович

(Ф.И.О.)

Руководитель проекта Е.А. Блинова

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

Заведующий кафедрой к.т.н., доц. Смелов В.В .

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

Консультант: Е.А. Блинова.

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

Нормоконтролер: Е.А. Блинова.

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

Курсовой проект защищен с оценкой

Минск 2023

[**ВВЕДЕНИЕ 3**](#_Toc135213678)

[**1. Аналитический обзор литературы и формирование требований 4**](#_Toc135213679)

[**1.1. Анализ прототипов 4**](#_Toc135213680)

[**1.1.1. Siver Screen 5**](#_Toc135213681)

[**1.1.2 By Card 5**](#_Toc135213682)

[**1.1.3 Afisha.me 6**](#_Toc135213683)

[**1.1.4 Afisha.relax.by 7**](#_Toc135213684)

[**1.2 Требования к проекту 8**](#_Toc135213685)

[**2. Разработка программной системы 9**](#_Toc135213686)

[**3. Описание технологии и её применения в базе данных 20**](#_Toc135213687)

[**4. Описание приложения для демонстрации. 24**](#_Toc135213688)

[**Заключение 28**](#_Toc135213689)

[**Cписок источников 28**](#_Toc135213690)

[**Приложение A 28**](#_Toc135213691)

**ВВЕДЕНИЕ**

В современном мире многие люди посещают кинотеатры. И для посетителей кинотеатров существует множество площадок, сервисов и программных средств для бронирования билетов и просмотра информации о кинофильмах.

Целью данной работы являлась разработка реляционной базы данных для посетителей кинотеатров. Эта база данных составлялась для обеспечения зрителям функции бронирования и просмотра информации о кинофильмах, для дальнейшего посещения кинофильма.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных — база данных, основанная на реляционной модели данных. В качестве СУБД для базы данных была выбрана Oracle Database, в связи с ее высокой производительностью и надежностью.

В основной части будут затронуты все аспекты разработки проекта и обоснованы некоторые технические приёмы, к которым приходилось прибегнуть, с целью реализации работы веб-сервера с базой данных.

Для обеспечения работы приложения использовалась технология полнотекстового поиска, предназначенного для поиска документов в базе данных текстов с использованием полнотекстового индекса

Основные требования к приложению:

* Применение полнотекстового поиска
* Зрителю доступна функция регистрации персональных данных и функция бронирования мест
* Зрителю доступна функция просмотра информации о сеансах, жанров, режиссёров и фильмах
* Работнику доступна функция редактирования, добавления и удаления информации о сеансах, фильмах, залах
* Работнику доступна функция проверки посещения зрителя определённого сеанса

В пояснительной записке вы сможете найти краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

## Аналитический обзор литературы и формирование требований

## Анализ прототипов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

# **Siver Screen**

Самое популярное альтернативное решение — интернет-ресурс «Silver Screen», являющийся одним из самых известных интернет-ресурсом, для посетителей кинотеатров. Главными преимуществами данного ресурса является его наполненность информацией и широкая известность. Практически все кинотеатры размещают свои услуги на данном ресурсе. Пользователь данного ресурса имеет возможность детально и подробно ознакомиться с каждым поставщиком услуг. Ещё одним немаловажным достоинством данного программного средства является возможность пользователей узнать всю информацию о кинофильме (жанр, страна, режиссёр и т.д.), доступную информацию о кинотеатрах, кинозалах, а также посмотреть афишу предстоящих сеансов. Данное программное средство предоставляет пользователю возможность приобретения билетов на кинофильм в определённом кинотеатре.

При анализе недостатков было выделено, что на данном ресурсе присутствует большое количество ненужной информации. А также ограниченное количество кинотеатров для бронирования.

Интерфейс интернет-ресурса представлен на рисунке 1.1.

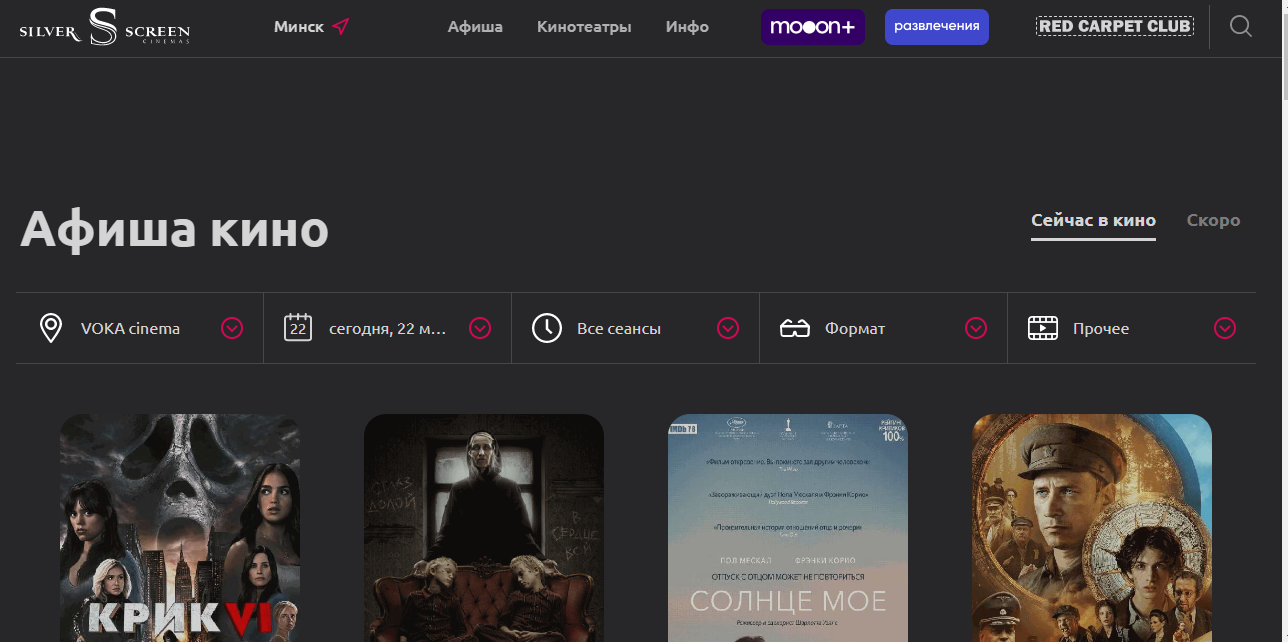


Рисунок 1.1 – Интернет-ресурс «Silver Screen»

# **1.1.2 By Card**

Еще одним альтернативным решением задач, поставленных в данном курсовом проекте, является интернет-ресурс «By Card». С помощью данного программного средства пользователь может посмотреть информацию о фильмах и забронировать билеты на киносеанс так же присутствует фильтр по датам. На данном ресурсе присутствует только полезная для пользователя информации ничего лишнего нет. Так же присутствует большое количество кинотеатров всех сетей.

При анализе ресурса не были выделены большие недостатки.

Интерфейс интернет-ресурса представлен на рисунке 1.2.

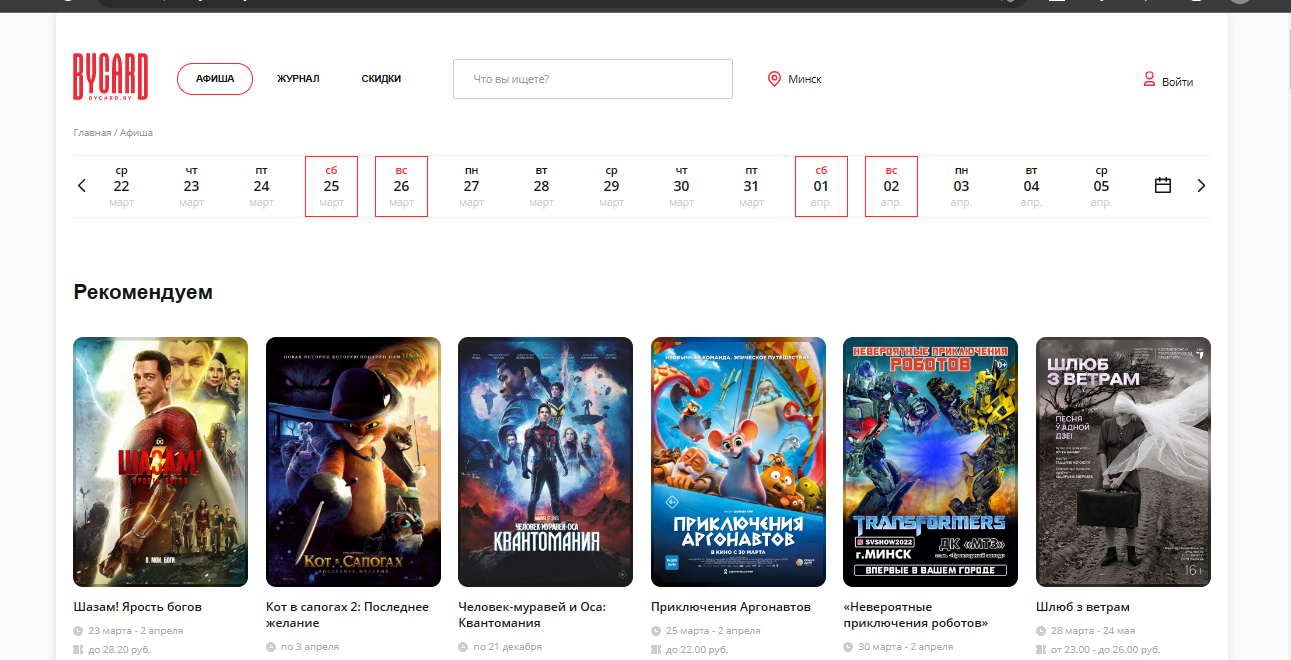


Рисунок 1.2 – Интернет-ресурс «By Card»

# **Afisha.me**

Ещё одной альтернативой решения поставленных задач является интернет-ресурс «Afisha.me». С помощью данного программного средства пользователь может ознакомиться со списком премьер в кинотеатрах, и время проведения сеансов, так же этот ресурс имеет множество разных афиш на другие виды деятельности (театр, вечеринки, цирк и др.). Немаловажным преимуществом является то, что сервис предоставляет возможность как сортировать кинофильмы по жанрам и датам проведения.

Однако, недостатками данной альтернативы являются малое количество информации о сеансах и кинофильмах.

Интерфейс интернет-ресурса представлен на рисунке 1.3.

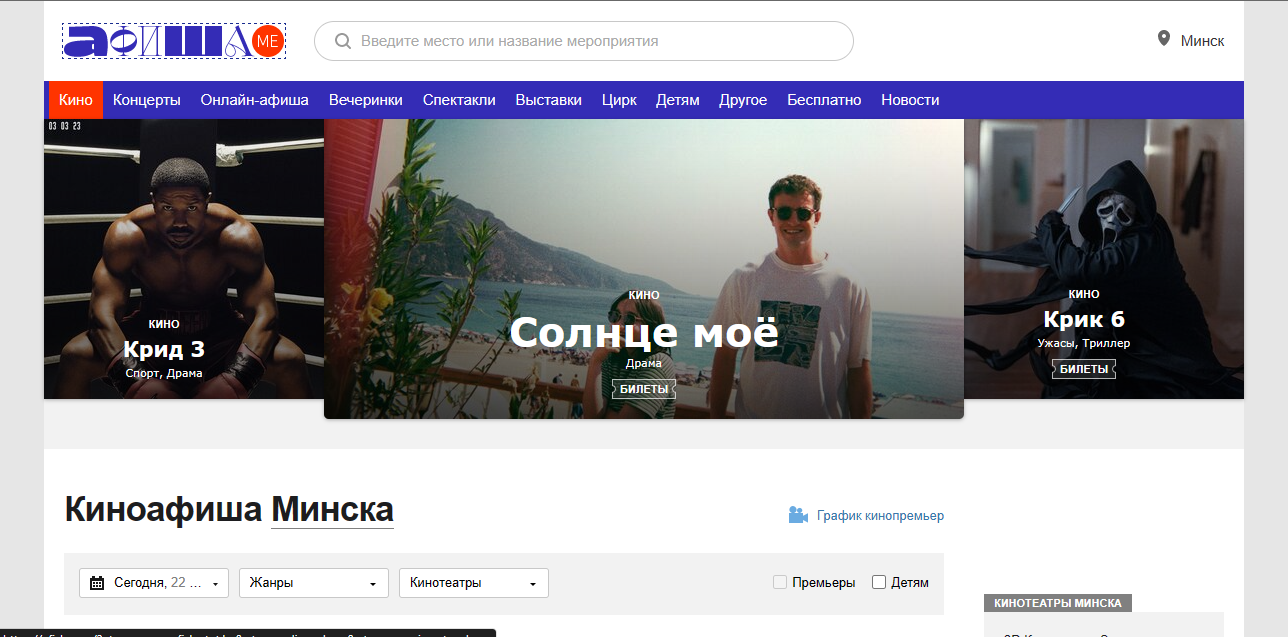


Рисунок 1.3 – Интернет-ресурс «Afisha.me»

# **Afisha.relax.by**

В список рассматриваемых альтернатив решения поставленных задач также можно включить малоизвестный интернет-ресурс «Afisha.relax.by». Данное программное средство предоставляет возможность найти практически любые услуги, в том числе и любые услуги по бронированию мест в кинотеатре. Важнейшим преимуществом является то, что в данном ресурсе присутствуют все кинотеатры и присутствует возможность покупки билета в любой из них.

Интерфейс интернет-ресурса представлен на рисунке 1.4.

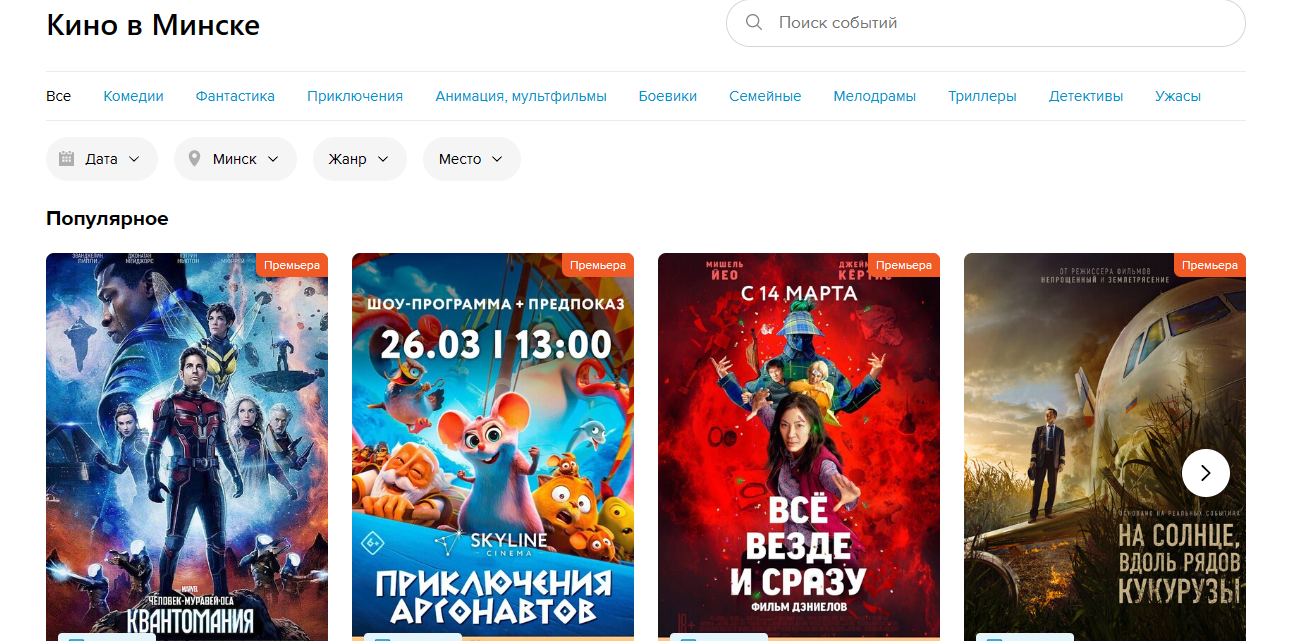


Рисунок 1.4 – Интернет-ресурс «Afisha.relax.by»

В результате аналитического обзора литературы было выведено несколько недостатков: у некоторых аналогов присутствует много лишней информации, а так в некоторых интернет-ресурсах нельзя бронировать места. Проанализировав данные недостатки можно учесть их при дальнейшем проектировании базы данных.

## Требования к проекту

Обзор вышеперечисленных известных аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформулировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте. Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

* Регистрация и авторизация пользователя;
* Шифрования пароля пользователя;
* Бронирование пользователем билетов;
* Пользователь может смотреть информацию о кинофильмах, сеансах, жанрах, режиссёрах;
* Работник может просматривать информацию о пользователе;
* Работник имеет функцию редактирования удаления и изменения информации о сеансах фильмах и залах
* Работник имеет функцию проверки посещения зрителя определенного сеанса;

## Разработка программной системы

1. **Проектирование баз данных**

Процесс проектирования базы данных состоит из следующих этапов:

* сбор информации;
* определение сущностей;
* определение атрибутов для каждой сущности;
* определение связей между сущностями;
* создание табличных пространств;
* создание ролей;
* создание профилей;
* создание пользователей;
* создание таблиц;

На этапе сбора информации вам необходимо точно определить, как будет использоваться база данных, и какая информация будет в ней храниться.

Далее следует этап определения сущностей и на нем определяются сущности, из которых будет состоять база данных.

Следующий этап — это определение атрибутов созданных сущностей. Атрибут представляет свойство, описывающее сущность. Атрибуты часто бывают числом, датой или текстом. Все данные, хранящиеся в атрибуте, должны иметь одинаковый тип и обладать одинаковыми свойствами.

На этапе определения связей работа происходит с ключами сущностей. Ключом называется набор атрибутов, однозначно определяющий запись. Ключи делятся на два класса: простые и составные.

На следующем этапе были созданы табличные пространства, в которых будет храниться все объекты базы данных.

Для правильной работы десктоп приложения с базой данных необходимо разграничение ролей. Для выполнения данной задачи в СУБД Oracle предусмотрены встроенные функции, такие как пользователи, роли и привилегии.

Чтобы составить визуальную взаимосвязанную структуры нашей базы данных, нам необходимо продумать, какая информация будет храниться в этих таблицах, после этого создать связи с помощью первичных и внешних ключей.

Далее представлена схема базы данных кинотеатра, которая состоит из 9 таблиц (Рис.2.1)

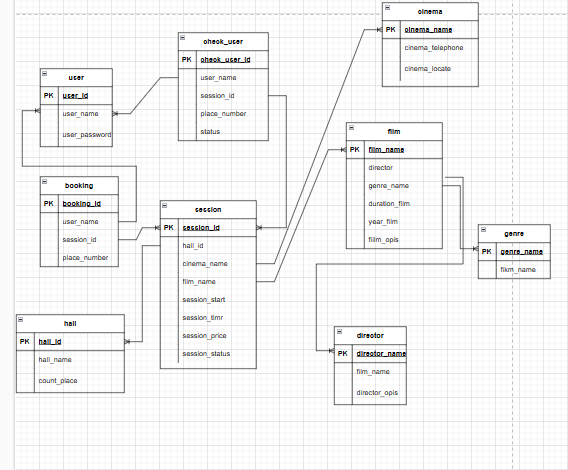
****

Рис. 2.1 – схема базы данных

Далее определена их структура и описание:

Таблица «**Users**» - хранит информацию о пользователях:

* user\_id – number;
* user\_name – varchar2(40);
* user\_password – varchar2(50);

Таблица «**Films**» - хранит информацию о фильмах:

* Film\_name - varchar2(1000);
* Director - varchar2(1000);
* Genre\_name - varchar2(20) CHECK (Genre\_name IN ('action', 'drama', 'melodrama', 'fantasy', 'family', 'documentary', 'western', 'adventure', 'comedy'));
* Duration\_film - number Check (Duration\_film <1000);
* Year\_film – DATE;
* Film\_opis - varchar2(2000);

Таблица «**Genre**» - хранит информацию о жанрах:

* Genre\_name - varchar2(20) CHECK (Genre\_name IN ('action', 'drama', 'melodrama', 'fantasy', 'family', 'documentary', 'western', 'adventure', 'comedy'));
* Film\_name - varchar2(1000);

Таблица «**Director**» - хранит информацию о режиссёрах:

* Director\_name - varchar2(1000);
* Film\_name - varchar2(1000);
* Director\_opis - varchar2(1000);

Таблица «**sessions**» - хранит информацию о сеансах:

* session\_id - number
* cinema\_name - varchar2(50);
* hall\_id number - Check (hall\_id = 1 or hall\_id = 2 or hall\_id = 3);
* Film\_name - varchar2(1000);
* session\_start - date;
* session\_time - number;
* session\_price - varchar2(20) check (session\_price like '%$');
* session\_status - varchar2(12) not null Check (session\_status = 'not started' or session\_status = 'started');

Таблица «**halls**» - хранит информацию о залах:

* hall\_id - number Check (hall\_id = 1 or hall\_id = 2 or hall\_id = 3);
* hall\_name - varchar2(20) Check (hall\_name = 'blue' or hall\_name = 'red' or hall\_name = 'black');
* count\_place – number;

Таблица «**cinema**» - хранит информацию о кинотеатрах:

* cinema\_name - varchar2(50);
* cinema\_telephone - varchar2(13) Check (cinema\_telephone like '+%');
* cinema\_locate - varchar2(50);

Таблица «**BOOKING**» - хранит информацию о бронировании мест на сеанс:

* booking\_id - number;
* user\_name - varchar2(40);
* session\_id - number not null;
* place\_number - number Check (place\_number <151);

Таблица «**check\_user**» - хранит информацию о посещении зрителя определенного сеанса:

* check\_user\_id – number;
* user\_name - varchar2(40);
* session\_id – number;
* place\_number - number Check (place\_number <151);
* status – varchar2(20) default 'none' check(status = 'none' or status = 'yes' or status = 'no');

В приложении имеется разделение пользователей на клиентов и работников.

Роль администратора в приложении не предусмотрена, она возлагается на администратора базы данных. Программное средство разделено на два приложения. Одно предусмотрено для клиентов, а второе для работников.

Диаграмма вариантов использования приложения для работника представлена на рисунке 2.2.

  
Рис. 2.2 – Диаграмма работника

Диаграмма вариантов использования приложения для пользователя

представлена на рисунке 2.3.

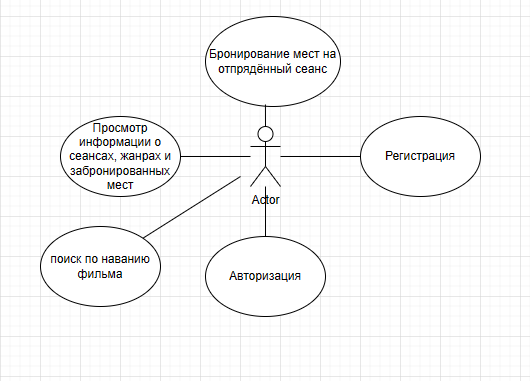


Рис. 2.3 – Диаграмма пользователя

* 1. **Разработка объектов баз данных**

**Таблицы**

Для работы в базе данных мы должны создать таблицы и организовать между ними связи, которые будут представлять логическую структуру БД. В нашу базу будут включены такие таблицы:

* Users;
* Films;
* Genre;
* Director;
* Halls;
* Cinema;
* Sessions;
* Booking;
* Check\_user.

Создание этих таблиц проходит исполнение команд блока PL/SQL.

Все таблицы создаются в отдельном файле TableView.sql, для дальнейшей возможности запускать скрипт в целом файле в новой базе данных.

**Создание ролей**

Для правильной работы приложения нужно разграничение ролей. Для этого были созданы два пользователя. Первый с именем XXXCORE1 (обычный пользователь) (Рис. 2.4) и второй с именем Syskino (Работник) (Рис.2.5)



Рис. 2.4 – пользователь XXXCORE1

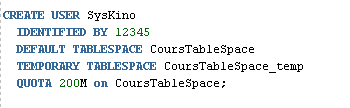


Рис. 2.5 – пользователь Syskino

Далее были выданы права для пользователя XXXCORE1 выполнение процедур из пакетов (Рис.2.6):



Рис. 2.6 – Выдача прав пользователю XXXCORE1

**Создание процедур**

**Процедура** — это подпрограмма, которая выполняет специфическое действие.

Наиболее используемые процедуры представлены в приложении А.

При разработке курсового проекта было создано большое количество процедур для следующих целей:

Для пользователя:

* регистрация;
* авторизация;
* просмотр информации о сеансах;
* просмотр информации о фильмах;
* просмотр информации о режиссёрах;
* просмотр информации о кинотеатрах;
* просмотр информации о залах;
* поиск фильма по имени;
* просмотр забронированных мест;
* бронирование мест.

Для работника:

* авторизация;
* информация о забронированных местах;
* проверка посещение пользователя определенного сеанса;
* добавление информации о режиссёрах;
* удаление информации о режиссёрах;
* изменение информации о сеансах (начался ли он);
* добавление и удаления информации о сеансах;
* проверка начала сеанса;
* добавление и удаление информации о фильмах.

Во всех процедурах присутствует проверка вводимых данных. Например, в процедуре авторизации пользователе идет проверка на существование пользователя, а для процедуры бронирование мест проверка на свободные места.

В процедуре регистрации есть функция шифрования пароля (Рис.2.7):

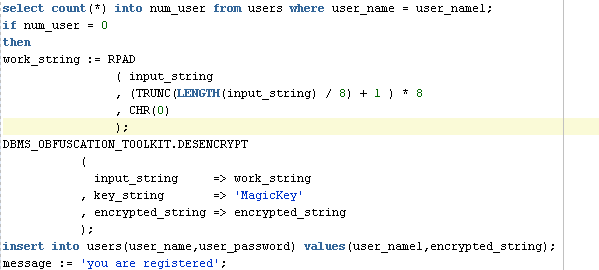


Рис. 2.7 – отрывок из процедуры регистрации – шифрование пароля

**Функции**

**Функция** — это подпрограмма, которая вычисляет значение. В данном курсовом проекте существует только 2 функции:

* генерация пароля пользователя;
* генерация логина пользователя.

**Представления**

Представление — это виртуальная таблица, основанная на результирующем наборе запроса.

Они содержит строки и столбцы, как и настоящая таблица. Поля в представлении — это поля из одной или нескольких реальных таблиц в базе данных.

Представления используются для абстракции от соединения таблиц и сокрытия их реализации внутри доступных пользователям процедур.

В данном курсовом проекте были созданы 3 представления:

* Представление для таблицы проверки посещения пользователя сеанса;
* Представление для таблицы кинотеатры;
* Представление для таблицы залы.

Далее приставлено представление «cinema\_v» (Рис. 2.7):

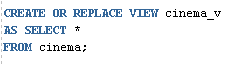


Рис. 2.7 – преставление «cinema\_v»

**Триггеры**

Триггер является именованным модулем PL/SQL, который хранится в базе данных и может быть вызван повторно. Вы можете включать и отключать триггер, но не можете явно вызывать его. Когда триггер включен, база данных автоматически вызывает его – то есть триггер срабатывает – всякий раз, когда происходит событие, которое вызывает триггер. Пока триггер отключен, он не срабатывает. В курсовом проекте присутствуют 4 After триггера:

* Триггер перед вставкой в таблицу фильмы вставляет данные в таблицу жанры;
* Триггер перед удалением из таблицы фильмы удаляет данные из таблицы жанры;
* Триггер перед вставкой в таблицу сеансы добавляет дополнительные данные в эту таблицу;
* Триггер перед вставкой в таблицу бронирования вставляет данные в таблицу проверки посещения пользователя;
* Триггер проверки даты сеанса и изменения её при входе в систему.

Пример триггера «insert\_check\_user» (Рис. 2.8):

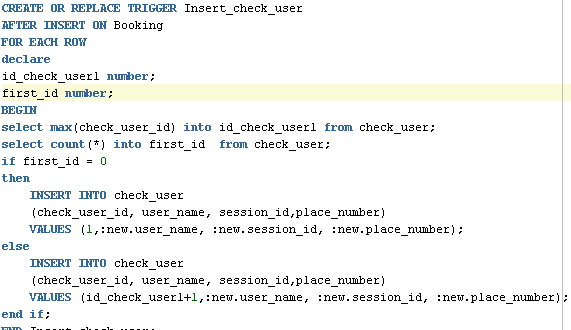


Рис. 2.8 – триггер «insert\_check\_user»

**Пакеты**

После создание пользователей, разбиение их на роли и создания процедур были созданы пакеты для хранения этих процедур для определенного пользователя.

В Oracle PL/SQL пакеты это – набор элементов: процедур, функций, определения типов; объявления переменных, констант можно объединить в пакет. После написания пакет PL/SQL компилируется, а затем сохраняется в базе данных Oracle, где его содержимое может использоваться многими приложениями.

Для логического разделения процедур были созданы четыре пакета

* PACK\_USER – пакет для хранения процедур для пользователя;
* PACK\_ROOT – пакет для хранения процедур для работника;
* PACK\_LOGIN\_USER – пакет для хранения процедур для авторизации и регистрации пользователей приложения;
* PACK\_OTHERS – пакет для остальных процедур.

Заголовок пакета, содержащий процедуры для работы клиента представлен на рисунке 2.9.

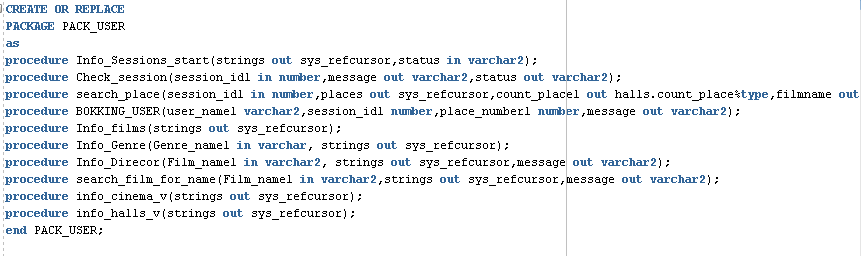


Рисунок 2.9 – Заголовок пакета PACK\_USER

Заголовок пакета, содержащий процедуры для работы работника представлен на рисунке 2.10.

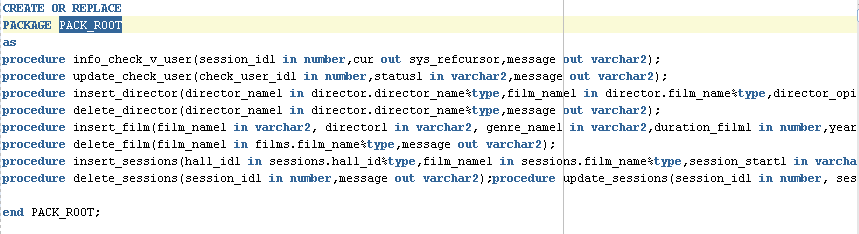


Рис. 2.10 – Заголовок пакета PACK\_ROOT

* 1. **Описание процедур импорта и экспорта**

**Процедура экспорта**

Для комфортного администрирования базы данных в ней необходимо реализовать возможность экспорта и импорта данных в удобный для администратора формат. В нашем случае это формат xml и txt.

Для экспорта данных из таблицы в файл xml и txt были созданы Directory для доступа к папке. (Рис. 2.11-2.12)

Объект Directory является логической ссылкой в базе данных на каталог файловой системы сервера, где установлена БД Oracle. Владельцем всех объектов Directory в базе данных является пользователь SYS, даже если объект Directory создан другим пользователем. Имена объектов Directory уникальны внутри всей БД.



Рис. 2.11 – директория для xml файла



Рис. 2.12 – директория для txt файла

Для экспорта данных использовался пакет UTL\_FILE. Процедура для записи в xml файл представлена на рисунке 2.13.

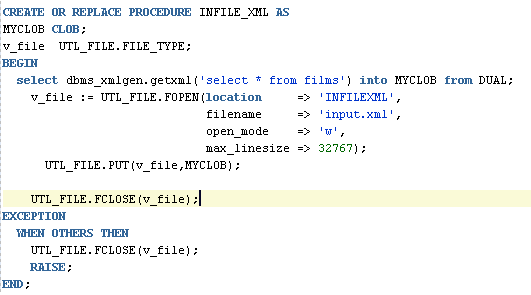


Рис. 2.13 – экспорт данных в xml из таблицы films

Для txt формата процедура имеет такую же структуру.

**Импорт данных**

Для импорта данных из xml так же была создана Directory которая осуществляет доступ к каталогу. Импорт осуществляется с помощью XML\_TABLE который приводит xml формат к формату таблицы. Процедура для импорта из xml формата приставлена на Рис. 2.14

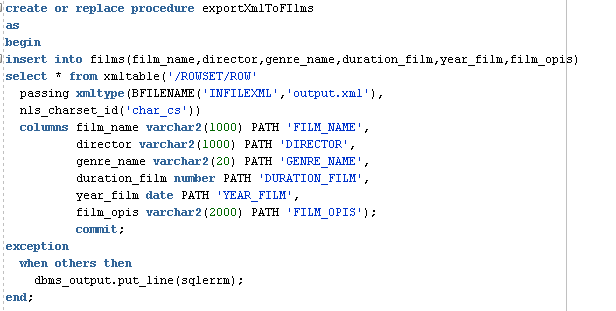


Рис. 2.14 – импорт данных их xml в таблицу films

Для импорта из txt формата использовалась sqlldr. Для этого был создан ctl файл с информацией о таблице и о записи (Рис. 2.15)

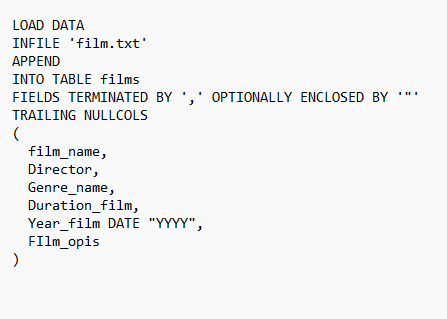


Рис. 2.15 – файл ctl для импорта из txt формата

Далее с помощью команды sqlldr были записаны три строки в таблицу films (Рис. 2.16)

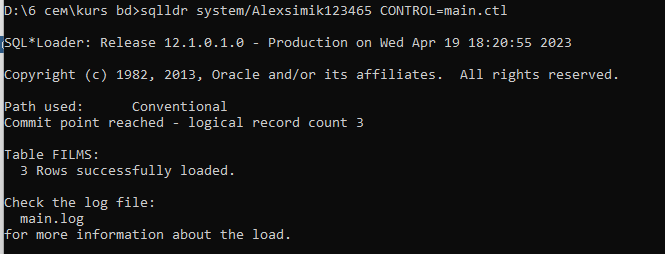


Рис. 2.16 – запись в таблицу Films

* 1. **Тестирование производительности**

Для тестирования производительности базы данных необходимо заполнить её большим количеством данных 100000 срок и выполнить запрос к ней. Для этого была создана функция RandomString, которая генерирует данные и определённого набора строк (Рис. 2.17)

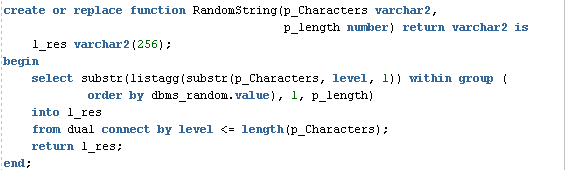


Рис. 2.17 – функция RandomString

Для записи 100000 был создан цикл от 1 до 100000 который с помощью цикла записывает данные, сгенерированные фикцией RandomString. Блок кода записи данных в таблицу Users представлен на Рис. 2.18:



Рис. 2.18 – запись 100000 в таблицу Users

Количество записей в таблице представлены на рисунке 2.19



Рис. 2.19 – число записей в таблице

Для начала был сделан запрос к таблице users и выполнен план запроса (Рис. 2.20)



Рис. 2.20 – план запроса для таблицы без индекса

Для улучшения производительности был создан B-Tree индекс для таблицы users (Рис. 2.21)



Рис. 2.21 – создание индекса для таблицы users

Далее ля проверки производительности был построен план запроса для таблицы уже с содранным индексом, плана запроса вы можете наблюдать на рис 2.22

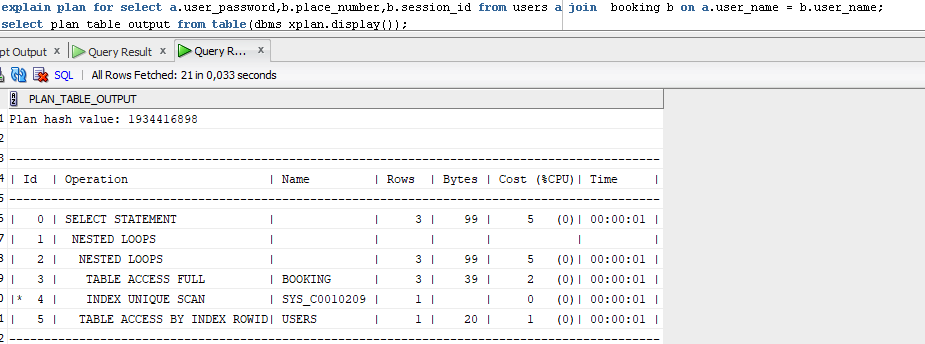


Рис. 2.22 – выполнение плана запроса.

Результат плана запроса показал, что стоимость запроса была 2297104002, применили индекс и стала 1934416898, поэтому производительность улучшилась.

Далее было произведено тестирование производительности с помощью секционирования. Для этого была создана таблица с секционированием LIST (Рис. 2.23)

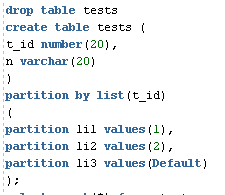


Рис. 2.23 – таблица с секционированием LIST

Далее таблица была заполнена 1000000 строк. И выведен план запроса для обращения к таблице с фильтрацией и обращение к определённой секции (Рис. 2.24–2.25)

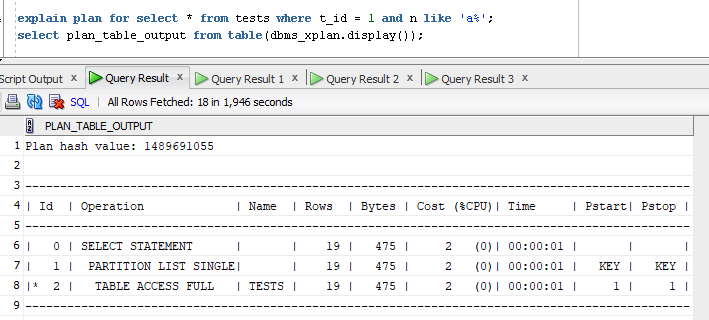


Рис. 2.24 – план запроса к таблице

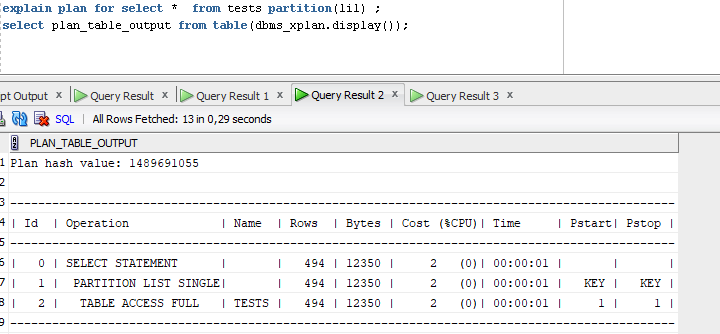


Рис. 2.25 – план запроса к секции таблицы

В результате тестирования получилось, что время выполнения запроса к таблице больше, чем выполнения запроса к секции таблицы, значит производительность улучшилась.

## Описание технологии и её применения в базе данных

Используемая технология – **полнотекстовый поиск (Full Text Search).**

Oracle Text — это бесплатная компонента, входящая в состав Oracle Database, предназначенная для создания приложений с использованием полнотекстового поиска. В предыдущих версиях Oracle имела также названия ConText и interMedia Text. Начиная с 11ой версии входит во все редакции СУБД.

Для использования полнотекстового поиска нужно выдать привилегии CTX\_DDL для использования технологии (Рис. 3.1):



Рис. 3.1 – предоставление привилегий

Далее нам нужно определить предварительные настройки. Для этого мы создаем Lexer – «коэффициентом соответствия». Есть несколько видов Lexer, но мы будем использовать AUTO\_LEXER – Идентифицирует индексируемый язык путем изучения содержимого и применяет подходящие параметры (включая выделение корней) для этого языка. Лучше всего работает, когда каждый документ содержит один язык и имеет по крайней мере пару абзацев текста для облегчения идентификации. И установим параметр BASIC\_WORDLIST – Используйте параметр списка слов, чтобы включить такие параметры запроса, как выделение корней и нечеткое сопоставление для вашего языка. Вы также можете использовать параметр «Список слов», чтобы включить индексирование подстрок и префиксов, что повышает производительность для запросов с подстановочными знаками. Лучше всего работает, когда каждый документ содержит один язык и имеет по крайней мере пару абзацев текста для облегчения идентификации. Далее устанавливаем атрибуты. Настройка представлена на Рис. 3.2

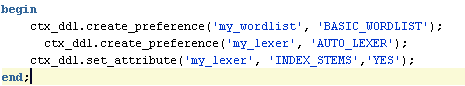


Рис. 3.2 – предварительная настройка для индекса

Далее нам нужно создать индекс для осуществления поиска они бывают 4 типов:

* CONTEXT – основной тип индекса. Применяется для индексирования содержимого одной или нескольких колонок таблицы;
* CTXCAT – применяется для индексации небольших текстовых фрагментов, не требует периодической переиндексации;
* CTXRULE – применяется при создании приложений для классификации документов;
* CTXXPATH – предназначен для ускорения работы с полями типа XMLType.

Создаем индекс CTXCAT для таблицы films и укажем имя LEXER и WORDLIST Рис. 3.3



Рис. 3.3 – создание индекса для таблицы films и столбца Film\_opis

Далее с помощью конструкции constrains мы выполним поиск отрывка слова в столбце film\_opis таблицы films

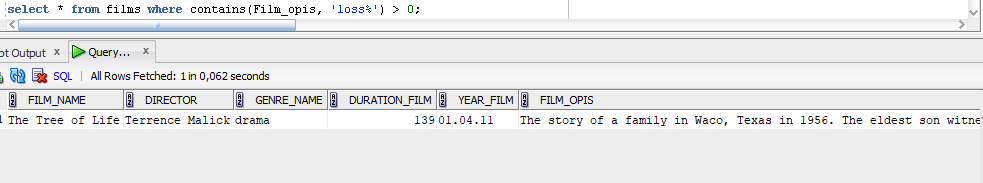


Рис. 3.4 – результат запроса

Так же можно использовать логические операторы, например выбор из нескольких значений (| или or или ,)

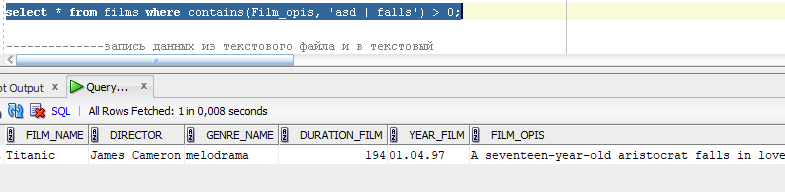


Рис. 3.5 – результат выполнения запроса с оператором or

Оператор and. Что бы использовать оператор and явна его нужно заключить в скобки {}, Пример выполнения оператора and на рис. 3.6

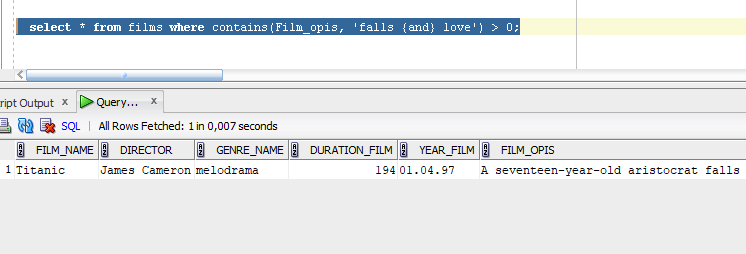


Рис. 3.6 – результат выполнения запроса с оператором and

Оператор not (~) Используется для поиска документов, содержащих одно условие запроса, а не другое. В примере на Рис. 3.7 мы выбираем что monster не red и запрос нам выдал, что монстер green.

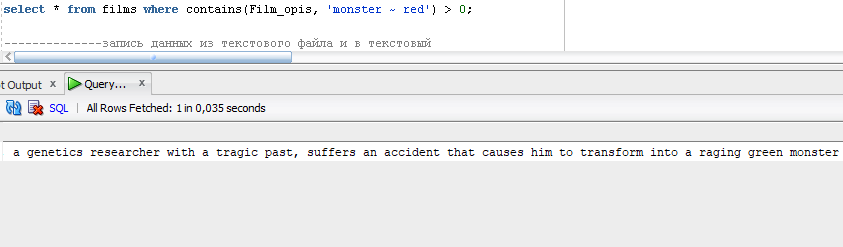


Рис. 3.7. – результат выполнения запроса с использованием оператора not

Оператор near предназначен для поиска на определенном расстоянии друг от друга слов. На рисунке поиск слов accident и raging которые находятся на расстоянии 8 слов друг от друга (Рис. 3.8). Так де если нужно найти слова которые находиться радом можно использовать знак (;) .

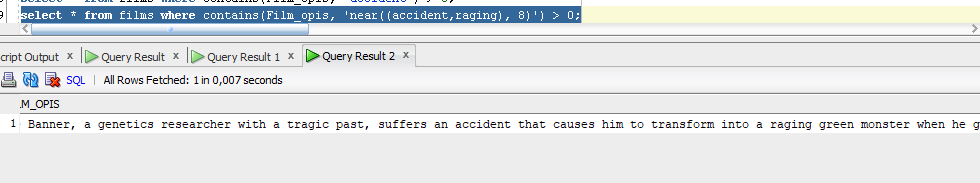


Рис. 3.8 – результат выполнения запроса используя оператор near

Оператор (!), расширяет поисковые запросы в зависимости от того, как звучит слово (Рис. 3.9).

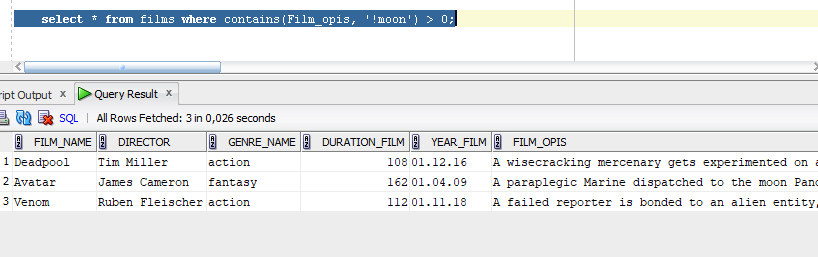


Рис. 3.9 – результат выполнения запроса используя звучание слова

**Сравнение Oracle Text и Like.**

Для сравнения проведем анализ производительности операторов. То есть построим план запроса для Oracle Text и Like и проведем сравнения. Будем использовать таблицу films и столбца film\_opis.

План запроса для Oracle Text представлен на рис. 3.10:

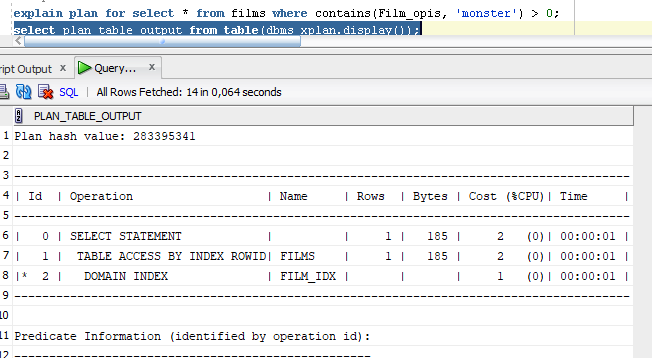


Рис. 3.10 – План запроса Full text search

План запроса для Like представлен на рис. 3.11:

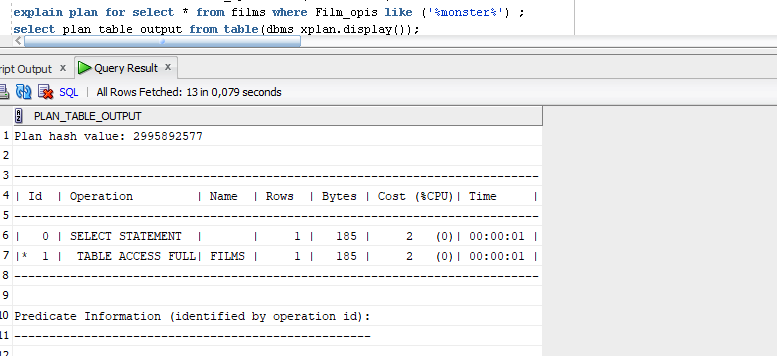


Рис. 3.11 – План запроса Like

В результате можно вывести что запрос с помощью Full text search быстрее, чем запрос с помощью оператора like. Это связанно с там что для full text search мы создавали индекс на определенный столбец так что для более быстрого поиска лучше использовать full text search. Однако для full text search нужна некоторая подготовка в виде создания индекса, а для like она не нужна. Так же full text search является более гибким он может использовать большое количество операторов для выбора: near, or, and, not и т.д.

## 4. Описание приложения для демонстрации.

**4.1 Тестирование клиентской области**

Приложение проектировалось и разрабатывалось в направлении минимизации человеческих ошибок. В приложении присутствует минимальное количество мест, где клиент может вызвать ошибку программы. Все варианты использования приложения продуманы и не допускают возможности сделать некорректное действие. При регистрации пользователя происходит проверка на совпадения пароля и существования пользователя. Если попробовать зарегистрироваться, не заполнив поля, появятся предупреждения, то есть у пользователя не получится зарегистрироваться пока он не заполнит все поля (Рис. 4.1)

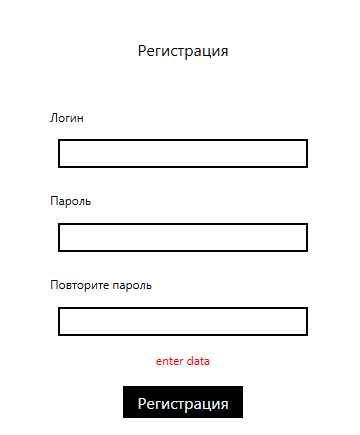


Рис. 4.1 – поля не заполнены

Если попробовать зарегистрироваться с разными паролями то приложение выдаст ошибку passwords don’t match что значит что пароли не совпадает (Рис. 4.2).



Рис. 4.2 – не совпадение паролей

* 1. **Тестирование области работника**

Что бы войти в систему как работник нужно ввести пароль и логин root. Тогда вы попадёте в окно работника.

При удалении фильма если введено несуществующее название то приложение выдаст ошибку no search film (Рис.4.3)

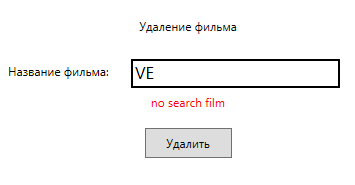


Рис. 4.3 – ошибка no search film

При изменении сеанса идет проверка на существование сеанса. Если сеанса не существует, то выводит ошибку no search session (Рис.4.4)

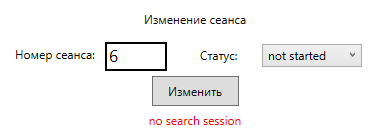


Рис. 4.4 – ошибка no search session

При добавлении режиссёра идет проверка на нулевые значения. Сели данные в каком-либо месте не введены то выводит ошибку press data (Рис. 4.5)

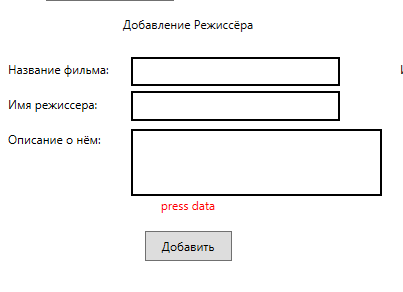


Рис.4.5 – ошибка press data

**4.3 Руководство пользователя**

**Область клиента**

Для начала пользователю нужно завести свой аккаунт. Для этого нужно придумать логин и пароль (Рис. 4.6):

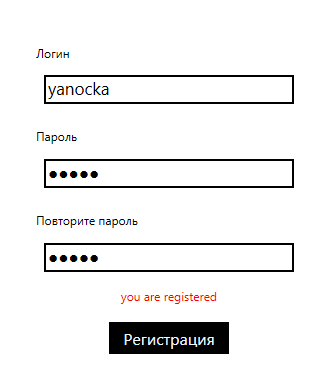


Рис. 4.6 – регистрация пользователя

Далее, чтобы войти в систему пользователю нужно корректно ввести логин и пароль. После входа нас встречает главное меню, на котором присутствует меню для переходов на страницы приложения. Первое в меню идет страница жанров, на которой присутствует функция выбора жанра. При выборе жанра приложение выводит соответствующий фильм по этому жанру. Пример с жанром action представлен Рис. 4.7:

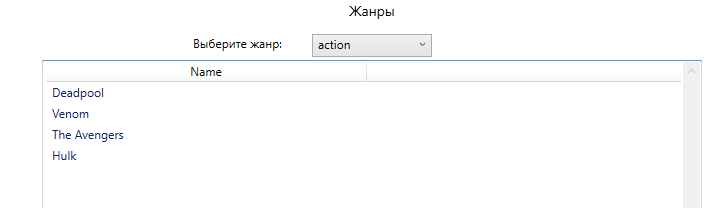


Рис. 4.7 – страница жанров

Далее в меню идет страница фильмы на которой присутствует информация о фильмах (Рис. 4.8)

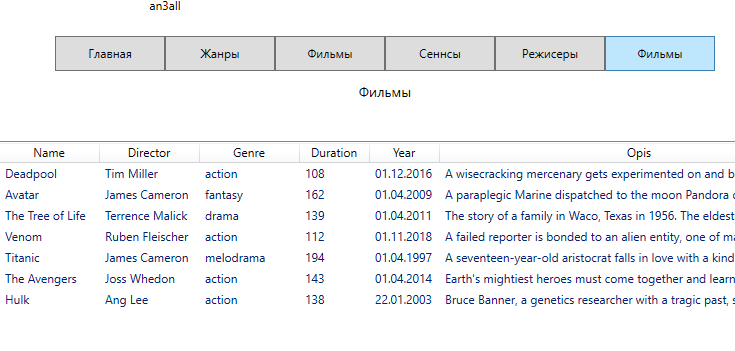


Рис. 4.8 – страница фильмы

Далее идет кладка сеансы на корой предложены все доступные сеансы. Если пользователь хочет забронировать место на сеанс для этого существует кнопка бронировать. При нажатии кнопки бронировать приложение спросит номер сеанса, на который вы хотите забронировать места. Далее вас перенаправит на страницу бронирования, где вы можете почитать подробную информацию о сеансе и просмотреть уже забронированные места (Рис. 4.9)

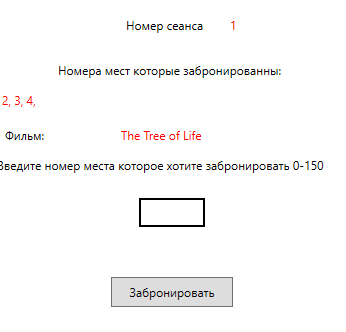


Рис. 4.10 – бронирование места на сеанс

Далее идет вкладка режиссёры на которой пользователь по названию фильма может найти доступную информацию о режиссёре этого фильма (Рис.4.11)

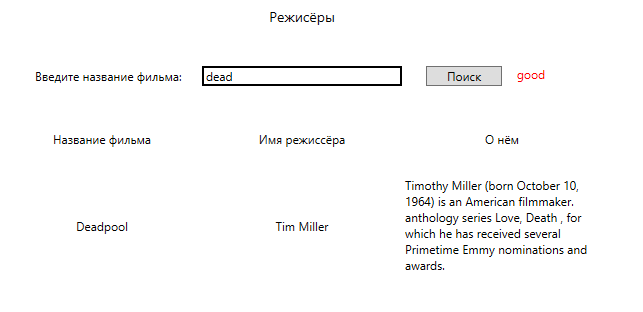


Рис. 4.11 – страница режиссёры

**4.4 Область работника**

Что бы войти в область работника нужно написать в окне авторизации логин и пароль root. В области работника есть функции добавления, удаления, изменения сеансов, режиссёров и фильмов. Главной функцией является проверка посещения пользователей определённого сеанса. Для этого в строке номера сеанса нужно ввести номер сеанса. После этого приложение выведет все забронированные места на этот сеанс. Далее работник может поставить в статусе места пришёл ли зритель на сеанс Рис. 4.12

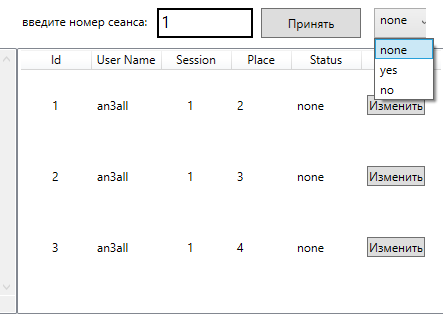


Рис. 4.12 – проверка пришёл ли зритель

В результате было разработано приложение сеть кинотеатров для взаимодействия с базой данных и выполнения ее процедур.

## Заключение

Данный курсовой проект был направлен на развитие навыков администрирования базы данных, а также на улучшение понимания взаимодействия программного средства с БД Oracle 12C на удалённом сервере.

В ходе выполнения работы были достигнуты следующие результаты:

* исследованы обучающие приложения-аналоги;
* сформулирована модель данных и приведено её графическое представление;
* подготовлена база данных Oracle с использованием технологии полнотекстового поиска;
* подготовлены процедуры для осуществления импорта и экспорта данных с форматом xml и txt;
* были созданы пользователи с различными привилегиями для работы с базой данных;
* были созданы скрипты для быстрого развертывания начальных таблиц компонентов и начальных таблиц пользователей базы данных с дальнейшим их заполнением;
* разработано приложение для взаимодействия с базой данных

## Cписок источников

1. Язык C# и .NET Framework сайт Microsoft [Электронный ресурс] – https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/introduction-to-wpf?view=netframeworkdesktop-4.8.
2. Руководство по WPF и работе с базами данных [Электронный ресурс] – https://metanit.com/sharp/wpf/.
3. Документация Oracle [Электронный ресурс] / Foundation, Inc. https://translated.turbopages.org/proxy\_u/en-ru.ru.fb77887c-6448f03d-9ab29a9e-74722d776562/https/docs.oracle.com/en/database/oracle/oracle-database/21/ccapp/indexing-with-oracle-text.html.
4. Официальный сайт Oracle [Электронный ресурс] / Foundation, Inc. <https://www.oracle.com/>.
5. Сайт Oraclesql [Электронный sресурс] / Foundation, Inc. <https://oracleplsql.ru/procedures.html>.

## Приложение A

Код процедуры pack\_root

|  |
| --- |
|  |
| create or replace package body PACK\_ROOT  as  ---1  procedure info\_check\_v\_user(session\_id1 in number,cur out sys\_refcursor,message out varchar2)  is  cursor curs is select \* from check\_user\_v where session\_id = session\_id1;  i number;  g number;  begin  select count(\*) into i from sessions where session\_id = session\_id1;  select count(\*) into g from check\_user\_v where session\_id = session\_id1;  if i > 0  then  if g = 0  then  message :='no search people';  dbms\_output.put\_line(message);  else  for fcurs in curs  loop  dbms\_output.put\_line('ID: '||fcurs.check\_user\_id||' User\_name: '||fcurs.user\_name||' session\_id: '||fcurs.session\_id||' num\_pace: '||fcurs.place\_number||' status: ' ||fcurs.status);  end loop;  message :='good';  dbms\_output.put\_line(message);  open cur for select \* from check\_user\_v where session\_id = session\_id1;  end if;  else  message :='no search session';  dbms\_output.put\_line(message);  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end info\_check\_v\_user;  ---2  procedure update\_check\_user(check\_user\_id1 in number,status1 in varchar2,message out varchar2)  is  begin  update check\_user set status = status1 where check\_user\_id = check\_user\_id1;  message :='updated';  dbms\_output.put\_line(message);  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end update\_check\_user;  --3  procedure insert\_director(director\_name1 in director.director\_name%type,film\_name1 in director.film\_name%type,director\_opis1 in director.director\_opis%type,message out varchar2)  is  num\_film number;  index\_name varchar2(40) := 'film\_namex';  begin  select count(\*) into num\_film from films where film\_name = film\_name1;  if num\_film = 1  then  insert into director(director\_name,film\_name,director\_opis) values(director\_name1,film\_name1,director\_opis1);  message := 'data add';  dbms\_output.put\_line(message);  execute immediate 'alter index ' || index\_name || ' rebuild';  else  message := 'no search film';  dbms\_output.put\_line(message);  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end insert\_director;  --4  procedure delete\_director(director\_name1 in director.director\_name%type,message out varchar2)  is  num\_director number;  begin  select count(\*)into num\_director from director where director\_name = 'Terrence Malick';  if num\_director > 0  then  delete director where director\_name = director\_name1;  message := 'data delete';  dbms\_output.put\_line(message);  else  message := 'no search film';  dbms\_output.put\_line(message);  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end delete\_director;  --5  procedure insert\_film(film\_name1 in varchar2, director1 in varchar2, genre\_name1 in varchar2,duration\_film1 in number,year\_film1 in varchar2,film\_opis1 in varchar2,message out varchar2)  is  year\_fi date := to\_date(year\_film1,'YYYY');  len number;  namee films.film\_name%type;  begin  SELECT INITCAP(film\_name1) into namee FROM DUAL;  SELECT LENGTH(year\_film1) into len FROM DUAl;  if len = 4  then  insert into films(film\_name,director,genre\_name,duration\_film,year\_film,film\_opis) values(namee,director1,genre\_name1,duration\_film1,year\_fi,film\_opis1);  message := 'data add';  dbms\_output.put\_line(message);  else  message := 'date bad';  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end insert\_film;  --6  procedure delete\_film(film\_name1 in films.film\_name%type,message out varchar2)  is  num number;  num\_session number;  begin  select count(\*) into num\_session from sessions where film\_name = film\_name1;  select count(\*) into num from films where film\_name = film\_name1;  if num = 1  then  if num\_session = 0  then  DELETE FROM films WHERE film\_name = film\_name1;  message := 'data delete';  dbms\_output.put\_line(message);  else  message := 'film use session';  dbms\_output.put\_line(message);  end if;  else  message := 'no search film';  dbms\_output.put\_line(message);  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end delete\_film;  --7  procedure insert\_sessions(  hall\_id1 in sessions.hall\_id%type  ,film\_name1 in sessions.film\_name%type  ,session\_start1 in varchar2  ,session\_price1 in sessions.session\_price%type  ,session\_status1 in sessions.session\_status%type  ,message out varchar2)  is  max\_id number;  duration1 number;  price varchar2(20);  num\_film number;  datee date := to\_date(session\_start1,'DD.MM.YYYY');  begin  price := session\_price1 || '$';  select count(\*) into num\_film from films where film\_name = film\_name1 and rownum = 1;  if num\_film > 0  then  select duration\_film into duration1 from films where film\_name = film\_name1;  select max(session\_id)into max\_id from sessions;  insert into sessions(session\_id,cinema\_name,hall\_id,film\_name,session\_start,session\_time,session\_price,session\_status) values(max\_id+1,'an3all',hall\_id1,film\_name1,datee,duration1,price,session\_status1);  message := 'data insert';  dbms\_output.put\_line(message);  else  message := 'no search film';  dbms\_output.put\_line(message);  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end insert\_sessions;  --8  procedure delete\_sessions(session\_id1 in number,message out varchar2)  is  num\_session number;  begin  select count(\*) into num\_session from sessions where session\_id=session\_id1;  if num\_session = 1  then  delete sessions where session\_id = session\_id1;  message := 'data delete';  dbms\_output.put\_line(message);  else  message := 'no search session';  dbms\_output.put\_line(message);  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end delete\_sessions;  --9  procedure update\_sessions(session\_id1 in number, session\_status1 in sessions.session\_status%type,message out varchar2)  is  num\_session number;  begin  select count(\*) into num\_session from sessions where session\_id=session\_id1;  if num\_session = 1  then  update sessions set session\_status = session\_status1 where session\_id = session\_id1;  message := 'data update';  dbms\_output.put\_line(message || 'изменено на: '||session\_status1);  else  message := 'no search session';  dbms\_output.put\_line(message);  end if;  EXCEPTION  WHEN OTHERS THEN  message :='error';  dbms\_output.put\_line(message||' '||sqlerrm);  end update\_sessions;  --10  procedure session\_status  is cursor curs is select session\_start from sessions;  sysdates date;  dates date;  begin  for fcurs in curs  loop  if sysdate >= fcurs.session\_start  then  update sessions set session\_status = 'started' where session\_start = fcurs.session\_start;  end if;  end loop;  EXCEPTION  WHEN OTHERS THEN  dbms\_output.put\_line(sqlerrm);  end;  end PACK\_ROOT; |
|  |
|  |

## Приложение Б

Пакет PACK\_USER

|  |
| --- |
|  |
| create or replace package body PACK\_USER  as  ---1  procedure Info\_Sessions\_start(strings out sys\_refcursor,status in varchar2)  is  begin  open strings for  select \* from sessions where session\_status = status;  end Info\_Sessions\_start;  ----2  procedure Check\_session(session\_id1 in number,message out varchar2,status out varchar2)  is  num number;  begin  select count(\*) into num from sessions where session\_id = session\_id1;  if num = 1  then  message:='good';  status := 'true';  else  message:='no search session';  status := 'false';  end if;  end Check\_session;  ----3  procedure search\_place(session\_id1 in number,places out sys\_refcursor,count\_place1 out halls.count\_place%type,filmname out sessions.film\_name%type)  is  cursor curs is select place\_number from BOOKING where session\_id = session\_id1;  i number := 1;  begin  select halls.count\_place, sessions.film\_name into count\_place1,filmname from sessions join halls on sessions.hall\_id = halls.hall\_id where sessions.session\_id = session\_id1;  dbms\_output.put\_line('Количество мест: ' || count\_place1);  dbms\_output.put\_line('Фильм: ' || filmname);  dbms\_output.put\_line('Номера занятых мест:');  for fcurs in curs  loop  dbms\_output.put\_line(fcurs.place\_number);  end loop;  open places for select place\_number from BOOKING where session\_id = session\_id1;  end search\_place;  ---4  procedure BOKKING\_USER(user\_name1 varchar2,session\_id1 number,place\_number1 number,message out varchar2)  is  user\_num number;  session\_num number;  session\_status varchar2(12);  place\_count number;  check\_count\_place number;  begin  select a.count\_place into check\_count\_place from halls a join sessions b on a.hall\_id = b.hall\_id where b.session\_id = session\_id1 and ROWNUM = 1;  select count(\*) into user\_num from users where user\_name = user\_name1;  select count(\*) into session\_num from sessions where session\_id = session\_id1;  select session\_status into session\_status from sessions where session\_id = session\_id1;  select count(\*) into place\_count from BOOKING where session\_id = session\_id1 and place\_number = place\_number1;  if place\_number1 < check\_count\_place+1  then  dbms\_output.put\_line('Место существует');  if user\_num > 0  then  dbms\_output.put\_line('хорошо');  if session\_num > 0  then  dbms\_output.put\_line('Сеанс есть');  if session\_status = 'not started'  then  dbms\_output.put\_line('сеанс не начался');  if place\_count = 0  then  dbms\_output.put\_line('место свободно и забранированно');  message := 'you reserved place';  insert into BOOKING(user\_name,session\_id,place\_number) values(user\_name1,session\_id1,place\_number1);  else  dbms\_output.put\_line('место занято');  message := 'place occupied';  end if;  else  dbms\_output.put\_line('сеанс начался');  message := 'session start';  end if;  else  dbms\_output.put\_line('такого сеанса нет');  message := 'no search session';  end if;  else  dbms\_output.put\_line('Такого логина нет');  message := 'loggin incorrect';  end if;  else  dbms\_output.put\_line('Места не существует');  message := 'no search place';  end if;  exception  when others then  dbms\_output.put\_line(sqlerrm);  end BOKKING\_USER;  ---5  procedure Info\_films(strings out sys\_refcursor)  is  cursor curs is select \* from films;  begin  for fcurs in curs  loop  dbms\_output.put\_line('Имя: '||fcurs.Film\_name||' Режисер: '||fcurs.Director||' Жанр: '||fcurs.Genre\_name||' Продолжительность: '||fcurs.Duration\_film||' Год: '||fcurs.Year\_film||' Описание: '||fcurs.Film\_opis);  end loop;  open strings for  select \* from Films;  end Info\_films;  ---6  procedure Info\_Genre(Genre\_name1 in varchar, strings out sys\_refcursor)  is  cursor curs is select \* from genre where Genre\_name = Genre\_name1;  begin  for fcurs in curs  loop  dbms\_output.put\_line('Имя: '||fcurs.Film\_name||' Жанр: '||fcurs.Genre\_name);  end loop;  open strings for  select \* from Genre where Genre\_name = Genre\_name1;  end Info\_Genre;  ----7  procedure Info\_Direcor(Film\_name1 in varchar2, strings out sys\_refcursor,message out varchar2)  is  cursor curs is select \* from director where contains(Film\_name, '%'||Film\_name1||'%') > 0;  num number;  begin  select count(\*) into num from director where contains(Film\_name, '%'||Film\_name1||'%') > 0;  message := 'no search';  if num = 0  then  message :='no search';  dbms\_output.put\_line(message);  else  for fcurs in curs  loop  dbms\_output.put\_line('Имя: '||fcurs.Director\_name||' Название фильма: '||fcurs.Film\_name||' Описание директора: '||fcurs.Director\_opis);  end loop;  open strings for  select \* from director where contains(Film\_name, '%'||Film\_name1||'%') > 0;  message :='good';  end if;  exception  when others  then  raise\_application\_error(-20001,'An error was encountered - '||SQLCODE||' -ERROR- '||SQLERRM);  end Info\_Direcor;  ---8  procedure search\_film\_for\_name(Film\_name1 in varchar2,strings out sys\_refcursor,message out varchar2)  is  cursor curs is select \* from films where contains(Film\_name, '%'||Film\_name1||'%') > 0;  num number;  index\_name varchar2(20) := 'film\_name\_film';  begin  execute immediate 'alter index ' || index\_name || ' rebuild';  select count(\*) into num from films where contains(Film\_name, '%'||Film\_name1||'%') > 0;  message := 'no search';  if num = 0  then  message :='no search';  dbms\_output.put\_line(message);  else  for fcurs in curs  loop  dbms\_output.put\_line('Название: '||fcurs.Film\_name||' Режисер: '||fcurs.Director||' Жанр: '||fcurs.Genre\_name||' Продолжительность: '||fcurs.Duration\_film||' Год: '||fcurs.Year\_film||' Описание: '||fcurs.Film\_opis);  end loop;  open strings for  select \* from films where contains(Film\_name, '%'||Film\_name1||'%') > 0;  message :='good';  end if;  exception  when others  then  raise\_application\_error(-20001,'An error was encountered - '||SQLCODE||' -ERROR- '||SQLERRM);  end search\_film\_for\_name;  ---9  procedure info\_cinema\_v(strings out sys\_refcursor)  is cursor curs is select \* from cinema\_v;  begin  for fcurs in curs  loop  dbms\_output.put\_line('Название: '||fcurs.cinema\_name||' Номер телефона: '||fcurs.cinema\_telephone||' где находится: '||fcurs.cinema\_locate);  end loop;  open strings for  select \* from cinema\_v;  exception  when others  then  raise\_application\_error(-20001,'An error was encountered - '||SQLCODE||' -ERROR- '||SQLERRM);  end info\_cinema\_v;  --10  procedure info\_halls\_v(strings out sys\_refcursor)  is cursor curs is select \* from halls\_v;  begin  for fcurs in curs  loop  dbms\_output.put\_line('Id: '||fcurs.hall\_id||' Название: '||fcurs.hall\_name||' количество мест: '||fcurs.count\_place);  end loop;  open strings for  select \* from halls\_v;  exception  when others  then  raise\_application\_error(-20001,'An error was encountered - '||SQLCODE||' -ERROR- '||SQLERRM);  end info\_halls\_v;  --11  procedure check\_booking\_user(user\_name1 in varchar2,cur out sys\_refcursor)  is cursor curs is select c.session\_id,c.film\_name,c.session\_start,c.session\_price,b.place\_number from users a join booking b on a.user\_name = b.user\_name join sessions c on b.session\_id = c.session\_id where a.user\_name = user\_name1 and c.session\_status = 'not started';  begin  for fcurs in curs  loop  dbms\_output.put\_line('ID: '||fcurs.session\_id||' Название фильма: '||fcurs.film\_name||' Начало сеанса: '||fcurs.session\_start||' Цена: '||fcurs.session\_price||' Номер места: '||fcurs.place\_number);  end loop;  open cur for  select c.session\_id,c.film\_name,c.session\_start,c.session\_price,b.place\_number from users a join booking b on a.user\_name = b.user\_name join sessions c on b.session\_id = c.session\_id where a.user\_name = user\_name1 and c.session\_status = 'not started';  exception  when others  then  raise\_application\_error(-20001,'An error was encountered - '||SQLCODE||' -ERROR- '||SQLERRM);  end check\_booking\_user;  end PACK\_USER; |
|  |